



BEA WebLogic Integration™

アダプタの開発

著作権

Copyright © 2002, BEA Systems, Inc. All Rights Reserved.

限定的権利条項

本ソフトウェアおよびマニュアルは、**BEA Systems, Inc.** 又は日本ビー・イー・エー・システムズ株式会社（以下、「**BEA**」といいます）の使用許諾契約に基づいて提供され、その内容に同意する場合にのみ使用することができ、同契約の条項通りにのみ使用またはコピーすることができません。同契約で明示的に許可されている以外の方法で同ソフトウェアをコピーすることは法律に違反します。このマニュアルの一部または全部を、**BEA Systems, Inc.** からの書面による事前の同意なしに、複写、複製、翻訳、あるいはいかなる電子媒体または機械可読形式への変換も行うことはできません。

米国政府による使用、複製もしくは開示は、**BEA** の使用許諾契約、および FAR 52.227-19 の「Commercial Computer Software-Restricted Rights」条項のサブパラグラフ (c)(1)、DFARS 252.227-7013 の「Rights in Technical Data and Computer Software」条項のサブパラグラフ (c)(1)(ii)、NASA FAR 補遺 16-52.227-86 の「Commercial Computer Software--Licensing」条項のサブパラグラフ (d)、もしくはそれらと同等の条項で定める制限の対象となります。

このマニュアルに記載されている内容は予告なく変更されることがあり、また **BEA** による責務を意味するものではありません。本ソフトウェアおよびマニュアルは「現状のまま」提供され、商品性や特定用途への適合性を始めとする（ただし、これらには限定されない）いかなる種類の保証も与えません。さらに、**BEA** は、正当性、正確さ、信頼性などについて、本ソフトウェアまたはマニュアルの使用もしくは使用結果に関していかなる確約、保証、あるいは表明も行いません。

商標または登録商標

BEA、**Jolt**、**Tuxedo**、および **WebLogic** は **BEA Systems, Inc.** の登録商標です。**BEA Builder**、**BEA Campaign Manager for WebLogic**、**BEA eLink**、**BEA Manager**、**BEA WebLogic Commerce Server**、**BEA WebLogic Enterprise**、**BEA WebLogic Enterprise Platform**、**BEA WebLogic Express**、**BEA WebLogic Integration**、**BEA WebLogic Personalization Server**、**BEA WebLogic Platform**、**BEA WebLogic Portal**、**BEA WebLogic Server**、**BEA WebLogic Workshop** および **How Business Becomes E-Business** は、**BEA Systems, Inc** の商標です。

その他の商標はすべて、関係各社が著作権を有します。

アダプタの開発

パート番号	日付	ソフトウェアのバージョン
なし	2002年6月	7.0

目次

このマニュアルの内容

対象読者	xiv
e-docs Web サイト	xv
このマニュアルの印刷方法	xv
関連情報	xvi
サポート情報	xvi
表記規則	xvii

1. ADK の概要

この章の目的	1-1
ADK の特徴	1-2
アダプタ開発の前提条件	1-2
ADK が提供する機能	1-3
アダプタの特徴	1-3
サービス アダプタ	1-4
イベント アダプタ	1-5
WebLogic Integration に限定されない J2EE 準拠アダプタ	1-6
設計時 GUI	1-6
アプリケーション ビュー	1-7
フレームワークのパッケージ化	1-7
始める前に	1-8

2. 基本開発コンセプト

実行時と設計時	2-1
実行時フレームワーク	2-2
設計時フレームワーク	2-2
イベントおよびサービス	2-3
イベントの概要	2-3
サービスの概要	2-4
アダプタによるロギングの使い方	2-5
ロギング ツールキット	2-5

ロギング フレームワーク	2-5
インターナショナルライゼーションとローカライゼーション	2-6
アダプタ論理名	2-6
アダプタ論理名の使用場所	2-7
アダプタのデプロイメントでアダプタ論理名を使用する	2-7
アダプタ論理名を構成基準として使用する	2-9
getAdapterLogicalName() の戻り値として使用されるアダプタ論理名 2-10	
共有接続ファクトリ	2-11
起動時の接続ファクトリへの参照	2-11
エンタープライズアーカイブ (EAR) ファイル	2-11
アダプタによる同時要求の処理方法	2-13
3. 開発ツール	
サンプル アダプタ	3-1
サンプル アダプタの使用目的	3-1
サンプル アダプタの内容	3-2
GenerateAdapterTemplate ユーティリティ	3-3
ADK Javadoc	3-3
Ant ベースのビルド プロセス	3-4
Ant を使用する理由	3-4
XML ツール	3-5
4. カスタム開発環境の作成	
アダプタ設定ワークシート	4-1
GenerateAdapterTemplate の使い方	4-2
手順 1. GenerateAdapterTemplate の実行	4-2
手順 1a: コンソール コード ページの指定 (Windows のみ)	4-5
手順 2. ツリーの再構築	4-5
手順 3. アダプタの WebLogic Integration へのデプロイ	4-6
5. ロギング ツールキットの使い方	
ロギング ツールキット	5-2
ロギング コンフィグレーション ファイル	5-2
ロギング コンセプト	5-3
メッセージ カテゴリ	5-3

メッセージ優先度	5-4
カテゴリへの優先度の割り当て	5-5
メッセージアペンダ	5-6
メッセージレイアウト	5-7
コンポーネントの結合	5-8
ロギングの設定方法	5-9
ロギング フレームワークのクラス	5-11
com.bea.logging.ILogger	5-11
com.bea.logging.LogContext	5-12
com.bea.logging.LogManager	5-12
ログ メッセージのインターナショナルライゼーションとローカライゼーション	5-16
マルチスレッド コンポーネントでのコンテキスト情報の保存	5-16

6. サービス アダプタの開発

WebLogic Integration に限定されない J2EE 準拠アダプタ	6-2
実行時環境におけるサービス アダプタ	6-2
イベントの処理フロー	6-5
手順 1 : 環境要件の調査	6-6
手順 2 : 開発環境のコンフィグレーション	6-7
手順 2a : ディレクトリ構造の設定	6-7
ディレクトリ構造の変更	6-9
手順 2b : アダプタ論理名の割り当て	6-10
手順 2c : ビルド プロセスの設定	6-10
Manifest ファイル	6-11
build.xml のコンポーネント	6-12
手順 2d : メッセージ バンドルの作成	6-24
手順 3 : SPI の実装	6-24
基本的な SPI の実装	6-24
ManagedConnectionFactory	6-25
トランザクション境界設定	6-25
ADK 実装	6-27
開発時に必要な AbstractManagedConnectionFactory プロパティ	6-32
ManagedConnection	6-34
ADK 実装	6-34

ManagedConnectionMetaData.....	6-35
ADK 実装	6-35
ConnectionEventListener.....	6-36
ADK 実装	6-36
ConnectionManager.....	6-36
ADK 実装	6-37
ConnectionRequestInfo.....	6-37
ADK 実装	6-37
LocalTransaction.....	6-37
ADK 実装	6-38
手順 4 : CCI の実装	6-38
この節の構成.....	6-38
基本的な CCI の実装	6-39
Connection	6-39
ADK 実装	6-40
Interaction.....	6-40
ADK 実装	6-41
XCCI を使用した CCI の実装	6-43
サービス	6-43
DocumentRecord	6-45
IDocument.....	6-45
ADK 付属の XCCI クラス.....	6-47
XCCI の設計パターン	6-48
非 XML J2EE 準拠アダプタの使用	6-49
ConnectionFactory.....	6-50
ADK 実装	6-50
ConnectionMetaData.....	6-51
ADK 実装	6-51
ConnectionSpec.....	6-51
ADK 実装	6-52
InteractionSpec.....	6-52
ADK 実装	6-53
LocalTransaction.....	6-54
Record	6-54
ADK 実装	6-55

ResourceAdapterMetaData	6-56
ADK 実装	6-56
手順 5 : アダプタのテスト	6-57
テスト支援機能の使用方法	6-57
ADK が提供するテスト ケースの拡張.....	6-58
sample.spi.NonManagedScenarioTestCase	6-58
sample.event.OfflineEventGeneratorTestCase	6-59
sample.client.ApplicationViewClient	6-59
手順 6 : アダプタのデプロイ	6-60

7. イベント アダプタの開発

イベント アダプタの概要	7-1
実行時環境におけるイベント アダプタ	7-2
イベントのフロー	7-4
手順 1 : アダプタの定義	7-5
手順 2 : 開発環境のコンフィグレーション	7-5
手順 2a : ファイル構造の設定	7-6
手順 2b : アダプタへの論理名の割り当て.....	7-6
手順 2c : ビルド プロセスの設定	7-6
手順 2d : メッセージ バンドルの作成	7-7
手順 2e : ロギングのコンフィグレーション	7-7
イベント生成ロギング カテゴリの作成.....	7-7
手順 3 : アダプタの実装	7-8
手順 3a : イベント ジェネレータの作成.....	7-9
データ抽出メカニズムの実装	7-9
イベント ジェネレータの実装方法	7-12
手順 3b : データ変換メソッドの実装	7-18
手順 4 : アダプタのテスト	7-20
手順 5 : アダプタのデプロイ	7-20

8. 設計時 GUI の開発

設計時フォーム処理の概要	8-2
フォーム処理クラス	8-3
RequestHandler	8-3
ControllerServlet.....	8-4
ActionResult	8-4

Word と子孫クラス.....	8-4
AbstractInputTagSupport と子孫クラス.....	8-5
フォーム処理シーケンス.....	8-6
前提条件.....	8-6
シーケンスの手順.....	8-7
設計時 GUI の機能.....	8-9
Java Server Pages.....	8-10
JSP テンプレート.....	8-11
JSP タグの ADK ライブラリ.....	8-12
JSP タグの属性.....	8-13
アプリケーション ビュー.....	8-15
ファイル構造.....	8-15
イベントの処理フロー.....	8-17
手順 1 : 設計時 GUI 要件の定義.....	8-19
手順 2 : ページフローの定義.....	8-20
画面 1 : ログイン.....	8-20
画面 2 : アプリケーション ビューの管理.....	8-21
画面 3 : 新しいアプリケーション ビューの定義.....	8-21
画面 4 : 接続のコンフィグレーション.....	8-21
画面 5 : アプリケーション ビューの管理.....	8-22
画面 6 : イベントの追加.....	8-23
画面 7 : サービスの追加.....	8-23
画面 8 : アプリケーション ビューのデプロイ.....	8-24
ユーザ アクセスの制御.....	8-25
アプリケーション ビューのデプロイ.....	8-25
アプリケーション ビューの保存.....	8-25
画面 9 : アプリケーション ビューの概要.....	8-26
手順 3 : 開発環境のコンフィグレーション.....	8-27
手順 3a : メッセージ バンドルの作成.....	8-27
手順 3b : WebLogic Server を再起動せずに JSP を更新する環境のコン フィグレーション.....	8-28
手順 4 : 設計時 GUI の実装.....	8-31
AbstractDesignTimeRequestHandler の拡張.....	8-31
インクルードするメソッド.....	8-32
手順 4a : ManagedConnectionFactory クラスの指定.....	8-33

手順 4b : <code>initServiceDescriptor()</code> の実装	8-33
手順 4c : <code>initEventDescriptor()</code> の実装	8-34
手順 5 : HTML フォームの作成	8-35
手順 5a : <code>confconn.jsp</code> フォームの作成	8-35
ADK タグ ライブラリのインクルード	8-36
ControllerServlet のポスト	8-37
[Form] フィールドのラベルの表示	8-38
テキスト フィールドのサイズの表示	8-38
フォームの [Submit] ボタンの表示	8-38
confconn() の実装	8-38
手順 5b : <code>addevent.jsp</code> フォームの作成	8-39
ADK タグ ライブラリのインクルード	8-39
ControllerServlet のポスト	8-39
[Form] フィールド ラベルの表示	8-40
テキスト フィールドのサイズの表示	8-40
フォームの [Submit] ボタンの表示	8-40
フィールドの追加	8-41
手順 5c : <code>addservc.jsp</code> フォームの作成	8-41
ADK タグ ライブラリのインクルード	8-41
ControllerServlet のポスト	8-42
[Form] フィールド ラベルの表示	8-42
テキスト フィールドのサイズの表示	8-42
フォームの [Submit] ボタンの表示	8-42
フィールドの追加	8-43
手順 5d : イベントおよびサービスの編集機能の実装 (省略可能)	8-43
アダプタ プロパティ ファイルの更新	8-43
edtservc.jsp と addservc.jsp の作成	8-45
メソッドの実装	8-46
手順 5e : Web アプリケーションのデプロイメント記述子 (WEB-INF/web.xml) の記述	8-47
手順 6 : ルック & フィールの実装	8-50
手順 7 : サンプルアダプタの設計時インタフェースのテスト	8-51
ファイルとクラス	8-51
テストの実行	8-52

9. アダプタのデプロイ

エンタープライズ アーカイブ (EAR) ファイルの使用	9-1
EAR ファイルにおける共有 JAR ファイルの使い方	9-2
EAR ファイルのデプロイメント記述子	9-3
WebLogic Server Administration Console を使用したアダプタのデプロイ ..	9-4
アダプタの自動登録	9-5
命名規約の使用	9-5
テキスト ファイルの使用	9-6
Web アプリケーションのデプロイメント記述子の編集	9-6
デプロイメント パラメータ	9-7
デプロイメント記述子の編集	9-7
WebLogic Integrator クラスタでのアダプタのデプロイメント	9-10

A. WebLogic Integration に限定されないアダプタの作成

この節の目的	A-1
アダプタの構築	A-2
構築プロセスの更新	A-3

B. XML Toolkit

Toolkit パッケージ	B-1
IDocument	B-2
Schema Object Model (SOM)	B-3
SOM の仕組み	B-4
スキーマの作成	B-5
結果として作成されるスキーマ	B-8
XML ドキュメントの検証	B-10
ドキュメントの検証方法	B-11
isValid() の実装	B-11
isValid() の実装例	B-12

C. WebLogic Integration 7.0 へのアダプタの移行

WebLogic Integration 7.0 ADK 向けのアダプタの再構築	C-1
アプリケーション統合 CLASSPATH およびアダプタのパッケージ化の変更 ..	C-2
アダプタによる共有接続ファクトリ ユーザ インタフェースのサポートの許 可	C-3

セキュリティ制約とログイン コンフィグレーションにおける変更	C-4
要求データを必要としないサービスに対する DBMS サンプルアダプタの変更	C-5
WebLogic Integration 7.0 での WebLogic Integration 2.1 アダプタの使用...	C-5

D. アダプタ設定ワークシート

アダプタ設定ワークシート	D-2
--------------------	-----

E. DBMS サンプル アダプタを使用したアダプタ開発方法の学習

DBMS サンプル アダプタの概要	E-1
DBMS サンプル アダプタの仕組み	E-2
始める前に	E-3
DBMS サンプル アダプタへのアクセス	E-3
DBMS サンプル アダプタ ツアー	E-3
DBMS サンプル アダプタの開発工程	E-25
手順 1 : DBMS サンプル アダプタについて	E-25
手順 2 : 環境の定義	E-26
手順 3 : Server Provider Interface パッケージの実装	E-28
ManagedConnectionFactoryImpl	E-29
ManagedConnectionImpl	E-30
ConnectionMetaDataImpl	E-31
LocalTransactionImpl	E-33
手順 4 : Common Client Interface パッケージの実装	E-34
ConnectionImpl	E-34
InteractionImpl	E-35
InteractionSpecImpl	E-37
手順 5 : イベント パッケージの実装	E-38
EventGenerator	E-38
手順 6 : DBMS サンプル アダプタのデプロイ	E-40
手順 6a : 環境のセットアップ	E-40
手順 6b : ra.xml ファイルの更新	E-40
手順 6c : RAR ファイルの作成	E-41
手順 6d : JAR および EAR ファイルの構築	E-41
手順 6e : EAR ファイルの作成とデプロイ	E-42
DBMS サンプル アダプタの設計時 GUI の開発工程	E-44

手順 1 : 要件の決定	E-44
手順 2 : 必要な Java Server Pages の決定	E-45
手順 3 : メッセージバンドルの作成	E-46
手順 4 : 設計時 GUI の実装	E-47
手順 5 : Java Server Pages の記述	E-49
カスタム JSP Tags の使用	E-49
オブジェクトのステートの保存	E-49
WEB-INF/web.xml のデプロイメント記述子の記述	E-50

索引

このマニュアルの内容

このマニュアル（アダプタの開発）の内容は以下のとおりです。

- 「ADK の概要」では、**WebLogic Integration Adapter Development Kit** に関する基本的な情報について説明します。サービス アダプタ、イベント アダプタ、設計時 GUI、およびアダプタの開発前に行うことについて説明します。
- 「基本開発コンセプト」では、アダプタの開発に関する主な **ADK** コンセプトについて説明します。具体的には、イベント、サービス、設計時と実行時の比較、ロギング、アダプタ論理名について説明します。
- 「開発ツール」では、アダプタの構築に使用できる **ADK** ツールについて説明します。具体的には、サンプルアダプタ、`GenerateAdapterTemplate` ユーティリティ、**Ant** ベースのビルド プロセス、**XML** ツール、**Javadoc** といったツールがあります。
- 「カスタム開発環境の作成」では、`GenerateAdapterTemplate` ユーティリティを使用して、サンプル アダプタを複製する方法、および新しいアダプタに合った開発環境のカスタマイズ方法について説明します。
- 「ロギング ツールキットの使い方」では、**ADK** ロギング ツールキットを使用して、ロギングを実装する方法について説明します。また、**ADK** ロギング フレームワークのコアとなる **Apache log4j** プロジェクトについても説明します。
- 「サービス アダプタの開発」では、サービスをサポートするアダプタの構築方法について説明します。詳しい手順と関連するコード サンプルの両方を提供します。
- 「イベント アダプタの開発」では、イベントをサポートするアダプタの構築方法について説明します。詳しい手順と関連するコード サンプルの両方を提供します。
- 「設計時 GUI の開発」では、アプリケーション ビューの定義、デプロイ、テストの際にアダプタ ユーザが必要とするグラフィカル ユーザ インタフェイスのビルド方法について説明します。詳しい手順と関連するコード サンプルの両方を提供します。

-
- 「アダプタのデプロイ」では、**WebLogic Integration** にアダプタをデプロイするときの手順について説明します。アダプタを手動でデプロイする手順、および **WebLogic Server Administration Console** からデプロイする手順を示します。
 - 「**WebLogic Integration** に限定されないアダプタの作成」では、第 6 章「サービス アダプタの開発」および第 7 章「イベント アダプタの開発」で説明した手順を変更して、**WebLogic Integration** 以外の **WebLogic Server** で使用できるアダプタを開発する方法について説明します。
 - 「**XML Toolkit**」では、XML ドキュメントの作成に利用できる **WebLogic Integration** のツールについて説明します。
 - 「**WebLogic Integration 7.0** へのアダプタの移行」では、**WebLogic Integration** のリリース 2.0 からのアダプタのデプロイ方法の変更について説明します。また、設計時 Web アプリケーションを **WebLogic Integration 2.1** で登録する方法についても説明します。
 - 「アダプタ設定ワークシート」では、実際にコーディングを始める前に、開発するアダプタのコンセプトを把握するためのワークシートを示します。特に、アダプタ論理名や Java パッケージ基本名などのコンポーネントの定義をサポートします。また、メッセージバンドルをローカライズする必要のあるロケールも決定できます。
 - 「**DBMS** サンプルアダプタを使用したアダプタ開発方法の学習」では、ADK を使用して **DBMS** サンプルアダプタを構築する方法について説明します。この節では、**DBMS** アダプタの使用方法について、単純なタスク駆動形式の例も紹介します。

対象読者

『アダプタの開発』は、ADK を使用してサービスアダプタ、イベントアダプタ、およびアプリケーションビューの作成を容易にする設計時 GUI を開発するプログラマを対象としています。

e-docs Web サイト

BEA 製品のドキュメントは、BEA Systems, Inc. の Web サイトで入手できます。BEA のホーム ページで [製品のドキュメント] をクリックするか、または「e-docs」という製品ドキュメント ページ (<http://edocs.beasys.co.jp/e-docs/index.html>) を直接表示してください。

このマニュアルの印刷方法

Web ブラウザの [ファイル | 印刷] オプションを使用すると、Web ブラウザからこのマニュアルを一度に 1 ファイルずつ印刷できます。

このマニュアルの PDF 版は、Web サイトで入手できます。WebLogic IntegrationPDF を Adobe Acrobat Reader で開くと、マニュアルの全体（または一部分）を書籍の形式で印刷できます。PDF を表示するには、WebLogic Integration ドキュメントのホーム ページを開き、[PDF 版] ボタンをクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader がない場合は、Adobe の Web サイト (<http://www.adobe.co.jp/>) で無料で入手できます。

関連情報

以下の関連情報があります。

- BEA WebLogic Server ドキュメント
(<http://edocs.beasys.co.jp/e-docs/index.html>)
- BEA WebLogic Integration のドキュメント
(<http://edocs.beasys.co.jp/e-docs/index.html>)
- Sun Microsystems, Inc. の J2EE コネクタ アーキテクチャの仕様
(<http://java.sun.com/j2ee/connector/>)
- XML スキーマの仕様 (<http://www.w3.org/TR/xmlschema-0/>)

サポート情報

BEA WebLogic Integration のドキュメントに関するユーザからのフィードバックは弊社にとって非常に重要です。質問や意見などがあれば、電子メールで **docsupport-jp@bea.com** までお送りください。寄せられた意見については、WebLogic Integration のドキュメントを作成および改訂する BEA の専門の担当者が直に目を通します。

電子メールのメッセージには、WebLogic Integration 7.0 リリースのドキュメントをご使用の旨をお書き添えください。

本バージョンの BEA WebLogic Integration について不明な点がある場合、または BEA WebLogic Integration のインストールおよび動作に問題がある場合は、BEA WebSupport (<http://websupport.bea.com/custsupp>) を通じて BEA カスタマサポートまでお問い合わせください。カスタマサポートへの連絡方法については、製品パッケージに同梱されているカスタマサポート カードにも記載されています。

カスタマサポートでは以下の情報をお尋ねしますので、お問い合わせの際はあらかじめご用意ください。

- お名前、電子メールアドレス、電話番号、ファクス番号

- 会社の名前と住所
- お使いの機種とコード番号
- 製品の名前とバージョン
- 問題の状況と表示されるエラー メッセージの内容

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
[Ctrl] + [Tab]	複数のキーを同時に押すことを示す。
<i>斜体</i>	強調または書籍のタイトルを示す。
等幅テキスト	コード サンプル、コマンドとそのオプション、データ構造体とそのメンバー、データ型、ディレクトリ、およびファイル名とその拡張子を示す。等幅テキストはキーボードから入力するテキストも示す。 <i>例</i> <pre>#include <iostream.h> void main () the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float</pre>
太字の等幅 テキスト	コード内の重要な箇所を示す。 <i>例</i> <pre>void commit ()</pre>

表記法	適用
斜体の等幅テキスト	コード内の変数を示す。 <i>例</i> <code>String expr</code>
すべて大文字のテキスト	デバイス名、環境変数、および論理演算子を示す。 <i>例</i> <code>LPT1</code> <code>SIGNON</code> <code>OR</code>
{ }	構文の中で複数の選択肢を示す。実際には、この括弧は入力しない。
[]	構文の中で任意指定の項目を示す。実際には、この括弧は入力しない。 <i>例</i> <code>buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...</code>
	構文の中で相互に排他的な選択肢を区切る。実際には、この記号は入力しない。
...	コマンドラインで以下のいずれかを示す。 <ul style="list-style-type: none"> ■ 引数を複数回繰り返すことができる ■ 任意指定の引数が省略されている ■ パラメータや値などの情報を追加入力できる 実際には、この省略記号は入力しない。 <i>例</i> <code>buildobjclient [-v] [-o name] [-f file-list]... [-l file-list]...</code>
.	コード サンプルまたは構文で項目が省略されていることを示す。実際には、この省略記号は入力しない。

1 ADK の概要

このガイドでは、WebLogic Integration Adapter Development Kit (ADK) の使用手順について説明します。具体的には、イベントアダプタ、サービスアダプタ、および設計時ユーザ インタフェースの開発、テスト、デプロイ方法について説明します。

この章の内容は以下のとおりです。

- ADK の特徴
- アダプタの特徴
- 設計時 GUI
- 始める前に

この章の目的

この章では、ADK によるイベントアダプタ、サービスアダプタ、および設計時 GUI の開発方法について概説し、以下の事項の習得を目的とします。

- アダプタの特徴と使用方法
- アダプタ開発の前提条件
- アダプタ開発に関連する用語

ADK の特徴

ADK は、BEA WebLogic Integration がサポートするイベントおよびサービス プロトコルを実装するためのツールセットです。これらのツールは、WebLogic Integration の各種リソースアダプタの開発、テスト、パッケージ化、配布をサポートする複数のフレームワークの集まりとして構成されています。特に、ADK には、4 つの目的のフレームワークがあります。

- 設計時動作
- 実行時動作
- ロギング
- パッケージ化

アダプタ開発の前提条件

ADK は、アダプタの開発に必要な次の 3 つの前提条件を備えています。

- 開発環境構造—どのような統合開発/デバッグ環境 (IDDE) でも、開発プロジェクトの構成が最も重要です。しっかりと構成された開発環境があれば、アダプタのコーディングにすぐに着手できます。ADK では、整った開発環境、ビルドプロセス、テスト体系、および直感的にイメージできるクラス名とクラス階層を提供しています。ADK を使用することで、ビルドプロセスの設計や構成に時間を費やす必要がなくなります。

ADK には非常に多くの先進技術が取り入れられているため、段階的な開発プロセス (コードを少し作成してはテストし、次に進む) が成功への鍵となります。ADK のテストプロセスを用いれば、開発者は簡単に変更を加えて、その変更をすぐにテストできます。

- 実装の周辺的な詳細をできる限り公開しない—周辺的な実装の詳細は、堅牢なソフトウェアプログラムが稼動するフレームワークのサポートに必要なコードセクションです。

たとえば、J2EE コネクタアーキテクチャの仕様では、`javax.resource.cci.InteractionSpec` 実装クラスで、JavaBeans 設計パターンの後にゲッター (取得) およびセッター (設定) メソッドを指定する

必要があります。JavaBeans 設計パターンをサポートするには、使用する実装クラスで `PropertyChangeListeners` および `VetoableChangeListeners` をサポートする必要があります。このサポート方法を習得するために JavaBeans の仕様を学習するのは大変です。それよりも、アダプタに関するエンタープライズ情報システム (EIS) 固有の実装を中心に作業するほうが望ましいことです。ADK は、アダプタを実装するための周縁的な詳細の大半に対応できる基本的な機能を備えています。

- 目的達成への明確なロード マップ 終了条件とは、「実装の完了はどのように判断すればよいのか」という問いに対する回答です。ADK は、アダプタの開発方法を明確な形で提供しています。その開発方法に従えば、開発者は、イベント、サービス、設計時動作、および実行時動作という主要コンセプトを軸にして、アイデアをまとめることができます。この開発方法を使用して、実装完了へのロード マップを形成する終了条件を確立できます。

ADK が提供する機能

ADK は、以下の機能を提供します。

- イベントとサービスの実行時サポート
- アダプタのユーザ インタフェースを WebLogic Integration Application View Console に統合するための API

ADK は、アダプタが単一のグラフィック コンソール アプリケーションの一構成要素となることを可能にし、ビジネス ユーザによる統合ソリューションの構築を実現することで、その価値をさらに高めています。

アダプタの特徴

リソース アダプタ (このマニュアルでは単に「アダプタ」と呼びます) は、相互に通信できるように設計されていないアプリケーションを連結するソフトウェア コンポーネントです。たとえば、ある会社が作成した注文入力システムと別の会社が作成した顧客情報システムとの間で通信を行うには、両者をつなぐアダプタが必要です。

ADK を使用することにより、次の 2 種類のアダプタを作成できます。

- メッセージを消費するサービス アダプタ
- メッセージを生成するイベント アダプタ

また、ADK により、J2EE コネクタ アーキテクチャの仕様に準拠していますが、Weblogic Integration 以外でも使用可能な J2EE 準拠アダプタを作成することもできます。

サービス アダプタ

サービス アダプタは、クライアントから XML 要求ドキュメントを受け取ると、基盤となるエンタープライズ情報システム (EIS) 内の特定の関数を呼び出します。このサービス アダプタはメッセージのコンシューマで、応答を返す場合と返さない場合があります。

サービスの呼び出しには、同期または非同期の 2 つの方法があります。非同期サービス アダプタを使用する場合、クライアント アプリケーションはサービス要求を発行した後も応答を待たずに処理を続行します。同期サービス アダプタを使用する場合は、クライアントは応答を待ってから処理を続行します。BEA WebLogic Integration では、サービス アダプタのこの 2 つの呼び出し方法をサポートしており、アダプタ開発者がこの機能を設定する必要はありません。

サービス アダプタでは、次の 4 つの機能が実行されます。

- 外部クライアントからサービス要求を受信します。
- 要求ドキュメントの XML フォーマットを EIS 固有のフォーマットに変換します。この要求ドキュメントは、サービスの要求 XML スキーマに従って作成されます。また、要求 XML スキーマは、EIS のメタデータに基づいています。
- EIS 内の該当する関数を呼び出し、応答を待ちます。
- EIS 固有のデータ形式からサービスの応答 XML スキーマに従った XML ドキュメントに、応答を変換します。応答 XML スキーマは、EIS のメタデータに基づいています。

イベントと同じように ADK にはすべてのサービスアダプタに共通するこの 4 つの機能に関する要素が実装されています。

サービスアダプタの開発方法については、第6章「サービスアダプタの開発」を参照してください。

イベント アダプタ

イベントアダプタは、EIS から WebLogic Server に情報を伝播します。このタイプのアダプタは、情報のパブリッシャと呼ばれます。

イベント アダプタには、プロセス内とプロセス外の2つの基本タイプがあります。プロセス内イベント アダプタは、EIS と同じプロセス内で機能します。プロセス外アダプタは、別のプロセス内で機能します。プロセス内イベント アダプタとプロセス外イベント アダプタの違いは、データ抽出処理の実行方法だけです。

WebLogic Integration 環境で稼働するイベント アダプタでは、次の3つの機能が実行されます。

- 外部のパーティにとって関心があると思われるイベントが稼働中の EIS で発生した場合に応答し、イベントに関するデータを EIS から抽出します。
- イベントの EIS 固有のデータ形式からイベントの XML スキーマに従った XML ドキュメントに、イベント データを変換します。XML スキーマは、EIS のメタデータに基づいています。
- アプリケーションビューから読み込んだイベント コンテキストに、各イベントを伝播します。

ADK には、すべてのイベント アダプタに共通するこの3つの機能に関する要素が実装されています。したがって、イベント アダプタ開発で、開発者は EIS 固有の箇所を中心に作業ができます。このコンセプトは、エンタープライズ JavaBean (EJB) の背景にあるコンセプトと同じです。コンテナが EJB 開発者に対してシステムレベルのサービスを提供するので、開発者はビジネス アプリケーション ロジックの実装に集中できます。

イベント アダプタの開発方法については、第7章「イベント アダプタの開発」を参照してください。

WebLogic Integration に限定されない J2EE 準拠アダプタ

このタイプのアダプタは **WebLogic Integration** 専用には設計されておらず、J2EE コネクタ アーキテクチャの仕様をサポートしているアプリケーション サーバであれば、どのアプリケーション サーバにでもプラグ インできます。このタイプのアダプタは、サービス アダプタの開発手順を少し変更するだけで開発できます。WebLogic Integration に限定されないアダプタの開発については、**附録 A 「WebLogic Integration に限定されないアダプタの作成」**を参照してください。

設計時 GUI

イベント アダプタとサービス アダプタの他にも、ADK の設計時フレームワークは、Web ベースの GUI を構築するのに使用するツールを提供しています。Web ベースの GUI は、アダプタのユーザがアプリケーション ビュー（後述の「アプリケーション ビュー」を参照）を定義、デプロイ、およびテストする際に必要となるものです。各アダプタには EIS 固有の機能がありますが、アプリケーション ビューをデプロイするには、すべてのアダプタで GUI が必要です。設計時フレームワークでは、主に 2 つのコンポーネントを使用して、各インタフェースの作成やデプロイメントの作業を最小限に抑えることができます。

- **Java Server Pages (JSP)** を使用して HT する Web アプリケーション コンポーネント。このコンポーネントの機能は、JSP テンプレート、JSP タグ ライブラリ、JavaScript ライブラリなどのツールによって拡張できます。
- **AbstractDesignTimeRequestHandler** という名称のデプロイメント ヘルパー コンポーネント。このコンポーネントは、WebLogic Server 上のアプリケーション ビューをデプロイ、アンデプロイ、および編集するための簡便な API を提供します。

設計時 GUI の開発方法については、第 8 章「設計時 GUI の開発」を参照してください。

アプリケーション ビュー

アダプタは、アプリケーションの全機能に対するシステムレベルのインタフェースと言えるのに対し、アプリケーション ビューはアプリケーション中の特定機能に対するビジネスレベルのインタフェースを表しています。

アプリケーション ビューは、単一のビジネスの目的のためにコンフィグレーションされ、そのビジネスの目的に関連したサービスだけが定義されます。それらのサービスは、要求ドキュメントで指定されるビジネス関連データだけを要求し、応答ドキュメントでビジネス関連データだけを返します。アプリケーション ビューは、ユーザの介入なしに、このビジネス関連データをアダプタ内に格納されているメタデータに結合します。アダプタは、このビジネス関連データと格納されているメタデータの両方を使用して、システムレベルの機能をアプリケーションに対して実行します。

また、アプリケーション ビューは、特定のビジネスの目的をサポートするイベントとサービスの両方を表します。これによって、ビジネス ユーザは、アプリケーション ビューを通じてアプリケーションと通信できます。この双方向通信は、実際にはイベント アダプタとサービス アダプタの2つのアダプタ コンポーネントによってサポートされています。アプリケーション ビューを使用することで、ユーザがこれを意識することなく、アプリケーションに対する統合ビジネス インタフェースが実現します。

アプリケーション ビューの詳細については、『*Application Integration ユーザーズ ガイド*』の「Application Integration ユーザーズ ガイドの概要」を参照してください。

フレームワークのパッケージ化

ADK のパッケージ化フレームワークは、顧客への配信用アダプタをパッケージ化するためのツールセットです。すべてのアダプタが、1つの WebLogic Server で同様にインストール、コンフィグレーション、およびアンインストールされるのが理想的です。また、すべてのサービス アダプタは J2EE 準拠である必要があります。パッケージ化フレームワークを使用すると、J2EE アダプタ アーカイブ (RAR) ファイル、Web アプリケーション アーカイブ (WAR) ファイル、エンタープライズ アーカイブ (EAR) ファイル、および WebLogic Integration 設計環境アーカイブを簡単に設計できるようになります。

始める前に

開発作業を始める前に、ご使用のコンピュータに **WebLogic Integration** がインストールされているか確認します。詳細については、『*BEA WebLogic Platform インストールガイド*』および『*BEA WebLogic Integration リリースノート*』を参照してください。

2 基本開発コンセプト

この章では、アダプタまたは設計時 GUI の開発を行う前に知っておく必要がある基本的なコンセプトについて説明します。この章の内容は以下のとおりです。

- 実行時と設計時
- イベントおよびサービス
- アダプタによるロギングの使い方
- アダプタ論理名
- 共有接続ファクトリ
- エンタープライズ アーカイブ (EAR) ファイル
- アダプタによる同時要求の処理方法

実行時と設計時

アダプタ アクティビティという言葉は、実行時および設計時アクティビティの両方を指します。実行時アクティビティは、アダプタ プロセスの実行です。アダプタのユーザによって実行される設計時アクティビティには、アプリケーションビューの作成、デプロイメントおよびテストが含まれます。

実行時および設計時アクティビティは、それぞれ ADK の実行時および設計時フレームワークによってサポートされています。実行時フレームワークは、アダプタを開発するためのツールで構成され、設計時フレームワークは Web ベースのユーザ インタフェースを設計するためのツールで構成されます。実行時と設計時アクティビティについては、この後でさらに詳しく説明します。

実行時フレームワーク

実行時フレームワークは、イベントアダプタおよびサービスアダプタの開発に使用するツールセットです。イベントアダプタの開発をサポートするため、実行時フレームワークでは、拡張可能な基本イベントジェネレータを提供しています。また、サービスアダプタの開発用に、J2EEに完全に準拠したアダプタを提供しています。

実行時フレームワークが提供するクラスには、以下の利点があります。

- J2EEではなく、EISの詳細を中心に作業ができます。
- ADK ロギングフレームワークを使用するのに必要な作業を最小限に抑えます。
- J2EE コネクタアーキテクチャを単純化します。
- 複数のアダプタで使用される冗長なコードを最小限に抑えます。

さらに、実行時フレームワークでは、イベントジェネレータの実装に役立つ抽象基本クラスを提供しています。イベントジェネレータを使用すると、ADK環境におけるイベントサポートが強化されます。

実行時フレームワークの主要コンポーネントは、実行時エンジンです。これはサービスの呼び出しを処理するアダプタコンポーネントのホストとなっており、次の3つのWebLogic Server機能を管理します。

- EISに対する物理接続
- ログイン認証
- トランザクション管理

これらは、すべてJ2EEコネクタアーキテクチャの規格に準拠しています。

設計時フレームワーク

設計時フレームワークは、アダプタユーザがアプリケーションビューの定義、デプロイ、およびテストの際に必要な、WebベースのGUIを作成するためのツールを提供します。各アダプタにはEIS固有の機能がありますが、アプリ

ケーションビューをデプロイするには、すべてのアダプタで GUI が必要です。設計時のフレームワークでは、そのような GUI の作成とデプロイにかかる手間を最小限に抑える 2 つのツールを使用します。

- JSP を使用して HTML ベースの GUI を構築できるようにする Web アプリケーション コンポーネント。このコンポーネントの機能は、JSP テンプレート、タグ ライブラリおよび JavaScript ライブラリなどのツールによって拡張できます。
- WebLogic Server 上のアプリケーションビューをデプロイ、アンデプロイ、および編集するための簡便な API を提供する、デプロイメント ヘルパー コンポーネント。

各アダプタの設計時インタフェースは、WAR ファイルとしてまとめられた J2EE 準拠の Web アプリケーションです。Web アプリケーションは、.jsp、.html およびイメージファイルなどをまとめたものです。Web アプリケーションの記述子は web.xml です。この記述子によって、Web アプリケーションのデプロイおよび初期化方法が、J2EE Web コンテナに与えられます。

各 Web アプリケーションには、デプロイメント時に指定されたコンテキストがあります。コンテキストは、Web コンテナのドキュメント ルート内にある Web アプリケーションに関連付けられたリソースを識別します。

イベントおよびサービス

ADK を使用して、イベント アダプタおよびサービス アダプタの両方を作成できます。ADK アーキテクチャでは、サービスおよびイベントは、それぞれの入出力の定義に XML スキーマを使用する自己記述オブジェクト（すなわち、名前がビジネス機能を示す）として定義されます。

イベントの概要

イベントは、関心イベントが EIS 内で発生したときにアプリケーションビューによってパブリッシュされる XML ドキュメントです。イベントの通知を希望するクライアントは、アプリケーションビューに登録して通知を要求します。登録が済むと、アプリケーションビューは、対象アプリケーションとクライアン

ト間のブローカとして機能します。クライアントがアプリケーションビューによってパブリッシュされたイベントをサブスクライブした場合、アプリケーションビューは対象アプリケーション内で関心イベントが発生するたびにクライアントに通知します。イベントサブスクライバに、関心イベントが発生したことが通知されると、サブスクライバにはそのイベントを記述したXMLドキュメントが渡されます。また、イベントをパブリッシュするアプリケーションビューは、クライアントに、パブリッシュ可能なイベントのXMLスキーマを提供することもできます。

注意： アプリケーションビューは、アプリケーションの特定機能に対するビジネスレベルのインタフェースを表しています。この機能の詳細は、『*Application Integration 入門*』を参照してください。

サービスの概要

サービスは、アプリケーション内のビジネスオペレーションのうち、アプリケーションビューによってエクスポートされるものです。サービスは要求/応答のメカニズムとして機能します。アプリケーションが、ビジネスサービスを呼び出す要求を受信すると、アプリケーションビューが対象アプリケーション内のサービスを呼び出し、結果を記述したXMLドキュメントを返します。

サービスを定義するには、入力要件、出力想定、および対話設定を定義する必要があります。

要求は2つの要素で送信されます。

- 要求に関する静的「セカンダリメタデータ」を含む対話設定。
- 任意の変数値を示す基本入力。たとえば、DBMSトランザクションでは、SQL文は対話設定に含まれ、変数の値は入力要件に含まれます。そして、トランザクションの結果が出力想定と考えられます。

アダプタによるロギングの使い方

ロギングは、アダプタに不可欠な機能です。一般にアダプタは、異なるアプリケーション同士の統合に使用され、データの処理時にエンド ユーザとは対話しません。フロントエンド コンポーネントと違って、アダプタは、エラーや警告状況が発生した場合に、処理を停止してエンドユーザの応答を待つことはできません。

また、アダプタが統合するビジネス アプリケーションは、通常、ミッションクリティカルなビジネス アプリケーションです。そこで、アダプタに、EIS とのすべてのトランザクションに関する監査レポートの保持が要求されるケースが考えられます。したがって、アダプタ コンポーネントは、正確なロギング情報と監査情報を提供する必要があります。ADK のロギング フレームワークは、ロギングと監査、両方の要求に対応できるように設計されています。

ロギング ツールキット

ADK が提供するツールキットを使用すると、ローカライズされたメッセージを複数の出力先にログとして記録できます。このロギング ツールキットにより、オープン ソース プロジェクト、Apache Log4j の機能が強化されます。

ロギング ツールキットは、Log4j 内の重要なクラスのラップを行い、J2EE 準拠アダプタの構築時に追加機能を提供します。このツールキットは `logtoolkit.jar` ファイルにあります。

ロギング ツールキットの使用の詳細については、第 5 章「ロギング ツールキットの使い方」を参照してください。

ロギング フレームワーク

ADK では、ロギング フレームワークを実装することにより、アダプタ アクティビティのログを記録できます。このフレームワークによって、インターナショナルライズされたメッセージおよびローカライズされたメッセージを、複数の出力先

に記録できます。また、メッセージのカテゴリ、優先度、フォーマット、および送り先の指定に使用できる一連のコンフィグレーションパラメータが用意されています。

ロギングフレームワークではカテゴリ階層が使用されており、アダプタ内のすべてのパッケージおよびクラスでロギングコンフィグレーションを継承できるようになっています。また、パラメータは実行時に簡単に変更できます。

インターナショナルライゼーションとローカライゼーション

ロギングフレームワークでは、ログメッセージをインターナショナルライズできます。インターナショナルライズされたアプリケーションは、コードを作成しなおすことなく、世界中のエンドユーザの言語および語法に簡単に変換できます。ローカライゼーションは、ロケール固有のコンポーネントおよびテキストを追加することにより、ソフトウェアを特定の地域または特定の言語用に変更する処理のことです。ロギングフレームワークでは、Javaプラットフォームによるインターナショナルライゼーションおよびローカライゼーション機能を使用されま

アダプタ論理名

各アダプタには、個別のアダプタを表し、また、すべてのアダプタの構成基準となるユニークな識別子であるアダプタ論理名を付ける必要があります。アダプタ論理名は、個別のアダプタおよび以下の関連項目を識別するために使用します。

- メッセージバンドル
- ロギングコンフィグレーション
- ログカテゴリ

アダプタ論理名は、ベンダ名、アダプタに接続されるEISのタイプ、およびEISのバージョン番号を組み合わせたものです。この名前は、通常、`vendor_EIS-type_EIS-version` という書式で表わされます。たとえば、アダプタ論理名 `BEA_WLS_SAMPLE_ADK` の場合は、以下のとおりです。

- BEA_WLS : ベンダと製品
- SAMPLE : EIS タイプ
- ADK : EIS バージョン

必要な情報が含まれていれば、この情報に対して別のフォーマットを使用することも可能です。

アダプタ論理名の使用場所

アダプタでは、アダプタ論理名が以下のように使用されます。

- アダプタのデプロイメント時に、WAR、RAR、JAR および EAR ファイルのファイル名の一部として使用されます。
- 2-9 ページの「アダプタ論理名を構成基準として使用する」に説明されているように、構成基準として使用されます。
- 2-10 ページの「getAdapterLogicalName() の戻り値として使用されるアダプタ論理名」に説明されているように、com.bea.adapter.web 内の抽象メソッド getAdapterLogicalName() の戻り値として使用されます。

アダプタのデプロイメントでアダプタ論理名を使用する

アダプタ論理名を割り当てるには、<ConnectorComponent> 要素を含む <Application> 要素の Name 属性の値としてこれを指定します。この値は、WebLogic Integration でアプリケーションビューとデプロイされたリソースアダプタを関連付けるためのキーとして使用します。コード リスト 2-1 でその例を示します。

コード リスト 2-1 ConnectorComponent 要素の Name 属性

```
<Application Deployed="true" Name="BEA_WLS_DBMS_ADK"  
  Path="<WLI_HOME>/adapters/dbms/lib/BEA_WLS_DBMS_ADK.ear"  
  TwoPhase="true">  
  <ConnectorComponent Name="BEA_WLS_DBMS_ADK" Targets="myserver"  
    URI="BEA_WLS_DBMS_ADK.rar" />  
  <WebAppComponent Name="DbmsEventRouter" Targets="myserver"  
    URI="BEA_WLS_DBMS_ADK_EventRouter.war" />  
  <WebAppComponent Name="BEA_WLS_DBMS_ADK_Web" Targets="myserver"
```

2 基本開発コンセプト

```
URI="BEA_WLS_DBMS_ADK_Web.war" />
</Application>
```

注意：アダプタ論理名を RAR ファイル名として任意に使用できますがこのような使用は URI 属性では必要ありません。

アプリケーションビューをデプロイすると、J2EE コネクタ アーキテクチャ CCI 接続ファクトリ デプロイメントと関連付けられます。たとえば、ユーザが abc.xyz アプリケーションビューをデプロイすると、WebLogic Integration では新しい ConnectionFactory がデプロイされ、この接続ファクトリと次の JNDI の場所がバインドされます。

```
com.bea.wlai.connectionFactories.abc.xyz.connectionFactoryInstance
```

この新しい接続ファクトリ デプロイメントでは、便宜性を高めるため、weblogic-ra.xml デプロイメント記述子の <ra-link-ref> 設定が使用されません。

必要に応じて、アプリケーションビューの設計時に、ユーザは共有可能な既存の接続ファクトリ デプロイメントをアプリケーションビューに関連付けることができます。接続ファクトリは、WebLogic Server Administration Console を使用して作成されたものである必要があります。詳細については、「共有接続ファクトリ」を参照してください。

<ra-link-ref> 要素を使用し、複数のデプロイ済み接続ファクトリを 1 つのデプロイ済みアダプタに論理的に関連付けることができます。必要に応じて、<ra-link-ref> 要素に別のデプロイ済み接続ファクトリを示す値を指定すると、新しくデプロイされるその接続ファクトリが、参照先の接続ファクトリと一緒にデプロイされたアダプタを共有します。また、参照先の接続ファクトリのデプロイメント時に定義したすべての値は、その他の値を指定しない限り、新しくデプロイされるこの接続ファクトリで継承されます。アダプタ論理名は、<ra-link-ref> 要素の値として使用されます。

アダプタ論理名を構成基準として使用する

表 2-1 は、アダプタ論理名が構成基準として使用される機能の種類のリストです。

表 2-1 どのようにアダプタ論理名が構成基準として使用されるか

機能領域	アダプタ論理名は次のように使用されます
ロギング	<p>アダプタ論理名は、アダプタ固有のすべてのログメッセージに対する基本ログカテゴリ名として使用される。その結果、アダプタ論理名は以下の XML ドキュメントの <code>RootLogContext</code> パラメータの値として渡される。</p> <ul style="list-style-type: none"> ■ <code>WLI_HOME/adapters/ADAPTER/src/eventrouter/WEB-INF/web.xml</code> ■ <code>WLI_HOME/adapters/ADAPTER/src/rar/META-INF/ra.xml</code> ■ <code>WLI_HOME/adapters/ADAPTER/src/rar/META-INF/weblogic-ra.xml</code> ■ <code>WLI_HOME/adapters/ADAPTER/src/war/WEB-INF/web.xml</code> <p>これらのパス名において、<code>ADAPTER</code> はアダプタ名を示す。たとえば、 <code>WLI_HOME/adapters/dbms/src/war/WEB-INF/web.xml</code></p> <p>さらに、アダプタ論理名は、アダプタの <code>Log4J</code> コンフィグレーションファイル名のベースとしても使用される。このとき、拡張子 <code>.xml</code> がアダプタ論理名に付けられる。 <code>.xml</code> が名前に加えられる。たとえば、サンプルアダプタの <code>Log4J</code> コンフィグレーションファイルは <code>BEA_WLS_SAMPLE_ADK.xml</code> となる。</p>

表 2-1 どのようにアダプタ論理名が構成基準として使用されるか (続き)

機能領域	アダプタ論理名は次のように使用されます
ローライゼーション	<p>アダプタの論理名は、アダプタのメッセージバンドルに対する基本名として使用される。たとえば、サンプルアダプタのデフォルトメッセージバンドルは <code>BEA_WLS_SAMPLE_ADK.properties</code> となる。その結果、アダプタ論理名は以下の XML ドキュメントの <code>MessageBundleBase</code> パラメータの値として渡される。</p> <ul style="list-style-type: none">■ <code>WLI_HOME/adapters/ADAPTER/src/eventrouter/WEB-INF/web.xml</code>■ <code>WLI_HOME/adapters/ADAPTER/src/rar/META-INF/ra.xml</code>■ <code>WLI_HOME/adapters/ADAPTER/src/rar/META-INF/weblogic-ra.xml</code>■ <code>WLI_HOME/adapters/ADAPTER/src/war/WEB-INF/web.xml</code> <p>これらのパス名において、<code>ADAPTER</code> の値はアダプタ名を示す。たとえば、 <code>WLI_HOME/adapters/dbms/src/war/WEB-INF/web.xml</code></p>

getAdapterLogicalName() の戻り値として使用されるアダプタ論理名

最後に、アダプタ論理名は、`com.bea.adapter.web.AbstractDesignTimeRequestHandler` の抽象メソッド `getAdapterLogicalName()` の戻り値として使用されます。この戻り値は、アプリケーションビューのデプロイメント時に、接続ファクトリに対する `RootLogContext` の値として使用されます。

共有接続ファクトリ

共有可能な既存の接続ファクトリ デプロイメントをアプリケーション ビューに関連付けることができます。接続ファクトリは、**WebLogic Server Administration Console** を使用して作成されたものである必要があります。共有可能な接続ファクトリとその JNDI の位置は、`ConnectorComponentMbeans` によって識別されます。JNDI の場所は、アプリケーション ビューのプロパティである `connectionFactoryJNDIName` に書き込まれます。アプリケーション ビューのデプロイヤは、デプロイメント時にこのプロパティを使用します。

起動時の接続ファクトリへの参照

すべての共有可能接続ファクトリは、起動プロセス時に参照されます。ユーザは、デプロイメント プロセスにおいて **WebLogic Administration Console** を使用してデプロイされた接続ファクトリが使用可能であることを確認する必要があります。接続ファクトリが見つからない場合、アプリケーション ビューのデプロイメントは失敗します。

エンタープライズ アーカイブ (EAR) ファイル

ADK では、アダプタのデプロイにエンタープライズアーカイブ ファイル、あるいは EAR ファイルを使用します。1 つの `.ear` ファイルには、アダプタのデプロイに必要な **WAR** および **RAR** ファイルが含まれます。EAR ファイルの例をコード リスト 2-2 に示します。

コード リスト 2-2 EAR ファイル構造

```
adapter.ear
  META-INF
    application.xml
  sharedJar.jar
```

2 基本開発コンセプト

```
adapter.jar
adapter.rar
  META-INF
    ra.xml
    weblogic-ra.xml
    MANIFEST.MF
designtime.war
  WEB-INF
    web.xml
  META-INF
    MANIFEST.MF
```

サンプルアダプタの EAR ファイルをコード リスト 2-3 に示します。

コード リスト 2-3 サンプルアダプタの EAR ファイル

```
sample.ear
  META-INF
    application.xml
  adk.jar (shared .jar between .war and .rar)
  bea.jar (shared .jar between .war and .rar)

  BEA_WLS_SAMPLE_ADK.jar (shared .jar between .war and .rar)

  BEA_WLS_SAMPLE_ADK.war (Web application with
    META-INF/MANIFEST.MF entry Class-Path:
    BEA_WLS_SAMPLE_ADK.jar adk.jar bea.jar log4j.jar
    logtoolkit.jar xcci.jar xmltoolkit.jar)

  BEA_WLS_SAMPLE_ADK.rar (Resource Adapter
    with META-INF/MANIFEST.MF entry Class-Path:
    BEA_WLS_SAMPLE_ADK.jar adk.jar bea.jar log4j.jar
    logtoolkit.jar xcci.jar xmltoolkit.jar)

log4j.jar (shared .jar between .war and .rar)
logtoolkit.jar (shared .jar between .war and .rar)
xcci.jar (shared .jar between .war and .rar)
xmltoolkit.jar (shared .jar between .war and .rar)
```

RAR と WAR ファイル内には、共有 JAR ファイルはなく、どちらも EAR ファイルのルート ディレクトリにある共有 JAR ファイルを参照しています。

EAR ファイルを使用したアダプタのデプロイ方法の詳細については、第 9 章「アダプタのデプロイ」を参照してください。

アダプタによる同時要求の処理方法

アダプタ設計者として、WebLogic Integration が 1 つのアダプタに対する複数の同時要求をどのように処理するかを、接続およびスレッドの観点から理解する必要があります。

アプリケーションビューは、Stateless Session EJB を使用しアダプタと通信します。アダプタ内でのすべての実行は、WebLogic Server 実行スレッドの適用範囲内で実行されます。このため、同時要求の数は、次の要因によって制限されます。

- WebLogic Server 実行スレッドの数
- コネクタの RAR デプロイメントのために接続ファクトリに関連付けられたプールの接続数

Sun Microsystems, Inc. による J2EE コネクタ仕様、バージョン 1.0 に定められているように、アダプタは独自のスレッドを作成しません。WebLogic Server 実行スレッドまたはプール内の接続のいずれかが酷使されると、スループットが低下します。有効なリソースの超過は次のような影響を及ぼします。

- WebLogic Server の実行スレッドが不足すると、`ApplicationView.invokeService()` が新たに呼び出され、空の実行スレッドが取得されるまでブロックされます。WebLogic Server Administration Console を使用し、サーバに対する実行スレッド数をコントロールできます。
- 接続プール内の接続が不足すると、`ApplicationView.invokeService()` が呼び出され、短時間のスリープの後、再試行して接続を取得します。アプリケーションビューは、1 秒間隔で最大 60 回再試行を行います。この再試行のロジックは、クライアントが処理しなければならない `pool empty` 例外の大部分を排除します。これは、通常、`pool empty` 条件の有効期限が短く、最終的には接続が有効になるためです。接続のホールド時間が異常に長い場合 (EIS のハングまたは過負荷)、`pool empty` 条件はクライアントに返されません。

3 開発ツール

ADK では、アダプタおよび設計時 GUI の開発を支援する堅牢なツール セットを提供しています。この章では、これらのツールについて説明します。この章の内容は以下のとおりです。

- サンプル アダプタ
- GenerateAdapterTemplate ユーティリティ
- ADK Javadoc
- Ant ベースのビルド プロセス
- XML ツール

サンプル アダプタ

開発者が、このコードを用いてアダプタの構築を始めることができるよう、ADK には、EIS に依存しないコード例を提供するサンプル アダプタが搭載されています。このサンプル アダプタと、同じく **WebLogic Integration** に搭載されている **DBMS** アダプタを混同しないよう注意してください。**DBMS** サンプル アダプタについては、付録 E 「**DBMS** サンプル アダプタを使用したアダプタ開発方法の学習」で説明します。**DBMS** サンプル アダプタの格納場所は、`WLI_HOME/adapters/dbms` です。

サンプル アダプタの使用目的

サンプルアダプタの目的は、アダプタの構築に必要なコーディング作業の負担を大幅に軽減することです。サンプルアダプタにより、主要な抽象クラスについては具象実装が提供されるため、開発者に必要な作業は、ご使用の **EIS** の要件を満たすためのカスタマイズだけです。さらに、ADK では、開発対象のアダプ

タで使用する、サンプルアダプタの開発ツリーを手早く複製できる `GenerateAdapterTemplate` ユーティリティが提供されています。3-3 ページの「`GenerateAdapterTemplate` ユーティリティ」を参照。

サンプルアダプタの内容

サンプルアダプタは、以下のクラスから構成されます。

`sample.cci.ConnectionImpl`

クライアントがその物理接続にアクセスするときに使用するアプリケーションレベルのハンドルを表す、`Connection` インタフェースの具象実装。

`sample.cci.InteractionImpl`

`DesignTimeInteractionSpecImpl` クラスを使用してどのように設計パターンを実装するかを示すクラス。

`sample.cci.InteractionSpecImpl`

基本実装を提供するインタフェース。この基本実装は標準対話プロパティにゲッターおよびセッターメソッドを使用することにより拡張できる。

`sample.client.ApplicationViewClient`

サービスを呼び出し、アプリケーションビューのイベントをリスンする方法を示すクラス。

`sample.eis.EIS`

`sample.eis.EISEvent`

`sample.eis.EISListener`

デモ用に簡単な `EIS` を表すクラス。

`sample.event.EventGenerator`

イベントジェネレータを作成する `ADK` 基本クラスの拡張方法を示す、`AbstractPullEventGenerator` の具象拡張。

`sample.event.OfflineEventGeneratorTestCase`

`Weblogic Server` 以外のイベントジェネレータの内部動作をテストするときに使用できるクラス。

`sample.spi.ManagedConnectionFactoryImpl`

特定の `EIS` のカスタマイズを行うことができる、`AbstractManagedConnectionFactory` の具象拡張。

`sample.spi.ManagedConnectionImpl`

特定の EIS のカスタマイズを行うことができる、
`AbstractManagedConnection` の具象拡張。

`sample.spi.ConnectionMetaDataImpl`

特定の EIS のカスタマイズを行うことができる、
`AbstractConnectionMetaData` の具象拡張。

`sample.spi.NonManagedScenarioTestCase`

非管理対象シナリオで SPI および CCI クラスをテストするために使用できるクラス。

`sample.web.DesignTimeRequestHandler`

設計時にイベントまたはサービスの追加方法を示す、
`AbstractDesignTimeRequestHandler` の具象拡張。

注意： サンプル アダプタ内のサンプルによって拡張されたクラスの詳細については、ADK Javadocs を参照してください。

GenerateAdapterTemplate ユーティリティ

サンプルアダプタを使いやすくするため、ADK では `GenerateAdapterTemplate` というコマンドライン ユーティリティを提供しています。このユーティリティを使用して、サンプルツリーを複製し、新しいアダプタ開発ツリーを作成できます。このアクションの使用法の詳細については、第 4 章「カスタム開発環境の作成」を参照してください。

ADK Javadoc

ADK のクラス、インタフェース、メソッド、およびコンストラクタは、開発キットの Javadoc に定義されています。Javadoc は、WebLogic Integration インストールで組み込まれ、`WLI_HOME/adapters/ADAPTER/docs/api` に保存されます。ADAPTER は、Sample や DMBS などのアダプタ名です。たとえば、Javadoc は `WLI_HOME/adapters/dbms/docs/api` にインストールされます。

Ant ベースのビルド プロセス

ADK では、Ant という Java 言語のみを使用して作成した Java のビルド ツールに基づくビルド プロセスが採用されています。ADK では、Ant により以下の処理が実行されます。

- アダプタの Java アーカイブ (JAR) ファイルを作成します。
- アダプタの Web アプリケーションに対する WAR ファイルを作成します。
- J2EE 準拠アダプタの RAR ファイルを作成します。
- リスト内の他のコンポーネントをデプロイメントのために EAR ファイルにバンドルします。

Ant を使用する理由

従来、ビルド ツールはシェル ベースでした。シェル コマンドと同様に、一連の依存関係を評価し、さまざまなタスクを実行します。これらのツールには、オペレーティング システム (OS) のプログラムを使用する、または作成することによって簡単に拡張できるという利点がありますが、逆にその OS に制限されるという欠点もあります。

Ant は、以下の理由でシェルベースの作成ツールよりも優れています。

- Ant は、シェル ベース コマンドではなく Java クラスを使用して拡張されません。
- コンフィグレーション ファイルは、シェル コマンドではなく XML ベースで、さまざまなタスクが実行されるターゲット ツリーを呼び出します。各タスクは、特定のタスク インタフェースを実装するオブジェクトによって実行されます。この方法では、シェル コマンドを作成する場合に本来の表現力がある程度失われますが、その代わりにアプリケーションはプラットフォームを通じてポータブルにできます。
- Ant により、さまざまな OS 固有のシェル コマンドを実行できます。

Ant の設定方法に関する詳細は、6-10 ページの「手順 2c : ビルド プロセスの設定」を参照してください。

XML ツール

ADK には 2 つの XML 開発ツールから成る XML ツールキットが付属しており、設計時フレームワークのメタデータ サポート レイヤの一部とみなされています。

- **XML スキーマ API** – この API はスキーマオブジェクト モデル (SOM) をベースとしており、XML スキーマを作成するときに使用されます。SOM は、構文的に複雑な XML スキーマ操作などの一般的な詳細処理の多くを抽出するツールセットで、これによって開発者はより基本的な側面に作業を集中できます。
- **XML ドキュメント API** – この API は IDocument をベースとしており、Document Object Model (DOM) のドキュメントに対する x-path インタフェースを提供します。

これらのツールの使用方法については、付録 B 「XML Toolkit」を参照してください。

WebLogic Integration は、両方の API に対する Javadoc を提供します。

- **SOM Javadoc** を参照するには
`WLI_HOME/docs/apidocs/com/bea/schema`
- **IDocument Javadoc** を参照するには
`WLI_HOME/docs/apidocs/com/bea/document`

4 カスタム開発環境の作成

警告： サンプルアダプタを直接変更しないでください。変更する場合は、この章で説明する `GenerateAdapterTemplate` ユーティリティを使用して、アダプタのコピーを作成し、そのコピーに必要な変更を加えます。サンプルアダプタそのものを変更すると（あるいは `GenerateAdapterTemplate` を使用せずにコピーを作成すると）、予期しない、あるいはサポートされない動作を起こすことがあります。

サンプルアダプタ（3-1 ページの「サンプルアダプタ」を参照）を使いやすくするため、ADK には `GenerateAdapterTemplate` というコマンドラインユーティリティが用意されています。このユーティリティを使用して、サンプルツリーを複製し、新しいアダプタ開発ツリーを作成できます。

この章の内容は以下のとおりです。

- アダプタ設定ワークシート
- `GenerateAdapterTemplate` の使い方

アダプタ設定ワークシート

アダプタ設定ワークシートは、開発対象のアダプタに関する重要なデータを特定して、収集するための質問表のようなものです。実際のワークシートの内容は、附録 D「アダプタ設定ワークシート」に記載されています。

このワークシートは、EIS タイプ、ベンダ、バージョン、デプロイメントのロケールおよび地域言語、アダプタ論理名、アダプタがサービスをサポートするかどうか、といったアダプタの重要情報を確認する 20 の質問で構成されています。`GenerateAdapterTemplate` を実行すると、これらの情報を入力するように要求されます。入力した情報が処理されると、アダプタのカスタム開発ツリーが作成されます。

GenerateAdapterTemplate の使い方

この節では、GenerateAdapterTemplate の使用方法について説明します。必要な手順は次のとおりです。

- 手順 1. GenerateAdapterTemplate の実行
- 手順 2. ツリーの再構築
- 手順 3. アダプタの WebLogic Integration へのデプロイ

手順 1. GenerateAdapterTemplate の実行

このツールを使用する手順は次のとおりです。

1. `WLI_HOME/adapters/utils` ディレクトリからコマンドラインを開き、以下のいずれかのコマンドを実行します。

- Windows NT の場合 : `GenerateAdapterTemplate.cmd`
- UNIX の場合 : `GenerateAdapterTemplate.sh`

システムから次のように応答があります。

```
WLI_HOME/adapters/utils>generateadapertemplate
*****
Welcome! This program helps you generate a new adapter
development tree by cloning the ADK's sample adapter development
tree.

Do you wish to continue? (yes or no); default='yes':
```

2. `[Enter]` を押して `[yes]` を選択します。

システムから次のように応答があります。

```
Please choose a name for the root directory of your adapter
development tree:
```

3. ユニークな覚えやすいディレクトリ名 (`dir_name`) を入力し、`[Enter]` を押します。

システムから次のように応答があります。

created directory *WLI_HOME/adapters/dir_name*

Enter the EIS type for your adapter:

システム出力で指定したパス名で、*dir_name* は新しいディレクトリの名前です。

注意： 既存のディレクトリ名を入力した場合、システムの応答は次のようになります。

```
WLI_HOME/adapters/dir_name already exists, please choose  
a new directory that does not already exist!
```

```
Please choose a name for the root directory of your adapter  
development tree:
```

- アダプタを接続する EIS タイプの識別子を入力します。〔Enter〕を押します。
システムから次のように応答があります。

```
Enter a short description for your adapter:
```

- 開発対象のアダプタの説明を簡潔に入力し、〔Enter〕を押します。

システムから次のように応答があります。

```
Enter the major version number for your adapter; default='1':
```

- 〔Enter〕を押してデフォルトをそのまま使用するか、適切なバージョン番号を入力し、〔Enter〕を押します。

システムから次のように応答があります。

```
Enter the minor version number for your adapter; default='0':
```

- 〔Enter〕を押してデフォルトをそのまま使用するか、適切なマイナーバージョン番号を入力し、〔Enter〕を押します。

システムから次のように応答があります。

```
Enter the vendor name for your adapter:
```

- ベンダ名を入力し、〔Enter〕を押します。

システムから次のように応答があります。

```
Enter an adapter logical name; default='default_name':
```

9. [Enter] を押してデフォルトをそのまま使用するか、または使用したいアダプタ論理名を入力します。[Enter] を押します。このデフォルトのアダプタ論理名 (「`default_name`」) は、WebLogic Integration で推奨されるフォーマットに従っています。

```
vendor_name_EIS-type_version-number.
```

システムから次のように応答があります。

```
Enter the Java package base name for your adapter
(e.g. sample adapter's is sample): Java パッケージ基本名
```

10. パッケージ形式に従って Java パッケージの基本名を入力し、[Enter] を押します。パッケージ形式の名前は、ドット区切りで、以下の文字列で構成されます。

- 組織の Web サイトの URL で使用される拡張子 (.com、.org または .edu など)。
- 会社名。
- 追加のアダプタ識別子。たとえば、`com.your_co.adapter.EIS` のようになります。

システムから次のように応答があります。

```
The following information will be used to generate your new
adapter development environment:
EIS Type = 'SAP R/3'
Description = 'description'
Major Version = '1'
Minor Version = '0'
Vendor = 'vendor_name'
Adapter Logical Name = 'adapter_logical_name'
Java Package Base = 'com.java.package.base'
Are you satisfied with these values? (enter yes or no or q to
quit);
default='yes':
```

11. データを確認し、問題がなければ [Enter] を押します。

登録したビルド情報を表示することによって、システム応答します。

注意: no と入力すると、4に戻ります。q(終了)を入力すると、アプリケーションが終了します。

手順 1a : コンソール コード ページの指定 (Windows のみ)

Windows システムの場合、以下のコード ページ リストからコンソールのコード ページの値を選択します。

```
437 - United States
850 - Multilingual (Latin I)
852 - Slavic (Latin II)
855 - Cyrillic (Russian)
857 - Turkish
860 - Portuguese
861 - Icelandic
863 - Canadian-French
865 - Nordic
866 - Russian
869 - Modern Greek
Enter your console's codepage; default='437':
```

コード ページが分からない場合は、コンソールプロンプトで `tchcp` と入力します。Windows のバージョンによって、このコマンドによりコンソールのコード ページの値が表示されます。

手順 2. ツリーの再構築

複製処理の終了後、新しいディレクトリに切り替え、ADK のビルド ツールである Ant を使用してツリー全体を再構築します。Ant の詳細については、3-4 ページの「Ant ベースのビルド プロセス」を参照してください。

Ant を使用してツリーを再構築するには次の手順に従います。

1. `WLI_HOME/adapters/ADAPTER/utils` にある `antEnv.cmd` (Windows) または `antEnv.sh` (UNIX) を編集します。
2. 以下の変数を該当するパスに設定します。
 - `BEA_HOME` - ご使用の BEA 製品の最上位ディレクトリ。例 : `c:/bea`
 - `WLI_HOME` - WebLogic Integration ディレクトリの場所。
 - `JAVA_HOME` - Java Development Kit の場所。

- `WL_HOME` – WebLogic Server ディレクトリの場所。
- `ANT_HOME` – Ant ディレクトリの場所。通常、
`WLI_HOME/adapters/utils`

注意： この操作はインストーラによって自動的に行われますが、これらの設定にもとづいて Ant 処理が実行されることを認識しておいてください。

UNIX システムの場合、`WLI_HOME/adapters/utils` の Ant ファイルに対し、すべての実行パーミッションを設定する必要があります。実行パーミッションを追加するには、以下のコマンドを入力します。

```
chmod u+x ant.sh
```

3. コマンドラインから `antEnv` を実行し、シェルに必要な環境変数を設定します。
4. `WLI_HOME/adapters/ADAPTER/project` ディレクトリから `ant release` を実行して、アダプタを構築します。(ADAPTER を新しいアダプタの開発ルート名と変える。)

`ant release` を実行すると、アダプタ用の Javadoc が生成されます。この Javadoc は、次の場所に表示できます。

```
WLI_HOME/adapters/ADAPTER/docs/
```

このファイルには、構築したアダプタを WebLogic Integration 環境でデプロイする場合の、それぞれの環境に即した指示が記載されています。具体的には、`config.xml` エントリと、既存パスの置換値が定義されます。さらに、このファイルでは、マッピング情報も定義されています。

4-6 ページの「手順 3. アダプタの WebLogic Integration へのデプロイ」で説明するように、アダプタを簡単にデプロイするために、`overview.html` の内容を直接 `config.xml` にコピーします。

手順 3. アダプタの WebLogic Integration へのデプロイ

アダプタのデプロイは、手動でも WebLogic Server Administration Console からでも実行できます。詳細については、第 9 章「アダプタのデプロイ」を参照してください。

5 ログイング ツールキットの使い方

ログイングは、アダプタ コンポーネントに不可欠な機能です。一般的にアダプタは、複数のアプリケーションの統合に使用され、データの処理時にエンド ユーザとは対話しません。フロントエンド コンポーネントの動作とは異なり、エラーや警告状況が発生した場合も、処理を停止して、エンドユーザの応答を待つことはありません。

ADK では、ログイング フレームワークを実装することにより、アダプタ アクティビティのログを記録できます。このフレームワークによって、インターナショナルライズされたメッセージおよびローカライズされたメッセージを、複数の出力先に記録できます。また、メッセージのカテゴリ、優先度、フォーマット、および送り先の指定に使用できる一連のコンフィグレーション パラメータが用意されています。

この章の内容は以下のとおりです。

- ログイング ツールキット
- ログイング コンフィグレーション ファイル
- ログイング コンセプト
- ログイング の設定方法
- ログイング フレームワークのクラス
- ログ メッセージのインターナショナルライゼーションとローカライゼーション
- マルチスレッド コンポーネントでのコンテキスト情報の保存

ログイング ツールキット

ADK が提供する ログイング ツールを使用すると、インターナショナルライズされたメッセージを複数の出力先にログとして記録できます。このログイング ツールキットにより、**Apache Log4j** オープン ソース プロジェクトの機能が強化されます。この製品には、**Apache Software Foundation** (<http://www.apache.org>) によって開発されたソフトウェアが組み込まれています。

ログイング ツールキットは、必要な **Log4j** クラスをラップし、**J2EE** 準拠アダプタに機能を追加するためのフレームワークです。このツールキットは、**WLI_HOME/lib** ディレクトリの **logtoolkit.jar** ファイル内にあります。この **JAR** ファイルは、**DOM**、**XERCES**、および **Log4j** に依存しています。**XERCES** の依存関係には、**WebLogic Server** と共に提供される **weblogic.jar** および **xmlx.jar** ファイルが必要です。**WebLogic Integration** は、**WLI_HOME/lib** の該当するバージョンである **Log4j (log4j.jar)** も提供します。

Log4j パッケージは、オープン ソース イニシアチブによって認可されたオープン ソースの完全ライセンス、**Apache** 公用ライセンスに従って配布されています。全ソース コード、クラス ファイル、およびドキュメント込みの最新の **Log4j** バージョンは、**Apache Log4j Web** サイト (<http://www.apache.org>) にあります。

ログイング コンフィグレーション ファイル

この節では、全般にわたってログイング コンフィグレーション ファイルの参照をしたりコードを抜粋したりしています。ログイング コンフィグレーション ファイルは、**BEA_WLS_DBMS_ADK.xml** などのアダプタ論理名で識別される **.xml** ファイルです。このファイルには、5-3 ページの「ログイング コンセプト」で説明されている 4 つのログイング コンセプトに関する基本情報が定義されていて、開発者のアダプタに合わせて変更できます。

ADK では、**WLI_HOME/adapters/sample/src** に基本的なログイング コンフィグレーション ファイル **BEA_WLS_SAMPLE_ADK.xml** が格納されています。このファイルを独自のアダプタ用に変更する場合は、**GenerateAdapterTemplate** を実行してください。このユーティリティを使用すると、サンプルバージョンのログ

ング コンフィグレーション ファイルを、開発者が新しく構築したアダプタに関連する情報にもとづいてカスタマイズし、このカスタマイズ バージョンのファイルを新しいアダプタの開発環境で使用できます。GenerateAdapterTemplate の詳細については、第 4 章「カスタム開発環境の作成」を参照してください。

ロギング コンセプト

ADK のロギング ツールキットを使う前に、ロギング フレームワークの主要コンセプトを理解しておいてください。ロギングには、次の 4 つの主要コンポーネントがあります。

- メッセージ カテゴリ
- メッセージ優先度
- メッセージアペンダ
- メッセージレイアウト

これらのコンポーネントは相互に連携して機能し、メッセージ タイプと優先度に従ってログ メッセージを記録し、メッセージのフォーマットおよび出力先を実行時に管理できます。

メッセージ カテゴリ

カテゴリは、定義された基準に従ってログ メッセージを識別する、ロギング フレームワークの中心的なコンセプトです。ADK では、カテゴリは、BEA_WLS_SAMPLE_ADK.DesignTime のように名前で識別されます。

カテゴリは、定義された階層で、どのカテゴリも親カテゴリからプロパティを継承できます。階層は次のように定義されます。

- あるカテゴリ名にドットが続き、そのカテゴリ名が子孫カテゴリのプレフィックスになっている場合、前者のカテゴリは、後者のカテゴリの祖先となります。
- あるカテゴリとその子孫カテゴリの間に祖先となるカテゴリがない場合、前者のカテゴリは後者のカテゴリ（子カテゴリ）の親となります。

たとえば、以下の図に示すように、`BEA_WLS_SAMPLE_ADK.DesignTime` は、`BEA_WLS_SAMPLE_ADK` の子孫で、さらに `BEA_WLS_SAMPLE_ADK` は、ルート カテゴリの子孫になっています。

```
ROOT CATEGORY
|
|-->BEA_WLS_SAMPLE_ADK
|   |
|   |-->BEA_WLS_SAMPLE_ADK.DesignTime
```

ルート カテゴリはカテゴリ階層の最上位にあり、名前で削除あるいは検索することができません。

カテゴリを作成する場合、属するアダプタ内のコンポーネントに対応した名前を割り当てます。たとえば、あるアダプタのコンポーネントが設計時ユーザインタフェース コンポーネントである場合、次のような名前にします。

```
BEA_WLS_SAMPLE_ADK.DesignTime
```

メッセージ優先度

各メッセージの重要度が、優先度によって示されています。メッセージの優先度は、メッセージのログに使用される `ILogger` インタフェース メソッドによって決まります。たとえば、`ILogger` インスタンスでデバッグ メソッドを呼び出すと、優先度のデバッグ メッセージが生成されます。

ログイング ツールキットでは、メッセージに対して 5 つの優先度をサポートしています。これらの優先度は、優先度の高い順に表 5-1 にリストされます。

表 5-1 ログイング ツールキットの優先度

優先度	意味
AUDIT	アダプタが実行するビジネス処理に関連する非常に重要なログ メッセージ。この重要度のメッセージは常にログに書き出される。
ERROR	アダプタにおけるエラーの発生。エラーメッセージは、ユーザに応じてインターナショナルライズおよびローカライズされる。
WARN	エラーではないが、アダプタに問題を引き起こす可能性がある状態。警告メッセージは、ユーザに応じてインターナショナルライズおよびローカライズされる。

表 5-1 ロギング ツールキットの優先度（続き）

優先度	意味
INFO	ユーザに応じてインターナショナルライズおよびローカライズされる情報メッセージ。
DEBUG	デバッグ メッセージ。デバッグ メッセージは、コンポーネントの内部要素がどのように機能しているかを調べるために使用される。通常、インターナショナルライズされない。

BEA_WLS_SAMPLE_ADK カテゴリには下記の子要素があるので、その重要度は「WARN」となっています。

```
<priority value='WARN' class='com.bea.logging.LogPriority' />
```

優先度を使用されるクラスは、com.bea.logging.LogPriorityである必要があります。

カテゴリへの優先度の割り当て

カテゴリには優先度を割り当てることができ、優先度が割り当てられていないカテゴリの場合、優先度が割り当てられている最も近い祖先から優先度を継承します。すなわち、あるカテゴリの継承優先度は、そのカテゴリの階層で最初に行き当たった NULL でない優先度になります。

ログ メッセージは、優先度がそのカテゴリの優先度以上の場合、ログ送り先に送られます。それ以外の場合は、メッセージはログには書き込まれません。優先度が割り当てられていないカテゴリは、階層から優先度を継承します。最終的にすべてのカテゴリが優先度を継承できるようにするため、ルート カテゴリには必ず優先度が割り当てられます。継承優先度 q のカテゴリ内の優先度 p のログステートメントは、 $p \geq q$ の場合、有効になります。このルールでは、優先度の順位が $\text{DEBUG} < \text{INFO} < \text{WARN} < \text{ERROR} < \text{AUDIT}$ のようになっているという想定に基づきます。

メッセージ アペンダ

ログイング フレームワークでは、アペンダと呼ばれるインタフェースを使用することにより、1つのアダプタで複数の送り先にログ メッセージを出力できます。Log4j には、以下の送り先に対応するアペンダがあります。

- コンソール
- ファイル
- リモート ソケット サーバ
- NT イベント ロガー
- リモート UNIX Syslog デーモン

さらに、ADK ログイング ツールキットには、ご使用の **WebLogic Server** ログへのログ メッセージの送信を呼び出せるアペンダも用意されています。

1つのカテゴリは、複数のアペンダを参照できます。あるカテゴリで有効なログイング要求は、カテゴリ階層の上位にある全アペンダとカテゴリ内にあるすべてのアペンダに対しても転送されます。すなわち、アペンダはカテゴリ階層から累加的に継承されます。

たとえば、コンソールアペンダがルート カテゴリに追加されると、有効なログイング要求はすべて、少なくともコンソールに表示されます。ここで、さらにファイルアペンダがカテゴリ **C** に追加されると、**C** および **C** の子カテゴリの有効なログイング要求はファイルに出力され、コンソールに表示されます。累加フラグを **false** に設定することにより、このデフォルト動作をオーバーライドして、アペンダの継承が累加的に付け加えられるのを止めることができます。

注意： この場合、さらにコンソールアペンダを直接 **C** に追加すると、同じメッセージが2回 (**C** からとルートから) コンソールに表示されます。ルート カテゴリが、常にコンソールにログを出力するためです。

コード リスト 5-1 は、**WebLogic Server** ログに対するアペンダを示しています。

コード リスト 5-1 WebLogic Server ログに対するアペンダを示すサンプルコー

ド

```
<!--
  WeblogicAppender がログ出力を Weblogic ログに送る。Weblogic 外で稼働する
  場合、
  アペンダはメッセージを System.out に書き出す
  -->
  <appender name="WebLogicAppender"
    class="com.bea.logging.WeblogicAppender"/>
</appender>
```

メッセージレイアウト

Log4j では、レイアウトをアペンダに関連付けることでログ メッセージのフォーマットをカスタマイズできます。レイアウトによってログ メッセージのフォーマットが決定され、アペンダは、フォーマットされたメッセージを送り先に転送します。ロギング ツールキットでは、通常 `PatternLayout` を使用してログ メッセージのフォーマットを設定します。`PatternLayout` は、標準 Log4j 配布キットに含まれており、これを使用することにより、C 言語の `printf` 関数に似た変換パターンに従って出力フォーマットを指定できます。

たとえば、変換パターンが「`%-5p%d{DATE} %c{4} %x - %m%n`」の `PatternLayout` を呼び出すと、次のようなメッセージが生成されます。

```
AUDIT 21 May 2001 11:00:57,109 BEA_WLS_SAMPLE_ADK - admin opened
connection to EIS
```

このパターンの各パラメータの意味は以下のとおりです。

- `%-5p` は、メッセージの優先度を示す。上記の例では `AUDIT` となっています。
- `%d{DATE}` は、メッセージの日付を示す。上記の例では `21 May 2001 11:00:57,109` となっています。
- `%c{4}` は、ログ メッセージのカテゴリを示す。上記の例では `BEA_WLS_SAMPLE_ADK` となっています。

「-」の後ろのテキストは、メッセージ内容です。

コンポーネントの結合

コード リスト 5-2 では、サンプルアダプタの新しいカテゴリを宣言し、新しいカテゴリに優先度を関連付け、アペンダを宣言することでログ メッセージの送信先のファイルのタイプを指定しています。

コード リスト 5-2 新しいログ カテゴリを宣言するサンプル XML コード

```
<!--
重要 !!! アダプタのルート カテゴリ、これを一意のものにすることで、
他のアダプタがカテゴリにログングされるのを防ぐ。
-->
<category name='BEA_WLS_SAMPLE_ADK' class='com.bea.logging.LogCategory'>
  <!--
  デフォルト優先度レベルで実行時に変更可
  DEBUG は、アダプタのコード ベースの全メッセージ
  INFO は、情報、警告、エラー、監査ログ
  WARN は、警告、エラー、監査ログ
  ERROR は、エラー、監査ログ
  AUDIT は、監査ログ
  -->
  <priority value='WARN' class='com.bea.logging.LogPriority' />
  <appender-ref ref='WebLogicAppender' />
</category>
```

注意： クラスは `com.bea.logging.LogCategory` と指定します。

ロギングの設定方法

注意： 次の手順は、`GenerateAdapterTemplate` ユーティリティを実行して開発環境の複製が作成されていることを前提としています。このユーティリティの詳細については、第4章「カスタム開発環境の作成」を参照してください。

アダプタに対してロギング フレームワークをセットアップするには

1. アダプタで使用されるすべての基本コンポーネントを確認します。たとえば、アダプタに `EventGenerator` があれば、`EventGenerator` コンポーネントが必要です。また、アダプタが設計時 GUI をサポートする場合は、設計時コンポーネントが必要となります。
2. 複製したアダプタから、基本ログ コンフィグレーション ファイルを開きます。このファイルは、`WLI_HOME/adapters/ADAPTER/src/` ディレクトリ内にあり、拡張子は `.xml` です。たとえば、`DBMS` サンプルアダプタ コンフィグレーション ファイルは、
`WLI_HOME/adapters/dbms/src/BEA_WLS_DBMS_ADK.xml` になります。
3. 基本ログ コンフィグレーション ファイルに、手順 1 で確認したアダプタ コンポーネントすべてのカテゴリ要素を追加します。各カテゴリ要素に優先度を設定します。コード リスト 5-3 は、優先度が `DEBUG` の `EventGenerator` のカテゴリを追加する方法を示しています。

コード リスト 5-3 `EventGenerator` ログ カテゴリを優先度 `DEBUG` で追加するサンプル コード

```
<category name='BEA_WLS_DBMS_ADK.EventGenerator'  
          class='com.bea.logging.LogCategory'>  
  <priority value='DEBUG'  
            class='com.bea.logging.LogPriority' />  
</category>
```

5 ログイング ツールキットの使い方

4. 必要なアペンダを決め、それをコンフィグレーションファイルで指定します。必要に応じて、メッセージフォーマット情報を追加します。コードリスト 5-4 は、<appender> 要素内に基本ファイルアペンダを追加する方法を示しています。<layout> 要素内の指定により、メッセージフォーマットが識別されます。

注意： デフォルトでは、WebLogicAppender は WebLogic Integration 7.0 が提供するすべてのサンプルアダプタで使用されます。

コード リスト 5-4 ファイルアペンダおよびレイアウトパターンを追加するサンプルコード

```
<!-- 基本ファイル アペンダ -->

<appender name='FileAppender'
  class='org.apache.Log4j.FileAppender' >

  <!-- 出力をファイルに送信 -->

  <param name='File' value='BEA_WLS_DBMS_ADK.log' />

  <!-- 既存を短縮 -->

  <param name="Append" value="true" />

  <!-- 基本 LOG4J パターン レイアウトを使用 -->

  <layout class='org.apache.Log4j.PatternLayout' >
    <param name='ConversionPattern' value='%-5p %d{DATE} %c{4}
      %x - %m%n' />
  </layout>

</appender>
```

ここで、以下のコンフィグレーションファイルの設定を確認します。

- `WLI_HOME/adapters/ADAPTER/src/eventrouter/web-inf/web.xml` : AbstractEventGenerator が、この基本コンフィグレーションファイルに入力されたログイング情報を使用して、初期化時にログフレームワークのコンフィグレーションを行います。

- `WLI_HOME/adapters/ADAPTER/src/rar/META-INF/ra.xml` および `weblogic-ra.xml` : `AbstractManagedConnectionFactory` が、この基本コンフィグレーション ファイルに入力されたロギング情報を使用して、初期化時にログ フレームワークのコンフィグレーションを行います。
- `WLI_HOME/adapters/ADAPTER/src/war/web-inf/web.xml` : `RequestHandler` (`AbstractDesignTimeRequestHandler` の親) が、この基本コンフィグレーション ファイルに入力されたロギング情報を使用して、初期化時にログ フレームワークのコンフィグレーションを行います。

これらのパスにおいて、`ADAPTER` はアダプタ名を示します。たとえば、`DBMS` サンプル アダプタの場合、関連付けられたコンフィグレーション ファイルのパス名は次のようになります。

`WLI_HOME/adapters/dbms/src/rar/META-INF/ra.xml`

ロギング フレームワークのクラス

ロギング フレームワークの基本コンセプトを理解した上で、さらにロギング ツールキットに用意されている 3 つのメイン クラスについても理解してください。

- `com.bea.logging.ILogger`
- `com.bea.logging.LogContext`
- `com.bea.logging.LogManager`

`com.bea.logging.ILogger`

このクラスは、ロギング フレームワークのメイン インタフェースです。ロギング メッセージ用にたくさんの便利なメソッドがあります。

「ロギングの設定方法」では、基本ログ コンフィグレーション ファイルでロギングのコンフィグレーション方法を説明しています。また、以下に示したロギングメソッドを実装することによってもプログラム上でロギングのコンフィグレーションを行うことができます。

- `logger.setPriority ("DEBUG")` – 現行の `ILogger` から出力されるメッセージの最低優先度を変更します。
- `logger.addRuntimeDestination (writer)` – コンテナが `PrintWriter` をアダプタに渡すときに使用されるアペンダを追加します。
- `logger.warn ("Some message", true)` – 優先度レベルが `WARN` のメッセージを、`ResourceBundle` を使用しないでログに記録します。ブール値は、文字列がメッセージであり、キーではないことを示します。
- `logger.warn ("someKey")` – 優先度レベルが `WARN` のメッセージを、`ResourceBundle` で "someKey" によってルックアップすることによってログに記録します。
- `logger.info ("someKey", anObjArray)` – 優先度レベルが `INFO` のメッセージを、someKey によってテンプレートを `ResourceBundle` でルックアップし、ブランクを `anObjArray` の要素で埋めてログに記録します。
- `logger.error (exception)` – 優先度レベルが `ERROR` のメッセージを、このメソッドに例外 (`Throwable`) を渡すことによってログに記録します。メソッドは、`getMessage()` を呼び出し、スタックトレースを組み込みます。(`Throwable` を引数として取るログिंग メソッドは、すべてスタックトレースをログに記録します。)

com.bea.logging.LogContext

このクラスは、ログिंग フレームワークの `ILogger` インスタンスの識別に必要な情報をカプセル化します。現行では、`LogContext` は、ログ カテゴリ名、および `en_US` などのロケールをカプセル化します。このクラスは、ログ マネージャ内の `ILogger` インスタンスを一意に識別するための主キーです。

com.bea.logging.LogManager

このクラスのメソッドによって、ログिंग フレームワークをコンフィグレーションできるようになり、`ILogger` インスタンスにアクセスすることもできます。

使用するアダプタに合うロギング ツールキットを適切にコンフィグレーションするために、ADK では、コード リスト 5-5 に示す引数を持つ `configure()` メソッドを `LogManager` に実装しています。

コード リスト 5-5 ロギング ツールキットのコンフィグレーション用のサンプルコード

```
public static LogContext
    configure(String strLogConfigFile,
              String strRootLogContext,
              String strMessageBundleBase,
              Locale locale,
              ClassLoader classLoader)
```

表 5-2 は、`configure()` によって渡される引数の説明です。

表 5-2 `configure()` の引数

引数	説明
<code>strLogConfigFile</code>	アダプタのログ コンフィグレーション情報が入っているファイル。ファイルは、クラスパス上におく必要がある。使用するアダプタのメイン JAR ファイルにインクルードして、そのアダプタの WAR および RAR ファイルにこのファイルがインクルードされるようにするとよい。このファイルは、 <code>Log4j.dtd</code> に準拠する必要がある。 <code>Log4j.dtd</code> ファイルは、 <code>WebLogic Integration</code> の <code>Log4j.jarfile</code> にある。
<code>strRootLogContext</code>	アダプタのカテゴリ階層の論理ルート名。サンプルアダプタの場合、この値は <code>BEA_WLS_SAMPLE_ADK</code> である。
<code>strMessageBundleBase</code>	アダプタのメッセージバンドルの基本名。ADK では、メッセージバンドルを使用する必要がある。サンプルアダプタの場合、この値は <code>BEA_WLS_SAMPLE_ADK</code> である。

表 5-2 configure() の引数 (続き)

引数	説明
locale	ユーザの国と言語。ロギング ツールキットでは、ロケールに基づいてカテゴリをさまざまな階層に編成する。たとえば、アダプタが en_US および fr_CA という 2 つのロケールをサポートする場合、ロギング ツールキットでは、en_US 用と fr_CA 用に 1 つずつ、2 つの階層を管理する。
classLoader	LogManager が ResourceBundles やログ コンフィグレーション ファイルなどのリソースをロードするときに使用する ClassLoader。

コンフィグレーションが完了すると、LogContext オブジェクトを指定することにより、アダプタで使用する ILogger インスタンスを取得できます。

コード リスト 5-6 LogContext オブジェクトを指定するサンプルコード

```
LogContext logContext = new LogContext("BEA_WLS_SAMPLE_ADK",
    java.util.Locale.US);

ILogger logger = LogManager.getLogger(logContext);
logger.debug("I'm logging now!");
```

ADK では、ほとんどのログ コンフィグレーションおよび設定は自動で行われません。com.bea.adapter.spi.AbstractManagedConnectionFactory クラスは、サービス アダプタのロギング ツールキットをコンフィグレーションし、AbstractEventGenerator はイベント アダプタのロギング ツールキットをコンフィグレーションします。さらに、ADK に用意されているクライアント コネクタ インタフェース (CCI) およびサービス プロバイダ インタフェース (SPI) の基本クラスはすべて、ILogger およびその関連付けられた LogContext へのアクセス機能を備えています。

アダプタには、EIS に対する通信を設定するのに使用するソケット レイヤなど、CCI/SPI レイヤをサポートするアダプタ内のレイヤも含まれる場合があります。アダプタが適切な ILogger オブジェクトにアクセスする方法は、2 つあります。

- CCI/SPI レイヤは、下位レイヤに `LogContext` オブジェクトを渡すことができます。この方法は、余分なオーバーヘッドがかかります。
- CCI レイヤは、現在実行しているスレッドに対して、コード内の可能な最も近い場所で `LogContext` を設定できます。ADK の `com.bea.adapter.cci.ConnectionFactoryImpl` クラスは、現在実行中のスレッドに対して、`getConnection()` メソッド中に `LogContext` を設定します。この `getConnection()` メソッドが、クライアント プログラムとアダプタの間の最初の接点になります。したがって、アダプタ内の下位のレイヤは、以下のコードを使用することにより、現在実行中のスレッドの `LogContext` に安全にアクセスできます。

コード リスト 5-7 現在のスレッドに対する `LogContext` にアクセスするためのコード

```
public static LogContext getLogContext(Thread t)
    throws IllegalStateException, IllegalArgumentException
```

また、`LogManager` には以下のような便利なメソッドがあります。

```
public static ILogger getLogger() throws IllegalStateException
```

このメソッドは、現在実行中のスレッドに対して、`ILogger` を提供します。この方法を使用するには注意点が1つあります。下位レイヤには、そのメンバーとして `LogContext` または `ILogger` を格納しないでください。下位レイヤが、これらを `LogManager` から動的に取り出すようにします。`LogContext` が現在実行中のスレッドに対して設定される前にこのメソッドが呼び出されると、`IllegalStateException` が返されます。

ログ メッセージのインターナショナルライゼーションとローカライゼーション

インターナショナルライゼーション (I18N) とローカライゼーション (L10N) は、ADK ログイング フレームワークの中心的なコンセプトです。ILogger インタフェースで、ログイングに使用するすべてのメソッドは、デバッグ メソッドを除き、I18Nを提供しています。その実装は、Java インターナショナルライゼーション規格に従っており、ResourceBundle オブジェクトを使用して、ロケール固有のメッセージまたはテンプレートを格納します。Java 言語の I18N および L10N 規格の使用方法については、Sun Microsystems 社による優れたオンラインチュートリアルがあります。

マルチスレッド コンポーネントでのコンテンツ情報の保存

現実に使用されているシステムの多くは、複数のクライアントを同時に処理する必要があります。そのようなシステムの典型的なマルチスレッド実装では、スレッドごとに異なるクライアントを処理します。ログイングは、特に複雑な分散アプリケーションのトレースおよびデバッグに適しています。2つのクライアント間のログイング出力を区別する一般的な方法は、クライアントごとに個別カテゴリをインスタンス化する方法ですが、この方法には欠点があります。カテゴリが増大し、これらを管理するオーバーヘッドが大きくなるという点です。

より負担の小さい方法は、同じクライアント対話で開始された各ログ要求に対してユニークな識別子でスタンプを付す方法です。Neil Harrison氏が、この方法を『*Pattern Languages of Program Design 3*』(R. Martin, D. Riehle, F. Buschmann 編集。Addison-Wesley, 1997) の「Patterns for Logging Diagnostic Messages」で説明しています。

各要求にユニークな識別子でスタンプを付けるには、ユーザはコンテキスト情報を **Nested Diagnostic Context (NDC)** にプッシュします。ロギング ツールキットには、NDC メソッドへのアクセスのための独立したインタフェースがあります。このインタフェースは、`getNDCInterface()` メソッドを使用して `ILogger` から取得されます。

NDC 出力は、XML コンフィグレーション ファイル (`%x` 記号による) でオンになります。ログ要求が行われると、適切なロギング フレームワーク コンポーネントが、現在のスレッドに対する全 NDC スタックをログ出力に組み込みます。ユーザはこの処理に関わる必要はありません。ユーザの作業は、コード内の数箇所決まった場所で `push` および `pop` メソッドを使用して、正しい情報を NDC に入力するだけです。

コード リスト 5-8 サンプル コード

```
public void someAdapterMethod(String aClient) {
    ILogger logger = getLogger();
    INestedDiagnosticContext ndc = logger.getNDCInterface();
    // 全ログ メッセージについてこのクライアント名を追跡する
    ndc.push("User name=" + aClient);
    // メソッド本体 . . .
    ndc.pop();
}
```

NDC は、使用するアダプタの `CCI Interaction` オブジェクト内で使用します。

6 サービスアダプタの開発

サービスアダプタは、クライアントから XML 要求ドキュメントを受け取ると、基本となる EIS で該当する関数を呼び出します。このサービスアダプタはメッセージのコンシューマで、応答を返す場合と返さない場合があります。サービスアダプタでは、次の 4 つの機能が実行されます。

- 外部クライアントからサービス要求を受信します。
- XML 要求ドキュメントを EIS 固有のフォーマットに変換します。この要求ドキュメントは、サービスの要求 XML スキーマに従って作成されます。また、要求 XML スキーマは、EIS のメタデータに基づいています。
- EIS で該当する関数を呼び出し、その関数からの応答を待ちます。
- EIS 固有のデータ形式からサービスの応答 XML スキーマに従った XML 形式に、応答を変換します。応答 XML スキーマは、EIS のメタデータに基づいています。

この章の内容は以下のとおりです。

- 実行時環境におけるサービスアダプタ
- イベントの処理フロー
- 手順 1: 環境要件の調査
- 手順 2: 開発環境のコンフィグレーション
- 手順 3: SPI の実装
- 手順 4: CCI の実装
- 手順 5: アダプタのテスト
- 手順 6: アダプタのデプロイ

WebLogic Integration に限定されない J2EE 準拠アダプタ

この章で説明することは、主に WebLogic Integration で使用するアダプタの開発手順です。ADK により WebLogic Integration 環境以外でも使用可能なアダプタを開発することはできますが、その場合は、所定の修正を加えた手順に従ってください。手順については、附録 A 「WebLogic Integration に限定されないアダプタの作成」を参照してください。

実行時環境におけるサービス アダプタ

図 6-1 および 図 6-2 は、実行時環境でサービス アダプタが使用される場合に実行されるプロセスを示しています。図 6-1 では非同期サービス アダプタ、図 6-2 では同期アダプタのプロセスを示しています。

図 6-1 実行時環境における非同期サービスアダプタ

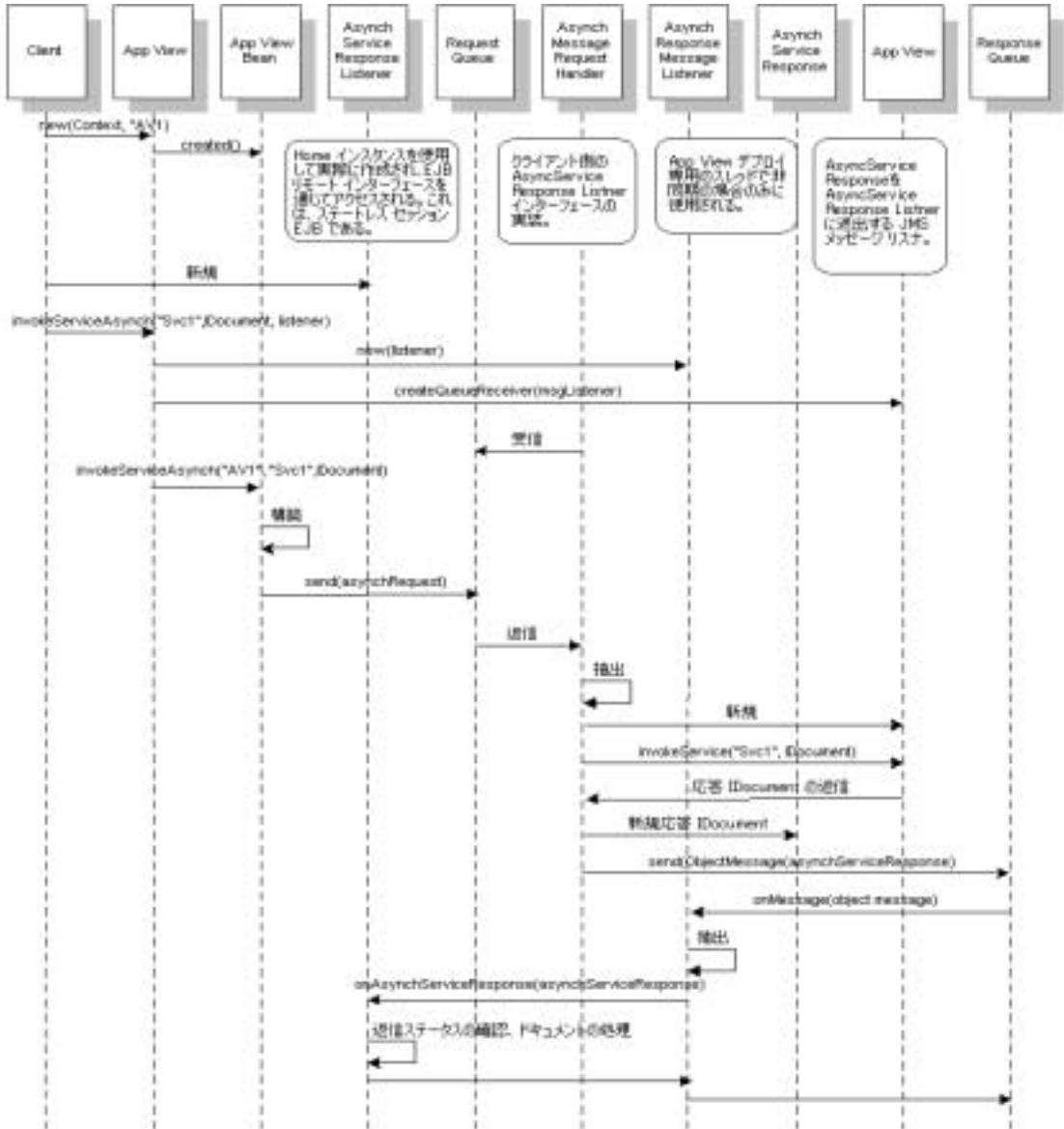
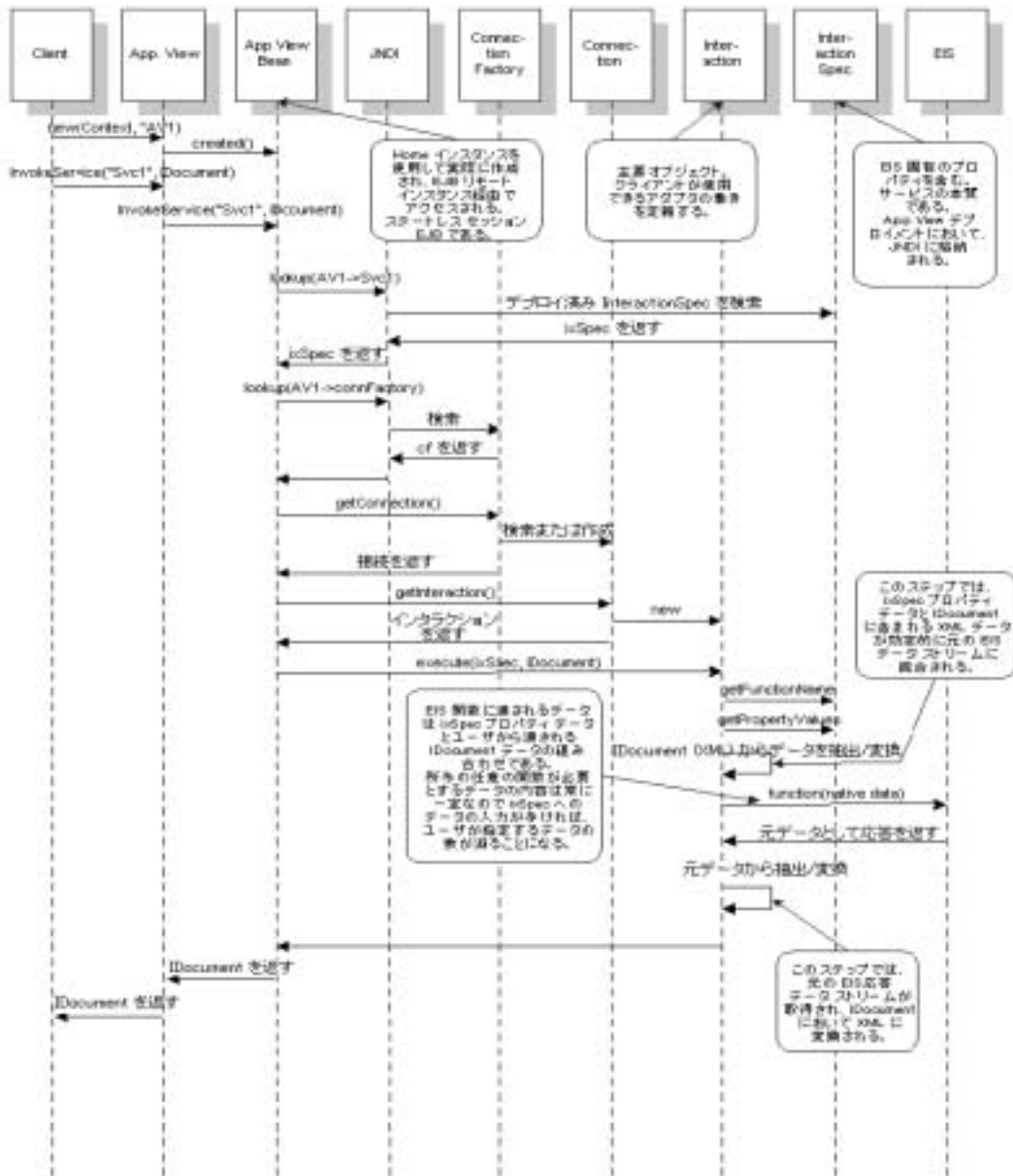


図 6-2 実行時環境における同期サービスアダプタ



イベントの処理フロー

図 6-3 は、サービスアダプタの開発手順の概要です。

図 6-3 サービスアダプタ開発プロセスにおけるイベントのフロー



手順 1 : 環境要件の調査

サービスアダプタの開発を開始する前に、サービスアダプタをサポートするために環境に必要なリソースを識別する必要があります。この手順では、開発環境における要件について詳しく説明します。必要なリソースの詳細なリストについては、附録 D 「アダプタ設定ワークシート」を参照してください。

- 必要な EIS とそれに適したサービスの識別

EIS に関する情報に基づいてバックエンド機能に対するインタフェースを決定します。

- 高コストの接続オブジェクトの決定

EIS 内の機能を呼び出すには、高コストの接続オブジェクトが必要です。この機能は、EIS と通信を行うために必要です。

高コストの接続オブジェクトには、ソケット接続や DBMS 接続のようなシステムリソースの割り当てが必要です。J2EE コネクタアーキテクチャの大きな利点の 1 つは、アプリケーションサーバがこのようなオブジェクトをプールしてくれることです。アダプタのオブジェクトはアプリケーションサーバによってプールされるため、このオブジェクトを決定する必要があります。

- 必要なセキュリティの決定

接続要求パスを通じて接続認証を渡すために、アダプタは `ConnectionRequestInfo` クラスを実装する必要があります。このような実装を容易にするため、ADK では `ConnectionRequestInfoMap` クラスを提供しています。このクラスを使用して、ユーザ名およびパスワードなどの認可情報を接続オブジェクトにマップすることができます。

ADK は、*J2EE Connector Architecture Specification 1.0* に準拠しています。接続アーキテクチャセキュリティの詳細については、このドキュメントの「Security」節を参照してください。以下の URL から簡単に印刷できる PDF 形式の仕様をダウンロードできます。

<http://java.sun.com/j2ee/>

- アダプタに必要なトランザクションサポートタイプの決定

以下のトランザクションの境界設定から、どのタイプをアダプタに実装するかを決定します。

- ローカルトランザクションの境界設定
- XA 準拠トランザクションの境界設定

注意： トランザクションの境界設定の詳細については、6-25 ページの「トランザクション境界設定」または次に示すサイトを参照してください。

http://java.sun.com/blueprints/guidelines/designing_enterprise_applications/transaction_management/platform/index.html

手順 2 : 開発環境のコンフィグレーション

この手順では、環境をコンフィグレーションする 4 つの手順（手順 2a から手順 2b まで）について説明します。

注意： この手順は、`GenerateAdapterTemplate` ユーティリティを実行することで簡単に行えます。詳細については、第 4 章「カスタム開発環境の作成」を参照してください。

手順 2a : ディレクトリ構造の設定

WebLogic Integration のインストール時に作成されるディレクトリ構造は、アダプタを実行するためだけでなく、ADK を使用するためにも必要です。ADK 関連のファイルは、`WLI_HOME/adapters/` ディレクトリ（`WLI_HOME` は、WebLogic Integration をインストールしたディレクトリ）に格納されています。インストール時に、`WLI_HOME` ディレクトリに必要なディレクトリおよびファイルが含まれているかを必ず確認してください。

次の表では、`WLI_HOME` のディレクトリ構造について説明します。

表 6-1 ADK ディレクトリ構造

パス名	説明
アダプタ	ADK を含むディレクトリ。

表 6-1 ADK ディレクトリ構造

パス名	説明
adapters/src/war	.jsp ファイル、イメージなどが格納されるディレクトリ。このディレクトリ内のすべてのファイルは、アダプタの WAR ファイルに組み込む必要がある。
adapters/utils	JAR ファイルにタイムスタンプするファイルを含め、ビルド プロセスで使用するファイルを含むディレクトリ。
adapters/dbms	ADK を使用して構築された J2EE 準拠のサンプル アダプタが格納されるディレクトリ。
adapters/dbms/docs	サンプルアダプタのユーザ ガイド、リリース ノート、およびインストール ガイドが格納されるディレクトリ。
adapters/sample	独自のアダプタを開発する際に使用できるサンプル アダプタが格納されるディレクトリ。
adapters/sample/project	Apache Jakarta Ant ビルド ファイルの build.xml が格納されるディレクトリ。このファイルには、ソースコードのコンパイル、JAR ファイルと EAR ファイルの生成、および Javadoc 情報の生成に使用されるビルド情報が格納される。アダプタの構築に関する詳細は、6-10 ページの「手順 2c: ビルド プロセスの設定」を参照。
adapters/sample/src	アダプタのあらゆるソースコードが格納されるディレクトリ。アダプタにソースコードを含めるかどうかは任意に選択する。
adapters/sample/src/ BEA_WLS_SAMPLE_ADK.properties	アダプタがインターナショナルライゼーションおよびローカライゼーションで使用するメッセージが定義されているファイル。
adapters/sample/src/ BEA_WLS_SAMPLE_ADK.xml	ロギングフレームワークに必要な基本的なコンフィグレーションファイル。独自のアダプタロギングコンフィグレーションファイルを開発するときこのファイルを使用する。
adapters/sample/src/ eventrouterer/WEB-INF/web.xml	イベント ルータ Web アプリケーション用のコンフィグレーションファイル。

表 6-1 ADK ディレクトリ構造

パス名	説明
adapters/sample/src/rar/ META-INF/ra.xml	J2EE 準拠アダプタに関するコンフィグレーション情報が定義されているファイル。このファイルは、ADK の実行時フレームワークで必要なパラメータを確認するための参考ガイドとして使用する。
adapters/sample/src/rar/META-INF/weblogic-ra.xml	Weblogic Server J2EE エンジンに固有の J2EE 準拠アダプタに関するコンフィグレーション情報が定義されているファイル。このファイルは、使用するアダプタに応じて weblogic-ra.xml ファイルを設定する際の例として使用する。このファイルは Weblogic Server で必要となる。
adapters/sample/src/sample	アダプタのソースコードが格納されるディレクトリ。
adapters/sample/src/war	.jsp ファイル、.html ファイル、イメージなどが格納されるディレクトリ。このディレクトリ内のすべてのファイルは、アダプタの Web アプリケーションアーカイブ (.war) ファイルに組み込む必要がある。
adapters/sample/src/war/WEB-INF/web.xml	Web アプリケーション記述子
adapters/sample/src/war/WEB-INF/weblogic.xml	Web アプリケーション用の WebLogic Server 固有の属性が定義されているファイル。
adapters/sample/src/ear/META-INF/application.xml	アダプタのアプリケーションビューをコンフィグレーションするためのコネクタおよび Web アプリケーションが収められている J2EE アプリケーション。

ディレクトリ構造の変更

GenerateAdapterTemplate を使用して開発ツリーを複製すると、adapters/sample のすべてのディレクトリの内容が自動的に複製されて、新しい開発環境に合わせて更新されます。

変更は、ファイル `WLI_HOME/adapters/ADAPTER/docs/api/index.html` (`ADAPTER` は、新しい開発ディレクトリ名) に反映されます。また、新しいアダプタに応じて **WebLogic Integration** を設定する際に、このファイル内のコードをコピーして、そのアダプタの `config.xml` ファイルに貼り付けることもできます。

手順 2b : アダプタ論理名の割り当て

アダプタに論理名を割り当てます。通常、アダプタの論理名は、次のようにアンダースコアで区切られた 3 つの項目 (ベンダ名、アダプタに接続される EIS のタイプおよび EIS のバージョン) で構成されます。

```
vendor_EIS-type_EIS-version
```

例 :

```
BEA_WLS_SAMPLE_ADK
```

アダプタ論理名の詳細については、2-6 ページの「アダプタ論理名」を参照してください。

手順 2c : ビルド プロセスの設定

ADK では、**Ant** という Java 言語のみを使用して作成した Java のビルド ツールに基づくビルド プロセスが採用されています。**Ant** の詳細については、3-4 ページの「**Ant** ベースのビルド プロセス」を参照してください。**Ant** の使用方法の詳細については、以下のサイトを参照してください。

<http://jakarta.apache.org/ant/index.html>

ADK に付属のサンプルアダプタ (`WLI_HOME/adapters/sample/project` に格納されている) には、`build.xml` というファイルが入っています。このファイルは、サンプルアダプタの **Ant** ビルド ファイルです。このファイルには、**J2EE** 準拠アダプタを構築するのに必要なタスクが収められています。`GenerateAdapterTemplate` ユーティリティを実行して、アダプタの開発ツリーを複製すると、使用するアダプタ専用の `build.xml` ファイルが作成されます。この自動ファイル生成により、サンプル `build.xml` ファイルをカスタマイズす

る手間が省け、かつコードの正確性が保証されます。
 GenerateAdapterTemplate ユーティリティの使用方法の詳細については、第 4 章「カスタム開発環境の作成」を参照してください。

Manifest ファイル

GenerateAdapterTemplate によって作成されるファイルの中には、MANIFEST.MF という manifest ファイルがあります。このファイルには、このファイルを使用する各コンポーネントに対するクラスロード指示が定義されています。manifest ファイルは、ear/META-INF を除くすべての /META-INF ディレクトリに作成されます。

コード リスト 6-1 は、サンプルアダプタに組み込まれている manifest ファイルの例を示しています。

コード リスト 6-1 Manifest ファイルの例

```
Manifest-Version: 1.0
Created-By:BEA Systems, Inc.
Class-Path: BEA_WLS_SAMPLE_ADK.jar adk.jar wlai-core.jar
            wlai-client.jar
```

このファイルの最初の 2 行には、バージョン情報およびベンダ情報が含まれます。3 行目では、関連クラスパスまたはクラスロード指示が定義されます。Class-Path プロパティには、コンポーネントが必要とするリソースへの参照と共有 JAR ファイルのリストが含まれます。(リストでは、ファイル名がスペースで区切られています。) JAR ファイルは、EAR ファイルの共有エリアに組み込む必要があります。(詳細については、2-11 ページの「エンタープライズアーカイブ (EAR) ファイル」を参照してください。)

JAR ツールでは、Class-Path: 行の長さの上限は 72 文字です。72 文字を超える行は次の行に回され、次に示すようにスペースで始まります。

```
Class-Path: .....72 文字のクラスパス
<スペース> クラスパスの続き
```

サンプルの ADK アダプタでは、以下の Ant コマンドによりすべての共有 JAR ファイルは、1 つの JAR ファイル (shared.jar) に統合されます。

コード リスト 6-2

```
<jar jarfile='${LIB_DIR}/shared.jar'>
  <zipfileset src='${LIB_DIR}/${JAR_FILE}'>
    <exclude name='META-INF/MANIFEST.MF' />
  </zipfileset>
  <zipfileset src='${WLI_LIB_DIR}/adk.jar'>
    <exclude name='META-INF/MANIFEST.MF' />
  </zipfileset>
  <zipfileset src='${WLI_LIB_DIR}/wlai-core.jar'>
    <exclude name='META-INF/MANIFEST.MF' />
  </zipfileset>
  <zipfileset src='${WLI_LIB_DIR}/wlai-client.jar'>
    <exclude name='META-INF/MANIFEST.MF' />
  </zipfileset>
</jar>
<jar jarfile='${LIB_DIR}/${EAR_FILE}'>
  <fileset dir='${basedir}' includes='version_info.xml' />
  <fileset dir='${SRC_DIR}/ear'
    includes='META-INF/application.xml' />
  <fileset dir='${LIB_DIR}'
    includes='shared.jar,${RAR_FILE},${WAR_FILE},
    ${EVENTROUTER_WAR_FILE}' />
</jar>
```

注意： ファイル名 MANIFEST.MF は、WAR ファイルに組み込む場合にはすべて大文字で表記する必要があります。正しく表記されていない場合、UNIX システムでは認識されず、エラーが発生します。

build.xml のコンポーネント

build.xml がどのように機能するかを知るには、ファイルを開いてコンポーネントをレビューします。ここでは、主なファイル要素について説明します。build.xml の内容をレビューする際にこれらの説明を参照してください。

注意： 以下の例はサンプルアダプタからの引用で、複製バージョンのものではありません。

1. 最初の行では、ルートプロジェクト要素の名前属性が設定されます。

```
<project name='BEA_WLS_SAMPLE_ADK' default='all' basedir='.'>
```

2. 名前は、次のリスト例で示すように、アーカイブファイル（JAR、WAR および RAR ファイル）に関連付けられます。

コード リスト 6-3 アーカイブ ファイル名の設定

```
<property name='JAR_FILE' value='BEA_WLS_SAMPLE_ADK.jar' />
<property name='RAR_FILE' value='BEA_WLS_SAMPLE_ADK.rar' />
<property name='WAR_FILE' value='BEA_WLS_SAMPLE_ADK_Web.war' />
<property name='EVENTROUTER_JAR_FILE'
  value='BEA_WLS_SAMPLE_ADK_EventRouter.jar' />
<property name='EVENTROUTER_WAR_FILE'
  value='BEA_WLS_SAMPLE_ADK_EventRouter.war' />
<property name='EAR_FILE' value='BEA_WLS_SAMPLE_ADK.ear' />
```

3. コード リスト 6-4 に、ADK の標準プロパティを示します。

コード リスト 6-4 標準 ADK プロパティ

```
<property name='ADK' value='${WLI_LIB_DIR}/adk.jar' />
<property name='ADK_WEB' value='${WLI_LIB_DIR}/adk-web.jar' />
<property name='ADK_TEST' value='${WLI_LIB_DIR}/adk-test.jar' />
<property name='ADK_EVENTGENERATOR' value='${WLI_LIB_DIR}/
  adk-eventgenerator.jar' />
<property name='BEA' value='${WLI_LIB_DIR}/bea.jar' />
<property name='LOGTOOLKIT' value='${WLI_LIB_DIR}/
  logtoolkit.jar' />
<property name='WEBTOOLKIT' value='${WLI_LIB_DIR}/
  webtoolkit.jar' />
<property name='WLAI_CORE' value='${WLI_LIB_DIR}/
  wlai-core.jar' />
<property name='WLAI_CLIENT' value='${WLI_LIB_DIR}/
  wlai-client.jar' />
<property name='WLAI_COMMON' value='${WLI_LIB_DIR}/
  wlai-common.jar' />
<property name='WLAI_EVENTROUTER' value='${WLI_LIB_DIR}/
  wlai-eventrouter.jar' />
<property name='XMLTOOLKIT' value='${WLI_LIB_DIR}/
  xmltoolkit.jar' />
<property name='XCCI' value='${WLI_LIB_DIR}/xcci.jar' />
```

これらのプロパティを変更する必要はありませんが、この後に、アダプタで必要な他の JAR ファイルやクラスを追加できます。

4. クラスパスは、次のリストに示すように、コンパイルのためにセットアップされます。

コード リスト 6-5 クラスパスの設定

```
<path id='CLASSPATH'>
  <pathelement location='${SRC_DIR}' />
  <pathelement path='${ADK}:${ADK_EVENTGENERATOR}:
    ${ADK_WEB}:${ADK_TEST}:${WEBTOOLKIT}:${WLAI_CORE}:
    ${WLAI_EVENTROUTER}:${WLAI_CLIENT}' />
  <pathelement path='${WEBLOGIC_JAR}:${env.BEA_HOME}' />
  <pathelement path='${JUNIT}:${HTTPUNIT}:${TIDY}' />
</path>
```

この情報の後に、任意で次の3つのファイルの組み合わせのいずれか呼び出すことができます。

- 以下のサンプル リストで示すアダプタのすべてのバイナリとアーカイブ

コード リスト 6-6 すべてのバイナリおよびアーカイブを呼び出すサンプル

```
<!-- このターゲットにより、アダプタのすべてのバイナリおよび
アーカイブが作成される -->
<target name='all' depends='ear' />
```

- アダプタのすべてのバイナリとアーカイブ、および Javadoc

```
<target name='release' depends='all,apidoc' />
```
- コード リスト 6-7 に示すように、`version_info` ファイルをアーカイブに組み込むように設定

コード リスト 6-7 サンプル `version_info` ファイル

```
<!-- このターゲットにより、version_info ファイルがアーカイブに
組み込まれる -->
<target name='version_info'>
  <java classname='GenerateVersionInfo'>
```

```
<arg line='-d${basedir}' />
<classpath>
  <pathelement path='${WLI_HOME}/adapters/utils:
    ${WEBLOGIC_JAR}' />
</classpath>
</java>
</target>
```

5. このアダプタに対する JAR ファイルの内容が指定されます。アダプタの実行時関係の要素はメインの JAR ファイルに含まれますが、設計時 GUI サポートクラスのような追加クラスは、以下のリストに示すように WAR ファイルまたは他の JAR ファイルに含まれます。

コード リスト 6-8 JAR ファイルに値を設定するためのサンプルコード

```
<target name='jar' depends='packages,version_info'>
  <delete file='${LIB_DIR}/${JAR_FILE}' />
  <mkdir dir='${LIB_DIR}' />
  <jar jarfile='${LIB_DIR}/${JAR_FILE}'>
```

6. アダプタのソースディレクトリからの includes リストが指定されます。ここで説明するサンプルアダプタでは、sample/cci および sample/spi パッケージ内のクラスがすべて含まれ、さらにロギング コンフィグレーションファイルとメッセージバンドルが含まれます。

コード リスト 6-9 Includes リストを組み込むサンプルコード

```
<fileset dir='${SRC_DIR}'
  includes='sample/cci/*.class,sample/spi/*.class,
  sample/eis/*.class,*.xml,*.properties' />
```

7. 以下のリストに示すように、JAR ファイルに関するバージョン情報を示します。

コード リスト 6-10 JAR ファイルのバージョン情報の設定

```
<!-- JAR ファイルのバージョン情報を組み込む -->
  <fileset dir='${basedir}'
           includes='version_info.xml' />
</jar>
```

8. J2EE アダプタ アーカイブ (RAR) ファイルが生成されます。このファイルには、アダプタに必要なすべてのクラスおよび JAR ファイルが含まれます。このファイルは、アダプタが依存する J2EE 準拠アプリケーション サーバにデプロイできます。この例には、以下のターゲットがあります。

- この RAR ファイルのバージョン情報
- アダプタの deployment descriptor

以下のリストに、サンプルアダプタに対し RAR ファイルがどのように作成されるかを示します。

コード リスト 6-11 接続アーキテクチャ RAR ファイルを作成するサンプルコード

```
<target name='rar' depends='jar'>
  <delete file='${LIB_DIR}/${RAR_FILE}' />
  <mkdir dir='${LIB_DIR}' />
  <jar jarfile='${LIB_DIR}/${RAR_FILE}'
      manifest='${SRC_DIR}/rar/META-INF/MANIFEST.MF'>
    <fileset dir='${SRC_DIR}/rar' includes='META-INF/ra.xml,
      META-INF/weblogic-ra.xml' excludes=
      'META-INF/MANIFEST.MF' />
  </jar>
</target>
```

9. J2EE Web アプリケーション アーカイブ (WAR) ファイルが生成されます。このファイルには、既存の環境をクリーンアップするコードも含まれています。

コード リスト 6-12 WAR ファイル生成のためのサンプル コード

```

<target name='war' depends='jar'>
<!-- 既存の環境をクリーンアップする -->

    <delete file='${LIB_DIR}/${WAR_FILE}' />
    <copy file='${WLI_HOME}/adapters/src/war/WEB-INF/taglibs/
        adk.tld' todir='${SRC_DIR}/war/WEB-INF/taglibs' />
    <java classname='weblogic.jspc' fork='yes'>
        <arg line='-d ${SRC_DIR}/war -webapp ${SRC_DIR}/
            war -compileAll -depend' />
        <classpath refid='CLASSPATH' />
    </java>

<!-- 最初のアダプタは共通 ADK JSP をコンパイルする必要がある -->

    <java classname='weblogic.jspc' fork='yes'>
        <arg line='-d ${WLI_HOME}/adapters/src/war -webapp
            ${WLI_HOME}/adapters/src/war -compileAll
            -depend' />
        <classpath refid='CLASSPATH' />
    </java>

<war warfile='${LIB_DIR}/${WAR_FILE}'
    webxml='${SRC_DIR}/war/WEB-INF/web.xml'
    manifest='${SRC_DIR}/war/META-INF/MANIFEST.MF'>

<!--
重要！ Exclude the WEB-INF/web.xml file from
the WAR as it already gets included via the webxml attribute
above
-->

    <fileset dir="${SRC_DIR}/war" >
        <patternset >
            <include name="WEB-INF/weblogic.xml" />
            <include name="**/*.html" />
            <include name="**/*.gif" />
        </patternset>
    </fileset>

<!--
重要！ ADK 設計時フレームワークをアダプタの設計時
Web アプリケーションに含める
-->

    <fileset dir="${WLI_HOME}/adapters/src/war" >
        <patternset >
            <include name="**/*.css" />

```

```
<include name="**/*.html"/>
<include name="**/*.gif"/>
<include name="**/*.js"/>
</patternset>
</fileset>

<!-- 設計時 UI をサポートするアダプタのクラスを
含める -->

<classes dir='${SRC_DIR}' includes='sample/web/*.class' />
<classes dir='${SRC_DIR}/war' includes='**/*.class' />
<classes dir='${WLI_HOME}/adapters/src/war' includes=
'**/*.class' />

<!--
アプリケーションで必要とされ、EAR に共有されないすべての JAR
ファイルを、WAR ファイルの WEB-INF/lib ディレクトリ下に
組み込む
-->
```

10. Web アプリケーションで必要とされるすべての JAR ファイルは、
build.xml ファイルの <lib> コンポーネント内に組み込まれています。

コード リスト 6-13 Web アプリケーションが必要とする JAR ファイルを組み込むサンプル コード

```
<lib dir='${WLI_LIB_DIR}' includes='adk-web.jar,
webtoolkit.jar,wlai-client.jar' />
```

11. EAR ファイルが組み込まれます。

コード リスト 6-14 EAR ファイルの組み込み

```
<target name='ear' depends='rar,eventrouterer_jar,war'>
<delete file='${LIB_DIR}/${EAR_FILE}' />
<!-- jar ファイルを組み込むのではなく、
jar ファイルを共有するイベント ルータを組み込む -->
```



```

<delete file='${LIB_DIR}/${EVENTROUTER_WAR_FILE}' />
<delete dir='${SRC_DIR}/eventrouter/WEB-INF/lib' />

<war warfile='${LIB_DIR}/${EVENTROUTER_WAR_FILE}'
    'webxml='${SRC_DIR}/eventrouter/WEB-INF/web.xml
    'manifest='${SRC_DIR}/eventrouter/META-INF/
    MANIFEST.MF'>

    <fileset dir='${basedir}' includes='version_info.xml' />
    <fileset dir="${SRC_DIR}/eventrouter" >
        <patternset>
            <exclude name="WEB-INF/web.xml" />
            <exclude name="META-INF/*.mf" />
        </patternset>
    </fileset>

    <lib dir='${LIB_DIR}' includes='${EVENTROUTER_JAR_
        FILE}' />
    <lib dir='${WLI_LIB_DIR}' includes=
        'adk-eventgenerator.jar,wlai-eventrouter.jar' />
</war>

<jar jarfile='${LIB_DIR}/${EAR_FILE}'>
    <fileset dir='${basedir}' includes='version_info.xml' />
    <fileset dir='${SRC_DIR}/ear' includes=
        'application.xml' />
    <fileset dir='${LIB_DIR}' includes='${JAR_FILE},
        ${RAR_FILE}, ${WAR_FILE}, ${EVENTROUTER_WAR_FILE}' />
    <fileset dir='${WLI_LIB_DIR}' includes='adk.jar,
        wlai-core.jar,wlai-client.jar' />
</jar>

<delete file='${LIB_DIR}/${EVENTROUTER_WAR_FILE}' />
<delete file='${LIB_DIR}/${EVENTROUTER_JAR_FILE}' />
<delete file='${LIB_DIR}/${WAR_FILE}' />
<delete file='${LIB_DIR}/${RAR_FILE}' />
<delete file='${LIB_DIR}/${JAR_FILE}' />

</target>

```

EAR デプロイメント固有のイベント ルータは自身でデプロイすることはできないため、以下のリストで示すように、EAR ターゲット内から呼び出されます。

コード リスト 6-15 EAR 固有の EventRouter を組み込むサンプルコード

```
<delete file='${LIB_DIR}/${EVENTROUTER_WAR_FILE}' />
<delete dir='${SRC_DIR}/eventrouter/WEB-INF/lib' />

<war warfile='${LIB_DIR}/${EVENTROUTER_WAR_FILE}'
    'webxml='${SRC_DIR}/eventrouter/WEB-INF/web.xml
    'manifest='${SRC_DIR}/eventrouter/META-INF/
    MANIFEST.MF'>

    <fileset dir='${basedir}' includes='version_info.xml' />
    <fileset dir="${SRC_DIR}/eventrouter" >
        <patternset >
            <exclude name="WEB-INF/web.xml" />
            <exclude name="META-INF/*.mf" />
        </patternset>
    </fileset>

    <lib dir='${LIB_DIR}' includes='${EVENTROUTER_
        JAR_FILE}' />
    <libdir='${WLI_LIB_DIR}'
        includes='adk-eventgenerator.jar,
        wlai-eventrouter.jar' />

</war>
```

上記リストの EAR ターゲット内に、以下のリストに示すように、すべての共通または共有 JAR ファイルもあります。

コード リスト 6-16 共通または共有 JAR ファイルの組み込みを示すサンプルコード

```
<jar jarfile='${LIB_DIR}/${EAR_FILE}'>

    <fileset dir='${basedir}' includes='version_info.xml' />
    <fileset dir='${SRC_DIR}/ear' includes='application.xml' />
    <fileset dir='${LIB_DIR}' includes='${JAR_FILE}, ${RAR_FILE},
        ${WAR_FILE}, ${EVENTROUTER_WAR_FILE}' />
    <fileset dir='${WLI_LIB_DIR}' includes='adk.jar,
        wlai-core.jar, wlai-client.jar' />

</jar>
```

12. このプロジェクトのすべての Java ソース ファイルがコンパイルされます。

コード リスト 6-17 Java ソースのコンパイル用サンプル コード

```
<target name='packages'>
  <echo message='Building ${ant.project.name}...' />
  <javac srcdir='${SRC_DIR}'
        excludes='war/jsp_servlet/**'
        deprecation='true' debug='true'>
    <classpath refid='CLASSPATH' />
  </javac>
</target>
```

13. 以下のリストに示すように、EventRouter JAR ファイルを構築します。

コード リスト 6-18 EventRouter JAR ファイル構築用サンプル コード

```
<target name='eventrouter_jar' depends='packages,version_info'>
  <delete file='${LIB_DIR}/${EVENTROUTER_JAR_FILE}' />
  <jar jarfile='${LIB_DIR}/${EVENTROUTER_JAR_FILE}'>
    <fileset dir='${SRC_DIR}'
            includes='sample/event/*.class' />
    <fileset dir='${basedir}'
            includes='version_info.xml' />
  </jar>
</target>
```

14. 以下のリストに示すように、J2EE WAR ファイルおよびスタンドアロン デプロイメントに使用されるイベント ルータを生成します。

コード リスト 6-19 スタンドアロン デプロイメント用の EventRouter ターゲットを生成するサンプル コード

```
<target name='eventrouter_war' depends='jar,eventrouter_jar'>
  <delete file='${LIB_DIR}/${EVENTROUTER_WAR_FILE}' />
  <delete dir='${SRC_DIR}/eventrouter/WEB-INF/lib' />
  <war warfile='${LIB_DIR}/${EVENTROUTER_WAR_FILE}' webxml=
```

```
    '${SRC_DIR}/eventrouter/WEB-INF/web.xml'>
    <fileset dir='${basedir}' includes='version_info.xml' />
    <fileset dir='${SRC_DIR}/eventrouter' excludes=
    'WEB-INF/web.xml' />
    <lib dir='${LIB_DIR}' includes='${JAR_FILE},
    ${EVENTROUTER_JAR_FILE}' />
    <lib dir='${WLI_LIB_DIR}' includes='adk.jar,
    adk-eventgenerator.jar,wlai-core.jar,
    wlai-eventrouter.jar,wlai-client.jar' />
    </war>
</target>
```

15. Javadoc が生成されます。

コード リスト 6-20 Javadoc を生成するサンプル コード

```
<target name='apidoc'>
    <mkdir dir='${DOC_DIR}' />
    <javadoc sourcepath='${SRC_DIR}'
    destdir='${DOC_DIR}'
    packagenames='sample.*'
    author='true'
    version='true'
    use='true'
    overview='${SRC_DIR}/overview.html'
    windowtitle='WebLogic BEA_WLS_SAMPLE_ADK Adapter
    API Documentation'
    doctitle='WebLogic BEA_WLS_SAMPLE_ADK Adapter
    API Documentation'
    header='WebLogic BEA_WLS_SAMPLE_ADK Adapter'
    bottom='Built using the WebLogic Adapter
    Development Kit (ADK)'>
    <classpath refid='CLASSPATH' />
    </javadoc>
</target>
```

16. 対応するオブジェクトにより作成されたファイルをクリーンアップするターゲットを示します。

コード リスト 6-21 クリーンアップコードを組み込むサンプルコード

```
<target name='clean' depends='clean_release' />
<target name='clean_release' depends='clean_all,clean_apidoc' />
<target name='clean_all' depends='clean_ear,clean_rar,clean_war,
  clean_eventrouter_war,clean_test' />
<target name='clean_test'>
  <delete file='${basedir}/BEA_WLS_SAMPLE_ADK.log' />
  <delete file='${basedir}/mcf.ser' />
</target>
<target name='clean_ear' depends='clean_jar'>
  <delete file='${LIB_DIR}/${EAR_FILE}' />
</target>
<target name='clean_rar' depends='clean_jar'>
  <delete file='${LIB_DIR}/${RAR_FILE}' />
</target>
<target name='clean_war' depends='clean_jar'>
  <delete file='${LIB_DIR}/${WAR_FILE}' />
  <delete dir='${SRC_DIR}/war/jsp_servlet' />
</target>
<target name='clean_jar' depends='clean_packages,clean_version_
  info'>
  <delete file='${LIB_DIR}/${JAR_FILE}' />
</target>
<target name='clean_eventrouter_jar'>
  <delete file='${LIB_DIR}/${EVENTROUTER_JAR_FILE}' />
</target>
<target name='clean_eventrouter_war' depends='clean_
  eventrouter_jar'>
  <delete file='${LIB_DIR}/${EVENTROUTER_WAR_FILE}' />
</target>
<target name='clean_version_info'>
  <delete file='${basedir}/version_info.xml' />
</target>
<target name='clean_packages'>
  <delete>
    <fileset dir='${SRC_DIR}' includes='**/*.class' />
  </delete>
</target>
<target name='clean_apidoc'>
  <delete dir='${DOC_DIR}' />
</target>
</project>
```

手順 2d : メッセージバンドルの作成

エンド ユーザ向けのメッセージは、メッセージバンドルに入れます。2 つ以上の言語でメッセージを生成できる「`key=value`」の組み合わせを含む `.properties` テキスト ファイル。実行時にロケールおよび地域言語が指定されると、メッセージの内容が該当する「`key=value`」の組み合わせに基づいて解釈され、メッセージがユーザのロケールに従って適切な言語で提示されます。

メッセージバンドルの作成方法については、次に示すサイトにある **JavaSoft** チュートリアルを参照してください。

<http://java.sun.com/docs/books/tutorial/i18n/index.html>

手順 3 : SPI の実装

サービスプロバイダ インタフェース (SPI) には、EIS への接続、管理、トランザクションの環境設定、およびサービス呼び出しのフレームワークが用意されているオブジェクトが入っています。J2EE 準拠アダプタには、`javax.resource.spi` パッケージにあるこれらのインタフェースの実装が必要です。

この節では、SPI の実装に使用できるインタフェースについて説明します。SPI の実装作業には、最低 3 つのインタフェースが必要です (6-24 ページの「基本的な SPI の実装」を参照してください)。ここでは、それぞれのインタフェースを詳しく説明し、さらに、ADK に含まれているサンプルアダプタによるそれらのインタフェース拡張方法を説明します。

まず、3 つのインタフェースについて説明します。次に、追加インタフェースについて詳しく説明し、さらにそれらを使用する理由や、アダプタで使用した場合の利点についても説明します。

基本的な SPI の実装

アダプタに SPI を実装するには、最低でも、次の 3 つのインタフェースの拡張が必要です。

- `ManagedConnectionFactory` – `ManagedConnection` インスタンスのマッチングと作成を行うメソッドによって、接続プールをサポートします。
- `ManagedConnection` – 基本となる EIS との物理的な接続を表します。
- `ManagedConnectionMetaData` – `ManagedConnection` インスタンスに関連付けられた基盤となる EIS インスタンスに関する情報を提供します。

3つのインタフェースは、ここで示す順番で実装するのが理想的です。

この3つのインタフェースのほか、使用するアダプタの要求に合わせて、この節で説明されたほかのインタフェースを任意に実装できます。

ManagedConnectionFactory

`javax.resource.spi.ManagedConnectionFactory`

`ManagedConnectionFactory` インタフェースは、`ManagedConnection` と EIS 固有の接続ファクトリ インスタンス両方のファクトリです。このインタフェースは、`ManagedConnection` インスタンスをマッチングしたり、作成したりするメソッドによって接続プールをサポートします。

トランザクション境界設定

`ManagedConnectionFactory` に不可欠なコンポーネントは、トランザクションの境界設定です。1つのトランザクションにインクルードされるプログラム文を決定します。J2EE では、アプリケーションサーバとアダプタ（およびそのベースのリソース マネージャ）との間のトランザクション管理規約を定義します。トランザクション管理規約は、2つの部分から構成されます。規約は、使用するトランザクションの種類によって異なります。トランザクションには、2つの種類があります。

- XA 準拠トランザクション
- ローカルトランザクション

XA 準拠トランザクション

`javax.transaction.xa.XAResource` ベース規約は、分散トランザクション処理環境 (DTP) におけるトランザクション マネージャとリソース マネージャ間の取り決めです。JDBC ドライバまたは JMS プロバイダは、グローバルトランザクションと、データベースまたはメッセージ サービス接続との間の関連付けをサポートするために、このインタフェースを実装します。

`XAResource` インタフェースは、トランザクションが外部トランザクション マネージャで管理されている環境内のアプリケーション プログラムでの使用を目的としたトランザクション リソースならばどのようなものでもサポート可能です。

そのようなリソースの例として、アプリケーションが複数のデータベース接続を経由してデータにアクセスするようなデータベース管理システムが挙げられます。それぞれのデータベース接続は、トランザクション リソースとして、トランザクション マネージャに登録されます。トランザクション マネージャは、グローバルトランザクションに関連する各接続に対して、`XAResource` を取得します。トランザクション マネージャは、`start()` メソッドを使用して、グローバルトランザクションとリソースを関連付け、`end()` メソッドによってトランザクションとリソースの関連付けを解除します。リソース マネージャは、グローバルトランザクションを、`start()` メソッド呼び出しと `end()` メソッド呼び出しの間にそのデータに対して行われたすべての作業に関連付けます。

トランザクションのコミット時、リソース マネージャは、トランザクションを 2 フェーズ コミット プロトコルに従って準備、コミット、またはロールバックするようにトランザクション マネージャから指示されます。

ローカルトランザクション

ローカルトランザクション管理規約は、アダプタが、そのリソース マネージャで実行されるローカルトランザクションをサポートするために、`javax.resource.spi.LocalTransaction` インタフェースを実装する場合の規約です。この規約によって、アプリケーションサーバがトランザクション管理のインフラストラクチャと実行時環境を整えられます。アプリケーション コンポーネントはこのトランザクション インフラストラクチャを利用して、コンポーネントレベルのトランザクション モデルをサポートします。

トランザクションの境界設定の詳細については、以下の URL を参照してください。

http://java.sun.com/blueprints/guidelines/designing_enterprise_applications/transaction_management/platform/index.html

ADK 実装

ADK では、アダプタに対して **AbstractManagedConnectionFactory** という抽象ファンデーションを提供しています。このファンデーションには、以下の機能があります。

- アダプタの例外およびログメッセージのインターナショナルライゼーションおよびローカライゼーションに関する基本サポート。
- ロギング ツールキットへのフック。
- 標準接続プロパティ（ユーザ名、パスワード、サーバ、接続 URL およびポート）に対するゲッター メソッドおよびセッター メソッド。
- アダプタの `java.util.ResourceBundle` から収集したアダプタ メタデータへのアクセス。
- アダプタプロバイダによる、ファクトリの初期化プロセスに対するライセンスチェックへのプラグインのサポート。ライセンス確認が失敗すると、クライアントアプリケーションは基本となる EIS に接続できなくなり、その結果、アダプタも使用できなくなります。
- **JavaBeans** スタイルのポストコンストラクタの初期化をサポートするための状態確認。

以下の主要メソッドに対し、独自の実装が必要です。

- `createConnectionFactory()`
- `createManagedConnection()`
- `checkState()`
- `equals()`
- `hashCode()`
- `matchManagedConnections()`

以下の節では、それらのメソッドについて説明します。

createConnectionFactory()

`createConnectionFactory()` は、アダプタに対するアプリケーションレベルの接続ハンドル用のファクトリです。すなわち、アダプタのクライアントは、このメソッドにより返されたオブジェクトを使用して EIS に対する接続ハンドルを取得します。

アダプタが CCI インタフェースをサポートする場合、

`com.bea.adapter.cci.ConnectionFactoryImpl` のインスタンス、または拡張されたクラスを返すことをお勧めします。このメソッドを正しく実装するポイントは、`ConnectionManager`、`LogContext`、および `ResourceAdapterMetaData` をクライアント API に伝播することです。

コード リスト 6-22 createConnectionFactory() の例

```
protected Object
    createConnectionFactory(ConnectionManager connectionManager,
                           String strAdapterName,
                           String strAdapterDescription,
                           String strAdapterVersion,
                           String strVendorName)
    throws ResourceException
```

createManagedConnection()

`createManagedConnection()` は、アダプタに対する `ManagedConnection` インスタンスの作成に使用します。以下のリストに、このメソッドの例を示します。

コード リスト 6-23 createManagedConnection() の例

```
public ManagedConnection
    createManagedConnection(Subject subject, ConnectionRequestInfo
                           info)
    throws ResourceException
```

`ManagedConnection` インスタンスは、EIS との通信に必要な高コスト リソースをカプセル化します。このメソッドは、`ConnectionManager` がクライアントの要求を満たすために新しい `ManagedConnection` が必要と判断した場合に、

ConnectionManager によって呼び出されます。アダプタの一般的な設計パターンでは、このメソッドで **EIS** との通信に必要なリソースを開き、それからそのリソースを新しい **ManagedConnection** インスタンスに渡すようにします。

checkState()

`checkState()` メソッドは、**AbstractManagedConnectionFactory** がファクトリ関係のタスクを実行する前に呼び出します。このメソッドを使用して、初期化の必要があるすべてのメンバーが正しく初期化されていることを確認しないと、**ManagedConnectionFactory** では分担された **SPI** タスクを実行できません。

このメソッドの実装は次のとおりです。

```
protected boolean checkState()
```

equals()

`equals()` メソッドは、オブジェクト引数が等しいかどうかをテストします。このメソッドは、**ConnectionManager** が接続プールの管理に使用しますので、必ず正しく実装してください。等価性比較において、このメソッドは重要なメンバーをすべて組み込む必要があります。

このメソッドの実装は次のとおりです。

```
public boolean equals(Object obj)
```

hashCode()

`hashCode()` メソッドは、ファクトリで使用するハッシュコードを提供します。また、**ConnectionManager** による接続プールの管理にも使用されます。したがって、このメソッドは、オブジェクトのユニーク性を決定するプロパティに基づいて `hashCode` を生成します。

このメソッドの実装は次のとおりです。

```
public int hashCode()
```

matchManagedConnections()

ManagedConnectionFactory は `matchManagedConnections()` メソッドの実装が必要です。**AbstractManagedConnectionFactory** は、**AbstractManagedConnection** の `compareCredentials()` メソッドに依存する `matchManagedConnections()` メソッドを実装します。

管理対象接続にロジックを適合させるためには、`AbstractManagedConnection` クラスが提供する `compareCredentials()` メソッドをオーバーライドします。このメソッドは、`ManagedConnectionFactory` が `ConnectionManager` に対する接続要求と接続とをマッチングする場合に呼び出されます。

現行では、`AbstractManagedConnectionFactory` の実装は、指定された `Subject/ConnectionRequestInfo` パラメータから `PasswordCredential` を抽出します。どちらのパラメータもヌルであれば、このインスタンスに対する `ManagedConnectionFactory` は適切なので、このメソッドは `true` を返します。この実装を、以下のリストに示します。

コード リスト 6-24 `compareCredentials()` の実装

```
public boolean compareCredentials(Subject subject,
                                 ConnectionRequestInfo info)
    throws ResourceException
{
    ILogger logger = getLogger();
```

次に、ADK の `ManagedConnectionFactory` を使用して、JAAS `Subject` または `SPI ConnectionRequestInfo` から、`PasswordCredential` を抽出する必要があります。以下のリストに例を示します。

コード リスト 6-25 `PasswordCredential` の抽出

```
PasswordCredential pc = getFactory().
getPasswordCredential(subject, info);
    if (pc == null)
    {
        logger.debug(this.toString() + ": compareCredentials
```

上記のリストでは、JAAS `Subject` および `ConnectionRequestInfo` がヌルで、一致しています。このメソッドは、このインスタンスのファクトリの適切性が確立されるまでは、呼び出されません。呼び出しの結果、`Subject` と `ConnectionRequestInfo` が両方ともヌルの場合、デフォルトにより資格一致となります。この接続の `ping` 結果により、比較結果の出力が決まります。以下のリストに、プログラムによる接続の `ping` を行う方法を示します。

コード リスト 6-26 接続の ping

```
return ping();
}
    boolean bUserNameMatch = true;
    String strPcUserName = pc.getUserName();
    if (m_strUserName != null)
    {

logger.debug(this.toString() + ": compareCredentials >>> comparing
my username ["+m_strUserName+"] with client username
["+strPcUserName+"]");
```

次に、**Subject** または `ConnectionRequestInfo` で指定されたユーザがこちらのユーザと同じかどうか確認する必要があります。このアダプタでは、再認証をサポートしていないので、この 2 つのユーザ名が一致しない場合、このインスタンスは要求を満たすことができなくなります。以下のコードが要求を満たしています。

```
bUserNameMatch = m_strUserName.equals(strPcUserName);
```

ユーザ名が一致すると、これが継続して良好かどうかを判別するため、接続の **ping** が行われます。名前が一致しない場合、**ping** する理由はありません。

接続を **ping** するには、以下のコードを使用します。

```
return bUserNameMatch ? ping() : false;
```

実装に関する説明

管理対象シナリオの場合、アプリケーション サーバは、アダプタの `ManagedConnectionFactory` に対して `matchManagedConnections()` メソッドを呼び出します。この指定では、接続要求を満たすためにアプリケーションサーバがどの `ManagedConnectionFactory` を使用するか、その決定については指示していません。ADK の `AbstractManagedConnectionFactory` は、`matchManagedConnections()` を実装します。

この実装を行う最初の手順は、「これ」（すなわち、`ConnectionManager` が `matchManagedConnections` を呼び出した `ManagedConnectionFactory` インスタンス）とアプリケーションサーバによって提供されたセット内の各 `ManagedConnection` に対する `ManagedConnectionFactory` との比較です。同一の `ManagedConnectionFactory` を持つセット内の各 `ManagedConnection` に対

しては、実装により `compareCredentials()` メソッドが呼び出されます。このメソッドにより、各 `ManagedConnection` オブジェクトは要求を満たすことができるかどうかを判別できます。

`ConnectionFactory` によって、`matchManagedConnections()` が呼び出され (コード リスト 6-27 を参照してください)、管理しているプール内で有効な接続が検索されます。このメソッドでヌルが返された場合、`ConnectionFactory` は、`createManagedConnection()` を呼び出し、EIS に新しい接続を割り当てます。

コード リスト 6-27 `matchManagedConnections()` メソッドの実装

```
public ManagedConnection
matchManagedConnections(Set connectionSet,
                          Subject subject,
                          ConnectionRequestInfo info)
    throws ResourceException
```

このクラスでは、接続のマッチングに次の方法を使用しています。

1. セット内の各オブジェクトに対して、一致が検出されるまで `connectionSet` を繰り返します。次に、オブジェクト が `AbstractManagedConnection` クラスかどうかを判別されます。
2. `AbstractManagedConnection` であった場合、この接続が、セットの `AbstractManagedConnection` に対する `ManagedConnectionFactory` と比較されます。
3. 2つのファクトリが同じ場合、`compareCredentials()` メソッドが `AbstractManagedConnection` に対して呼び出されます。
4. `compareCredentials()` メソッドが `true` を返すと、インスタンスが返されます。

開発時に必要な `AbstractManagedConnectionFactory` プロパティ

`AbstractManagedConnectionFactory` の基本実装を使用するには、デプロイメント時に次の表のプロパティを指定する必要があります。

表 6-2 AbstractManagedConnectionFactory プロパティ

プロパティ名	プロパティの種類	可能な値	説明	デフォルト値
LogLevel	java.lang.String	ERROR、WARN、INFO、DEBUG	冗長性レベルのログ	WARN
LanguageCode	java.lang.String	有効な ISO 言語コードについては、 http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt を参照。	ログメッセージに対して適切なロケールを指定する。	en
CountryCode	java.lang.String	有効な ISO 国コードについては、 http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html を参照。	ログメッセージに対して適切なロケールを指定する。	US
MessageBundleBase	java.lang.String	有効な任意の Java クラス名またはファイル名	ログメッセージに対してメッセージバンドルを指定する。	なし、必須
LogConfigFile	java.lang.String	有効な任意のファイル名	LOG4J システムのコンフィグレーション	なし、必須
RootLogContext	java.lang.String	有効な任意の Java 文字列	この接続ファクトリのログメッセージを分類する。	なし、必須
AdditionalLogContext	java.lang.String	有効な任意の Java 文字列	このファクトリのメッセージを一意に識別する追加情報を付加する。	なし、省略可能

ADKにおける他の主要 ManagedConnectionFactory 機能

ADK のサンプルアダプタは、`sample.spi.ManagedConnectionFactoryImpl` と呼ばれる、`AbstractManagedConnectionFactory` を拡張するクラスを提供します。このクラスは、ADK の基本クラスを拡張する方法例として使用してください。

`ManagedConnectionFactory` と呼ばれる、サンプルアダプタの実装コードの完全リストについては、次を参照してください。

```
WLI_HOME/adapters/sample/src/sample/spi/  
ManagedConnectionFactoryImpl.java
```

ManagedConnection

`javax.resource.spi.ManagedConnection`

`ManagedConnection` オブジェクトは、EIS に対する接続設定に必要なすべての高コストリソースをカプセル化します。`ManagedConnection` インスタンスは、基本となる EIS に対する物理的な接続を表します。`ManagedConnection` オブジェクトは、アプリケーションサーバによって管理対象環境にプールされます。

ADK 実装

ADK では `ManagedConnection` の抽象実装を提供しています。この基本クラスは、接続イベントリスナおよび複数アプリケーションレベルの接続ハンドルを `ManagedConnection` インスタンスごとに管理するロジックを提供します。

`ManagedConnection` インタフェースを実装する場合、基本となる EIS が提供するトランザクション境界設定サポートを決める必要があります。トランザクション境界設定の詳細については、6-25 ページの「トランザクション境界設定」を参照してください。

ADK では、`AbstractManagedConnection` という、以下の機能を持つ `javax.resource.spi.ManagedConnection` インタフェースの抽象実装を提供しています。

- ADK ロギング フレームワークへのアクセスの提供
- 接続イベントリスナの集合の管理

- すべての接続イベント リスナに接続関連のイベントを通知する便利なメソッドの提供
- `ManagedConnection` インスタンスのクリーンアップおよび破棄を簡素化

ADK に付属しているサンプルアダプタには、`AbstractManagedConnection` を拡張する `ManagedConnectionImpl` が入っています。サンプルアダプタの `ManagedConnection` 実装コードの完全リストについては、次を参照してください。

```
WLI_HOME/adapters/sample/src/sample/spi/  
ManagedConnectionFactoryImpl.java
```

ManagedConnectionMetaData

`javax.resource.spi.ManagedConnectionMetaData`

`ManagedConnectionMetaData` インタフェースは、`ManagedConnection` インスタンスに関連付けられている基本となる `EIS` インスタンスに関する情報を提供します。アプリケーションサーバでは、この情報に基づいて、接続対象の `EIS` インスタンスに関する実行時情報が取得されます。

ADK 実装

ADK では、`AbstractManagedConnectionMetaData` という、以下の機能を持つ `javax.resource.spi.ManagedConnectionMetaData` および `javax.resource.cci.ConnectionMetaData` インタフェースの抽象実装を提供しています。

- 例外処理の簡素化
- `AbstractManagedConnection` インスタンスへのアクセス
- `EIS` 固有ロジックの実装に開発者が専念可能
- `CCI` と `SPI` 実装に対して別個のメタデータ クラスを持つ必要性の除去

ADK に付属しているサンプルアダプタには、`AbstractManagedConnectionMetaData` を拡張する `ConnectionMetaDataImpl` が入っています。アダプタの詳細なコード リストについては、次を参照してください。

```
WLI_HOME/adapters/sample/src/sample/spi/ConnectionMetaDataImpl.java
```

ConnectionEventListener

`javax.resource.spi.ConnectionEventListener`

`ConnectionEventListener` インタフェースは、アプリケーションサーバが `ManagedConnection` インスタンスからの通知を受信できるようにするイベントコールバックメカニズムを提供します。

ADK 実装

ADK では、`ConnectionEventListener` の 2 つの具象実装を提供しています。

- `com.bea.adapter.spi.ConnectionEventLogger`: ADK ロギングフレームワークを使って、接続関連のイベントをアダプタのログに記録します。
- `com.bea.adapter.spi.NonManagedConnectionEventListener`: アダプタが非管理対象環境で実行されている場合に、`javax.resource.spi.ManagedConnection` インスタンスを破棄します。この実装には以下の利点があります。
 - ADK ロギングフレームワークを使って接続関連イベントのログを記録します。
 - 接続関連のエラーが発生した場合に、`ManagedConnection` インスタンスを破棄します。

たいていの場合は、ADK で提供するこの 2 つの実装で十分なので、このインタフェースの独自の実装を用意する必要はありません。

ConnectionManager

`javax.resource.spi.ConnectionManager`

`ConnectionManager` インタフェースは、アダプタが接続要求をアプリケーションサーバに渡すのに使用できるフックを提供します。

ADK 実装

ADK では、このインタフェースの具象実装を提供しています。これは、`com.bea.adapter.spi.NonManagedConnectionManager` と呼ばれます。この実装は、非管理対象環境で実行中のアダプタに基本接続マネージャを提供します。管理対象環境では、このインタフェースはアプリケーション サーバによって提供されます。ほとんどの場合に、ADK が用意する実装を使用できます。

`NonManagedConnectionManager` は、`javax.resource.spi.ConnectionManager` インタフェースの具象実装で、アダプタが非管理シナリオにおいて `ConnectionManager` の役割を果たしますが、接続プールなどの高品質なサービスは提供しません。

ConnectionRequestInfo

`javax.resource.spi.ConnectionRequestInfo`

`ConnectionRequestInfo` インタフェースは、アダプタが独自の要求に固有なデータ構造を接続要求フローに渡せるようにします。アダプタは、空のインタフェースを拡張して、接続要求に対する独自のデータ構造をサポートします。

ADK 実装

ADK は、`javax.resource.spi.ConnectionRequestInfo` インタフェースの具象実装を提供します。このインタフェースは、`ConnectionRequestInfoMap` と呼ばれます。これは、ユーザ名やパスワードなど、接続が確立された場合に要求される情報への `java.util.Map` インタフェースを提供します。

LocalTransaction

`javax.resource.spi.LocalTransaction`

`LocalTransaction` インタフェースは、EIS リソース マネージャの内部で管理されているトランザクションのサポートを提供し、外部トランザクション マネージャを必要としません。

ADK 実装

ADK では、`AbstractLocalTransaction` という、このインタフェースの抽象実装を提供しており、EIS 固有の事項を中心に `LocalTransaction` の実装作業を行うことができます。特に、次のような作業を行います。

- 例外処理の簡素化
- EIS 固有のトランザクション ロジックの実装にアダプタ プロバイダが専念可能
- CCI と SPI 実装に対して別個のメタデータ クラスを持つ必要性の除去

手順 4 : CCI の実装

クライアント インタフェースにより、J2EE 準拠アプリケーションは、バックエンド システムにアクセスできます。クライアント インタフェースは、クライアント アプリケーションとバックエンド システム間のデータ フローを管理しますが、コンテナまたはアプリケーション サーバとアダプタとのやり取りに関する可視性はありません。クライアント インタフェースは、EIS との対話における要求および応答の両レコードのフォーマットを指定します。

まず、アダプタが J2EE 準拠 **Common Client Interface (CCI)** をサポートする必要があるかどうかを判断しなければなりません。現行の J2EE 仕様では必須要件にはなっていませんが、これから先のバージョンでは必須要件になることが予想されます。したがって、ADK では、アダプタの CCI インタフェースの実装を支援することに重点を置いています。

この節の構成

この節（「手順 4 : CCI の実装」）では、CCI の実装に使用できるインタフェースについて説明します。CCI の実装作業には、最低でも 2 つのインタフェースが必要です（6-39 ページの「基本的な CCI の実装」を参照）。ここでは、それぞれのインタフェースを詳しく説明し、さらに、ADK に含まれているサンプルアダプタによるそれらのインタフェース拡張方法を説明します。

この 2 つの必須インタフェースの説明に続いて、その他のインタフェースの詳細やこれらのインタフェースを使用する理由や利点についても説明しています。

基本的な CCI の実装

アダプタに CCI を実装するには、最低でも、次の 2 つのインタフェースを拡張する必要があります。

- `Connection` – クライアントが基盤となる物理接続にアクセスするときに使用する、アプリケーションレベルのハンドルを表します。
- `Interaction` – コンポーネントが EIS 関数を実行できるようにします。

これらのインタフェースは、できればここで示す順序で実装してください。

また、アダプタに必要な以下のインタフェースはいずれも実装可能です。

- `ConnectionFactory`
- `ConnectionMetaData`
- `ConnectionSpec`
- `InteractionSpec`
- `LocalTransaction`
- `Record`
- `ResourceAdapterMetaData`

Connection

`javax.resource.cci.Connection`

`Connection` は、クライアントが基盤となる物理接続にアクセスするときに使用する、アプリケーションレベルのハンドルを表します。`Connection` インスタンスに関連付けられた実際の物理接続は、`ManagedConnection` インスタンスによって表されます。

クライアントは、`ConnectionFactory` インスタンスで `getConnection()` メソッドを使用することにより、`Connection` インスタンスを取得します。`Connection` は、ゼロ個以上の `Interaction` インスタンスと関連付けることができます。

ADK 実装

ADK では、`AbstractConnection` という、このインタフェースの抽象実装を提供しています。この実装には以下の機能があります。

- ADK ロギング フレームワークへのアクセス
- `AbstractManagedConnection` インスタンスへのアクセス
- 状態管理およびアサーション チェック

このクラスは、次のメソッドを実装して拡張する必要があります。

```
public Interaction createInteraction()  
    throws ResourceException
```

このメソッドは、この接続に関連付けられた対話を作成します。この対話により、アプリケーションが **EIS** 関数を実行できるようになります。このメソッドの戻り値および例外は、以下のとおりです。

- `Interaction` インスタンスを返す
- 作成操作が失敗した場合、`ResourceException` が発生

Interaction

```
javax.resource.cci.Interaction
```

`javax.resource.cci.Interaction` は、コンポーネントが **EIS** 関数を実行できるようにします。`Interaction` インスタンスは、**EIS** インスタンスとの対話を、以下のメソッドによってサポートします。

- 入力 `Record`、出力 `Record` および `InteractionSpec` を取り込む `execute()` メソッド。このメソッドは、`InteractionSpec` によって表される **EIS** 関数を実行し、出力 `Record` を更新します。

- 入力 Record および InteractionSpec を取り込む execute() メソッド。このメソッドは、InteractionSpec によって表される EIS 関数を実行し、戻り値として出力 Record を生成します。

Interaction インスタンスは、接続から作成され、Interaction と Connection インスタンスとの関連付けの保守に必要です。close() メソッドは、対話に必要なアダプタが保守するすべてのリソースを解放します。Interaction インスタンスをクローズしても、関連付けられた Connection インスタンスはクローズをトリガしません。

ADK 実装

ADK では、AbstractInteraction という、このインタフェースの実装を提供しています。この実装には以下の利点があります。

- ADK ロギング フレームワークへのアクセスの提供
- 警告の管理

このインタフェースには、execute() を実装する AbstractInteraction の具象拡張を提供する必要があります。execute() には 2 つのバージョンがあります。これらについては、次に説明します。

execute() バージョン 1

コード リスト 6-28 で宣言されている execute() メソッドは、InteractionSpec で表された対話を示しています。

コード リスト 6-28 execute() バージョン 1 のコード例

```
public boolean execute(InteractionSpec ispec,  
                      Record input,  
                      Record output)  
    throws ResourceException
```

この形の呼び出しでは、execute() は入力レコードを取り込んで、出力レコードを更新します。以下を返します。

- EIS 関数の実行が正常に行われ出力 (Record) が更新された場合の戻り値は true、そうでない場合は false
- 例外 `ResourceException` – 実行操作が失敗した場合

以下の表に、`execute()` バージョン 1 のパラメータを示します。

表 6-3 `execute()` バージョン 1 のパラメータ

パラメータ	説明
<code>ispec</code>	対象 EIS データまたは関数モジュールを表す <code>InteractionSpec</code>
入力	入力 <code>Record</code>
出力	出力 <code>Record</code>

`execute()` バージョン 2

コード リスト 6-29 で宣言されている `execute()` メソッドでも、`InteractionSpec` で表された `Interaction` が実行されます。

コード リスト 6-29 `execute()` バージョン 2 のコード例

```
public Record execute(InteractionSpec ispec,  
                    Record input)  
    throws ResourceException
```

この形式の呼び出しでは、`execute()` は入力 `Record` を取り込み、`Interaction` の実行が正常に完了した場合、出力 `Record` を返します。

このメソッドの戻り値および例外は、以下のとおりです。

- EIS 関数の実行が正常に行われた場合は出力 `Record` を返し、そうでない場合は例外を送出します。
- 例外 `ResourceException` – 実行操作が失敗した場合

例外が発生した場合、このメソッドによって `Connection` に通知され、`Connection` によって自己のクローズなどの適切な処置が行われます。

以下の表に、`execute()` バージョン 2 のパラメータを示します。

表 6-4 `execute()` バージョン 2 のパラメータ

パラメータ	説明
<code>ispec</code>	対象 EIS データまたは関数モジュールを表す <code>InteractionSpec</code>
入力	入力 <code>Record</code>

XCCI を使用した CCI の実装

XCCI (XML-CCI) は、データ表示に XML ベースのレコード形式を使用する `Client Connector Interface` の固有言語です。これらのフォーマットは、フレームワークおよびツールによってサポートされています。XML-CCI は、通常、XCCI という略称で呼ばれます。

XCCI には、`Services` および `DocumentRecords` という 2 つの主要コンポーネントがあります。

サービス

サービスは、EIS で使用可能な機能を表し、次の 4 つのコンポーネントで構成されます。

- ユニークなビジネス名

各サービスには、統合ソリューションにおける役割を示すユニークなビジネス名があります。たとえば、カスタマリレーションシップ マネージメント (CRM) に関する統合ソリューションでは、`CreateNewCustomer` といったサービス名が考えられます。サービス名がそのサービスのビジネス目的を表すようにすること、また、EIS でサービスによって呼び出される関数名を使用して付けることが重要です。

- 要求ドキュメント定義

要求ドキュメント定義は、サービスの入力要件を記述します。`com.bea.document.IDocumentDefinition` インタフェースにより、ドキュメントタイプに関するすべてのメタデータが具体的に表されます。これには、ドキュメントスキーマ (構成と用途) と、このタイプ of 全ドキュメント

に対するルート要素名が含まれます。ルート要素名が必要なのは、XMLスキーマでは複数のルート要素が定義可能なためです。

- 応答ドキュメント定義

応答ドキュメント定義は、サービスの出力を記述します。

- 追加メタデータ

サービスは、EISの機能実行に関連する複雑な処理をほとんど意識する必要のない、統合ソリューションにおける高次元のコンポーネントです。つまり、サービスがEISとの対話に必要な詳細処理の多くはパブリックインタフェースに公開されません。その結果、EISの関数呼び出しに必要な一部の情報が、クライアントからの要求では提供されません。したがって、多くのサービスでは追加メタデータを格納しておく必要があります。WebLogic Integrationでは、この追加メタデータは、アダプタの `javax.resource.cci.InteractionSpec` 実装クラスによってカプセル化されます。

サービスが要求または応答データを必要としないことを示すには、`DesignTimeRequestHandler`の要求または応答に対し、空またはヌルの `IDocumentDefinition` を作成します。また、空またはヌルの `IDocumentDescriptor` インスタンスを持つサービスに対し、`IServiceDescriptor`の要求または応答に `IDocumentDescriptor` を設定します。静的 `DocumentFactory.createNullDocumentDefinition()` メソッドを使用して空またはヌルの `IDocumentDefinition` インスタンスを作成し、静的 `DescriptorFactory.createNullDocumentDescriptor()` メソッドを使用して空またはヌルの `IDocumentDescriptor` インスタンスを作成します。

空またはヌルのドキュメント定義または記述子を生成された `IServiceDescriptor`、あるいは設計時にアダプタにより生成された `IApplicationViewDescriptor` で使用する場合、これらのサービスに対するヌル要求または応答ドキュメントを実行時に処理する必要があります。つまり、空またはヌルのドキュメント記述子を使用するアダプタは、実行時に要求または応答ドキュメントが非ヌルであるとは想定しないからです。

アプリケーションビューの実行時エンジンにより、要求または応答を必要とするサービスが非ヌルの要求または応答ドキュメントを受け取り、要求や応答を必要としないサービスがヌルの要求や応答ドキュメントを受け取ることができません。

DocumentRecord

`com.bea.connector.DocumentRecord`

実行時に、XCCI レイヤは `DocumentRecord` オブジェクトをサービスに対する入力と想定し、`DocumentRecord` オブジェクトをサービスからの出力として返します。`DocumentRecord` は、`javax.resource.cci.Record` および `com.bea.document.IDocument` インタフェースを実装しています。`Record` インタフェースの説明については、6-54 ページの「`Record`」を参照してください。

`IDocument` は、アダプタ内の CCI レイヤからの XML 入力および出力を促進するインタフェースですが、これは次に説明します。

IDocument

`com.bea.document.IDocument`

`IDocument` は、W3C Document Object Model (DOM) を取り巻く高次元のラップです。`IDocument` インタフェースの最も重要な付加価値は、XML ドキュメントの要素に `Xpath` インタフェースを提供する点です。すなわち、`IDocument` オブジェクトは、`XPath` 文字列を使用してクエリおよび更新が可能です。たとえば、コード リスト 6-30 に示す XML ドキュメントは、「*Bob*」という名前の人物に関する詳細を記録するために XML をどのように使用するかを示しています。

コード リスト 6-30 XML の例

```
<Person name="Bob">
  <Home squareFeet="2000"/>
  <Family>
    <Child name="Jimmy">
      <Stats sex="male" hair="brown" eyes="blue"/>
    </Child>
    <Child name="Susie">
      <Stats sex="female" hair="blonde" eyes="brown"/>
    </Child>
  </Family>
</Person>
```

`IDocument` を使用し、コード リスト 6-31 に示されている `XPath` コードで「`Jimmy`」の髪の色を検索できます。

コード リスト 6-31 IDocument Data を検索するサンプルコード

```
System.out.println("Jimmy's hair color: " +
    person.getStringFrom("//Person[@name=\"Bob\"]/Family/Child
    [@name=\"Jimmy\"]/Stats/@hair");
```

一方、DOM を使用する場合、コード リスト 6-32 に示すコードを使用してクエリを発行する必要があります。

コード リスト 6-32 DOM Data を検索するサンプルコード

```
String strJimmysHairColor = null;
org.w3c.dom.Element root = doc.getDocumentElement();
if (root.getTagName().equals("Person") &&
    root.getAttribute("name").equals("Bob")) {
    org.w3c.dom.NodeList list = root.
        getElementsByTagName("Family");
    if (list.getLength() > 0) {
        org.w3c.dom.Element family = (org.w3c.dom.
            Element)list.item(0);

        org.w3c.dom.NodeList childList =
family.getElementsByTagName("Child");
        for (int i=0; i < childList.getLength(); i++) {
            org.w3c.dom.Element child = childList.item(i);
            if (child.getAttribute("name").equals("Jimmy")) {
                org.w3c.dom.NodeList statsList =
                    child.getElementsByTagName("Stats");
                if (statsList.getLength() > 0) {
                    org.w3c.dom.Element stats = statsList.item(0);
                    strJimmysHairColor = stats.getAttribute("hair");
                }
            }
        }
    }
}
```

IDocument によりコードが簡素化されます。

ADK 付属の XCCI クラス

アダプタに対し XCCI を実装するために、ADK は以下のクラスおよびインタフェースを提供します。

- `AbstractDocumentRecordInteraction`
- `DocumentDefinitionRecord`
- `DocumentInteractionSpecImpl`
- `IProxiedMarker`
- `IProxiedConnection`

ここでは、それらのクラスおよびインタフェースについて説明します。

AbstractDocumentRecordInteraction

`com.bea.adapter.cci.AbstractDocumentRecordInteraction`

このクラスは、ADK の抽象基本 `Interaction`、`com.bea.adapter.cci.AbstractInteraction` を拡張します。このクラスの目的は、`DocumentRecord` の操作に便利なメソッドを提供し、実装時のエラー処理の量を減らすことです。このクラスは以下の宣言をします。

```
protected abstract boolean execute(  
    InteractionSpec ixSpec,  
    DocumentRecord inputDoc,  
    DocumentRecord outputDoc  
    ) throws ResourceException
```

および

```
protected abstract DocumentRecord execute(  
    InteractionSpec ixSpec,  
    DocumentRecord inputDoc  
    ) throws ResourceException
```

これらのメソッドは、出力レコードが `DocumentRecord` オブジェクトであることが確認されてから、具象実装で呼び出されます。

DocumentDefinitionRecord

`com.bea.adapter.cci.DocumentDefinitionRecord`

このクラスを使用すると、アダプタが `DocumentRecordInteraction` 実装から `IDocumentDefinition` を返すことができるようになります。このクラスは、サービスに対する要求と応答ドキュメント定義の少なくとも一方を作成する設計時要求を満たすのに便利です。

DocumentInteractionSpecImpl

`com.bea.adapter.cci.DocumentInteractionSpecImpl`

このクラスを使用すると、特定のサービスに対する要求ドキュメント定義および応答ドキュメント定義を、実行時に実行メソッドに対して提供された `InteractionSpec` に保存できます。この機能は、アダプタの `Interaction` が実行時にサービスの XML スキーマにアクセスする必要がある場合に便利です。

IProxyedMarker および IProxyedConnection インタフェース

`com.bea.connector.IProxyedConnection`

`com.bea.connector.IProxyedMarker`

`IProxyedMarker` インタフェースは、`com.bea.adapter.cci.ConnectionFactoryImpl` クラスによって実装されます。このマーカは、関連付けられた接続がプロキシオブジェクトであるかどうかを判別するために使用されます。`IProxyedConnection` インタフェースは、`com.bea.adapter.cci.AbstractConnection` クラスによって実装され、プロキシに関連付けられた実際の接続を返すために使用されます。`IProxyedConnection` インタフェースには、`getAdapterConnection()` という 1 つのメソッドがあります。`getAdapterConnection` メソッドは、ポインタを返す `AbstractConnection` クラスで定義されています。`IProxyedConnection` インタフェースが必要になるのは、プロキシが実装するインタフェースのみ返すことができるからです。プロキシは、派生ツリーのクラスオブジェクトを区別できません。

XCCI の設計パターン

XCCI による方法を使用した場合に使用される一般的な設計パターンは、`Interaction` 実装でのサービスの定義をサポートすることです。このデザインパターンを使用する場合、アダプタの `javax.resource.cci.Interaction` 実装により、クライアントプログラムは、`WebLogic Integration` サービスを定義するために、基本にある `EIS` からメタデータを検索できます。その結果、具体的には、この対話によって、サービスに対する要求と応答 XML スキーマおよび追加

メタデータの生成を可能にする必要があります。Interaction により、クライアント プログラムは EIS の提供する関数のカタログを参照できます。この方法により、アダプタのシンクライアント アーキテクチャが促進されます。

ADK では、この設計パターンの実装を支援する

`com.bea.adapter.cci.DesignTimeInteractionSpecImpl` クラスを提供しています。`sample.cci.InteractionImpl` クラスは、`DesignTimeInteractionSpecImpl` クラスを使用してこの設計パターンの実装方法を示します。

非 XML J2EE 準拠アダプタの使用

ADK では、WebLogic Integration で非 XML アダプタを使用するためのプラグイン メカニズムを提供しています。あらかじめ構築されたアダプタのすべてが、その `javax.resource.cci.Record` データ型に XML を使用するわけではありません。たとえば、次のような場合、XML は使用できません。

- 独自のレコード形式で J2EE 準拠アダプタを開発した場合
- アダプタの CCI レイヤに独自のレコード形式を使用する外部のパーティの J2EE 準拠アダプタを購入した場合

このようなタイプのアダプタの実装を容易にするため、ADK では `com.bea.connector.IRecordTranslator` インタフェースを提供しています。実行時に、アプリケーション統合エンジンは、アダプタの `IRecordTranslator` 実装を使用して、アダプタのサービスを実行する前に要求および応答レコードを変換します。

Application Integration エンジンでは、`com.bea.connector.DocumentRecord` タイプの `javax.resource.cci.Record` しかサポートしないので、この独自形式を要求および応答レコード用のドキュメント レコードに変換する必要があります。この場合、アダプタの CCI 対話レイヤを書き直す必要はありません。`IRecordTranslator` インタフェースを実装するアダプタの EAR ファイルにクラスを含めることにより、アプリケーション統合エンジンは要求および応答の各レコードに対して、トランスレータ クラスにあるトランスレーション メソッドを実行できます。

InteractionSpec 実装クラスと IRecordTranslator 実装クラスの間には、1 対 1 の相関関係があります。複数の種類の InteractionSpec 実装を持つアダプタは、それぞれに対して IRecordTranslator 実装クラスを必要とします。プラグインアーキテクチャは、アダプタの InteractionSpec の完全クラス名およびフレーズ RecordTranslator を使用してトランスレータクラスを名前ロードします。たとえば、アダプタの InteractionSpec クラスの名前が `com.bea.adapter.dbms.cci.InteractionSpecImpl` である場合、エンジンは `com.bea.adapter.dbms.cci.InteractionSpecImplRecordTranslator` クラスをロードします（後者のクラスが有効の場合）。

実装する必要があるメソッドの詳細については、以下のディレクトリの `com.bea.connector.IRecordTranslator` の Javadoc を参照してください。

WLI_HOME/docs/apidocs/com/bea/connector/IRecordTranslator.html

ConnectionFactory

`javax.resource.cci.ConnectionFactory`

ConnectionFactory は、EIS インスタンスへの接続を取得するためのインタフェースを提供します。ConnectionFactory インタフェースの実装は、アダプタによって必ず提供されます。

アプリケーションは、JNDI ネームスペースから ConnectionFactory インスタンスをロックアップし、これを使用して EIS 接続を取得します。

JNDI 登録をサポートするために、`java.io.Serializable` および `javax.resource.Referenceable` インタフェースを実装する必要があります。このためには、ConnectionFactory の実装クラスが必要となります。

ADK 実装

ADK では、`ConnectionFactoryImpl` という

`javax.resource.cci.ConnectionFactory` インタフェースの具象実装を提供します。このインタフェースには以下の機能があります。

- ADK ログイン フレームワークへのアクセス
- アダプタ メタデータへのアクセス

■ getConnection() メソッドの実装

このクラスは通常、拡張せずにそのまま使用できます。

ConnectionMetaData

`javax.resource.cci.ConnectionMetaData`

`ConnectionMetaData` は、`Connection` インスタンスによって接続された EIS インスタンスに関する情報を提供します。コンポーネントは、`Connection.getMetaData` メソッドを呼び出して `ConnectionMetaData` インスタンスを取得します。

ADK 実装

デフォルトで、ADK は、`com.bea.adapter.spi.AbstractConnectionMetaData` クラスを通じてこのクラスの実装を提供します。この抽象クラスを拡張して、その 4 つの抽象メソッドを使用するアダプタに実装する必要があります。

ConnectionSpec

`javax.resource.cci.ConnectionSpec`

`ConnectionSpec` は、アプリケーション コンポーネントが接続要求固有のプロパティを `ConnectionFactory.getConnection()` メソッドに渡すときに使用されます。

この `ConnectionSpec` インタフェースを `JavaBean` として実装し、ツールをサポートできるようにすることをお勧めします。`ConnectionSpec` 実装クラスのプロパティは、ゲッター メソッドおよびセッター メソッドのパターンによって定義します。

CCI 仕様により、`ConnectionSpec` の標準プロパティのセットが定義されます。プロパティの定義は、派生インタフェースまたは空の `ConnectionSpec` インタフェースの実装クラスに対して行われます。さらに、アダプタでは基本となる EIS に固有の追加プロパティを定義できます。

ADK 実装

ConnectionSpec 実装は、JavaBean でなければならないので、ADK ではこのクラスに対しては実装を提供していません。

InteractionSpec

javax.resource.cci.InteractionSpec

InteractionSpec は、EIS インスタンスとの対話を行うのに必要なプロパティを格納しています。特に、これは基本となる EIS の指定関数を対話で実行するとき 사용됩니다。

CCI 仕様により、InteractionSpec の標準プロパティのセットが定義されます。標準プロパティを基本の EIS に適用しない場合、標準プロパティのサポートのために InteractionSpec を実装する必要はありません。

InteractionSpec 実装クラスは、サポートされる各プロパティに対してゲッター メソッドおよびセッター メソッドを提供する必要があります。このゲッター メソッドおよびセッター メソッド規約は、JavaBean 設計パターンに基づいている必要があります。

InteractionSpec インタフェースは、ツールをサポートするためには、JavaBean として実装する必要があります。InteractionSpec インタフェースの実装クラスは、java.io.Serializable インタフェースを実装する必要があります。

InteractionSpec には、Record にはないが、どの EIS 関数を呼び出すかを判断するときに役立つ情報が入っています。

次の表では、標準プロパティについて説明します。

表 6-5 標準 InteractionSpec プロパティ

プロパティ	説明
FunctionName	EIS 関数の名前
InteractionVerb	EIS インスタンスとの対話モードで、SYNC_SEND、SYNC_SEND_RECEIVE、または SYNC_RECEIVE がある。

表 6-5 標準 InteractionSpec プロパティ (続き)

プロパティ	説明
ExecutionTimeout	Interaction が、EIS で指定関数を実行するのを待つ時間 (ミリ秒)

以下の標準プロパティは、対話インスタンスの ResultSet 要件に関する判断材料として使用されます。

- FetchSize
- FetchDirection
- MaxFieldSize
- ResultSetType
- ResultSetConcurrency

CCI 実装では、InteractionSpec インタフェースに記述されたプロパティ以外の追加プロパティを指定できます。

注意： 追加プロパティのフォーマットおよびタイプは、EIS に固有のもので、CCI 仕様のスコープ外です。

ADK 実装

ADK には、InteractionSpecImpl という、`javax.resource.cci.InteractionSpec` の具象実装が入っています。このインタフェースは、表 6-5 で説明されている標準対話プロパティにゲッター メソッドおよびセッター メソッドを使用することによって拡張できる、基本実装を提供します。

LocalTransaction

`javax.resource.cci.LocalTransaction`

`LocalTransaction` インタフェースは、アプリケーションレベルのローカルトランザクション境界設定に使用されます。これは、リソースマネージャのローカルトランザクションに対するトランザクション境界設定インタフェースです。システム規約レベルの `LocalTransaction` インタフェース (`javax.resource.spi` package で定義されている) は、コンテナのローカルトランザクション管理に使用されます。

ローカルトランザクションは、リソースマネージャの内部で管理されます。ローカルトランザクションの調整には、外部トランザクションマネージャは関与しません。

CCI 実装は、`LocalTransaction` インタフェースを実装できます (必須ではありません)。 `LocalTransaction` インタフェースが CCI 実装でサポートされている場合、`Connection.getLocalTransaction()` メソッドは、`LocalTransaction` インスタンスを返すこととなります。すると、コンポーネントは、返された `LocalTransaction` を使用して、基本の `EIS` インスタンスに対して、(`Connection` インスタンスに関連付けられた) リソースマネージャのローカルトランザクションの境界設定を行うことができます。

`com.bea.adapter.spi.AbstractLocalTransaction` クラスでもこのインタフェースを実装します。

ローカルトランザクションの詳細については、6-25 ページの「トランザクション境界設定」を参照してください。

Record

`javax.resource.cci.Record`

`javax.resource.cci.Record` インタフェースは、`Interaction` で定義される `execute()` メソッドからの入力またはメソッドへの出力表現に使用する基本インタフェースです。 `execute()` メソッドの詳細については、6-41 ページの「`execute()` バージョン 1」および 6-42 ページの「`execute()` バージョン 2」を参照してください。

MappedRecord または IndexedRecord には、追加の Record を入れることができます。これは、MappedRecord および IndexedRecord を使用して任意の深さの階層構造を作成できることを示しています。MappedRecord または IndexedRecord で表される階層構造のリーフ要素としては、基本 Java タイプが使用されます。

Record インタフェースは、以下の表に示されている表現の 1 つになるように拡張できます。

表 6-6 Record インタフェースの表現

表現	説明
MappedRecord	キーと値がペアになったレコードの集まり。このインタフェースは、 <code>java.util.Map</code> に基づいている。
IndexedRecord	順序付き、インデックス付きのレコードの集まり。このインタフェースは、 <code>java.util.List</code> に基づいている。
EIS 抽象の <code>JavaBean</code> ベース表現	例として ERP システムの発注書作成のために生成されるカスタム レコードがある。
<code>javax.resource.cci.ResultSet</code>	このインタフェースは、 <code>java.sql.ResultSet</code> と <code>javax.resource.cci.Record</code> の両方を拡張する。 <code>ResultSet</code> は、表データで示される。

アダプタが CCI インタフェースを実装する場合、次に考慮する事項はサービスに対しどのレコード形式を使用するかです。各サービスに対し、要求レコード（サービスへの入力を提供する）および応答レコード（EIS 応答を提供する）の形式を指定する必要があります。

ADK 実装

ADK では、CCI レイヤで XML ベースのレコード形式を実装する支援に重点を置いています。この目的のため、ADK では `DocumentRecord` クラスを提供しています。さらに、BEA のスキーマ ツールキットを使用すれば、サービスに対する要求および応答ドキュメントを記述するスキーマを開発できます。

ADK では、レコード名とレコード記述に対するゲッター メソッドおよびセッター メソッドを提供する `javax.resource.cci.Record` インタフェースの具象実装、`RecordImpl` を用意しています。

アダプタプロバイダが XML ベースのレコード形式を使用する（使用を強く推奨します）場合、ADK では `com.bea.adapter.cci.AbstractDocumentRecordInteraction` クラスも用意しています。このクラスによって、クライアントが `DocumentRecord` オブジェクトを渡すことができます。さらに、このクラスは `DocumentRecord` の内容へのアクセスに便利なメソッドを提供しています。

ResourceAdapterMetaData

`javax.resource.cci.ResourceAdapterMetaData`

インタフェース `javax.resource.cci.ResourceAdapterMetaData` は、アダプタ実装の機能に関する情報を提供します。CCI クライアントは、`ConnectionFactory.getMetaData` を使用してアダプタのメタデータ情報を取得します。`getMetaData()` メソッドは、EIS インスタンスに対するアクティブな接続を必要としません。`ResourceAdapterMetaData` インタフェースは、アダプタ実装に固有な情報を追加して提供するように拡張できます。

注意： このインタフェースは、アダプタを通じて接続されている EIS インスタンスに関する情報は提供しません。

ADK 実装

ADK では、アダプタメタデータをカプセル化し、すべてのプロパティに対してゲッターおよびセッターを提供するインタフェース、`ResourceAdapterMetaDataImpl` を用意しています。

手順 5 : アダプタのテスト

アダプタのテストを行うため、ADK では、`com.bea.adapter.test.TestHarness`、ユニット テスト用のオープンソース ツール JUnit を強化するテスト支援機能を提供しています。

`com.bea.adapter.test.TestHarness` は、以下の処理を行います。

- テスト コンフィグレーション情報を含むプロパティ ファイルの読み取り
- ロギング ツールキットの初期化
- JUnit TestSuite の初期化
- テスト クラスのロードと、JUnit を使用したテスト クラスの実行
- オフラインおよび Weblogic Server 外でのコードのテスト

JUnit に関する詳細については、次のサイトを参照してください。

<http://www.junit.org>

テスト支援機能の使用法

ADK テスト支援機能を使用する手順は次のとおりです。

1. `junit.framework.TestCase` を拡張するクラスを作成します。このクラスは、新しい `junit.framework.TestSuite` を返す、`suite` という名前の静的メソッドを提供する必要があります。
2. テスト メソッドを実装します。各メソッドの名前は、`test` から始まります。
3. プロジェクト ディレクトリで、`test.properties` を作成または変更します。(サンプル アダプタを複製した場合、アダプタにはすでに `project` ディレクトリに基本 `test.properties` があります。) このプロパティ ファイルには、テスト ケースに必要な任意のコンフィグレーションプロパティが入ります。
4. Ant を使ってテストを呼び出します。Ant `build.xml` ファイルには、アダプタのプロパティ ファイルを持つ `com.bea.adapter.test.TestHarness` クラスを呼び出すテスト ターゲットが必要です。たとえば、サンプル アダプタでは、コード リスト 6-33 に示されている Ant ターゲットを使用しています。

コード リスト 6-33 サンプルアダプタによる Ant ターゲットの指定

```
<target name='test' depends='packages'>
  <java classname='com.bea.adapter.test.TestHarness'>
    <arg value='-DCONFIG_FILE=test.properties' /><classpath
      refid='CLASSPATH' />
  </java>
```

このターゲットは、メインクラス `com.bea.adapter.test.TestHarness` を使用して JVM を呼び出します。このクラスは、サンプルアダプタに対して確立されたクラスパスを使用し、以下のコマンドライン引数を渡します。

```
-DCONFIG_FILE=test.properties
```

ADK が提供するテスト ケースの拡張

サンプルアダプタには、次の 2 つの `TestCase` 拡張が付属しています。

- `sample.spi.NonManagedScenarioTestCase`
- `sample.event.OfflineEventGeneratorTestCase`

`sample.spi.NonManagedScenarioTestCase`

`NonManagedScenarioTestCase` を使用すると、非管理対象シナリオの SPI および CCI クラスをテストできます。具体的には、次の 4 つについてテストします。

- `ManagedConnectionFactory` 実装の初期化
- `ManagedConnectionFactory` インスタンスのシリアライゼーションまたはデシリアライゼーション
- EIS に対する接続オープン
- EIS に対する接続クローズ接続のクローズ時に関連付けられたすべてのリソースがクローズされることを確認します。

sample.event.OfflineEventGeneratorTestCase

`sample.event.OfflineEventGeneratorTestCase` を使用すると、Weblogic Server 外のイベント ジェネレータの内部機能をテストできます。具体的には、イベント ジェネレータの以下の内容をテストします。

- イベント ルータのシミュレーションを行い、アダプタのイベント ジェネレータの新しいインスタンスをインスタンス化します。
- 初期化のため `test.properties` をイベント ジェネレータに渡します。これによって、初期化ロジックをテストできます。
- イベント ジェネレータを無作為に更新します。これによって、`setupNewTypes()` メソッドおよび `removeDeadTypes()` メソッドをテストできます。
- イベント通知を受け取り、アダプタのログ ファイルに表示します。

sample.client.ApplicationViewClient

`sample.client.ApplicationViewClient` は、追加のアダプタ テスト手段を提供します。このクラスは、サービスを呼び出し、アプリケーション ビューのイベントをリスンする方法を示す Java プログラムです。Ant `build.xml` ファイルは、クライアント ターゲットを提供し、`ApplicationViewClient` プログラムを使用できるようにします。ant `client` を実行する場合のデフォルト コンフィグレーションは、プログラムの使用方法を表示することです。クライアント プログラムに対する入力パラメータは、`build.xml` ファイルを編集し、変更します。

`sample.client.ApplicationViewClient.java` の例を確認するには、`WLI_HOME/adapters/sample/src/sample/client` を参照してください。

注意： `sample.client.ApplicationViewClient` は、テスト支援機能に組み込まれていません。

手順 6 : アダプタのデプロイ

SPI および CCI インタフェースを実装し、アダプタをテストしたときに、手動または **WebLogic Server Administration Console** からアダプタを **WebLogic Integration** 環境にデプロイできます。詳細は、第 9 章「アダプタのデプロイ」を参照してください。

7 イベント アダプタの開発

この章の内容は以下のとおりです。

- イベント アダプタの概要
- 実行時環境におけるイベント アダプタ
- イベントのフロー
- 手順 1: アダプタの定義
- 手順 2: 開発環境のコンフィグレーション
- 手順 3: アダプタの実装
- 手順 4: アダプタのテスト
- 手順 5: アダプタのデプロイ

イベント アダプタの概要

イベント アダプタは、情報を EIS から WebLogic Integration 環境に伝播します。このタイプのアダプタは、情報のパブリッシャといえます。

すべての WebLogic Integration イベント アダプタには、次の 3 つの機能があります。

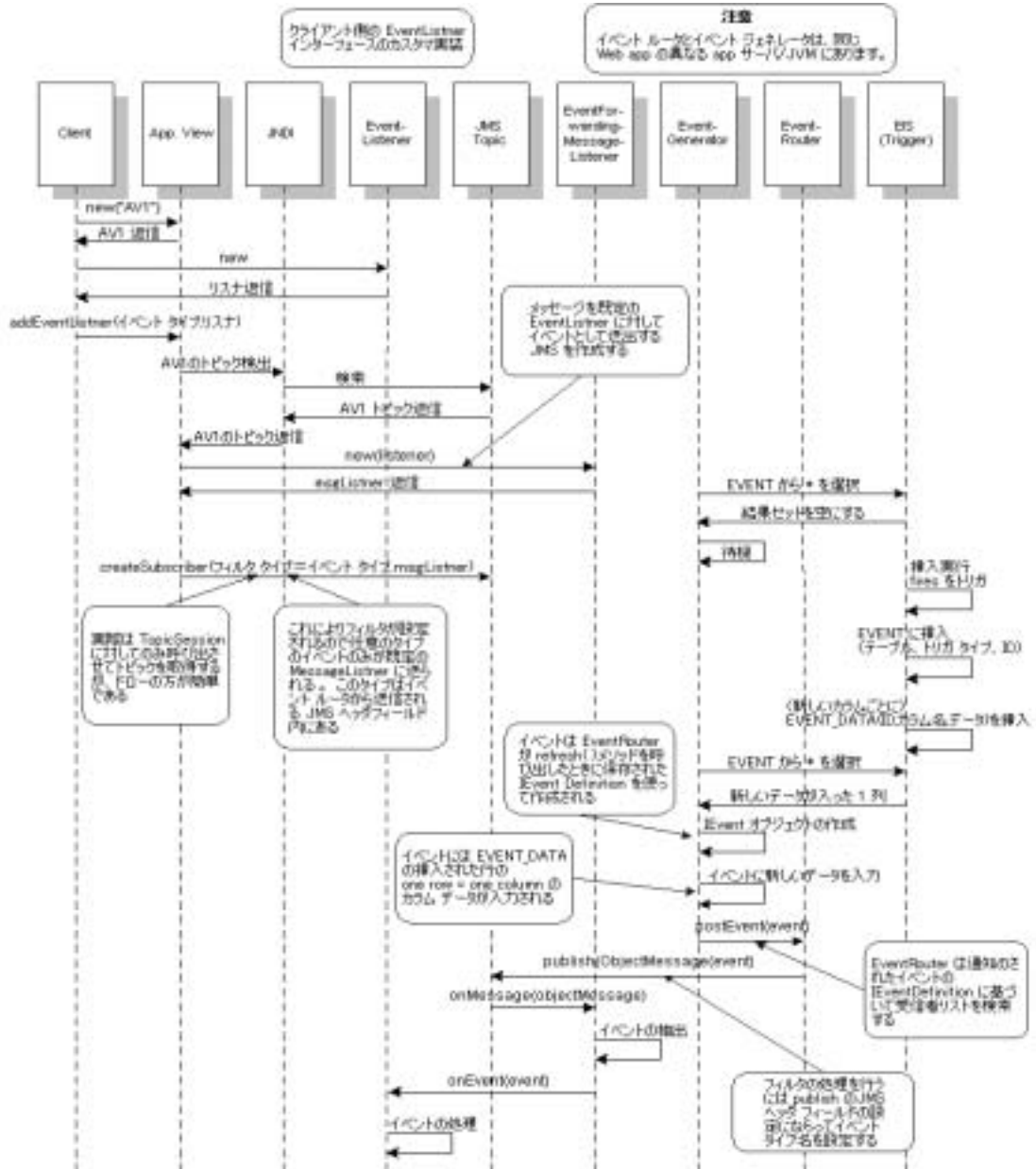
- EIS からイベントに関するデータを抽出しそれを格納することによって実行中の EIS 内部で発生したイベントに応答します。
- イベント データを、EIS 固有のデータ形式から、イベントの XML スキーマに従った XML ドキュメントに変換します。XML スキーマは、EIS のメタデータに基づいています。
- イベント ルータを使って、イベントを WebLogic Integration 環境に伝播します。

WebLogic Integration には、すべてのイベントアダプタに共通の3つの機能に関する要素が実装されており、アダプタの EIS 固有の要素のみに焦点をあてることができます。

実行時環境におけるイベントアダプタ

図 7-1 は、実行時環境におけるイベントの動作を示しています。

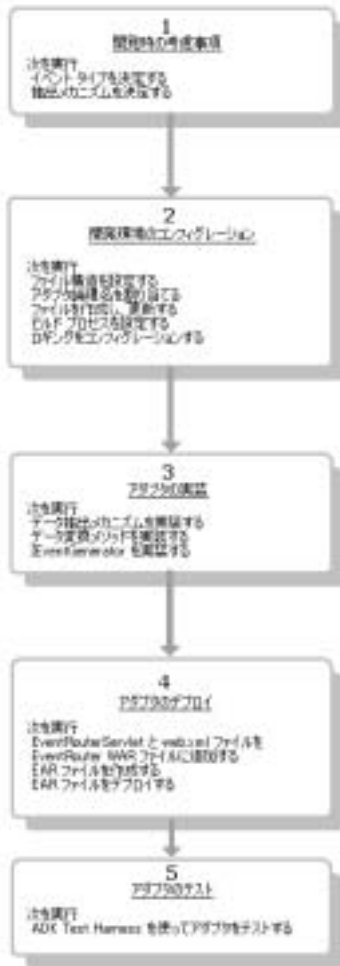
図 7-1 実行時環境におけるイベント アダプタ



イベントのフロー

図 7-2 は、イベント アダプタの開発手順の概要を示しています。

図 7-2 イベントアダプタのイベントフロー



手順 1 : アダプタの定義

イベントアダプタの開発を始める前に、その要件を定義する必要があります。これに必要な情報の詳細については、附録 D 「アダプタ設定ワークシート」を参照してください。ここでは、手順 1 を行う際に最も重要なタスクの概要を説明します。

1. 以下の事項を考慮して、イベントを定義します。
 - イベントの内容は何ですか？
 - XML スキーマの中でイベントはどのように定義されますか？
 - 何によってイベントを発生させますか？
2. 以下のデータ抽出メソッドのうち 1 つを選択します。
 - プッシュ EIS がアダプタにイベントを通知します。アダプタで EIS をポーリングしてステートの変化を調べる場合は、このメソッドを使用します。
 - プルアダプタによって EIS をポーリングし、EIS からイベント データを抽出します。パブリッシュおよびサブスクライブ モデルのように機能するイベント生成を実装する場合は、このメソッドを使用します。

手順 2 : 開発環境のコンフィグレーション

この手順では、アダプタの開発環境に応じてコンピュータを設定するための 5 つの手順について説明します。

- 手順 2a : ファイル構造の設定
- 手順 2b : アダプタへの論理名の割り当て
- 手順 2c : ビルド プロセスの設定
- 手順 2d : メッセージ バンドルの作成
- 手順 2e : ロギングのコンフィグレーション

手順 2a : ファイル構造の設定

イベント アダプタ開発環境に必要なファイル構成は、サービスアダプタの開発に必要なファイル構成と同じです。詳細は、第 6 章「サービス アダプタの開発」の「手順 2a : ディレクトリ構造の設定」を参照してください。

手順 2b : アダプタへの論理名の割り当て

アダプタに論理名を割り当てます。一般的に、この名前は、ベンダ名、アダプタに接続される EIS のタイプ、および EIS のバージョン番号で構成され、*vendor_EIS-type_EIS version* という形式で表されます。たとえば、

BEA_WLS_SAMPLE_ADK となります。

この名前には、以下の構成要素が含まれます。

- BEA_WLS : ベンダ
- SAMPLE : EIS タイプ
- ADK : EIS バージョン

手順 2c : ビルド プロセスの設定

WebLogic Integration では、Ant という Java 言語のみで作成した Java ベースのビルド ツールに基づくビルド プロセスを採用しています。Ant がどのように機能するかについては、3-4 ページの「Ant ベースのビルド プロセス」を参照してください。Ant の使用方法については、以下のサイトを参照してください。

<http://jakarta.apache.org/ant/index.html>

WebLogic Integration に付属のサンプルアダプタには、Ant ビルド ファイルが格納されています (*WLI_HOME/adapters/sample/project/build.xml*)。このファイルには、J2EE 準拠アダプタを構築するのに必要なタスクが収められています。GenerateAdapterTemplate ユーティリティを実行して、アダプタの開発ツリーを複製すると、使用するアダプタ専用の build.xml ファイルが作成されます。このファイルは自動的に生成されるため、サンプル build.xml ファイル

をカスタマイズする必要はなく、コードの正当性も保証されています。
GenerateAdapterTemplate ユーティリティの使用方法の詳細については、第 4 章「カスタム開発環境の作成」を参照してください。

ビルド プロセスの詳細については、第 6 章「サービスアダプタの開発」の「手順 2c: ビルド プロセスの設定」を参照してください。

手順 2d : メッセージバンドルの作成

エンド ユーザ向けのメッセージは、メッセージバンドルに入れます。
「key=value」の組み合わせから構成される .properties テキスト ファイルで、メッセージをインターナショナルライズできます。実行時に地理的ロケールと地域言語が指定されると、メッセージの内容が、「key=value」の組み合わせに基づいて解釈され、指定された言語でメッセージが表示されます。

メッセージバンドルの作成方法については、次に示すサイトにある **JavaSoft** チュートリアルを参照してください。

<http://java.sun.com/docs/books/tutorial/i18n/index.html>

手順 2e : ロギングのコンフィグレーション

ロギングは、**Apache Jakarta** プロジェクトの一環として開発された、**Log4j** と呼ばれるロギング ツールによって実行されます。

この手順を開始する前に、第 2 章「基本開発コンセプト」でロギングについて、そして第 5 章「ロギング ツールキットの使い方」で **Log4j** の使用方法について学ぶことをお勧めします。

イベント生成ロギング カテゴリの作成

イベント アダプタを使用する場合、そのイベント生成に対してロギング カテゴリを作成する必要があります（ロギング カテゴリの詳細については、5-3 ページの「メッセージ カテゴリ」を参照してください）。特定のアダプタに対するロギング コンフィグレーション ファイルを編集するには

(`WLI_HOME/adapters/YOUR_ADAPTER/src/adapter_logical_name.xml`)、以下のリストに示すコードを追加してください。

コード リスト 7-1 イベント生成ロギング カテゴリを作成するサンプルコード

```
<category name='BEA_WLS_SAMPLE_ADK.EventGenerator' class='com.bea.  
    logging.LogCategory'>  
  
</category>
```

BEA_WLS_SAMPLE_ADK を開発対象のアダプタの論理名に置き換えます。

このカテゴリにパラメータを設定しない場合、親カテゴリのすべてのプロパティ設定を継承します。この例では、親カテゴリは *BEA_WLS_SAMPLE_ADK* です。ルート カテゴリとしてアダプタ論理名を使用する必要はありませんが、複数アダプタ環境で他のアダプタに影響を及ぼさないよう、ユニークな識別子を使用する必要があります。

手順 3 : アダプタの実装

イベントアダプタを実装するには、以下の2つの手順を実行する必要があります。

1. イベントジェネレータを作成します。このプロセスでは、データ抽出メソッド（プッシュまたはプルモード）と *IEventGenerator* インタフェースを実装します（このインタフェースは、イベント生成処理の実行時にイベントルータで使用されます）。この手順については、「[手順 3a : イベントジェネレータの作成](#)」で説明しています。
2. データ変換メソッドを実装します。この手順については、「[手順 3b : データ変換メソッドの実装](#)」で説明しています。

手順 3a : イベント ジェネレータの作成

イベント生成機能により、EIS から通知を受信したり、または EIS をポーリングしたりして特定のイベントの発生をチェックするメカニズムがアダプタに装備されます。WebLogic Integration エンジンには、複数タイプのイベントをサポートする強力なイベント ジェネレータを提供します。イベント タイプは、イベントのコンフィグレーション プロパティで定義されます。

イベント プロパティは、通常、設計時にイベントに関連付けられたプロパティで定義されます。イベント アダプタをコンフィグレーションする場合、アダプタがイベント プロパティを収集する Web ページを 1 ページ以上指定することができます。これらのプロパティは、アプリケーション ビュー記述子とともに保存され、実行時にイベントに戻されます。WebLogic Integration エンジンでは、このプロパティおよびソース アプリケーション ビューを使用して、リスナへ戻すルーティングが決定されます。たとえば、同じプロパティが定義された同一のイベント ジェネレータ 2 つを別々にデプロイしても、WebLogic Integration エンジンでは 1 つの IEventDefinition しか作成されませんが、異なるプロパティが指定された場合は、単一のイベント アダプタの各デプロイメントに対し、1 つの IEventDefinition が作成されます。イベント ジェネレータは、ルーティング処理でどの IEventDefinition を使用するかを決定する必要があります。この決定は、通常、プロパティの値と特定のイベント発生に基づいて行われます。

IEventDefinition オブジェクトは、イベント ジェネレータの実装で使用され、特定のイベントをリスナに戻すルーティングを行います。別に説明するとおり、WebLogic Integration エンジンによってイベントを含むデプロイされたアプリケーション ビューの IEventDefinition オブジェクトを作成します。

IEventDefinition オブジェクトを使用して、アプリケーション ビューのデプロイメントに関する特定のプロパティを抽出したり、スキーマおよびルーティング オブジェクトにアクセスしたりできます。これらの属性は、イベントのルーティング時に使用します。

データ抽出メカニズムの実装

WebLogic Integration では、次に示す 2 つのデータ抽出モードをサポートしています。

- プッシュ イベントによる生成—イベント生成オブジェクトがイベント ジェネレータに通知を送信すると、ステートの変更が認識されます。プッシュ イベント ジェネレータがイベントを受信すると、WebLogic Integration エンジン

ンによって、デプロイされたアプリケーションビューにイベントが転送されます。プッシュ イベント ジェネレータでは、パブリッシュ モデルおよびサブスクライブ モデルが使用されています。

- プル イベントによる生成—ステートが変更したかどうかの判断にポーリングが必要なときに使用します。データ抽出プロセスでは、ステートの変更が認識されるまで、継続的にオブジェクトに対するクエリが行われます。ステートの変更が認識された時点でイベントが作成され、**WebLogic Integration** エンジンによって、デプロイされたアプリケーションビューにこのイベントが転送されます。

プル モード

プル モードでは、ポーリング手法によりイベントの発生が判断されます。これを実装するには、`com.bea.adapter.event` パッケージの `AbstractPullEventGenerator` からイベント ジェネレータを取得します。

注意： `adk-eventgenerator.jar` ファイルには、イベント ジェネレータの実装に必要な ADK ベース クラスが含まれます。これは、**WAR** メイク ファイルに含まれます。

ADK では、`AbstractPullEventGenerator` に、抽象メソッドをいくつか用意していますが、実際の実装ではこれをオーバーライドします。これらのメソッドについて以下の表に示します。

表 7-1 AbstractPullEventGenerator メソッド

メソッド	説明
<code>postEvents()</code>	他のイベント生成、メッセージ変換、およびルーティング コードの制御メソッドで、ポーリングおよびルーティング コードを追加できる。イベント ルータ コンフィグレーション ファイルで指定された間隔で、 <code>AbstractPullEventGenerator</code> の実行メソッドから呼び出される。
<code>setupNewTypes()</code>	デプロイされる <code>IEventDefinition</code> オブジェクトを前処理するためのメソッド。有効な新しい <code>IEventDefinition</code> オブジェクトだけがこのメソッドに渡される。

表 7-1 AbstractPullEventGenerator メソッド (続き)

メソッド	説明
<code>removeDeadTypes()</code>	アンデプロイされる <code>IEventDefinition</code> オブジェクトに対して要求されたクリーンアップを処理する。 WebLogic Integration エンジンでは、関連イベントのアプリケーションビューがアンデプロイされた場合にこのメソッドを呼び出す。
<code>doInit()</code>	イベント ジェネレータが作成されるときに呼び出される。初期化プロセスで、イベント ジェネレータによって事前定義されたコンフィグレーション値が使用されて、イベント生成処理に必要なステートまたは接続情報が設定される。
<code>doCleanupOnQuit()</code>	イベント生成プロセスで割り当てられたリソースを解放する。イベント生成処理を行っているスレッドの終了前に呼び出される。

プッシュ モード

プッシュ モードは、イベントのルーティングをトリガする通知を使用します。これを実装するには、`com.bea.adapter.event` パッケージの `AbstractPushEventGenerator` クラスからイベント ジェネレータを取得します。このイベント パッケージには、他にもいくつかのサポート クラスが入っています。表 7-2 でサポート クラスを説明しています。

注意： `adk-eventgenerator.jar` ファイルには、イベント ジェネレータの実装に必要な **WebLogic Integration** ベース クラスが含まれます。これは、**WAR** メイク ファイルに含まれます。

表 7-2 AbstractPushEventGenerator クラス

クラス	説明
AbstractPushEventGenerator	AbstractPullEventGenerator と同じ抽象メソッドおよび具象メソッドが入っているクラス。両方の実装 (AbstractPullEventGenerator and AbstractPushEventGenerator) におけるメソッドの使用目的は同じである。それぞれに割り当てられているメソッドと役割のリストについては、表 7-1 を参照。
IPushHandler	おもにイベントのルーティングからイベントの生成を行うためのインタフェース。データ抽出のプッシュモードの実装には必要とされない。IPushHandler は、PushEventGenerator と密に結合して使用するために設計されている。PushEventGenerator は、PushHandler の実装の初期化、サブスクライブ、クリーンアップを行う。IPushHandler は、生成ロジックを行う簡単なインタフェースである。このインタフェースによって、リソースの初期化、プッシュ イベントへのサブスクライブ、およびクリーンアップを行う。
PushEvent	java.util.EventObject から派生するイベントオブジェクト。PushEvent オブジェクトは EIS 通知のラップとして設計されており、あらゆる IPushEventListener オブジェクトに送信する。
EventMetaData	イベント生成に必要なデータをラップする。このクラスは、初期化時に IPushHandler に渡される。

イベントジェネレータの実装方法

イベントジェネレータの実装は、通常、次の制御フローに従って行います。

1. `doInit()` メソッドは、EIS に対する接続を作成し、有効性を検証します。
2. `setupNewTypes()` メソッドは、処理に必要なデータ構造を作成する `IEventDefinition` オブジェクトを処理します。
3. `postEvents()` メソッドは、以下のデータ抽出モードのいずれか 1 つを繰り返し呼び出します。
 - プッシュ `postEvents()` メソッドはイベントが存在する場合に EIS をポーリングし、`postEvent()` ほどの `IEventDefinition` オブジェクトがそれを受け取るかを決定します。その後、関連スキーマを使用して、イベント データを `IDocument` オブジェクトに変換し、`IEventDefinition` オブジェクトに関連する `IEvent` を使用して `IDocument` オブジェクトをルーティングします。
 - プル `postEvents()` メソッドはイベントの通知を待ちます。通知を受け取ると、イベント データを `PushEvent` オブジェクトから抽出し、それを、イベント アダプタに割り当てられたスキーマに従って `IDocument` オブジェクトに変換します。`IDocument` に必要なイベント データがすべて含まれている場合、`IDocument` は適切な `IEventDefinition` オブジェクトに転送されます。
4. `removeDeadTypes()` メソッドは、イベント処理に使用されているデータ構造から無効な `IEventDefinition` オブジェクトを削除します。これらのオブジェクトに関連付けられたリソースも解放されます。`IEventDefinition` オブジェクトは、アプリケーション ビューがデプロイされていない場合に、無効とみなされます。
5. `doCleanUpOnQuit()` メソッドは、イベント処理中に割り当てられたリソースを削除します。

コード リスト 7-2 は、サンプルアダプタの（プル モード）イベント ジェネレータのクラス宣言を示しています。

コード リスト 7-2 データ抽出のプル モードの実装例

```
public class EventGenerator
    extends AbstractPullEventGenerator
```

注意: AbstractPullEventGenerator は、独自のスレッドで実行できるようにするため、Runnable インタフェースを実装します。

「手順 3a: イベントジェネレータの作成」では、さらに、データ抽出のプルモードでイベントジェネレータの実装コード例を示します。

サンプル EventGenerator

コードリスト 7-3 は、イベントジェネレータ用の簡単なコンストラクタを示しています。親のメンバーが正しく初期化されるように、親のコンストラクタを呼び出します。また、doInit() メソッドを使用して、map 変数からコンフィグレーション情報を受信し、パラメータの有効性を検証するための方法をこのリストで示します。サンプルでは、設計時にイベントジェネレータに関連付けられたパラメータが使用されています。

コード リスト 7-3 EventGenerator 用のサンプル コンストラクタ

```
public EventGenerator()
{
    super();
}

protected void doInit(Map map)
    throws java.lang.Exception
{
    ILogger logger = getLogger();

    m_strUserName = (String)map.get("UserName");
    if (m_strUserName == null || m_strUserName.length() == 0)
    {
        String strErrorMsg =
            logger.getI18NMessage("event_generator_no_UserName");
        logger.error(strErrorMsg);
        throw new IllegalStateException(strErrorMsg);
    }
    m_strPassword = (String)map.get("Password");
    if (m_strPassword == null || m_strPassword.length() == 0)
    {
        String strErrorMsg = logger.getI18NMessage
            ("event_generator_no_Password");
        logger.error(strErrorMsg);
        throw new IllegalStateException(strErrorMsg);
    }
}
```

コード リスト 7-4 に示されているように、`postEvents()` が親クラスの実行メソッドから呼び出されます。このメソッドが、EIS をポーリングして、新しいイベント発生の有無を判別します。このメソッドは、一定間隔（この間隔は、イベント ルータの `web.xml` ファイルで定義する）で呼び出されます。

コード リスト 7-4 `postEvents ()` の実装例

```
*/ protected void postEvents(IEventRouter router)
    throws java.lang.Exception
{
    ILogger logger = getLogger();

    // 作業 : 実際のアダプタでは EIS を呼び出して、
    // 前回このメソッドが呼び出された後に新しいイベントが発生
    // したかどうかの判別が必要となる。例示のため、
    // このメソッドが呼び出されるたびに 1 つのイベントを通知する・・・
    // イベント データは現在イベント定義に従って
    // フォーマットされたシステムにあるいくつかの・・・
    // イベント タイプを検索する・・・

    Iterator eventTypesIterator = getEventTypes();
    if (eventTypesIterator.hasNext())
    {
        do
        {
            // イベント ルータはまだこのタイプのイベントに関連性がある

            IEventDefinition eventDef = (IEventDefinition)
                eventTypesIterator.next();
            logger.debug("Generating event for " + eventDef.getName());

            // デフォルト イベント (ブランクまたはデフォルト データ) を作成する

            IEvent event = eventDef.createDefaultEvent();

            // イベントのフォーマットを取得する

            java.util.Map eventPropertyMap = eventDef.
                getPropertySet();
            String strFormat = (String)eventPropertyMap.get
                ("Format");
            if( logger.isDebugEnabled() )
                logger.debug("Format for event type '"+eventDef.
                    getName()+"' is '"+strFormat+"'");
            java.text.SimpleDateFormat sdf =
                new java.text.SimpleDateFormat(strFormat);
            IDocument payload = event.getPayload();
            payload.setStringInFirst("/SystemTime", sdf.format(new
                Date()));
        }
    }
}
```

7 イベントアダプタの開発

```
// ここで、監査メッセージのログを取る ...

try
{
    logger.audit(toString() + ": postEvents >>> posting event
        ["+payload.toXML()+"] to router");
}

    catch (Exception exc)

{
    logger.warn(exc);
}

// この呼び出しによって実際に IEventRouter にイベントが通知される

router.postEvent(event);
} while (eventTypesIterator.hasNext());
}

} // postEvents の終了
```

実際のアダプタは、EIS をクエリして、前回このメソッドが呼び出された後に新しいイベントが発生したかどうかを判別する必要があります。ADK に付属の DBMS サンプルアダプタで提供される、このような呼び出しの具体例は、EventGenerator.java ファイルの postEvent() メソッドです。

WLI_HOME/adapters/dbms/src/com/bean/adapters/dbms/event/EventGenerator.java

新しいイベントタイプの追加

新しいイベントタイプの処理更新時には、setupNewTypes() が呼び出されます。通常、イベントジェネレータは、EIS からのイベントを受け取るために EIS にリソースを割り当てる必要があります。たとえば、DBMS サンプルアダプタでは、新しいイベントタイプを処理するために、DBMS でトリガが作成されます。setupNewTypes() メソッドにより、新しいタイプを処理するのに必要な定義をセットアップできます。親クラスですでに listOfNewTypes() ファイルの健全性チェックがなされ、ログに記録されているので、ここでチェックは行いません。

コード リスト 7-5 setupNewTypes() のテンプレート サンプル

```
protected void setupNewTypes(java.util.List listOfNewTypes)
{
    Iterator iter = listOfNewTypes.iterator();
```

```
        while (iter.hasNext())
        {
            IEventDefinition eventType = (IEventDefinition)iter.next();
        }
    }
```

アンデプロイされているアプリケーション ビューのイベント タイプの削除

アンデプロイされたアプリケーション ビューのイベント タイプの削除時には、`removeDeadTypes()` が呼び出されます。

クリーンアップ処理を実行する必要があります。

未使用イベント タイプが今後処理されないようにするため、クリーンアップ処理を行う必要があります。たとえば、未使用のイベント タイプの処理に必要なリソースをクローズします。コード リスト 7-6 に `removeDeadTypes()` の実装方法を示します。

コード リスト 7-6 `removeDeadTypes()` テンプレートに基づくサンプル コード

```
protected void removeDeadTypes(java.util.List listOfDeadTypes)
{
    Iterator iter = listOfDeadTypes.iterator();
    while (iter.hasNext())
    {
        IEventDefinition eventType = (IEventDefinition)iter.next();
    }
}
```

リソースの削除

イベント ジェネレータのシャットダウン時に、`doCleanUpOnQuit()` が呼び出されます。このメソッドはイベント処理中に割り当てられたリソースを削除します。サンプル アダプタはこのメソッドで停止します。以下のリストでは、このメソッドを実装するテンプレートを示します。

コード リスト 7-7 `doCleanUpOnQuit()` メソッドの呼び出し例

```
protected void doCleanUpOnQuit()
    throws java.lang.Exception
{
}
```

```
        ILogger logger = getLogger();
        logger.debug(this.toString() + ": doCleanUpOnQuit");
    }
}
```

手順 3b : データ変換メソッドの実装

データ変換は、EIS からデータを取り出してアプリケーション サーバが読み取れる XML スキーマに変換する処理です。各イベントに対し、スキーマが SOM および IDocument クラス ライブラリを使用して XML 出力の外観を定義します。以下のコード リストに、データ変換処理中のイベントのシーケンスを示します。

- コード リスト 7-8 は、EIS から XML スキーマへのデータ変換に使用されるコードを示しています。
- コード リスト 7-9 は、コード リスト 7-8 のコードで作成された XML スキーマを示しています。
- コード リスト 7-10 は、コード リスト 7-9 のスキーマで作成された有効な XML ドキュメントを示しています。

コード リスト 7-8 EIS データを XML スキーマに変換するサンプル コード

```
SOMSchema schema = new SOMSchema();
SOMElement root = new SOMElement("SENDINPUT");
SOMComplexType mailType = new SOMComplexType();
root.setType(mailType);
SOMSequence sequence = mailType.addSequence();
SOMElement to = new SOMElement("TO");
to.setMinOccurs("1");
to.setMaxOccurs("unbounded");
sequence.add(to);
SOMElement from = new SOMElement("FROM");
from.setMinOccurs("1");
from.setMaxOccurs("1");
sequence.add(from);
SOMElement cc = new SOMElement("CC");
cc.setMinOccurs("1");
cc.setMaxOccurs("unbounded");
sequence.add(cc);
SOMElement bcc = new SOMElement("BCC");
bcc.setMinOccurs("1");
bcc.setMaxOccurs("unbounded");
sequence.add(bcc);
```

```

SOMElement subject = new SOMElement("SUBJECT");
subject.setMinOccurs("1");
subject.setMaxOccurs("1");
sequence.add(bcc);
SOMElement body = new SOMElement("BODY");
if (template == null)
    { body.setMinOccurs("1");
      body.setMaxOccurs("1");
    }
else
    { Iterator iter = template.getTags();
      if (iter.hasNext())
        { SOMComplexType bodyComplex = new SOMComplexType();
          body.setType(bodyComplex);
          SOMAll all = new SOMAll();
          while (iter.hasNext())
            { SOMElement eNew = new SOMElement((String)iter.next());
              all.add(eNew);
            }
          bodyComplex.setGroup(all);
        }
    }
sequence.add(body);
schema.addElement(root);

```

コード リスト 7-9 コード リスト 7-8 のコードで作成された XML スキーマ

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="SENDINPUT">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="TO" maxOccurs="unbounded"
        type="xsd:string"/>
      <xsd:element name="FROM" type="xsd:string"/>
      <xsd:element name="CC" maxOccurs="unbounded"
        type="xsd:string"/>
      <xsd:element name="BCC" maxOccurs=
        "unbounded" type="xsd:string"/>
      <xsd:element name="BCC" maxOccurs="unbounded"
        type="xsd:string"/>
      <xsd:element name="BODY" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

コード リスト 7-10 コード リスト 7-9 のスキーマで作成された有効な XML ド

コメント

```
</xsd:schema>
<?xml version="1.0"?>
<!DOCTYPE SENDINPUT>
<SENDINPUT>
  <TO/>
  <FROM/>
  <CC/>
  <BCC/>
  <BCC/>
  <BODY/>

</SENDINPUT> <xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

手順 4 : アダプタのテスト

WebLogic Integration に用意されているアダプタ テスト 支援機能を使用して、アダプタをテストできます。このツールとその使用方法の詳細については、第 6 章「サービス アダプタの開発」の「手順 5 : アダプタのテスト」を参照してください。

手順 5 : アダプタのデプロイ

新しいアダプタを再構築したら、そのアダプタを WebLogic Integration 環境にデプロイします。アダプタのデプロイは、手動でも WebLogic Server Administration Console からでも実行できます。詳細は、第 9 章「アダプタのデプロイ」を参照してください。

8 設計時 GUI の開発

ADK の設計時フレームワークは、アダプタ ユーザがアプリケーション ビューの定義、デプロイ、およびテストのための Web ベースの GUI を作成するためのツールを提供します。各アダプタには EIS 固有の機能がありますが、アプリケーション ビューをデプロイするための GUI は、すべてのアダプタで必要です。設計時フレームワークでは、主に以下のコンポーネントを使用して、このような GUI の作成やデプロイメントの作業を最小限に抑えることができます。

- **Web アプリケーション コンポーネント** : Java Server Pages (JSP) を使用して HTML ベースの GUI を作成できます。このコンポーネントの機能は、JSP テンプレート、タグ ライブラリ、JavaScript ライブラリなどのツールによって拡張できます。
- **abstractDesignTimeRequestHandler** : WebLogic Server のアプリケーション ビューをデプロイ、アンデプロイ、コピー、および編集する際に利用できる簡単な API を提供します。

この章の内容は以下のとおりです。

- 設計時フォーム処理の概要
- 設計時 GUI の機能
- ファイル構造
- イベントの処理フロー
- 手順 1 : 設計時 GUI 要件の定義
- 手順 2 : ページフローの定義
- 手順 3 : 開発環境のコンフィグレーション
- 手順 4 : 設計時 GUI の実装
- 手順 5 : HTML フォームの作成
- 手順 6 : ルック & フィールの実装

設計時フォーム処理の概要

Java サーブレットと JSP を組み合わせることにより、さまざまな方法でフォームを処理できます。いずれの方法でフォームを処理する場合でも、以下の基本的な必要条件が適用されます。

- HTML フォームの表示。

この機能を作成するには、以下の作業を行います。

 - HTML を使用してフォームレイアウトを生成
 - 必須フィールドをユーザに指示
 - 必要に応じて、フィールドにデフォルト値を事前に設定
- ユーザによって発行されるフォームのデータに含まれる HTTP 要求のフィールド値の検証。

この機能を作成するには、以下の作業を行います。

- すべての必須フィールドが入力されているかどうかを判断するロジックの作成
 - 一連の制約に対して発行された各値の検証。たとえば、年齢のフィールドに 1 ~ 120 までの有効な整数が入力されているかどうか Web application による確認。
- 無効な値が入力されたフォームを、誤ったフィールドの横にエラーメッセージを付けて再表示。Web アプリケーションで各国のロケールがサポートされていれば、エラーメッセージは、ユーザが希望するロケールに合わせてローカライズする必要があります。

また、Web アプリケーションは、有効な情報の再入力を要求せずにユーザによる最後の入力内容を再表示することもできる必要があります。入力されたフィールド値がすべて有効になるまで、Web アプリケーションでは手順 2 を繰り返す必要があります。
 - すべてのフィールドの有効性を大まかに検証した後、フォームを処理する。データの処理中に、個々のフィールドの有効性には無関係なエラー（Java 例外など）が生じることがあります。この場合、ページの最上段にローカライズ済みのエラーメッセージを付けて、フォームを再表示します。手順 3 と同様に、入力された情報が有効な場合は、ユーザが再入力しなくても済むように入力フィールドの値をすべて保存しておく必要があります。

Web アプリケーション開発者は、以下の事項を決定する必要があります。

- フォーム処理 API を実装するオブジェクトまたはメソッド。
- Web アプリケーションで次のページを表示する方法とタイミング。
- フォーム処理が正しく実行されたら、Web アプリケーションで次のページを表示すること。

フォーム処理クラス

Web アプリケーションで使用する各フォームに対してすべてのフォーム処理機能を実装するのは、単調であると同時に、エラーの原因になりやすいプロセスです。ADK 設計時フレームワークでは、「モデル/ビュー/コントローラ (MVC)」パラダイムによって、このプロセスが簡略化されています。このパラダイムでは、以下の 5 つのクラスが使用されます。

- RequestHandler
- ControllerServlet
- ActionResult
- Word と子孫クラス
- AbstractInputTagSupport と子孫クラス

RequestHandler

`com.bea.web.RequestHandler`

このクラスにより HTTP 要求処理ロジックが提供されます。このクラスは、MVC ベース メカニズムのモデル コンポーネントです。RequestHandler オブジェクトは ControllerServlet によってインスタンス化され、HTTP セッション中に handler というキーの下に保存されます。ADK には

`com.bea.adapter.web.AbstractDesignTimeRequestHandler` というクラスがあります。この抽象基本クラスにより、アプリケーション ビューをデプロイする際にすべてのアダプタに共通して必要となる機能が実装されます。アダプタまたは EIS 固有のロジックを定義するには、このクラスを拡張する必要があります。

ControllerServlet

`com.bea.web.ControllerServlet`

このクラスは HTTP 要求を受け取ると、要求中の各フィールド値の有効性を検証し、要求の処理を `RequestHandler` に委託します。また、表示するページを決定します。`ControllerServlet` は **Java Reflection** を使って、`RequestHandler` で呼び出すメソッドを特定します。`ControllerServlet` はフォーム処理ロジックを実装するメソッド名を指定するために、`doAction` という HTTP 要求パラメータを探します。このパラメータがなければ、`ControllerServlet` は `RequestHandler` でメソッドを呼び出しません。

`ControllerServlet` は、Web アプリケーションに合わせて `web.xml` ファイルでコンフィグレーションされます。`ControllerServlet` は、`RequestHandler` で実行するメソッドに HTTP 要求を委託します。`ControllerServlet` を使用する場合、コードを指定する必要はありません。ただし、37 ページの表 8-5 に示す初期パラメータを指定する必要があります。

ActionResult

`com.bea.web.ActionResult`

`ActionResult` では、要求の処理結果がカプセル化されます。また、`ControllerServlet` には、次に表示するページを決定するための情報が提供されます。

Word と子孫クラス

`com.bea.web.validation.Word`

Web アプリケーションのすべてフィールドについて、その有効性を一定の範囲で検証する必要があります。`com.bea.web.validation.Word` クラスおよびその子孫オブジェクトにより、フォーム中の各フィールドの有効性を検証するためのロジックが提供されます。無効なフィールドがあると、`Word` オブジェクトはメッセージバンドルを使用して、該当するフィールドに応じてインターナショナルライズまたはローカライズされたエラーメッセージを検索します。ADK では、表 8-1 に示すカスタムバリデータが提供されます。

表 8-1 Word オブジェクトのカスタムバリデータ

バリデータ	フィールドに対する値の決定
Integer	指定された範囲内の整数値であるかどうかを判断する。
Float/Double	指定された範囲内の浮動小数点数になっているかどうかを判断する。
Identifier	有効な Java 識別子であるかどうかを判断する。
Perl 5 Regular Expression	Perl 5 の正規表現に一致するかどうかを判断する。
URL	(ユーザにより入力されたものが) 有効な URL であるかどうかを判断する。
Email	(ユーザにより入力された値が) 有効な電子メール アドレスかどうかを判断する。
Date	(ユーザによって入力された値が) 指定の日付/時間フォーマットに従っており、有効な日時になっているかどうかを判断する。

AbstractInputTagSupport と子孫クラス

`com.bea.web.tag.AbstractInputTagSupport`

Web ツールキットで提供されるタグ クラスには以下の機能があります。

- フォーム フィールド用の HTML を生成し、必要に応じてフィールドにデフォルト値を設定しておく。
- 入力された値が無効な場合に、フォーム フィールドの横にローカライズしたエラー メッセージを表示する。
- `ControllerServlet` がフォームのフィールド名を使って、検証オブジェクトにアクセスできるように、`com.bea.web.validation.Word` を初期化して Web アプリケーション スコープに保存する。

発行タグ

これ以外に、ADK では次のような発行タグを使用できます。

```
<adk:submit name='xyz_submit' doAction='xyz' />
```

このタグにより、doAction パラメータが要求中の ControllerServlet に送られます。この結果、ControllerServlet から登録済の RequestHandler で xyz() メソッドが呼び出されます。

フォーム処理シーケンス

この節では、フォームを処理する順序について説明します。

前提条件

フォームを処理するための前提条件は、以下のとおりです。

1. カスタム ADK 入力タグを使用した JSP が HTTP 応答オブジェクトに書き込まれるとき、このタグにより com.bea.web.validation.Word のインスタンスがオブジェクトによって初期化され、入力フィールド名をキーとして、Web アプリケーション スコープにこのインスタンスが格納されること。このようなタグにより、ControllerServlet で検証オブジェクトが使えるようになるため、まず HTTP 要求を大まかに検証してから、RequestHandler に要求を出すことができます。次にその例を示します。

```
<adk:int name='age' minInclusive='1' maxInclusive='120'  
required='true' />
```

2. JSP エンジンが com.bea.web.tag.IntegerTagSupport のインスタンスで doStartTag() メソッドを呼び出した時点で、このタグの HTML が生成されること。IntegerTagSupport インスタンスにより、com.bea.web.validation.IntegerWord の新しいインスタンスが作成され、これがキー age に基づいて Web アプリケーション スコープに追加されます。また、ControllerServlet では、対応する ServletContext から IntegerWord インスタンスを検索できます。検証では、年齢に送られた値が 1 以上、かつ 120 以下であることが確認されます。

3. HTML フォームでは、doAction という非表示フィールドを発行する必要があります。このフィールドの値は、ControllerServlet がフォーム処理を実行する RequestHandler 上のメソッドを特定するときに使用されます。

これら前提条件に従うと、コード リスト 8-1 のような JSP フォームが表示されます。

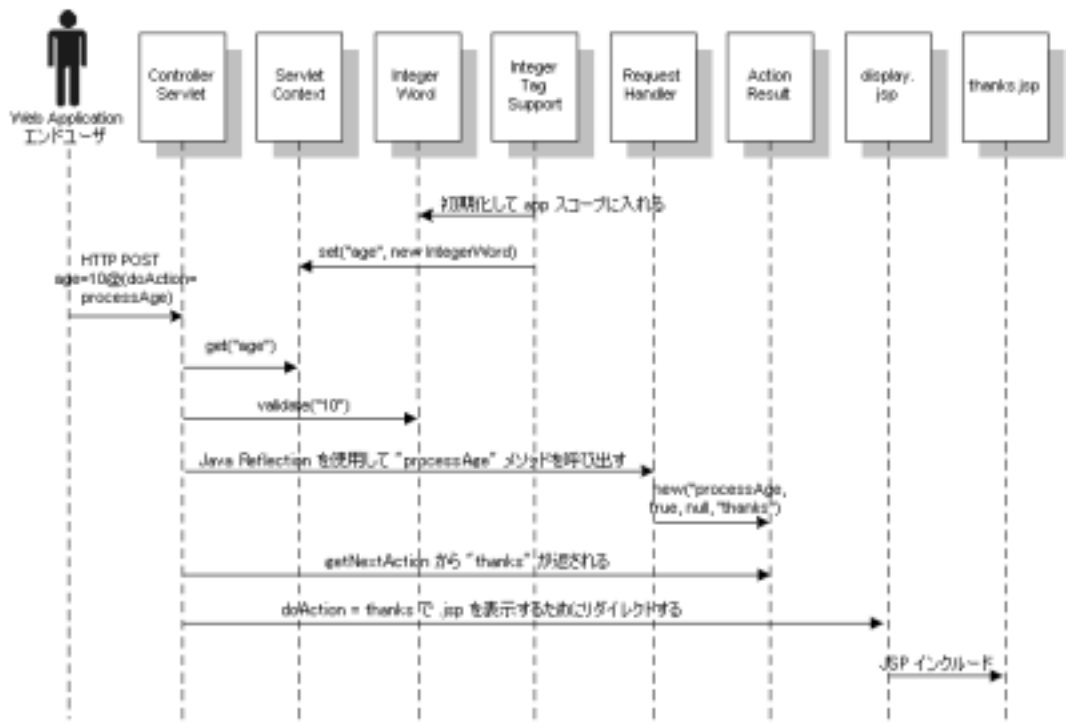
コード リスト 8-1 JSP サンプル フォーム

```
<form method='POST' action='controller'>
  Age: <adk:int name='age' minInclusive='1' maxInclusive='120'
        required='true' />
  <adk:submit name='processAge_submit' doAction='processAge' />
</form>
```

シーケンスの手順

フォーム処理の実行方法を、順を追って次のダイアグラムに示します。

図 8-1 UI フォーム処理



シーケンスは次のとおりです。

1. ユーザが、以下の内容のフォームを発行します。age=10, doAction=processAge.
2. ControllerServlet が HTTP 要求から年齢フィールドのデータを読み込みます。
3. ControllerServlet は age をキーとして使用して、ServletContext から com.bea.web.validation.Word を検索します。このオブジェクトは、com.bea.web.validation.IntegerWord のインスタンスです。
4. ControllerServlet が Word インスタンスで validate() メソッドを呼び出し、パラメータとして 10 を渡します。

5. **Word** インスタンスにより、この値 10 が 1 以上 120 以下の値であることが判断されます。
6. `ControllerServlet` がセッションから `RequestHandler` を検索、または作成して、`handler` としてこれをセッションに追加します。
7. `ControllerServlet` は、**Java Reflection API** を使用して、`processAge()` を探し、`RequestHandler` でこのメソッドを呼び出します。このメソッドが存在しない場合は、例外が生成されます。メソッドのシグネチャは、次のとおりです。

```
public ActionResult processAge(HttpServletRequest request)
throws Exception
```
8. `RequestHandler` は、フォームの入力データを処理した後、`ActionResult` オブジェクトを返して処理の結果を示します。`ActionResult` には、`ControllerServlet` が次に表示するページを判断するための情報が格納されます。次のページの情報には、**Web** アプリケーション内に表示する別の **JSP** や **HTML** のページの名前などがあります。たとえば `thanks` により、`thanks.jsp` ページが表示されます。
9. `ActionResult` が正常に実行されると、`ControllerServlet` により **HTTP** 応答が **Web** アプリケーションの表示ページにリダイレクトされます。**ADK** では、表示ページは通常、`display.jsp` となります。
10. `display.jsp` ページには、`content` パラメータ（たとえば、`thanks.jsp`）により指定される **JSP** があります。これにより、**JSP** がユーザに表示されます。

設計時 GUI の機能

設計時 GUI の開発では、実行時アダプタを開発する場合とは異なる独自の機能を使用します。この節では、設計時 GUI の機能について説明します。

Java Server Pages

設計時 GUI は、一連の Java Server Pages (JSP) で構成されます。JSP とは、特定のトランザクションを開始する目的で Java サーブレットを呼び出すための HTML ページのことです。ユーザ側から見た JSP は、通常の Web ページと変わりません。

設計時 GUI を構成する JSP を以下の表に示します。

表 8-2 設計時 GUI JSP

ファイル名	説明
display.jsp	アダプタ ホームページとも呼ばれる表示ページ。このページには、ルック & フィールドを作成するのに必要な HTML が表示される。
login.jsp	アダプタの設計時 GUI ログイン ページ
confconn.jsp	[Confirm Connection] ページは、EIS の接続パラメータを指定するためのフォーム。
appvwadmin.jsp	[Application View Administration] ページには、アンデプロイされたアプリケーション ビューの概要が表示される。
addevent.jsp	[Add Event] ページではイベントをアプリケーション ビューに追加できる。
addservc.jsp	[Add Service] ページではサービスをアプリケーション ビューに追加できる。
edtevent.jsp	[Edit Event] ページでは必要に応じてユーザがイベントを編集できる。
edtservc.jsp	[Edit Service] ページでは必要に応じてユーザがサービスを編集できる。
depappvw.jsp	[Deploy Application View] ページでは、ユーザがデプロイメント プロパティを指定できる。

これらの JSP の実装方法については、8-20 ページの「手順 2: ページフローの定義」を参照してください。

JSP テンプレート

テンプレートとは、HTTP 要求で指定するパラメータに基づいて Java サーブレットが動的に生成する HTML ページのことです。テンプレートを使用することにより、Web アプリケーションで使用するカスタム ページやカスタム HTML の量を最小限に抑えることができます。

設計時フレームワークの JSP テンプレートを使用すると、アダプタの新しいアプリケーション ビューを定義、デプロイ、およびテストするための Web アプリケーションを効率的に作成できます。ADK のテンプレートでは、アダプタの開発者向けに次の 3 つの利点を提供します。

- アプリケーション ビューをデプロイする際に必要となる HTML フォームの大部分が用意されています。通常、次の 3 つのカスタム フォームを作成すれば、ほとんどの要求に対応できます。
 - EIS 固有の接続パラメータを収集するフォーム
 - イベントの追加に必要な EIS 固有の情報を収集するフォームイベントの編集に必要な情報の収集は、同じフォームでも異なるフォームでも使用できます。
 - サービスの追加に必要な EIS 固有の情報を収集するフォーム EIS のメタデータ カタログを参照するためのカスタム JSP を作成することもできます。サービスの編集に必要な情報の収集は、同じフォームでも異なるフォームでも使用できます。
- これらは、Java プラットフォームのインターナショナルライゼーションおよびローカライゼーション機能にも利用できます。Web アプリケーションで使用される各ページのコンテンツはメッセージバンドルに格納されます。したがって、アダプタの Web インタフェースを効率的にインターナショナルライズできます。
- これにより、すべてのテンプレート間での一貫したルック & フィールが保証されます。

ADK で提供される JSP テンプレートのリストについては、8-11 ページの「JSP テンプレート」を参照してください。

JSP タグの ADK ライブラリ

ADK で提供される JSP タグ ライブラリを使用することにより、使いやすい HTML フォームの作成が容易になります。アダプタ ページを開発する際、HTML フォーム入力コンポーネントのカスタム タグを使用して、検証メカニズムへシームレスにリンクできます。ADK が提供するカスタム タグについて、以下の表に示します。

表 8-3 ADK JSP タグ

タグ	説明
adk:check box	フォームを表示するときに、チェック ボックスのフォーム フィールドをチェックする必要があるかどうかを決定する。(このタグはデータ検証を実行しない。)
adk:content	メッセージ バンドル中の特定のメッセージにアクセスする。
adk:date	日付値が特定のフォーマットに従って入力されているか検証する。
adk:double	2 バイト値が入力されているか検証する。
adk:email	有効な電子メールアドレス (複数も可) が入力されているかを検証する。
adk:float	浮動小数点の値が入力されているか検証する。
adk:identifier	有効な Java の識別子が入力されているか検証する。
adk:int	整数値が入力されているか検証する。
adk:label	メッセージ バンドルからラベルを表示する。
adk:password	ユーザが入力したテキスト フィールドと Perl 5 の正規表現を照合して、入力にアスタリスク (*) を付ける。
adk:submit	フォームを検証メカニズムにリンクする。
adk:text	入力値と Perl 5 の正規表現を照合する。
adk:textarea	入力されたテキスト領域と Perl 5 の正規表現を照合する。

表 8-3 ADK JSP タグ（続き）

タグ	説明
adk:url	有効な URL が入力されているか検証する。

JSP タグの属性

表 8-4 に記載されている属性を使用して JSP タグをさらにカスタマイズできます。

表 8-4 JSP タグの属性

タグ	必須の属性	使用可能な属性
adk:int, adk:float, adk:double	name - フィールド名	default - ページに表示するデフォルト値 maxlength - 値の最大長 size - ディスプレイサイズ minInclusive - ユーザが指定する値はこの値以上でなければならない maxInclusive - ユーザが指定する値はこの値以下でなければならない minExclusive - ユーザが指定する値はこの値を下回ってはならない maxExclusive - ユーザが指定する値はこの値を上回ってはならない required - true または false（デフォルトは False、省略可能フィールド） attrs - 追加用の HTML 属性

表 8-4 JSP タグの属性 (続き)

タグ	必須の属性	使用可能な属性
adk:date	name - フィールド名	default - ページに表示するデフォルト値 maxlength - 値の最大長 size - ディスプレイサイズ required - true または false (デフォルトは False、省略可能フィールド) attrs - 追加用の HTML 属性 lenient - true または false (デフォルトは False、つまり構文解析時に日付フォーマットの基準を厳しく設定する。) format - ユーザ入力で予期されるフォーマット (デフォルトは <i>mm/dd/yyyy</i>)
adk:email, adk:url, adk:identifier	name - フィールド名	default - ページに表示するデフォルト値 maxlength - 値の最大長 size - ディスプレイサイズ required - true または false (デフォルトは False、省略可能フィールド) attrs - 追加用の HTML 属性
adk:text, adk:password	name - フィールド名	default - ページに表示するデフォルト値 maxlength - 値の最大長 size - ディスプレイサイズ required - true または false (デフォルトは False、省略可能フィールド) attrs - 追加用の HTML 属性 pattern - Perl 5 の正規表現

表 8-4 JSP タグの属性（続き）

タグ	必須の属性	使用可能な属性
adk:textarea	name - フィールド名	default - ページに表示するデフォルト値 required - true または false（デフォルトは False、省略可能フィールド） attrs - 追加用の HTML 属性 pattern - Perl 5 の正規表現 rows - 表示する行数 columns - 表示するカラム数

注意： タグの使用方法については、次の場所にある `adk.tld` を参照してください。

`WLI_HOME/adapters/src/war/WEB-INF/taglibs`

アプリケーション ビュー

アプリケーション ビューはアプリケーション中の特定機能に対するビジネスレベルのインタフェースを表しています。詳細については、1-7 ページの「アプリケーション ビュー」を参照してください。

ファイル構造

設計時 GUI アダプタを構築する場合、サービス アダプタの構築時と同じファイル構造が必要になります。6-7 ページの「手順 2a: ディレクトリ構造の設定」を参照してください。この参照ページで説明されているファイル構造以外に、以下の点にも注意する必要があります。

- 各アダプタの設計時インタフェースは、WAR ファイルとしてまとめられた J2EE 準拠の Web アプリケーションです。
- Web アプリケーションは `.jsp`、`.html`、および画像ファイルをバンドルしたものであること。

- Web アプリケーション記述子は、`WLI_HOME/adapters/ADAPTER/src/war/WEB-INF/web.xml` です。この記述子により、**J2EE Web** コンテナに対して、**Web** アプリケーションをデプロイおよび初期化する方法が指示されること。

イベントの処理フロー

図 8-2 は、設計時 GUI の開発手順を表しています。

図 8-2 設計時 GUI の開発のイベント フロー



手順 1 : 設計時 GUI 要件の定義

設計時 GUI の開発を始める前に、以下の質問に答えてその要件を定義する必要があります。

- この GUI でサポートされるのはイベント アダプタですか、サービス アダプタですか、またはその両方ですか？
- ユーザはどのようにしてイベントおよびサービス カタログをブラウズしますか？

EIS では、イベントおよびサービス カタログにアクセスするための関数を定義する必要があります。EIS にこれらの関数がない場合は、カタログをブラウズできません。EIS がこれらの関数に対応していない場合は、次の設計原理に従うことをお勧めします。EIS のメタデータを入手するために設計時 UI を呼び出します。これは事実上、実行時コンポーネントを呼び出すことと同じです。いずれの場合も、バックエンド EIS で実行されます。

したがって、設計時メタデータの機能を提供するには、可能な限り実行時アーキテクチャを利用する必要があります。CCI Interaction オブジェクトを使用する設計時 GUI 固有の関数を呼び出す必要があります。ADK に用意されたサンプルアダプタでは、このアプローチに基づく例やフレームワークが提供されます。サンプルアダプタは、`WLI_HOME/adapters/sample` にあります。

- アダプタは、サービスに対する要求 / 応答スキーマをどのように生成しますか？EIS を呼び出しますか、またはその他の手法を使いますか？

一般に、特定の関数やイベントに関するメタデータを入手するには、アダプタから EIS を呼び出す必要があります。これにより、アダプタで EIS メタデータが XML スキーマフォーマットに変換されます。このとき、SOM API を呼び出す必要があります。サンプルアダプタは SOM API の実装にも対応しています。この API の詳細は、8-12 ページの「JSP タグの ADK ライブラリ」を参照してください。

- 特定のサービス テストをサポートしますか？設計時 GUI がサービス テストをサポートするようにする場合、以下を設定する必要があります。
 - XML 応答スキーマを HTML フォームに変換するクラス。このクラスの例については下記を参照してください。

```
WLI_HOME/adapters/dbms/docs/api/com/bea/adapter/dbms/utills/  
class-use/TestFormBuilder.html
```

A 変換機能を呼び出して、HTML フォームを表示する JSP 指定の `testform.jsp`。このファイルの例については、次を参照してください。
`WLI_HOME/adapters/dbms/src/war/`

手順 2 : ページ フローの定義

ユーザがアプリケーション ビューを呼び出した場合の JSP の表示順序を指定する必要があります。この節では、効果的なアプリケーション ビューで必要になる基本的なページのフローについて説明します。なお、フローには具体的な要求に応じてページを追加できるので、ここでは最低限の条件についてのみ説明します。

画面 1 : ログイン

アプリケーション ビューはセキュリティ対策が施されたシステムであるため、ビューを実装するには、まずログインが必要です。このため、最初は必ず [Application View Console - Logon] ページを表示します。

このページを使用するには、有効なユーザ名とパスワードを入力する必要があります。この情報の有効性が検証されて、ユーザが、デフォルトの **WebLogic Server** セキュリティレルムに定義されたアダプタグループのメンバーであることを確認します。

注意： アプリケーションビューの Web アプリケーションに対するセキュリティ要件は、`WLI_HOME/adapters/ADAPTER/src/war/WEB-INF/web.xml` ファイルで指定します。これは、`adapter.war` ファイルに組み込まれています。

画面 2 : アプリケーション ビューの管理

ユーザが正しくログインすると、[Application View Console] ページが表示されます。このページには、アプリケーション ビュー、各フォルダのステータス、およびこれらに対して実行されたアクションを示すフォルダが表示されます。ユーザはこのページから既存のアプリケーション ビューを表示したり、新しいアプリケーション ビューを追加できます。

- 既存のアプリケーション ビューを表示するには、該当するフォルダを選択して、アプリケーション ビューを指定します。アプリケーション ビューを選択すると、[Application View Summary] ページ (appvwsum.jsp) が表示されます。このページの詳細については、8-26 ページの「画面 9 : アプリケーション ビューの概要」を参照してください。
- 新しいアプリケーション ビューを追加するには、[Add Application View] をクリックします。[Define New Application View] 画面が表示されます。

画面 3 : 新しいアプリケーション ビューの定義

[Define New Application View] ページ (defappvw.jsp) を使用して、クライアントが置かれたあらゆるフォルダに新しいアプリケーション ビューを定義できます。これには、アプリケーション ビューとアダプタを関連付けるための説明が必要になります。このフォームには、アプリケーション ビュー名と説明を入力するフィールドに加え、アプリケーション ビューと関連付けるアダプタを表示するドロップダウン リスト ボックスがあります。

新しいアプリケーション ビューを定義した後に [OK] を選択すると、[Configure Connection] ページが表示されます。

画面 4 : 接続のコンフィグレーション

新しいアプリケーション ビューが有効であれば、ユーザは接続のコンフィグレーションを行う必要があります。すなわち、アプリケーション ビューの有効性が検証された後、[Configure Connection Parameters] ページ (confconn.jsp)

が表示されます。このページは、EIS の接続パラメータを指定するためのフォームです。接続パラメータは EIS 固有のものであるため、このページの外観はアダプタごとに異なります。

ユーザが接続パラメータを発行すると、アダプタはこのパラメータを使用して EIS への新しい接続を試みます。EIS に接続されると、ユーザには次の [Application View Administration] ページが表示されます。

画面 5 : アプリケーションビューの管理

ユーザには、新しいアプリケーションビューを管理する手段が必要になります。[Application View Administration] ページ (appvwadmin.jsp) には、アンデプロイされたアプリケーションビューの概要が表示されます。具体的には、以下の情報を示します。

- 接続条件—接続条件セクションでは、ユーザが接続パラメータを変更するための [Configure Connection] ページに戻るリンクを設定します。
- イベントのリスト—アプリケーションビューの各イベントについて、ユーザは以下の操作を実行できます。
 - XML スキーマの表示
 - イベントの削除
 - イベント プロパティの指定
- サービスのリスト—アプリケーションビューの各サービスについて、ユーザは以下の操作を実行できます。
 - 要求 XML スキーマの表示
 - 応答 XML スキーマの表示
 - サービスの削除
 - サービス プロパティの指定

このページでは、アプリケーションビューにイベントリストやサービスリストを提供する以外に新しいイベントやサービスを追加できるページへのリンクも設定します。

画面 6 : イベントの追加

イベントをアプリケーション ビューに追加する必要があります。[Add Event] ページ (addevent.jsp) で、これを行うことができます。

新しいイベントには以下のルールが適用されます。

- 各イベントにはそれぞれユニークな名前を割り当てる。
 - イベント名には、a ~ z、A ~ Z、0 ~ 9、アンダースコア (_) の文字を使用すること。先頭の文字は、英字でなければなりません。それ以外の文字は無効です。
 - 名前は 256 文字以内で指定すること。
 - イベント名が、アプリケーション ビューにおいてユニークな名前になっていること。ユーザが重複したイベント名を指定すると、イベントがすでに定義されているという旨のエラー メッセージが表示され、フォームが再ロードされます。
- 必要に応じてイベントの説明を入力できます。この説明は、2048 (2K) 文字以内で指定します。
- 各イベントには、名前と説明以外に EIS 固有のパラメータを指定すること。一連の EIS 固有のパラメータによって、アダプタに対するイベント タイプを定義します。
- 必要に応じて、EIS のイベント カタログを参照するメカニズムをアダプタに搭載すること。

新しいイベントを定義し、保存したときに [Application View Administration] ページに戻ります。

画面 7 : サービスの追加

アプリケーション ビューに新しいサービスを追加したい場合があります。[Add Service] ページ (addservc.jsp) で、これを行うことができます。

新しいイベントには以下のルールが適用されます。

- 各サービスにはそれぞれユニークな名前を割り当てる。

- サービス名には、**a ~ z, A ~ Z, 0 ~ 9, アンダースコア (_)** の文字を使用すること。先頭の文字は、英字でなければなりません。それ以外の文字は無効です。
- 名前は **256 文字以内** で指定すること。
- サービス名がアプリケーションビューに固有の名前になっていること。ユーザが重複したサービス名を指定すると、サービスがすでに定義されているという旨のエラーメッセージが表示され、フォームが再ロードされます。
- 必要に応じてサービスの説明を入力できます。この説明は、**2048 (2K)** 文字以内で指定します。
- 各サービスには、名前と説明以外に **EIS 固有のパラメータ** を指定すること。一連の **EIS 固有のパラメータ** によって、アダプタに対するサービスタイプを定義します。
- 必要に応じて、**EIS のサービスカタログ** を参照するメカニズムをアダプタに搭載すること。

新しいサービスを定義し、保存すると [Application View Administration] ページに戻ります。

画面 8 : アプリケーションビューのデプロイ

1つ以上のサービス、またはイベントを追加したら、アプリケーションビューをデプロイできます。アプリケーションビューをデプロイすると、イベントやサービスの処理にこれを利用できるようになります。アプリケーションビューをデプロイすると、[Deploy Application View] ページ (depappvw.jsp) が表示されます。

このページで、ユーザは以下のデプロイメントプロパティを指定します。

- 接続プーリングパラメータ
 - 最小プールサイズ - 0 以上
 - 最大プールサイズ - 1 以上
 - 最大プールサイズのターゲット小数部 - 0 以上、1 未満
 - プールの縮小 - 接続プールは縮小できるかどうかをコントロール

- ログイン レベル—次の 4 つのログイン レベルのいずれかを指定できます。
 - [Log all message]
 - [Log informational messages, warnings, errors, and audit messages]
 - [Log warnings, errors, and audit messages]
 - [Log errors and audit messages]
- セキュリティー Restrict Access リンクをクリックしてアプリケーション ビューのセキュリティ制限を適用するフォームにアクセスできます。

ユーザ アクセスの制御

表示されたフォームでユーザ名、またはグループ名を指定することにより、別のユーザのアクセス特権を許可したり、無効にしたりすることができます。アプリケーション ビューでは 2 つの機能（読み込みと書き込み）へのアクセスを制御します。

- 読み込みアクセス権を持つユーザは、サービスの実行、およびイベントのサブスクライブができます。
- 書き込みアクセス権を持つユーザは、アプリケーション ビューをデプロイ、編集、およびアンデプロイできます。

アプリケーション ビューのデプロイ

ユーザは、デプロイ オプションを選択してアプリケーション ビューをデプロイします。ユーザは、アプリケーション ビューを永続的にデプロイするかどうかを決定する必要があります。永続的デプロイメントが選択された場合、アプリケーション サーバを再起動するたびにアプリケーション ビューを再デプロイします。

アプリケーション ビューの保存

アプリケーション ビュー コンソールを介して、アンデプロイしたアプリケーション ビューを保存したり、後でこれを復元したりできます。このプロセスでは、デプロイされたアプリケーション ビューがすべて、リポジトリの中に保存されていることが前提になります。すなわち、保存されていないアプリケーション ビューをデプロイすれば自動的に保存されます。

画面 9 : アプリケーション ビューの概要

アプリケーション ビューが正しくデプロイされると、[Application View Summary] (appvwsum.jsp) ページが表示されます。このページにはアプリケーション ビューに関する以下の情報が表示されます。

- デプロイ ステート (デプロイされているかアンデプロイされているか)

- アプリケーション ビューがデプロイされている場合

ページには、アプリケーション ビューをアンデプロイするためのオプションがあります。ユーザが [Undeploy] リンクをクリックすると、この選択を確認する子ウィンドウが表示されます。ユーザが確認した後、アプリケーション ビューはアンデプロイされ、概要ページが再表示されます。このようにしてアンデプロイされたアプリケーション ビューは、続けてリポジトリに保存されるため、ユーザはアプリケーション ビューを編集したり、削除したりできます。

アダプタでイベントのテストがサポートされる場合は、[Summary] ページに各イベントのテスト リンクが表示されます。イベントのテストは、ADK で直接的にはサポートされていません。

アダプタでサービスのテストがサポートされる場合は、[Summary] ページに各サービスのテスト リンクが表示されます。ADK では、testservc.jsp と testrsflt.jsp のファイルを提供することにより、サービスのテストに使用できる 1 つのアプローチが紹介されています。これらのページを自由に使用して自分のサービス テスト方法を作成できます。

- アプリケーション ビューがデプロイされていない場合

ページには、アプリケーション ビューをデプロイするためのオプションがあります。ユーザが [Deploy] リンクを選択すると、アプリケーション ビューがデプロイされ [Application View Summary] ページが再ロードされます。

ページには、アプリケーション ビューを編集するためのオプションがあります。ユーザが [Edit] リンクをクリックすると、[Application View Administration] ページが表示されます。

ページには、アプリケーション ビューを削除するためのオプションがあります。ユーザが [Remove] リンクをクリックすると、子ウィンドウが表示され、実際に ADK リポジトリからアプリケーション ビューを削除す

るかどうかが確認されます。ユーザが確認した後、アプリケーションビューがリポジトリから削除され、アプリケーションビュー コンソールに戻ります。

- 接続条件
- デプロイメント情報 (プーリング コンフィグレーション、ログレベル、セキュリティ)
- イベント リスト-各イベントに対し、[Summary] ページでは、スキーマを表示し、テストがサポートされてる場合は、テストを行うオプションがあります。ユーザはこのページからイベントの削除はできません。まず編集が必要です。
- サービス リスト-各サービスに対し、[Summary] ページでは、要求スキーマおよび応答スキーマを表示し、サービスのテストがサポートされてる場合は、テストを行うオプションがあります。ユーザは、このページからサービスの削除はできません。まずアンデプロイして、編集する必要があります。

手順 3 : 開発環境のコンフィグレーション

ここでは、設計時 GUI 開発をサポートするためのソフトウェア環境をセットアップします。

手順 3a : メッセージ バンドルの作成

エンド ユーザ向けのメッセージは、メッセージバンドルに入れます。メッセージバンドルは、「*key=value*」の組み合わせから構成される単純な `.properties` テキスト ファイルで、メッセージをインターナショナルライズできます。実行時にロケールと母国語を指定すると、メッセージの内容が、キー値の組み合わせの指定に従って翻訳され、ロケールに合わせた言語でメッセージが表示されます。

メッセージバンドルの作成方法については、次に示すサイトにある **JavaSoft** チュートリアルを参照してください。

<http://java.sun.com/docs/books/tutorial/il8n/index.html>

手順 3b : WebLogic Server を再起動せずに JSP を更新する環境のコンフィグレーション

設計時 UI は WAR ファイルの J2EE Web アプリケーションとしてデプロイされます。WAR ファイルは、JAR ファイルにある WEB-INF/web.xml 中で Web アプリケーション記述子が付いた JAR ファイルです。ただし、WAR ファイルでは、WebLogic Server 中の J2EE Web コンテナを使用してそのまま JSP を再コンパイルすることはできません。したがって、通常は単に JSP ファイルを変更するだけでも、WebLogic Server を再起動する必要があります。これは JSP 本来の考え方とは異なるため、ADK では WebLogic Server を再起動しなくても JSP を更新できるように次の解決策をお勧めします。

1. アダプタの設計時 UI 用に有効な WAR ファイルを作成します。サンプルアダプタでは、Ant を使用して、このファイルを作成します。コード リスト 8-2 に J2EE WAR ファイルを生成するターゲットを示します。

コード リスト 8-2 WAR ファイルの作成ターゲット

```
<target name='war' depends='jar'>

<!-- 既存の環境を削除する -->
  <delete file='${LIB_DIR}/${WAR_FILE}'/>

<war warfile='${LIB_DIR}/${WAR_FILE}'
  webxml='${SRC_DIR}/war/WEB-INF/web.xml'
  manifest='${SRC_DIR}/war/META-INF/MANIFEST.MF'>

<!--
重要 !WEB-INF/web.xml ファイルを WAR から削除する
このファイルは上記の webxml 属性によりすでに含まれている
-->
  <fileset dir="${SRC_DIR}/war" >
    <patternset >
      <include name="WEB-INF/weblogic.xml"/>
      <include name="**/*.html"/>
      <include name="**/*.gif"/>
    </patternset>
  </fileset>

<!--
重要 !ADK 設計時フレームワークをアダプタの設計時
Web アプリケーションに含める
-->
  <fileset dir="${WLI_HOME}/adapters/src/war" >
```

```

<patternset >
  <include name="**/*.css"/>
  <include name="**/*.html"/>
  <include name="**/*.gif"/>
  <include name="**/*.js"/>
</patternset>
</fileset>

<!--
設計時 UI をサポートするアダプタのクラスを含める
-->
<classes dir='${SRC_DIR}' includes='sample/web/*.class' />

<classes dir='${SRC_DIR}/war' includes='**/*.class' />
<classes dir='${WLI_HOME}/adapters/src/war'
  includes='**/*.class' />

<!--
アプリケーションで必要とされ、EAR に共有されないすべての JAR
ファイルを、WAR ファイルの WEB-INF/lib ディレクトリ下に含める
-->
<lib dir='${WLI_LIB_DIR}'
  includes='adk-web.jar,webtoolkit.jar,wlai-client.jar' />
</war>
</target>

```

この Ant ターゲットは、*PROJECT_ROOT/lib* ディレクトリ内に設計時インタフェース用に有効な WAR ファイルを作成します。このディレクトリで、*PROJECT_ROOT* は WebLogic Integration がインストールされている場所で、開発者はここにアダプタを構築します。たとえば、DBMS アダプタは、次の場所で構築されます。

WLI_HOME/adapters/DBMS

2. Web アプリケーションを WebLogic Server Administration を使用して WebLogic Server にロードします。
3. 開発環境のコンフィグレーションサンプル開発環境情報をコード リスト 8-3 に示します。

コード リスト 8-3 アダプタ開発ツリーの名前

```

<Application Deployed="true" Name="BEA_WLS_SAMPLE_ADK_Web"
  Path="WLI_HOME\adapters\PROJECT_ROOT\lib">

```

```
<WebAppComponent Name="BEA_WLS_SAMPLE_ADK_Web"
  ServletReloadCheckSecs="1" Targets="myserver" URI=
  "BEA_WLS_SAMPLE_ADK_Web" />
</Application>
```

アダプタ論理名およびディレクトリの値を以下のように設定します。

- a. `BEA_WLS_SAMPLE_ADK_Web` を開発対象のアダプタの論理名に置き換えます。
- b. `WLI_HOME` を、**WebLogic Integration** がインストールされているディレクトリのパス名に置き換えます。コード リスト 8-3 に示すように、`PROJECT_ROOT` を、アダプタ開発ツリーのトップレベル ディレクトリの名前に置き換えます。

注意： `GenerateAdapterTemplate` を実行すると、コード リスト 8-3 の情報が自動的に更新されます。`WLI_HOME/adapters/ADAPTER/src/overview.html` を開いて、この情報をコピーし、`config.xml` エントリに貼り付けることができます。

4. JSP を変更する場合、`src/war` ディレクトリ内で変更し、**WAR** ターゲットを再構築します。一時ディレクトリ中の **JSP** は変更しないでください。**WAR** ファイルが作成されると、これは、特定の **JSP** への変更のみを取り上げる **WebLogic Server** によって監視されるディレクトリにも抽出されます。**WebLogic Server** が実行する監視の間隔は、`WEB-INF/weblogic.xml` にある `pageCheckSeconds` パラメータによって指定します。コード リスト 8-4 にその方法を示します。

コード リスト 8-4 監視間隔の設定

```
<jsp-descriptor>
  <jsp-param>
    <param-name>compileCommand</param-name>
    <param-value>/jdk130/bin/javac.exe</param-value>
  </jsp-param>
  <jsp-param>
    <param-name>keepgenerated</param-name>
    <param-value>>true</param-value>
  </jsp-param>
  <jsp-param>
    <param-name>pageCheckSeconds</param-name>
```

```
<param-value>1</param-value>
</jsp-param>
<jsp-param>
  <param-name>verbose</param-name>
  <param-value>true</param-value>
</jsp-param>
</jsp-descriptor>
```

このアプローチにより、WAR ファイルが正しく構築されているかもテストされます。

手順 4 : 設計時 GUI の実装

Web アプリケーションで使用するフォームごとに、「設計時フォーム処理の概要」で説明する手順を実行するのは、単調であると同時に、エラーの原因になりやすいプロセスです。設計時フレームワークでは、「モデル/ビュー/コントローラ」パラダイムをサポートすることになり、このプロセスが簡略化されています。

設計時 GUI を実装するには、`DesignTimeRequestHandler` クラスを実装する必要があります。このクラスでは、フォームからのユーザ入力を受け入れ、設計時 GUI のアクションを実行します。このクラスを実装するには、ADK の `AbstractDesignTimeRequestHandler` を拡張する必要があります。このオブジェクトが提供するメソッドの詳細については、`DesignTimeRequestHandler` クラスの Javadoc を参照してください。

AbstractDesignTimeRequestHandler の拡張

`AbstractDesignTimeRequestHandler` により、WebLogic Server でアプリケーション ビューのデプロイ、編集、コピー、削除を行うためのユーティリティ クラスが提供されます。これにより、アプリケーション ビュー記述子にアクセスできます。アプリケーション ビュー記述子では、アプリケーション ビューに適

用する接続パラメータ、イベントリスト、サービスリスト、ログレベル、およびプールの設定が提供されます。[Application View Summary] ページには、各パラメータが表示されます。

ユーザレベルから見ると、AbstractDesignTimeRequestHandler により、アダプタ全体に共通するすべてのアクションに適用できる実装が提供されます。主なアクションは以下のとおりです。

- アプリケーションビューの定義

- 接続のコンフィグレーション

注意： ADK には、CCI 接続を設定するための接続パラメータを処理するメソッドはありますが、confconn.jsp ページは提供されません。このフォームの作成方法については、8-35 ページの「手順 5a : confconn.jsp フォームの作成」を参照してください。

- アプリケーションビューのデプロイ

- アプリケーションビューセキュリティの定義

- アプリケーションビューの編集

- アプリケーションビューのアンデプロイ

インクルードするメソッド

これらのアクションを正しく実行するには、AbstractDesignTimeRequestHandler を実装する際に、次のメソッドを指定する必要があります。

- `initServiceDescriptor();`

このメソッドにより、設計時にアプリケーションビューにサービスが追加されます (8-33 ページの「手順 4b : `initServiceDescriptor()` の実装」を参照してください)。

- `initEventDescriptor();`

このメソッドにより、設計時にアプリケーションビューにイベントが追加されます (8-34 ページの「手順 4c : `initEventDescriptor()` の実装」を参照してください)。

AbstractDesignTimeRequestHandler の具象実装のたびに、次の 2 つのメソッドを指定する必要があります。

- `protected String getAdapterLogicalName();`
このメソッドにより、アダプタの論理名が返されます。これは、アプリケーションビューをその名前でデプロイする際に使用されます。
- `protected Class getManagedConnectionFactoryClass();`
このメソッドは、アダプタに対する SPI の `ManagedConnectionFactory` 実装クラスを返します。

手順 4a : `ManagedConnectionFactory` クラスの指定

`ManagedConnectionFactory` クラスを指定するには、次のメソッドを実装する必要があります。

```
protected Class getManagedConnectionFactoryClass();
```

このメソッドはアダプタに対する SPI の `ManagedConnectionFactory` 実装クラスを返します。このクラスは、`AbstractManagedConnectionFactory` が EIS への接続を試みる場合に必要です。

手順 4b : `initServiceDescriptor()` の実装

サービスアダプタについて、アダプタのユーザが設計時にサービスを追加できるように、`initServiceDescriptor()` を実装する必要があります。このメソッドは、コード リスト 8-5 のように実装します。

コード リスト 8-5 `initServiceDescriptor()` の実装

```
protected abstract void initServiceDescriptor(ActionResult result,  
                                             IServiceDescriptor sd,  
                                             HttpServletRequest request)  
    throws Exception
```

このメソッドは、`AbstractDesignTimeRequestHandler` の `addservc()` の実装によって呼び出されます。このメソッドには、`IServiceDescriptor` パラメータに関連付けられた EIS 固有の情報を初期化する役割があります。`addservc()` の基本クラスが、エラー処理などを実行します。ユーザが `addservc JSP` を発行すると、`addservc()` メソッドが呼び出されます。

手順 4c : `initEventDescriptor()` の実装

イベントアダプタは、アダプタのユーザが設計時にイベントを追加できるように、`initEventDescriptor()` を実装する必要があります。このメソッドは、コードリスト 8-6 のように実装します。

コード リスト 8-6 `initEventDescriptor()` の実装

```
protected abstract void
    initEventDescriptor(ActionResult result,
                       IEventDescriptor ed,
                       HttpServletRequest request)
    throws Exception;
```

このメソッドは、`AbstractDesignTimeRequestHandler` の `addevent()` の実装によって呼び出されます。このメソッドには、`IServiceDescriptor` パラメータに関連付けられた EIS 固有の情報を初期化する役割があります。`addevent()` の基本クラスを実装することによりエラー処理などのコンセプトに対応できます。ユーザが `addevent JSP` を発行すると、`addevent()` メソッドが呼び出されます。`addevent` には共通のロジックが含まれていることに加えて EIS 固有のロジックが `initEventDescriptor()` に委託されるので、オーバーライドしないようにしてください。

注意： サービス記述子にプロパティを追加する場合は、`Bean` 属性の命名規約に従って名前を割り当てる必要があります。これに従っていない場合、サービス記述子では、`InteractionSpec` は正しく更新されません。

手順 5 : HTML フォームの作成

設計時 GUI を実装する際の最後の手順は、インタフェースを構成する各種のフォームを作成することです。フォームの作成に習熟するために、以下の節を参照してください。

- 必要なフォームのリストと詳しい説明については、8-10 ページの「Java Server Pages」を参照してください。
- 各フォームの具体的な説明については、8-20 ページの「手順 2 : ページフローの定義」を参照してください。

以下の節では、これらのフォームのコードの作成方法について説明します。フォームのサンプルコードもあります。

手順 5a : confconn.jsp フォームの作成

このページでは、ユーザが EIS で接続パラメータを指定するための HTML フォームを示します。このページには、アダプタの設計時 Web アプリケーションを表示する必要があります。このフォームは `doAction=confconn` によって `ControllerServlet` にポストされます。すなわち、設計時インタフェースの `RequestHandler` では次のメソッドを指定する必要があります。

```
public ActionResult confconn(HttpServletRequest request) throws  
    例外
```

このメソッドを実装すると、指定した接続パラメータにもとづいてアダプタの `ManagedConnectionFactory` の新しいインスタンスを作成できます。`ManagedConnectionFactory` により、EIS への接続を設定するための `CCI ConnectionFactory` が指定されます。したがって、発行された `confconn` フォームを処理することにより EIS に接続するために必要なパラメータが正しく指定されているかどうかを検証されます。

コード リスト 8-7 にサンプルアダプタの `confconn` フォームを示します。

コード リスト 8-7 confconn.jsp のコーディング

```
1 <%@ taglib uri='/WEB-INF/taglibs/adk.tld' prefix='adk' %>
2 <form method='POST' action='controller'>
3   <table>
4     <tr>
5       <td><adk:label name='userName' required='true' /></td>
6       <td><adk:text name='userName' maxlength='30' size='8' /></td>
7     </tr>
8     <tr>
9       <td><adk:label name='password' required='true' /></td>
10      <td><adk:password name='password' maxlength='30' size='8' /></td>
11    </tr>
12    <tr>
13      <td colspan='2'><adk:submit name='confconn_submit'
14        doAction='confconn' /></td>
15    </tr>
16 </form>
```

コード リスト 8-7 の内容を以下の節に示します。

- ADK タグ ライブラリのインクルード
- ControllerServlet のポスト
- [Form] フィールドのラベルの表示
- テキスト フィールドのサイズの表示
- フォームの [Submit] ボタンの表示
- confconn() の実装

ADK タグ ライブラリのインクルード

コード リスト 8-7 のライン 1 では、JSP エンジンに ADK タグ ライブラリをインクルードするように指定しています。

```
<%@ taglib uri='/WEB-INF/taglibs/adk.tld' prefix='adk' %>
```

ADK が提供するタグを表 8-3 に示します。

ControllerServlet のポスト

コードリスト 8-7 のライン 2 では、フォームが ControllerServlet にポストされます。

```
<form method='POST' action='controller'>
```

ControllerServlet は、Web アプリケーションに合わせて web.xml ファイルでコンフィグレーションされます。これは、RequestHandler で実行するメソッドに HTTP 要求を委託します。ControllerServlet を使用するのに、いずれのコードも指定する必要はありません。ただし、表 8-5 にリストされた初期パラメータは指定する必要があります。

表 8-5 ControllerServlet の初期パラメータ

パラメータ	説明
MessageBundleBase	アダプタで使用するすべてのメッセージバンドルに基本名を指定する。ADK は、サンプルアダプタには常に論理名を使用する。ただし、メッセージバンドルには独自の命名規約を選択できる。なお、このプロパティは、ra.xml でも設定できる。
DisplayPage	アプリケーションのページのフローとルック & フィールの両方をコントロールする JSP の名前を指定する。サンプルアダプタでは、このページで display.jsp が表示される。
LogConfigFile	アダプタの log4j コンフィグレーションファイルを指定する。
RootLogContext	ルート ログの内容を指定する。ログの内容を利用して、プログラムの各モジュールに従い、ログメッセージをカテゴリ別に分類できる。ADK では、特定のアダプタのメッセージがすべて、特定のカテゴリ別に分類されるように、ルート ログの内容にアダプタの論理名を使用する。
RequestHandlerClass	アダプタの要求ハンドラ クラスに完全修飾名を付ける。サンプルアダプタの場合、この値は sample.web.DesignTimeRequestHandler である。

[Form] フィールドのラベルの表示

コード リスト 8-7 のライン 5 では、フォーム中のフィールドのラベルを表示します。

```
<adk:label name='userName' required='true' />
```

表示する値は、ユーザのメッセージバンドルから検索します。required 属性は、ユーザのパラメータ指定が必須であるかどうかを示します。

テキスト フィールドのサイズの表示

コード リスト 8-7 のライン 6 では、テキスト フィールドにサイズ 8、最大長 30 が設定されます。

```
<adk:text name='userName' maxlength='30' size='8' />
```

フォームの [Submit] ボタンの表示

コード リスト 8-7 のライン 13 では、アダプタ ユーザが入力を発行するためのフォームにボタンが表示されます。

```
<adk:submit name='confconn_submit' doAction='confconn' />
```

ボタンのラベルは、confconn_submit を使用してメッセージバンドルから検索されます。フォーム データを発行すると、ControllerServlet が、登録済みのリクエスト ハンドラ（「RequestHandlerClass プロパティ」を参照）で confconn メソッドを検索し、リクエスト データをリクエスト ハンドラに送ります。

confconn() の実装

AbstractDesignTimeRequestHandler により、confconn() メソッドが実装されます。このメソッドの実装によって、Java Reflection API を利用して、アダプタの ManagedConnectionFactory インスタンスのセッター メソッドに、ユーザが指定した接続パラメータがマップされます。アダプタの ManagedConnectionFactory に具象なクラスを指定するだけで済むようになります。このクラスを指定するには、次のメソッドを実装します。

```
public Class getManagedConnectionFactoryClass()
```

手順 5b : addevent.jsp フォームの作成

このフォームを使用してアプリケーション ビューに新しいイベントを追加できます。このフォームは EIS 固有のもので、コード リスト 8-8 はサンプル アダプタの addevent.jsp フォームを示しています。

コード リスト 8-8 addevent.jsp フォームを作成するサンプル コード

```
1 <%@ taglib uri='/WEB-INF/taglibs/adk.tld' prefix='adk' %>
2 <form method='POST' action='controller'>
3     <table>
4         <tr>
5             <td><adk:label name='password' required='true' /></td>
6             <td><adk:text name='eventName' maxlength='100' size='50' /></td>
7         </tr>
8         <tr>
9             <td colspan='2'><adk:submit name='addevent_submit'
10                doAction='addevent' /></td>
11         </tr>
12     </table>
</form>
```

以下の節に、addevent.jsp の内容を説明します。

ADK タグ ライブラリのインクルード

コード リスト 8-8 のライン 1 では、JSP エンジンに ADK タグ ライブラリをインクルードするように指定しています。

```
<%@ taglib uri='/WEB-INF/taglibs/adk.tld' prefix='adk'%>
```

ADK が提供するタグを表 8-3 に示します。

ControllerServlet のポスト

コード リスト 8-8 のライン 2 では、フォームが ControllerServlet にポストされます。

```
<form method='POST' action='controller'>
```

ControllerServlet は、Web アプリケーションに合わせて web.xml ファイルでコンフィグレーションされます。これは、RequestHandler で実行するメソッドに HTTP 要求を委託します。ControllerServlet を使用するのに、いずれのコードも指定する必要はありません。ただし、表 8-5 「ControllerServlet パラメータ」にリストされた初期パラメータは指定する必要があります。

[Form] フィールド ラベルの表示

コード リスト 8-8 のライン 5 では、フォーム中のフィールドのラベルを表示します。

```
<adk:label name='eventName' required='true' />
```

表示する値は、ユーザのメッセージバンドルから検索します。required 属性は、ユーザのパラメータ指定が必須であるかどうかを示します。

テキスト フィールドのサイズの表示

コード リスト 8-8 のライン 6 では、テキスト フィールドにサイズ 50、最大長 100 が設定されます。

```
<adk:text name='eventName' maxlength='100' size='50' />
```

フォームの [Submit] ボタンの表示

コード リスト 8-8 のライン 9 では、アダプタ ユーザが入力を発行するためのフォームにボタンが表示されます。

```
<adk:submit name='addevent_submit' doAction='addevent' />
```

ボタンのラベルは、addevent_submit を使用してメッセージバンドルから検索されます。フォーム データを発行すると、ControllerServlet が、登録済みのリクエスト ハンドラ（「RequestHandlerClass プロパティ」を参照）で addevent() メソッドを検索し、リクエスト データをリクエスト ハンドラに送ります。

フィールドの追加

イベントを定義するときに必要なその他のフィールドを追加する必要があります。複数のフィールドがあるフォームの例については、附録 E 「DBMS サンプルアダプタを使用したアダプタ開発方法の学習」を参照してください。

手順 5c : addservc.jsp フォームの作成

このフォームを使用して、アプリケーションビューに新しいサービスを追加できます。このフォームは EIS 固有のもので、コード リスト 8-9 にサンプルアダプタの addservc.jsp フォームを示します。

コード リスト 8-9 addservc.jsp のコーディング

```
1 <%@ taglib uri='/WEB-INF/taglibs/adk.tld' prefix='adk' %>
2 <form method='POST' action='controller'>
3     <table>
4         <tr>
5             <td><adk:label name='serviceName' required='true' /></td>
6             <td><adk:text name='serviceName' maxlength='100' size='50' /></td>
7         </tr>
8         <tr>
9             <td colspan='2'><adk:submit name='addservc_submit'
10                doAction='addservc' /></td>
11         </tr>
12 </table>
</form>
```

ADK タグ ライブラリのインクルード

コード リスト 8-9 のライン 1 では、JSP エンジンに ADK タグ ライブラリをインクルードするように指定しています。

```
<%@ taglib uri='/WEB-INF/taglibs/adk.tld' prefix='adk' %>
```

タグ ライブラリでは、ADK で提供される使いやすいフォーム検証機能がサポートされます。ADK タグ ライブラリには、表 8-3 に示すタグが用意されています。

ControllerServlet のポスト

コード リスト 8-9 のライン 2 では、フォームが ControllerServlet にポストされます。

```
<form method='POST' action='controller'>
```

ControllerServlet は、Web アプリケーションに合わせて web.xml ファイルでコンフィグレーションされます。これは、RequestHandler で実行するメソッドに HTTP 要求を委託します。ControllerServlet を使用するのに、いずれのコードも指定する必要はありません。ただし、表 8-5 「ControllerServlet パラメータ」にリストされた初期パラメータは指定する必要があります。

[Form] フィールド ラベルの表示

コード リスト 8-9 のライン 5 では、フィールドのラベルを表示します。

```
<adk:label name='serviceName' required='true' />
```

表示する値は、ユーザのメッセージバンドルから検索します。required 属性は、ユーザのパラメータ指定が必須であるかどうかを示します。

テキスト フィールドのサイズの表示

コード リスト 8-9 のライン 6 では、テキスト フィールドにサイズ 50、最大長 100 が設定されます。

```
<adk:text name='serviceName' maxlength='100' size='50' />
```

フォームの [Submit] ボタンの表示

コード リスト 8-9 のライン 9 では、アダプタ ユーザが入力を発行するためのフォームにボタンが表示されます。

```
<adk:submit name='addservc_submit' doAction='addservc' />
```


ボタンのラベルは、[addservc_submit] を使用してメッセージバンドルから検索されます。フォーム データを発行すると、ControllerServlet が、登録済みのリクエスト ハンドラ（「RequestHandlerClass プロパティ」を参照）で addservc メソッドを検索し、リクエスト データをリクエスト ハンドラに送ります。

フィールドの追加

サービスを定義するときに必要なその他のフィールドを追加する必要があります。複数フィールドのあるフォームの例については、附録 E 「DBMS サンプルアダプタを使用したアダプタ開発方法の学習」を参照してください。

手順 5d : イベントおよびサービスの編集機能の実装（省略可能）

設計時にユーザに対しイベントやサービスを編集する機能を許可する場合は、アダプタ プロパティを編集し、edtservc.jsp と edtevent.jsp のフォームを作成して、具体的なメソッドを実装する必要があります。ここでは、以下のタスクについて説明します。

注意： この手順は省略可能です。ユーザにこれらの機能を提供する必要はありません。

アダプタ プロパティ ファイルの更新

まず、アダプタ プロパティ ファイルを次のように変更してサンプル アダプタに合わせてシステム プロパティを更新します。

- 以下のプロパティを追加します。
 - edtservc_title=Edit Service
 - edtservc_description=On this page, you edit service properties.
 - edtevent_description=On this page, you edit event properties.edtevent_title=Edit Event
 - glossary_description=This page provides definitions for commonly used terms.

- service_submit_add=Add
 - service_label_serviceDesc=Description:
 - service_submit_edit=Edit
 - service_label_serviceName=Unique Service Name:
 - event_submit_add=Add
 - event_label_eventDesc=Description:
 - event_label_eventName=Unique Event Name:
 - event_submit_edit=Edit
 - eventLst_label_edit=Edit
 - serviceLst_label_edit=Edit
 - event_does_not_exist=Event {0} does not exist in application view {1}.
 - service_does_not_exist=Service {0} does not exist in Application View {1}.
 - no_write_access={0} does not have write access to the Application View.
- 以下のプロパティを削除します。
- addservc_submit_add=Add
 - addevent_label_eventDesc=Description:
 - addservc_label_serviceName=Unique Service Name:
 - addevent_submit_add=Add
 - pingTable_invalid=The ping table cannot be reached. Please enter a valid table in the existing database to ping.
 - pingTable=Ping Table
 - addevent_label_eventName=Unique Event Name:
 - addservc_label_serviceDesc=Description:

アダプタプロパティファイルを更新した後、新しいファイルを元のファイルと比較し、同期されていることを確認します。

edtservc.jsp と addservc.jsp の作成

これらの Java サーバ ページは編集機能を提供するために呼び出されるものです。編集 JSP ファイルと追加 JSP ファイルの主な違いは、記述子の値のロード方法です。編集 JSP ファイルは、既存の記述子の値をロードします。このため、DBMS サンプルアダプタでの編集および追加には、同じ HTML ファイルを使用します。

これらの HTML ファイルは各 JSP ページに静的にインクルードされます。これにより、JSP/HTML とプロパティを複製しなくても済むようになります。記述子の値は編集ページに表示するコントロールにマップされます。ここから、いずれの変更も発行できます。

記述子に定義された値でコントロールを初期化するには、`AbstractDesignTimeRequestHandler` で

`loadEvent/ServiceDescriptorProperties()` メソッドを呼び出します。このメソッドにより、サービスのプロパティがすべて、`RequestHandler` に設定されます。これらの値を設定すると、`RequestHandler` により JSP ファイルで使用する ADK コントロールにそれぞれの値がマップされます。

`loadEvent/ServiceDescriptorProperties()` をデフォルトの設定で実装すると、ADK タグに対応するプロパティ名を使用して、記述子の値をマップします。ADK タグ名以外の値を使用してサービスやイベントのプロパティをマップすると、これらの値をオーバーライドして ADK タグ名マッピングに記述子を送ります。

また、HTML を解決する前に、`RequestHandler` を初期化する必要があります。この初期化は、1 回だけ実行します。コード リスト 8-10 に `edtevent.jsp` のロード時に使用するコード例を示します。

コード リスト 8-10 `edtevent.jsp` のロード時に使用するサンプル コード

```
if(request.getParameter("eventName") != null){
handler.loadEventDescriptorProperties(request);
}
```

`edtservc.jsp` ファイルを、`edtservc` に発行する必要があります。

```
<adk:submit name='edtservc_submit' doAction='edtservc' />
```

edtevent.jsp ファイルを、edtevent に発行する必要があります。

```
<adk:submit name='edtevent_submit' doAction='edtevent' />
```

以下の場所の DBMS サンプル アダプタに例を示します。

```
WLI_HOME/adapters/dbms/src/war
```

メソッドの実装

最後に、表 8-6 に示されているメソッドを実装します。

表 8-6 edtservc.jsp および edtevent.jsp で実装するメソッド

メソッド	説明
loadServiceDescriptorProperties および loadEventDescriptorProperties	これらのメソッドにより、RequestHandler に ADK のタグ値マッピング機能が提供される。開発者が同じ値を使用して ADK タグに名前を付け、サービス / イベント記述子をロードする場合、マッピングはフリーになる。これ以外の場合は、開発者が自分の DesignTimeRequestHandler をオーバーライドして、これらのマッピングを行う必要がある。
boolean supportsEditableServices() および boolean supportsEditableEvents()	これらのメソッドはマーカーとして使用する。これらのメソッドが返す値が true の場合は、[Application View Administration] ページに編集リンクが表示される。DesignTimeRequestHandler のオーバーライドがサポートされている。
editServiceDescriptor および editEventDescriptor	これらのメソッドを使用して、編集済みのサービスデータまたはイベントデータの永続性を設定する。これらのメソッドにより、要求によって ADK タグ値が抽出され、これがサービス記述子、またはイベント記述子に追加される。また、これらのメソッドではイベントまたはサービスに関連するスキーマを特別に処理する。スキーマの変更が必要な場合は、ここで更新する必要がある。要求から値を読み取った後で、これがなくなったら場合は、RequestHandler から削除する必要がある。

各メソッドの実装例については、サンプル アダプタを参照してください。

手順 5e : Web アプリケーションのデプロイメント 記述子 (WEB-INF/web.xml) の記述

アダプタに対して WEB-INF/web.xml Web アプリケーションを作成する必要があります。GenerateAdapterTemplate を使用してサンプルアダプタからアダプタを複製する場合は、新しいアダプタの web.xml ファイルが自動的に生成されます。

以下のコード リスト (コード リスト 8-11 から コード リスト 8-15) は、このファイルの重要コンポーネントです。

コード リスト 8-11 web.xml サブレット コンポーネント

```
<servlet>
  <servlet-name>controller</servlet-name>
  <servlet-class>com.bea.web.ControllerServlet</servlet-class>
  <init-param>
    <param-name>MessageBundleBase</param-name>
    <param-value>BEA_WLS_SAMPLE_ADK</param-value>
    <description>The base name for the message bundles
      for this adapter. The ControllerServlet uses this
      name and the user's locale information to
      determine which message bundle to use to
      display the HTML pages.</description>
  </init-param>

  <init-param>
    <param-name>DisplayPage</param-name>
    <param-value>display.jsp</param-value>
    <description>The name of the JSP page
      that includes content pages and provides
      the look-and-feel template. The ControllerServlet
      redirects to this page to let it determine what to
      show the user.</description>
  </init-param>

  <init-param>
    <param-name>LogConfigFile</param-name>
    <param-value>BEA_WLS_SAMPLE_ADK.xml</param-value>
    <description>The name of the sample adapter's
      LOG4J configuration file.</description>
  </init-param>
</servlet>
```

```
<init-param>
  <param-name>RootLogContext</param-name>
  <param-value>BEA_WLS_SAMPLE_ADK</param-value>
  <description>The root category for log messages
    for the sample adapter. All log messages created
    by the sample adapter will have a context starting
    with this value.</description>
</init-param>

<init-param>
  <param-name>RequestHandlerClass</param-name>
  <param-value>sample.web.DesignTimeRequestHandler
</param-value>
  <description>Class that handles design
    time requests</description>
</init-param>

<init-param>
  <param-name>Debug</param-name>
  <param-value>on</param-value>
  <description>Debug setting (on|off, off is
    default)</description>
</init-param>

  <load-on-startup>1</load-on-startup>
</servlet>
```

コード リスト 8-12 に示すこのコンポーネントでは、ControllerServlet が controller という名前にマップされます。このアクションには、ADK JSP フォームで ControllerServlet が controller という論理名にマップされるという前提に基づくため、重要な意味があります。

コード リスト 8-12 web.xml ControllerServlet マッピング コンポーネント

```
<servlet-mapping>
  <servlet-name>controller</servlet-name>
  <url-pattern>controller</url-pattern>
</servlet-mapping>
```

コード リスト 8-13 に示すこのコンポーネントでは ADK タグ ライブラリを宣言します。

コード リスト 8-13 web.xml ADK タグ ライブラリ コンポーネント

```
<taglib>
  <taglib-uri>adk</taglib-uri>
  <taglib-location>/WEB-INF/taglibs/adk.tld</taglib-location>
</taglib>
```

コード リスト 8-14 に示すこのコンポーネントでは Web アプリケーションに対するセキュリティ制約を宣言します。前のリリースでは、ユーザはアダプタグループに属している必要がありました。リリース 7.0 以降では、ユーザは Administrators グループに属している必要があります (コード リスト 8-14 およびコード リスト 8-15 のロール名を参照してください)。これは、デプロイメントに、ユーザが Administrators グループに属することが必須の MBeans へのアクセスが必要となるためです。

コード リスト 8-14 web.xml セキュリティ制約のコンポーネント

```
<Security-constraint>
  <web-resource-collection>
    <web-resource-name>AdapterSecurity</web-resource-name>
    <url-pattern>*.jsp</url-pattern>
  </web-resource-collection>

  <auth-constraint>
    <role-name>Administrators</role-name>
  </auth-constraint>

  <user-data-constraint>
    <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

コード リスト 8-15 に示すこのコンポーネントでは、ログイン コンフィグレーションを宣言します。

コード リスト 8-15 web.xml ログイン コンフィグレーションのコンポーネント

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>default</realm-name>
```

```
<form-login-config>
  <form-login-page>/login.jsp</form-login-page>
  <form-error-page>/login.jsp?error</form-error-page>
</form-login-config>

</login-config>

<security-role>
  <role-name>Administrators</role-name>
</security-role>
```

手順 6 : ルック & フィールの実装

設計時 GUI の開発時に従うべき重要なプログラミング手法として、アプリケーションビューで使用するすべてのページで一貫したルック & フィールを実装する必要があります。ルック & フィールは `display.jsp` によって決定されます。このページは ADK に組み込まれており、設計時 Web アプリケーションに以下の利点をもたらします。

- すべてのページのルック & フィールを決定するテンプレートを設定します。
- content HTTP 要求パラメータに基づいてその他の JSP を組み込みます。content HTTP 要求パラメータがない場合は、`display.jsp` ファイルに `main.jsp` を組み込む必要があります。
- Java 例外のエラー ページを ADK の `error.jsp` として登録します。

全体のページを通じて一貫したルック & フィールを実装するには、次の手順を実行します。

1. サンプルアダプタの `display.jsp` 基準として使用します。例については、`WLI_HOME/adapters/sample/src/war/WEB-INF/web.xml` を参照してください。
2. HTML を使用して、独自のルック & フィールか社内の ID 基準に従って、ルック & フィール マークアップを変更します。
3. HTML マークアップのいずれかの場所に、次の行を追加します。

```
<%pageContext.include(sbPage.toString());%>
```


このコードは、別のページを組み込むときに使用するカスタム JSP タグです。このタグでは、JSP スクリプトレット `sbPage.toString()` を使用して、HTML または JSP ページを表示ページに含めます。`sbPage.toString()` は、実行時 `content` (HTTP 要求パラメータ) の値の妥当性を検証します。

手順 7: サンプル アダプタの設計時インタフェースのテスト

WebLogic Integration では、サンプルアダプタの設計時インタフェースの基本的な機能を検証するための、テストドライバが用意されています。このテストドライバは、<http://www.httpunit.org> にあるテスト用 Web インタフェースのフレームワークである HTTP ユニットを基準に作成されています。HTTP ユニットは、JUnit テスト フレームワーク (<http://www.junit.org> にあるフレームワーク) に関連しています。HTTP ユニットと JUnit の両バージョンとも WebLogic Integration の中に組み込まれています。

テストドライバではいくつかのテストが実行されます。これによりアプリケーション ビューが作成されてイベントとサービスの両方がアプリケーション ビューに追加され、アプリケーション ビューにデプロイ/アンデプロイを実行し、イベントとサービスの両方についてテストを実行します。正常に実行が終了すると、テストドライバはすべてのアプリケーション ビューを削除します。

ファイルとクラス

すべてのテスト ケースが、`DesignTimeTestCase` クラス、または対応する親クラス (`AdapterDesignTimeTestCase`) で有効です。`DesignTimeTestCase` クラス (`sample.web` パッケージ、および

`WLI_HOME/adapters/sample/src/sample/web` フォルダにあります) には、サンプルアダプタに固有のテストが組み込まれています。

`AdapterDesignTimeTestCase` (`com.bea.adapter.web` パッケージ、および `WLI_HOME/lib/adk-web.jar` ファイルにあります) には、すべてのアダプタと一部の便利なメソッドに適用されるテストが組み込まれています。

テストの実行

設計時インタフェースをテストするには、次の一連の手順を実行します。

1. サンプルアダプタをデプロイして、**WebLogic Server** を起動します。次に、現在の作業フォルダを具体的なプロジェクト フォルダに設定して、以下の手順に示すとおり `setenv` コマンドを実行します。

2. `WLI_HOME` に移動し、コマンド プロンプトで `setenv` と入力します。

`setenv` コマンドにより、手順 3 の実行に必要な環境が設定されます。

3. コマンド プロンプトで次のように入力して、サンプルアダプタの **Web** フォルダに移動します。

```
cd WLI_HOME/adapters/sample/project
```

4. `designTimeTestCase.properties` ファイルを編集します。実行するテストケースのリストを含む行に、`web.DesignTimeTestCase` を追加します。次のような行になります。

```
test.case=web.DesignTimeTestCase
```

5. ファイルの終わり付近で、ユーザ名とパスワードの 2 つのエントリの値の変更が必要になる場合があります。テスト ドライバが、**WebLogic Integration** に接続する際に必要となるユーザ名とパスワードを指定します。

6. `test.properties` ファイルを編集した後、**WebLogic Server** を起動します。

7. コマンド プロンプトで次のようにコマンドを入力してテストを実行します。

```
ant designtimetest
```

9 アダプタのデプロイ

アダプタを作成した後は、エンタープライズアーカイブ (EAR) ファイルを使用して、これをデプロイする必要があります。EAR ファイルを使用すると、すべてのアダプタ コンポーネントを 1 度にデプロイできるため、このタスクを効率的に行えます。EAR ファイルのデプロイは、WebLogic Server Administration Console から実行できます。

この章の内容は以下のとおりです。

- エンタープライズアーカイブ (EAR) ファイルの使用
- WebLogic Server Administration Console を使用したアダプタのデプロイ
- Web アプリケーションのデプロイメント記述子の編集

エンタープライズアーカイブ (EAR) ファイルの使用

各アダプタは、1 つのエンタープライズアーカイブ (EAR) ファイルを使用してデプロイされます。EAR ファイルには、デプロイに必要な設定時 Web アプリケーションの WAR ファイル、アダプタ RAR ファイル、アダプタ JAR ファイルおよび共有の JAR ファイルが含まれています。この EAR ファイルは、コードリスト 9-1 のような構造にします。

コードリスト 9-1 EAR ファイル構造

```
adapter.ear
  application.xml
  sharedJar.jar
  adapter.jar
  adapter.rar
  META-INF
    ra.xml
```

```
        weblogic-ra.xml
        MANIFEST.MF
designotime.war
    WEB-INF
        web.xml
    META-INF
        MANIFEST.MF
```

サンプルアダプタの EAR ファイルをコードリスト 9-2 に示します。

コード リスト 9-2 サンプルアダプタの EAR ファイル

```
sample.ear
  application.xml
  adk.jar (shared .jar between .war and .rar)
  bea.jar (shared .jar between .war and .rar)
  BEA_WLS_SAMPLE_ADK.jar (shared .jar between .war and .rar)

  BEA_WLS_SAMPLE_ADK.war (Web application with
    META-INF/MANIFEST.MF entry Class-Path:
    BEA_WLS_SAMPLE_ADK.jar adk.jar bea.jar log4j.jar
    logtoolkit.jar xcci.jar xmltoolkit.jar)

  BEA_WLS_SAMPLE_ADK.rar (Resource Adapter with
    META-INF/MANIFEST.MF entry Class-Path:
    BEA_WLS_SAMPLE_ADK.jar adk.jar bea.jar log4j.jar
    logtoolkit.jar xcci.jar xmltoolkit.jar)

  log4j.jar (shared .jar between .war and .rar)
  logtoolkit.jar (shared .jar between .war and .rar)
  xcci.jar (shared .jar between .war and .rar)
  xmltoolkit.jar (shared .jar between .war and .rar)
```

RAR ファイルと WAR ファイルのいずれも共有 JAR ファイルは含んでおらず、`<manifest.classpath>` 属性を使用して共有 JAR ファイルを参照します。

EAR ファイルにおける共有 JAR ファイルの使い方

設計時アプリケーションは、アダプタの SPI クラスを非管理対象のシナリオで使用します。そのため、アダプタの SPI および CCI クラスは、EAR ファイルと同じディレクトリにある共有 JAR ファイルに含まれます。WAR および RAR クラ

スローダから共有 JAR ファイル中のクラスにアクセスできるようにするには、MANIFEST.MF ファイルで、共有 EAR ファイルのインクルードを要求する必要があります。MANIFEST.FM の詳細については、6-11 ページの「Manifest ファイル」または次の URL で参照してください。

<http://developer.java.sun.com/developer/Books/JAR/basics/manifest.html>

BEA_WLS_SAMPLE_ADK.rar および BEA_WLS_SAMPLE_ADK.war には、コード リスト 9-3 のような META-INF/MANIFEST.MF が含まれています。

コード リスト 9-3 Manifest ファイルの例

```
Manifest-Version: 1.0
Created-By: BEA Systems, Inc.
Class-Path: BEA_WLS_SAMPLE_ADK.jar adk.jar wlai-core.jar
            wlai-client.jar
```

注意: ファイル名 MANIFEST.MF は、すべて大文字で表記します。正しく表記されていない場合、UNIX システムでは認識されず、エラーが発生します。

EAR ファイルのデプロイメント記述子

コード リスト 9-4 に、EAR ファイルのコンポーネントを宣言するデプロイメント記述子を示します。この場合、これらのコンポーネントには設計時 WAR およびアダプタ RAR モジュールが含まれます。

コード リスト 9-4 EAR ファイルのデプロイメント記述子

```
<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE
Application 1.3//EN"
'http://java.sun.com/dtd/application_1_3.dtd'>
<application>
  <display-name>BEA_WLS_SAMPLE_ADK</display-name>
  <description>This is a J2EE application that contains a sample
    connector and Web application for configuring
    application views for the adapter.</description>
```

```
<module>
  <connector>BEA_WLS_SAMPLE_ADK.rar</connector>
</module>
<module>
  <web>
    <web-uri>BEA_WLS_SAMPLE_ADK.war</web-uri>
    <context-root>BEA_WLS_SAMPLE_ADK_Web</context-root>
  </web>
</module>
</application>
```

アダプタのデプロイは、WebLogic Server Administration Console から実行できます。この手順については、「WebLogic Server Administration Console を使用したアダプタのデプロイ」で説明しています。

WebLogic Server Administration Console を使用したアダプタのデプロイ

WebLogic Server Administration Console を使用して、アダプタをコンフィグレーションまたはデプロイする手順は次のとおりです。

1. WebLogic Server Administration Console を起動します。
2. ナビゲーションツリー（左ペイン）で、[デプロイメント | アプリケーション] を選択します。
[アプリケーション] ページが表示されます。
3. [新しい Application のコンフィグレーション] を選択します。
[新しい Application のコンフィグレーション] ページが表示されます。
4. 以下のフィールドに値を入力します。
 - [名前] フィールドにアダプタの論理名を入力します。
 - [パス] フィールドで、適切な EAR ファイルのパスを入力します。
 - [デプロイ済み] フィールドで、チェック ボックスが選択されていることを確認します。
5. [適用] をクリックして、新しいエントリを作成します。

6. [コンポーネントの構成] を選択します。
7. 各コンポーネントに個別のターゲットを設定します。

WebLogic Server Administration Console からアプリケーション（またはアプリケーション コンポーネント）をインストールすると、該当するドメインのコンフィグレーション ファイル（/config/*DOMAIN_NAME*/config.xml -ここで *DOMAIN_NAME* はドメイン名）にもそのアプリケーションまたはコンポーネントのエントリが作成されます。また、アプリケーションとアプリケーション コンポーネントをコンフィグレーションしモニタするための JMX Management Beans (MBeans) も生成されます。

アダプタの自動登録

WebLogic Integration では、アダプタのデプロイ時に、自動登録プロセスが実行されます。自動登録は、アダプタのデプロイメント フェーズ中に実行されます。このプロセスは、以下の 2 つのうちいずれかの方法で呼び出せます。

- 命名規約の使用
- テキスト ファイルの使用

命名規約の使用

設計時 Web アプリケーションとコネクタのデプロイメントには、命名規約を使用することをお勧めします。

EAR ファイルを WebLogic Integration 環境でデプロイする際、アダプタの論理名をファイル名として使用し、config.xml でファイルを定義します。コード リスト 9-5 で、その例を示します。

コード リスト 9-5 config.xml ファイルにアダプタの論理名を追加

```
<Application Deployed="true" Name="ALN"  
  Path="WLI_HOME/adapters/ADAPTER/lib/ALN.ear">  
<ConnectorComponent Name="ALN" Targets="myserver"  
  URI="ALN.rar"/>
```

```
<WebAppComponent Name="ALN_EventRouter" Targets="myserver"
  URI="ALN_EventRouter.war" />
<WebAppComponent Name="ALN_Web" Targets="myserver"
  URI="ALN_Web.war" />
</Application>
```

リストでは、*ALN* フィールドにアダプタの論理名を入力します。この名前を、`<ConnectorComponent>` 要素の `Name` 属性の値として使用してください。

設計時 **Web** アプリケーションのデプロイメントに *ALN_Web* という名前を割り当てると、デプロイメントの際に、設計時 **Web** アプリケーションが自動的に **Application View Management Console** に登録されます。この命名規約は、**DBMS** およびサンプルアダプタで使用されます。

テキスト ファイルの使用

あるいは、`webcontext.txt` という名前のテキスト ファイルを **EAR** ファイルのパス名のルート ディレクトリにインクルードできます。`webcontext.txt` ファイルには、アダプタの設計時 **Web** アプリケーションのコンテキストが含まれます。このファイルは、**UTF-8** でエンコーディングする必要があります。

Web アプリケーションのデプロイメント記述子の編集

いくつかのアダプタに対し、イベント ルータの **Web** アプリケーションで使用されるデプロイメント パラメータを変更しなければならない場合があります。たとえば、**DBMS** アダプタの場合、対応するイベント ジェネレータで使用されるデータ ソースの変更が必要になります。

この節では、**WebLogic Server Administration Console** の **Deployment Descriptor Editor** を使用して、以下の **Web** アプリケーション デプロイメント記述子を編集する方法について説明します。

- `web.xml`

- `weblogic.xml`

デプロイメント パラメータ

イベント ルータ サーブレットのパラメータは、どれでも変更できます。以下のパラメータがあります。

- `eventGeneratorClassName`
- `userID`
- `password`
- `dataSource`
- `jdbcDriverClassName`
- `dbURL`
- `dbAccessFlag`
- `eventCatalog`
- `eventSchema`
- `RootLogContext`
- `AdditionalLogContext`
- `LogConfigFile`
- `LogLevel`
- `MessageBundleBase`
- `LanguageCode`
- `CountryCode`
- `sleepCount`

デプロイメント記述子の編集

Web アプリケーションの デプロイメント記述子を編集する手順は次のとおりです。

1. 次の URL にアクセスして、ブラウザから **WebLogic Server Administration Console** を開きます。

`http://host:port/console`

この URL で、*host* は **WebLogic Server** が稼動するコンピュータ名、*port* は **WebLogic Server** のリスンポート番号に置き換えます。例：

`http://localhost:7001/console`

2. 左ペインで、[デプロイメント] ノードおよびその下の [Web アプリケーション] ノードを展開します。
3. デプロイメント記述子を編集する Web アプリケーションの名前を右クリックします。ドロップダウンメニューから、[Web アプリケーションデプロイメント記述子の編集] を選択します。**WebLogic Server Administration Console** が、新しいブラウザに表示されます。

Console は、2つのペインで構成されています。左ペインには、2つの Web アプリケーションデプロイメント記述子のすべての要素で構成されたナビゲーションツリーが含まれます。右ペインには、`web.xml` ファイルの説明要素のフォームが含まれます。

4. Web アプリケーションデプロイメント記述子の要素の編集、削除、または追加には、左ペインで編集対象のデプロイメント記述子に対応するノードを展開します。以下のノードがあります。
 - **Web App Descriptor** ノードには、`web.xml` デプロイメント記述子の要素があります。
 - **WebApp Ext** ノードには、`weblogic.xml` デプロイメント記述子の要素があります。
5. Web アプリケーションデプロイメント記述子の既存の要素を編集する手順は次のとおりです。
 - a. 左ペインのツリーで、親の要素を順にクリックして展開し、編集する要素を見つけます。
 - b. 該当する要素の名前をクリックします。フォームは、右ペインに選択した要素の属性または下位要素のリストと共に表示されます。
 - c. 右ペインのフォーム内のテキストを編集します。
 - d. [適用] をクリックします。
6. Web アプリケーションデプロイメント記述子に新しい要素を追加する手順は次のとおりです。

- a. 左ペインのツリーで、親の要素を順にクリックして展開し、作成する要素の名前を見つけます。
 - b. 適切な要素名を右クリックし、ドロップダウン メニューから [新しい要素のコンフィグレーション] を選択します。フォームは右ペインに表示されます。
 - c. 右ペインのフォームに、要素の情報を入力します。
 - d. [作成] をクリックします。
7. Web アプリケーション デプロイメント 記述子から既存の要素を削除する手順は次のとおりです。
- a. 左ペインのツリーで、親の要素を順にクリックして展開し、削除する要素の名前を見つけます。
 - b. 適切な要素名を右クリックし、ドロップダウン メニューから [要素の削除] を選択します。確認ページが表示されます。
 - c. [削除] ページで [はい] をクリックして、要素の削除を確定します。
8. Web アプリケーション デプロイメント 記述子の変更がすべて完了したら、左側のペインでツリーのルート要素をクリックします。ルート要素は、Web アプリケーションの *.war アーカイブ ファイルの名前または Web アプリケーションの表示名です。
9. Web アプリケーション デプロイメント 記述子のエントリが有効かどうかを確認する場合は、[検証] をクリックします。
10. [永続化] をクリックして、編集したデプロイメント記述子ファイルを WebLogic Server のメモリだけでなく、ディスクに書き込みます。

WebLogic Integrator クラスタでのアダプタのデプロイメント

アダプタは、WebLogic Integration クラスタにデプロイできます。クラスタ化 WebLogic Integration 環境におけるアダプタのデプロイの詳細については、『*WebLogic Integration ソリューションのデプロイメント*』の「WebLogic Integration クラスタについて」を参照してください。

A WebLogic Integration に限定されないアダプタの作成

第 6 章「サービス アダプタの開発」および第 7 章「イベント アダプタの開発」で概説した J2EE 準拠のアダプタの開発手順は、主に WebLogic Integration 向けのアダプタを開発するためのものです。これらの章で説明した手順を変更することで、J2EE コネクタアーキテクチャの仕様に準拠しながら、WebLogic Integration 以外でも使用可能なアダプタを構築できます。

この章では、そのための変更点について説明します。この章の内容は以下のとおりです。

- この節の目的
- アダプタの構築
- 構築プロセスの更新

この節の目的

この節では、J2EE 準拠のアダプタの開発手順を変更して、WebLogic Integration 以外でも実行可能なアダプタを構築する方法を示します。この節では、第 6 章「サービス アダプタの開発」に記載されている各手順を示し、その手順の変更方法について説明します。各手順をしっかりと理解し、ここで説明する変更方法を行ってください。

アダプタの構築

アダプタを構築するには、以下の手順に従ってください。この構築は、『*WebLogic Integration インストール ガイド*』の説明に従って、**WebLogic Integration** がインストールされているという前提とします。

1. 第 6 章「サービス アダプタの開発」の「手順 1: 環境要件の調査」の説明に従って、開発環境における要件を決定します。トランザクション サポートを参照するリストの最後の項目は無視してください。**WebLogic Server** は、ローカルまたは **XA** トランザクションはサポートしていないためです。
2. 第 4 章「カスタム開発環境の作成」の説明に従って、**GenerateAdapterTemplate** を実行します。
3. 6-10 ページの「手順 2b: アダプタ論理名の割り当て」の説明に従って、アダプタに論理名を割り当てます。
4. 6-24 ページの「基本的な SPI の実装」の説明に従って **SPI** を実装します。以下のクラスを拡張します。
 - **AbstractManagedConnectionFactory** (6-25 ページの「**ManagedConnectionFactory**」を参照)。
 - **AbstractManagedConnection** (6-34 ページの「**ManagedConnection**」を参照)。
 - **AbstractConnectionMetaData** (6-35 ページの「**ManagedConnectionMetaData**」を参照)。これらのクラスを拡張する際には、以下の点に注意してください。
 - **WebLogic Server** は、トランザクション セマンティクスが使用されるアダプタをサポートしていません。
 - **ConnectionManager** インタフェースは実装しないでください。ここで開発するアダプタは管理対象アダプタですので、**WebLogic Server** にプラグインするような設計となります。
5. **AbstractConnectionFactory** を拡張します。

構築プロセスの更新

WebLogic Integration に依存しないアダプタを作成するには、A-2 ページの「アダプタの構築」で説明されている手順の他に、`build.xml` ファイルを変更する必要があります。構築プロセスを更新するには、次の手順に従います。

1. コード エディタで、ADK の `build.xml` ファイルを開きます。
2. 6-10 ページの「手順 2c:ビルド プロセスの設定」を参照します。この手順には、6-12 ページの「`build.xml` のコンポーネント」と呼ばれる節が含まれます。その節では、`build.xml` ファイルのコンテンツが一連のコード リストによって示されています。
3. コード リスト 6-12 および コード リスト 6-13 を参照します。
4. これらのリストに示されるコードを、アダプタの `build.xml` ファイルから削除します。

B XML Toolkit

BEA WebLogic Integration の Adapter Development Kit (ADK) に付属している XML ToolKit を使用すると、有効な XML ドキュメントを作成し、アダプタを介して EIS から別のアプリケーションに情報を送信できます。XML 操作に必要な処理の多くを 1 箇所に統合できるので、作業を軽減できます。

この章の内容は以下のとおりです。

- Toolkit パッケージ
- IDocument
- Schema Object Model (SOM)

Toolkit パッケージ

XML Toolkit は、主に 2 つの Java パッケージから構成されます。

- `com.bea.document`
- `com.bea.schema`

これらのパッケージは、`xmltoolkit.jar` ファイル内にあり、WebLogic Integration のインストール時に、ADK と一緒にインストールされます。ここには、クラス、インタフェース、およびメソッドに関する詳細な Javadoc が格納されています。Javadoc を参照するには、次のディレクトリに移動してください。

`WLI_HOME/docs/apidocs/index.html`

この URL で、`WLI_HOME` のフォルダには、WebLogic Integration がインストールされています。

IDocument

`com.bea.document.IDocument`

IDocument は、W3C Document Object Model (DOM) と XPath インタフェースを XML ドキュメント内の要素に統合するコンテナです。この統合により、XPath 文字列を使用するだけで、IDocument オブジェクトをクエリおよび更新できます。XPath 文字列を使用すると、クエリする要素だけを指定し、そのクエリへの応答を返すことができるため、XML ドキュメント全体を解析して、特定の情報を探する必要がなくなります。

たとえば、コード リスト B-1 で示す XML ドキュメントには、「Bob」という人物に関する記述がされています。

コードリスト B-1 XML の例

```
<Person name="Bob">
  <Home squareFeet="2000"/>
  <Family>
    <Child name="Jimmy">
      <Stats sex="male" hair="brown" eyes="blue"/>
    </Child>
    <Child name="Susie">
      <Stats sex="female" hair="blonde" eyes="brown"/>
    </Child>
  </Family>
</Person>
```

ここで、<child> 要素から、「Jimmy」の髪の色を検索するとします。DOM を使用する場合、コード リスト B-2 に示すようなコードを使用する必要があります。

コードリスト B-2 DOM データのサンプル検索

```
String strJimmysHairColor = null;
org.w3c.dom.Element root = doc.getDocumentElement();
if (root.getTagName().equals("Person") && root.getAttribute("name").
    equals("Bob")) {
    org.w3c.dom.NodeList list = root.getElementsByTagName("Family"); if
        (list.getLength() > 0) {
```

```

org.w3c.dom.Element family = (org.w3c.dom.Element)list.item(0);
org.w3c.dom.NodeList childList = family.getElementsByTagName ("Child");
for (int i=0; i < childList.getLength(); i++) {
    org.w3c.dom.Element child = childList.item(i);
    if (child.getAttribute("name").equals("Jimmy")) {
        org.w3c.dom.NodeList statsList = child.
            getElementsByTagName("Stats");
        if (statsList.getLength() > 0) {
            org.w3c.dom.Element stats = statsList.item(0);
            strJimmysHairColor = stats.getAttribute("hair");
        }
    }
}
}
}
}

```

IDocument を使用すれば、コード リスト B-3 に示すような XPath 文字列を作成し、同じように「Jimmy」の髪の色を検索できます。

コードリスト B-3 IDocument データのサンプル検索

```

System.out.println("Jimmy's hair color: " + person.getStringFrom
    ("//Person[@name=\"Bob\"] /Family/Child[@name=\"Jimmy\"]/Stats/@hair");

```

このように、IDocument を使用すると、簡単なコードでドキュメント内の情報をクエリおよび検索できます。

Schema Object Model (SOM)

SOM は、XML スキーマを構築するためのインタフェースです。アダプタによって、特定の要求メタデータや応答メタデータの EIS が呼び出されます。このメタデータは、その後 XML スキーマにプログラマティックに変換する必要があります。SOM は、複雑なスキーマ構文などの詳細な一般 XML スキーマ情報を数多く抽出し、有効性を検証するための一連のツールです。このツールを使うことで、XML ドキュメントの重要度のより高い作業に専念できます。

SOM の仕組み

XML スキーマは、アダプタで接続している EIS とアダプタ側のアプリケーションとの間で、いわば契約書のような役割を果たします。この契約書では、アプリケーションで操作を行うために、EIS のデータをどのように表示するかが定義されます。ドキュメント（XML で記述された EIS のメタデータの集合）は、スキーマで定義されたルールに準拠していれば、ドキュメントの XML コードが正しいかどうかに関係なく、有効とみなされます。たとえば、<name> 要素に名前を表示するスキーマを定義していて、この要素が <firstname> と <lastname> の 2 つの子要素を必要とする場合、EIS のドキュメントを有効にするには、このドキュメントをコード リスト B-4 に示すフォームで表示し、スキーマをコード リスト B-5 のように定義する必要があります。

コードリスト B-4 ドキュメント例

```
<name>
  <firstname>Joe</firstname>
  <lastname>Smith</lastname>
</name>
```

コードリスト B-5 スキーマ例

```
<スキーマ >
  <element name="name">
    <complexType>
      <sequence>
        <element name="firstname" />
        <element name="lastname" />
      </sequence>
    </complexType>
  </element>
</スキーマ >
```

正しい XML コードとして作成されていても、<name></name> 以外のフォームは無効です。次に示すフォームは、無効の一例です。

```
<name>Joe Smith</name>
```

スキーマの作成

SOMにあるクラスおよびメソッドを使用して、プログラムでXMLスキーマを作成できます。このツールを使用することの利点は、プログラムに変数を入力するだけで、スキーマを要求に応じて調整できることです。たとえば、次のコード例は購買発注ドキュメントを検証するスキーマを作成するためのものです。コードリスト B-6 によって、スキーマが設定されて必要な要素が追加されます。

コードリスト B-6 購買発注スキーマ

```
import com.bea.schema.*;
import com.bea.schema.type.SOMType;

public class PurchaseOrder
{
    public static void main(String[] args)
    {
        System.out.println(getSchema().toString());
    }

    public static SOMSchema getSchema()
    {
        SOMSchema po_schema = new SOMSchema();

        po_schema.addDocumentation("Purchase order schema for
        Example.com.\nCopyright 2000 Example.com.\nAll rights
        reserved.");

        SOMElement purchaseOrder =
            po_schema.addElement("purchaseOrder");

        SOMElement comment = po_schema.addElement("comment");

        SOMComplexType usAddress =
            po_schema.addComplexType("USAddress");

        SOMSequence seq2 = usAddress.addSequence();

        // adding an object to a SOMSchema defaults to type="string"
        seq2.addElement("name");
        seq2.addElement("street");
        seq2.addElement("city");
        seq2.addElement("state");
        seq2.addElement("zip", SOMType.DECIMAL);
    }
}
```

属性の設定は、コード リスト B-7 で示すように、要素の作成と同じようにして行うことができます。属性を正しく設定するには、アドレスを正確に記述する必要があります。

コードリスト B-7 親属性の設定

```
SOMAttribute country_attr = usAddress.addAttribute("country",
    SOMType.NMTOKEN);
country_attr.setUse("fixed");
country_attr.setValue("US");
```

コード リスト B-8 で示すように、`complexType` と同様に、`simpleTypes` をスキーマのルートに追加できます。

コードリスト B-8 SimpleTypes をスキーマ ルートに追加

```
SOMSimpleType skuType = po_schema.addSimpleType("SKU");
SOMRestriction skuRestrict = skuType.addRestriction
    (SOMType.STRING);
skuRestrict.setPattern("\\d{3}-[A-Z]{2}");

SOMComplexType poType =
    po_schema.addComplexType("PurchaseOrderType");

purchaseOrder.setType(poType);
poType.addAttribute("orderDate", SOMType.DATE);
```

`SOMComplexType` オブジェクトの `addSequence()` メソッドによって、`SOMSequence` 参照が返されるため、ユーザはそのスキーマに追加された要素を変更できます。コード リスト B-9 で示すように、このようにしてオブジェクトがスキーマに追加されます。

コードリスト B-9 要素を変更するための `addSequence()` の実装

```
SOMSequence poType_seq = poType.addSequence();
poType_seq.addElement("shipTo", usAddress);
poType_seq.addElement("billTo", usAddress);
```

スキーマ内の要素の属性を設定するには、SOMElement オブジェクトのセッターメソッドを呼び出します。たとえば、コード リスト B-10 は、setMinOccurs() と setMaxOccurs() の実装方法を示します。

コードリスト B-10 setMinOccurs() および setMaxOccurs() の実装

```
SOMElement commentRef = new SOMElement(comment);
    commentRef.setMinOccurs(0);
    poType_seq.add(commentRef);
SOMElement poType_items = poType_seq.addElement("items");

SOMComplexType itemType = po_schema.addComplexType("Items");
SOMSequence seq3 = itemType.addSequence();
SOMElement item = new SOMElement("item");
    item.setMinOccurs(0);
    item.setMaxOccurs(-1);
    seq3.add(item);
SOMComplexType t = new SOMComplexType();
    item.setType(t);
SOMSequence seq4 = t.addSequence();
    seq4.addElement("productName");
SOMElement quantity = seq4.addElement("quantity");
SOMSimpleType st = new SOMSimpleType();
    quantity.setType(st);
SOMRestriction restrict =
    st.addRestriction(SOMType.POSITIVEINTEGER);
    restrict.setMaxExclusive("100");
```

この例では、PurchaseOrderType の items 要素は、Items タイプの前で作成されています。このため、コード リスト B-11 に示すコードを使用して、Items タイプ オブジェクトが利用可能になったら、参照を作成してタイプを設定する必要があります。

コードリスト B-11 品目タイプ オブジェクトが使用可能になった時点でのタイプの設定

```
poType_items.setType(itemType);
```

最後に、要素をスキーマに追加します。要素を追加するには、`SOMSequence` の `addElement()` メソッド、または以前に作成した `SOMElement` の `add()` メソッドを実装します。コードリスト B-12 は、この2つのメソッドを示します。

コードリスト B-12 スキーマへの要素の追加

```
seq4.addElement("USPrice", SOMType.DECIMAL);

    SOMElement commentRef2 = new SOMElement(comment);
    commentRef2.setMinOccurs(0);
    seq4.add(commentRef2);

    SOMElement shipDate = new SOMElement("shipDate", SOMType.DATE);
    shipDate.setMinOccurs(0);
    seq4.add(shipDate);
    t.addAttribute("partNum", skuType);

    return po_schema;
}
}
```

結果として作成されるスキーマ

前の7つのリスト（コードリスト B-6 からコードリスト B-12 まで）で示したコードを実行すると、コードリスト B-13 に示すスキーマが作成されます。

コードリスト B-13 XML スキーマ定義ドキュメント

```
<?xml version="1.0" ?>
<!DOCTYPE schema (View Source for full doctype...)>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/XMLSchema">
  <xsd:annotation>

    <xsd:documentation>Purchase order schema for Example.com.
      Copyright 2000 Example.com. All rights
      reserved.</xsd:documentation>

  </xsd:annotation>

  <xsd:simpleType name="SKU">
    <xsd:annotation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{3}-[A-Z]{2}" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```



```

        </xsd:restriction>
    </xsd:simpleType>

    <xsd:complexType name="PurchaseOrderType">
        <xsd:sequence>
            <xsd:element type="USAddress" name="shipTo" />
            <xsd:element type="USAddress" name="billTo" />
            <xsd:element ref="comment" minOccurs="0" />
            <xsd:element type="Items" name="items" />
        </xsd:sequence>

        <xsd:attribute name="orderDate" type="xsd:date" />
    </xsd:complexType>

    <xsd:complexType name="Items">
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" name="item"
                minOccurs="0">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element type="xsd:string"
                            name="productName"/>
                        <xsd:element name="quantity">
                            <xsd:simpleType>
                                <xsd:restriction base=
                                    "xsd:positiveInteger">
                                    <xsd:maxExclusive value="100"/>
                                </xsd:restriction>
                            </xsd:simpleType>
                        </xsd:element>
                        <xsd:element type="xsd:decimal" name=
                            "USPrice" />
                        <xsd:element ref="comment"
                            minOccurs="0" />
                        <xsd:element type="xsd:date"
                            name="shipDate" minOccurs="0" />
                    </xsd:sequence>
                    <xsd:attribute name="partNum" type="SKU" />
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="USAddress">
        <xsd:sequence>
            <xsd:element type="xsd:string" name="name" />
            <xsd:element type="xsd:string" name="street" />
            <xsd:element type="xsd:string" name="city" />
            <xsd:element type="xsd:string" name="state" />
            <xsd:element type="xsd:number" name="zip" />
        </xsd:sequence>

        <xsd:attribute name="country" use="fixed" value="US"
            type="xsd:NMTOKEN" />
    </xsd:complexType>
</xsd:element type="PurchaseOrderType" name="purchaseOrder" />

```

```
<xsd:element type="xsd:string" name="comment" />
</xsd:schema>
```

XML ドキュメントの検証

次に、コード リスト B-13 のスキーマを使用して、EIS から送られたドキュメントを検証します。たとえば、コード リスト B-14 に示すドキュメントでは、先ほど作成したスキーマに基づいて検証します。

コードリスト B-14 検証対象の XML ドキュメント

```
<?xml version="1.0" ?>
<!DOCTYPE PurchaseOrder (View Source for full doctype...)>

<purchaseOrder orderDate="1/14/00">
  <shipTo Country="US">
    <name>Bob Jones</name>
    <street>1000 S. 1st Street</street>
    <city>Denver</city>
    <state>CO</state>
    <zip>80111</zip>
  </shipTo>

  <billTo Country="US">
    <name>Bob Jones</name>
    <street>1000 S. 1st Street</street>
    <city>Denver</city>
    <state>CO</state>
    <zip>80111</zip>
  </billTo>

  <comment>None</comment>

  <items>
    <item partNum="123-AA">
      <productName>Washer</productName>
      <quantity>20</quantity>
      <USPrice>0.22</USPrice>
      <comment>Only shipped 10</comment>
      <shipDate>1/14/00</shipDate>
    </item>
    <item partNum="123-BB">
      <productName>Screw</productName>
      <quantity>10</quantity>
      <USPrice>0.30</USPrice>
      <comment>None</comment>
      <shipDate>1/14/00</shipDate>
    </item>
  </items>
</purchaseOrder>
```

```
</items>  
</purchaseOrder>
```

ドキュメントの検証方法

SOM では、SOMSchema メソッドの `isValid()` を使用して、XML DOM ドキュメントの検証ができます。SOMElement クラスには対応する `isValid()` メソッドがあり、DOM ドキュメントではない要素の検証を行います。

この `isValid()` メソッドでは、ドキュメントまたは要素が有効の場合には確定し、無効の場合にはエラー リストをコンパイルします。ドキュメントが有効な場合、`isValid()` によって `True` が返され、エラー リストは空白になります。

`isValid()` の実装

コード リスト B-15 では、`isValid()` を実装するための 2 つの方法を示します。

コードリスト B-15 `isValid()` の実装例

```
public boolean isValid(org.w3c.dom.Document doc,  
                      java.util.List errorList)  
public boolean isValid(IDocument doc,  
                      List errorList)
```

以下のパラメータを使用します。

- `doc` - 検証対象のドキュメント インスタンス
- `errorList` - `doc` ドキュメントで検出されたエラーのリスト

このスキーマに基づいて、ドキュメントが有効であると判断されると、`isValid()` は、`True` のブール値を返します。有効でないと判断されると、`isValid()` は `False` を返し、`errorList` が作成されます。

`errorList` は、`doc` ドキュメント内で検出されたエラーを報告するためのもので、`java.util.List` です。ドキュメントの検証前に、エラー リストはクリアされます。このため、使用するリスト実装は、`clear()` メソッドをサポートしている

必要があります。isValid() が False を返す場合、エラー リストには、バリデーションプロシージャの実行中に検出されたエラーが入力されます。リスト内の項目は、クラス com.bea.schema.SOMValidationException のインスタンスです。isValid() が True を返す場合、errorList は空白になります。

API の詳細については、次の URL の isValid() の Javadoc を参照してください。

WLI_HOME/docs/apidocs/com/bea/SOMSchema.html

isValid() の実装例

コード リスト B-16 に、isValid() の実装例を示します。

コードリスト B-16 isValid() の実装例

```
SOMSchema schema = ...;

IDocument doc = DocumentFactory.createDocument(new FileReader(f));
java.util.LinkedList errorList = new java.util.LinkedList();
boolean valid = schema.isValid(doc, errorList);...
if (! valid){
    System.out.println("Document was invalid. Errors were:");
    for (Iterator i = errorList.iterator; i.hasNext(); )
    {
        System.out.println(((SOMValidationException) i.next()).
            toString());
    }
}
```

C WebLogic Integration 7.0 へのアダプタの移行

WebLogic Integration 2.1 から WebLogic Integration 7.0 に移行する場合、アダプタの移行タスクを実行する必要はありません。WebLogic Integration 2.1 向けに開発され、テストされたアダプタは、変更を加えなくても WebLogic Integration 7.0 で正常に稼動しますが、このリリースの新機能を最大限に活かすために、この章で説明するタスクを実行したほうが良い場合があります。

この章の内容は以下のとおりです。

- WebLogic Integration 7.0 ADK 向けのアダプタの再構築
- アプリケーション統合 CLASSPATH およびアダプタのパッケージ化の変更
- アダプタによる共有接続ファクトリ ユーザ インタフェースのサポートの許可
- セキュリティ制約とログイン コンフィグレーションにおける変更
- 要求データを必要としないサービスに対する DBMS サンプル アダプタの変更
- WebLogic Integration 7.0 での WebLogic Integration 2.1 アダプタの使用

WebLogic Integration 7.0 ADK 向けのアダプタの再構築

WebLogic Integration 7.0 ADK 向けにアダプタを再構築する必要がある場合、WebLogic Integration 7.0 で提供される新しいバイナリ ファイルを参照するよう、手順を再構築する必要があります。特に、以下の新しい JAR ファイルを参照する必要があります。

```
<property name='WLAI_CORE' value='${WLI_LIB_DIR}/wlai-core.jar' />
<property name='WLAI_CLIENT' value='${WLI_LIB_DIR}/wlai-client.jar' />
<property name='WLAI_EVENTROUTER' value='${WLI_LIB_DIR}/wlai-eventrouter.jar' />
```

以下の JAR ファイルは、アダプタでの有効性がなくなっています。

```
<property name='WLAI_CLIENT' value='${WLI_LIB_DIR}/wlaiclient.jar' />
<property name='WLAI_COMMON' value='${WLI_LIB_DIR}/wlai-common.jar' />
<property name='WLAI_EJB_CLIENT' value='${WLI_LIB_DIR}/wlai-ejb-client.jar' />
<property name='WLAI_SERVLET_CLIENT'
  value='${WLI_LIB_DIR}/wlai-servlet-client.jar' />
<property name='WLAI_EVENTROUTER_CLIENT'
  value='${WLI_LIB_DIR}/wlai-eventrouter-client.jar' />
```

環境プロパティの宣言方法も、以下のフォームに合うよう変更する必要があります。

```
<property environment='env' />
```

以下のような文は、無効となった Ant 文であるため、削除してください。

```
<property name='WL_HOME' environment='env' />
```

注意： 無効な Ant 文があると、以下のようなエラー メッセージが表示されません。

```
You must specify value, location or refid with the name
attribute.
```

アプリケーション統合 CLASSPATH および アダプタのパッケージ化の変更

WebLogic Integration 2.1 および WebLogic Integration 2.1 SP1 では、WebLogic Server のインスタンスに対し、システム CLASSPATH にアダプタの java クラスが必要とされていましたが、WebLogic Integration 7.0 では、アダプタ java クラスは単一の独立した EAR ファイルにパッケージ化されている必要があります。アダプタ java クラスまたは JAR ファイルを WebLogic Integration 7.0 のインストールに移動したり、アダプタのクラスを WebLogic Integration CLASSPATH に追加しないでください。アダプタ EAR ファイルのコンフィグレーション手順について

は、『WebLogic Integration 移行ガイド』「移行に関するその他のトピック」の「Application Integration アダプタ EAR ファイルのコンフィグレーション」を参照してください。

アダプタによる共有接続ファクトリ ユーザ インタフェースのサポートの許可

WebLogic Integration は、共有接続ファクトリをサポートしています。アダプタが、関連付けられたユーザ インタフェースと対話するのを許可するには、アダプタのプロパティ ファイルに以下のプロパティを追加します。nav.jsp プロパティは、Application View Console に表示されるツールバー項目に対応しており、その他のプロパティは共有接続ファクトリの表示ラベルとして使用されます。共有接続ファクトリを使用する場合、必ず最新の ADK および設計時インタフェースを使用してください。

```
#nav.jsp#
nav_label_summary=Summary
nav_label_service=Add&nbsp;Service
nav_label_main=Home
nav_label_event=Add&nbsp;Event
nav_label_deploy=Deploy&nbsp;Application&nbsp;View
nav_label_define=Define&nbsp;Application&nbsp;View
nav_label_connection=Configure&nbsp;Connection
nav_label_admin=Administration
nav_label_select=Select&nbsp;Connection&nbsp;Type

# owned connection hdr #
connhdr_label_username=User Name:
connhdr_label_eisproductname=EIS Product Name:
connhdr_label_eisproductversion=EIS Product Version:

# referenced connection hdr #
connhdr_label_referenceConnectionCaption=Referenced Connection
connhdr_label_connection=Connection:
connhdr_label_adaptername=Name:
connhdr_label_adapterdesc=Description:
connhdr_label_adapterversion=Version:
connhdr_label_adapterlocaltrans=Supports local transactions

depappvw_label_sharedconnection=Shared Connection
depappvw_label_adaptername=Name:
depappvw_label_adaptervendor=Vendor:
```

```
depappvw_label_adapterdesc=Description:  
depappvw_label_adapterversion=Version:
```

WebLogic Integration 2.1 アダプタとの下位互換性をサポートするには、アプリケーション統合エンジンがユーザ インタフェースを識別する必要があります。以下のような変更を行い、ユーザ インタフェースにバージョン番号をマーキングします。

アダプタの Web コンポーネントの web.xml ファイルに以下のエントリを加えます。

```
<context-param>  
  <param-name>version</param-name>  
  <param-value>7.0</param-value>  
</context-param>  
  
<サーブレット >  
  <servlet-name>contextinfo</servlet-name>  
  <servlet-class>jsp_servlet.__contextinfo</servlet-class>  
</servlet>  
  
<servlet-mapping>  
  <servlet-name>contextinfo</servlet-name>  
  <url-pattern>contextinfo</url-pattern>  
</servlet-mapping>
```

contextinfo JSP により、アプリケーション統合エンジンによるユーザ インタフェースフレームワークのバージョンの識別が可能になります。この JSP は、WebLogic Integration 7.0 を使用してアダプタを再コンパイルするときに追加されます。

セキュリティ制約とログイン コンフィグレーションにおける変更

アダプタの開発者は、アダプタグループではなく Administrators グループを使用するよう設計時ロールの制約を更新する必要があります。前のリリースでは、ユーザはアダプタグループに属している必要がありました。リリース 7.0 以降では、ユーザは Administrators グループに属している必要があります（「設計時 GUI の開発」のコード リスト 8-14 および コード リスト 8-15 のロール名を参照してください）。この変更は、デプロイメントに、ユーザが Administrators グループに属することが必須の MBeans へのアクセスが必要となるためです。

要求データを必要としないサービスに対する DBMS サンプル アダプタの変更

WebLogic Integration 7.0 では、要求データを必要としないサービスに対し、DBMS サンプルアダプタにより、空またはヌルの要求ドキュメント定義が生成されます。たとえば、where 文節にパラメータを供給しないシンプルな SQL select 文に基づくサービスは、実行時に要求ドキュメントの実行を必要としません。このようなサービスは、空またはヌルのドキュメント定義に関連付けられます。これは、DBMS サンプルアダプタの設計時 Web インタフェースの [Summary and Administration] ページの [要求はありません] ラベルに表示されます。

ApplicationView インスタンスで要求データを必要としないサービスの名前で ApplicationView.getRequestDocumentDefinition() を呼び出すと、isNull() メソッドが true を返す IDocumentDefinition インスタンスが返されます。IDocumentDefinition インスタンスで getDocumentSchema()、getDocumentSchemaName() または getRootElementName() を呼び出すと、送出時に IllegalStateException が発生します。

これらの変更は、WebLogic Integration 7.0 の DBMS サンプルアダプタで定義されたアプリケーションビューにのみ反映されます。この変更は、既存のアプリケーションビューには影響はありませんが、サンプルアダプタの動作の変化を、進行中の開発において考慮する必要があります。

WebLogic Integration 7.0 での WebLogic Integration 2.1 アダプタの使用

WebLogic Integration 2.1 を使用して開発されたアダプタは、コンポーネントの再コンパイルを行わなくても WebLogic Integration 7.0 で使用できますが、これらのアダプタを WebLogic Integration 7.0 で使用する前に、コンフィグレーションを以下のように変更する必要があります。

- クラスパスに wlai-client.jar を追加する。

- イベント ルータをプライムする。

まず、クラスパスに `wlai-client.jar` を追加します。 `startWeblogic` スクリプトを編集して、Windows システムの [スタート] コマンドに以下のテキストを追加します。

```
%WLI_HOME%\lib\wlai-client.jar  
  
%JAVA_HOME%\bin\java -classic %DB_JVMARGS% -Xmx256m -classpath  
%WLI_HOME%\lib\wlai-client.jar;%SVRCP%
```

UNIX システムの場合は、以下のテキストを追加します。

```
$WLI_HOME\lib\wlai-client.jar
```

クラスパスを変更する他に、WebLogic Integration 2.1 アプリケーション ビューをデプロイする前に、イベント ルータをプライムする必要があります。これは、WebLogic Integration 2.1 アダプタを WebLogic Integration 7.0 環境で実行する場合にのみ、そしてアプリケーション ビューをデプロイメントする前にのみ必要です。イベント ルータをプライムすることで、ルータは、サーバとの通信が途切れた場合に自身を再初期化します。

イベント ルータをプライムするには、イベント ルータ サーブレットにアクセスする必要があります。イベント ルータ サーブレットにアクセスするには、デプロイされているイベント ルータの URL を使用します。イベント ルータは、アプリケーション統合エンジンと同じ物理マシン上で、つまり別の WebLogic Server インスタンスに対してデプロイされている場合があります。ローカルにデプロイされたイベント ルータの URL は、通常、以下のようなパターンになります。

```
http://localhost:7001/EventRouterContext/EventRouter
```

ここで、`EventRouterContext` は `application.xml` ファイルで定義されたコンテキストです。たとえば、DBMS サンプルアダプタのイベント ルータのコンフィグレーション ページは以下のようになります。

```
http://localhost:7001/DbmsEventRouter/EventRouter
```

イベント ルータは、ルータを WAR ファイルにコンパイルすることでスタンドアロン モジュールとしてデプロイされている場合もあります。DBMS サンプルアダプタは、`eventrouter_war` ターゲットに示すように、このメソッドを `build.xml` ファイルで使用します。以下は DBMS サンプルアダプタに対する `build.xml` ファイルからの抜粋です。ここでルータがどのように WAR ファイルにコンパイルされるかを示します。

```
<target name='eventrouter_war' depends='jar,eventrouter_jar'>  
  <delete dir='${SRC_DIR}/eventrouter/WEB-INF/lib'/>
```

```

<war warfile='${LOCAL_LIB_DIR}/${EVENTROUTER_WAR_FILE}'
  webxml='${SRC_DIR}/eventrouter/WEB-INF/web.xml'>
  <fileset dir='.' includes='version_info.xml' />
  <fileset dir='${SRC_DIR}/eventrouter'
    excludes='WEB-INF/web.xml' />
  <lib dir='${LOCAL_LIB_DIR}'
    includes='${JAR_FILE},${EVENTROUTER_JAR_FILE}' />
  <lib dir='${WLI_LIB_DIR}'
    includes='adk.jar,adk-eventgenerator.jar,
    wlai-eventrouter.jar,wlai-core.jar,wlai-client.jar' />
</war>
</target>

```

この URL を使用してイベント ルータ サーブレットにアクセスすると、イベント ルータのコンフィグレーションが表示されます。

Event Router

The screenshot shows a web interface for configuring an Event Router. At the top right, there is a 'Save Changes' button. Below it, there are radio buttons for 'Add' and 'Remove'. The main part of the interface is a table with three columns: 'Server Name', 'UserID', and 'Password'. Each column has a text input field. To the right of the 'Password' field is a 'Do it' button.

このページで、イベント ルータのサーバをコンフィグレーションすることができます。イベント ルータのサーバ情報を追加するには、

1. [Add] ラジオ ボタンを選択します。
2. サーバ名 (DNS または TCPIP) とポートを入力します。
3. ユーザ id とパスワードを入力します。
注意： ユーザは必ず管理者特権を持つ必要があります。
4. [Do it] をクリックします。
5. [Save Changes] をクリックします。パラメータの情報が変更され、変更が保存されたことを示す確認メッセージが表示されます。

初期化ファイルで、イベント ルータが初期化されたことを確認できます。このファイルは、通常、_Servletcontextidname で始まる長い名前を持ちます。このファイルは、必要に応じていつでも安全に削除し、再作成することができます。

D アダプタ設定ワークシート

この付録にあるワークシートを使用して、開発対象のアダプタに関する重要な情報を収集できます。ワークシートに記載された質問に回答していくことで、アダプタの論理名や **Java** パッケージの基本名などコンポーネントの定義ができます。また、メッセージバンドルをローカライズする必要のあるロケールも決定できます。これらの質問に回答することで、コーディングを開始する前にアダプタの定義を容易に行うことができます。

注意： `GenerateAdapterTemplate` ユーティリティを使用している場合、このワークシートの使用は特に重要です。質問への答えが、このユーティリティを正しく稼働させるのに必須となるためです。

アダプタ設定ワークシート

アダプタの開発を始める前に、次の質問についてできる限りお答えください。先頭にアスタリスク (*) の付いたすべての質問の回答は、**GenerateAdapterTemplate** ユーティリティの使用を予定している場合、必ず必要になります。

1. * アダプタの開発に使用する **EIS** の名前は何ですか？
2. * 使用する **EIS** のバージョンは何ですか？
3. * 使用する **EIS** のタイプ (DBMS または ERP など) は何ですか？
4. * アダプタのベンダ名は何ですか？
5. * 使用するアダプタのバージョンは何ですか？
6. * アダプタの論理名は何ですか？
7. アダプタで **EIS** 内部の機能呼び出す必要がありますか？
その必要がある場合、アダプタはサービスをサポートしますか？
8. 外部プログラムから **EIS** の機能呼び出せるようにするために、**EIS** ではどのようなメカニズムや **API** を提供していますか？
9. このメカニズムのために **EIS** にセッションや接続を設定するのに、どのような情報が必要ですか？
10. 特定のサービスについて **EIS** で呼び出す機能を特定するときに、どのような情報が必要ですか？
11. **EIS** では特定の機能に対する入出力の要件のために、**EIS** をクエリすることができますか？
可能な場合、サービスの入力要件を特定するのにどのような情報が必要になりますか？
12. 入力要件のうち、どの要件にも出てくる静的な情報は何か？アダプタでは、このような静的な情報を **InteractionSpec** オブジェクトにエンコーディングする必要があります。

13. 入力要件のうち、各要件に対する動的な情報は何か？アダプタには、各要件に対して、このサービスで必要とする入力パラメータを記述した XML スキーマを提供する必要があります。
14. サービスの出力要件を特定するのに必要な情報は何か？
15. アダプタから呼び出せる機能のカタログを参照するメカニズムが EIS によって提供されていますか？提供されている場合は、お使いのアダプタがサービスの参照をサポートしている必要があります。
16. EIS 内部での変更通知を受け取る必要がありますか？必要がある場合は、お使いのアダプタがイベントをサポートしている必要があります。
17. 外部プログラムから EIS のイベント通知を受け取るために、EIS ではどのようなメカニズムや API が提供されていますか？この質問の回答によって、プル型とプッシュ型のいずれのメカニズムを開発すべきかを定めることができます。
18. EIS では、お使いのアダプタがサポートできるイベントを特定できますか？
19. EIS では特定のイベントに対するメタデータをクエリできますか？
20. お使いのアダプタでサポート可能な（言語および国によって定義される）ロケールは何ですか？

E DBMS サンプル アダプタを使用したアダプタ開発方法の学習

この章の内容は以下のとおりです。

- DBMS サンプル アダプタの概要
- DBMS サンプル アダプタの仕組み
- DBMS サンプル アダプタの開発工程
- DBMS サンプル アダプタの設計時 GUI の開発工程

DBMS サンプル アダプタの概要

DBMS サンプル アダプタは JSP ベースの GUI を含む J2EE 準拠のアダプタです。これは、WebLogic Integration ADK を使用してアダプタがどのように構築されるかを示す具体例を提供します。リレーショナルデータベースは、アダプタのプロバイダが個々の EIS について学習する時間を省き、アダプタと ADK に集中できるよう、アダプタの EIS として使用されます。

DBMS サンプル アダプタは、独自のアダプタを設計、開発するために必要なタスクの理解をサポートすることを目的としています。プロダクション環境での使用向けではなく、そのような環境ではサポートされていません。アダプタは、プロダクション向けに準備の整ったアダプタとしてではなく、例として機能するものであるため、すべての機能が含まれているわけではなく、以下のような制限があります。このアダプタは、複雑なクエリやストアド プロシージャの実行はできません。

DBMS サンプル アダプタは、開発者やビジネス アナリストが ADK を使用してアダプタを構築する場合のさまざまな可能性について理解できるようサポートします。ビジネス アナリストの方は、E-2 ページの「DBMS サンプル アダプタの仕組み」で説明するように、インタフェースの操作を通じてアプリケーション ビュー、サービスおよびイベントについて理解できます。

アダプタ開発者は、J2EE 準拠のアダプタを構築するための ADK クラスの拡張および使用方法の学習から始めることをお勧めします。手順は、以下を参照してください。

- E-25 ページの「DBMS サンプルアダプタの開発工程」
- E-44 ページの「DBMS サンプルアダプタの設計時 GUI の開発工程」
- DBMS サンプルアダプタ コード
- DBMS サンプルアダプタ Javadoc

DBMS サンプル アダプタは以下の要件を満たしています。

- エンドユーザが GUI を使用して、Pointbase、Oracle、SQLServer、または Sybase データベースに接続できること。
- ADK のクラスおよびツールを使用すること。
- ユーザがイベントおよびサービスに基づいてアプリケーション ビューを作成できること。
- ユーザがイベントおよびサービスをテストできること。
- GUI からベースとなるデータベースのカタログ、スキーマ、テーブル、およびカラムを参照できること。
- データベース (EIS) で、選択、挿入、削除、および更新サービスの作成がサポートされること。

DBMS サンプル アダプタの仕組み

この節では、DBMS サンプルアダプタの仕組みについて説明します。ビジネスアナリストの方はインタフェースの操作を通じてアダプタの仕組みを理解できません。この節の例ではベースとなるデータベースに顧客を挿入といったサービスを作成する方法について説明します。次に、イベントがどのように生成されてアクションの実行が通知されるのかを示します。

この節の内容は以下のとおりです。

- 始める前に
- DBMS サンプルアダプタへのアクセス

- DBMS サンプル アダプタ ツアー

始める前に

DBMS サンプル アダプタにアクセスする前に、以下のタスクを完了してください。

- **WebLogic Integration** をインストールします。手順については、『*WebLogic Platform インストールガイド*』を参照してください。
- **ADK Ant** ベースの作成プロセスの設定。手順については、6-10 ページの「手順 2c : ビルド プロセスの設定」を参照してください。
- 設計時 GUI がアクセス可能になるよう **DBMS アダプタ** をデプロイ。詳細については、『*WebLogic Platform インストールガイド*』を参照してください。

DBMS サンプル アダプタへのアクセス

DBMS サンプル アダプタにアクセスする手順は次のとおりです。

1. 新しいブラウザ ウィンドウを開きます。
2. 該当するシステムの **Application View Management Console** の URL を入力します。

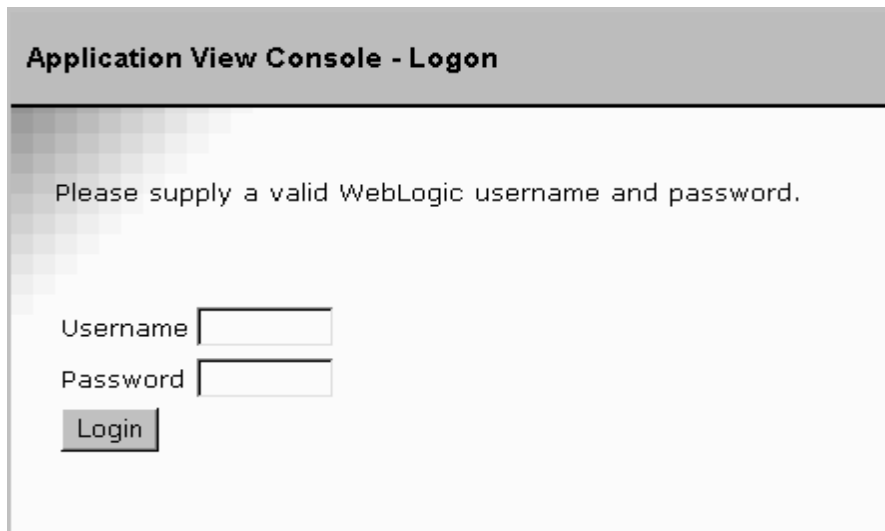
```
http://HOSTNAME:7001/wlai
```

図 E-1 に示すように、[Application View Console - Logon] 画面が表示されます。

DBMS サンプル アダプタ ツアー

ここでは DBMS サンプルアダプタの操作方法について簡単に説明します。まず、ブラウザで DBMS サンプル アダプタの [Application View Console - Logon] ページを開きます。手順については、E-3 ページの「DBMS サンプルアダプタへのアクセス」を参照してください。

図 E-1 [Application View Console - Logon]



Application View Console - Logon

Please supply a valid WebLogic username and password.

Username

Password

1. Application View Management Console にログオンするには、WebLogic Server のユーザ名とパスワードを入力し、[Login] をクリックします。[Application View Management Console] が表示されます。

図 E-2 [Application View Management Console]



2. [Add Application View] をクリックします。[Define New Application View] 画面が表示されます。アプリケーション ビューを作成する場合、アプリケーション ビューと DBMS サンプル アダプタを関連付けるための説明を指定します。

アプリケーション ビューおよびその定義に関する詳細については、『*Application Integration ユーザーズガイド*』の「アプリケーション ビューの定義」を参照してください。

図 E-3 [Define New Application View] ページ



3. アプリケーション ビューを定義する手順は次のとおりです。
 - a. [Application View Name] フィールドに `AppViewTest` と入力します。

入力する名前は、このアプリケーションが実行する一連の機能を表すものにする必要があります。また、アダプタに対してユニークなアプリケーション ビュー名にする必要があります。ピリオド (.)、ハッシュマーク (#)、バックスラッシュ (\)、プラス記号 (+)、アンパサンド (&)、カンマ (,)、アポストロフィ (')、二重引用符 (") およびスペースは無効です。
 - b. [Description] フィールドにアプリケーション ビューの簡単な説明を入力します。
 - c. [Associated Adapters] リストからこのアプリケーション ビューを作成する際に使用する DBMS サンプル アダプタを選択します。
 - d. [OK] をクリックします。[Select Existing Connection] 画面が表示されます。

図 E-4 [Select Existing Connection] ページ



[Select Existing Connection] ページでは、アプリケーション ビューに関連付ける接続ファクトリのタイプを選択できます。

- 新しい接続ファクトリを作成するには、[New Connection] ラジオ ボタンを選択します。
- 他のアプリケーション ビューと接続ファクトリを共有する場合は、既存の接続ファクトリのラジオ ボタンを選択します。既存の接続ファクトリの横の [Reference] リンクをクリックして、既存の接続ファクトリを使用してデプロイされているアプリケーション ビューの名前を表示します。

図 E-5 [Connection Factory Reference] ページ



[Connection Factory Selection] ページから、[Select Connection] または [Connection Configuration] ページをいつでも表示できます。アプリケーションビューをデプロイする前であれば、新しい接続ファクトリと既存の接続ファクトリを切り替えることができます。

4. [Continue] をクリックします。新しい接続ファクトリの作成を選択した場合、[Configure Connection Parameters] ページが表示されます。既存の接続ファクトリの使用を選択した場合、[Application View Administration] ページが表示されます。（[Application View Administration] ページの詳細については、手順 5. を参照してください。）

図 E-6 [Configure Connection Parameters] ページ

The screenshot shows a web-based configuration interface. The title bar reads 'Configure Connection Parameters'. Below the title, there is a navigation menu on the left with items: 'Configure Connection', 'Administration', 'Add Service', 'Add Event', and 'Display Application View'. The main content area has a heading 'Configure Connection' and a sub-heading 'Administration'. Below this, there is a message: 'On this page, you supply parameters to connect to your DBMS'. The form contains three input fields: 'WebLogic User Name*' with the value 'admin', 'WebLogic Password*', and 'Data Source Name (JNDI)*' with the value 'WLAI_DataSource'. A 'Continue' button is positioned below the form. The top right corner of the page features the 'bea' logo and links for 'Home' and 'Logout'.

5. [Configure Connection Parameter] 画面では、アプリケーション ビューと対象の EIS との対話を可能にするネットワーク関連情報を入力します。これを入力する必要があるのは、各アプリケーション ビューで 1 度だけです。
 - a. WebLogic Server の ユーザ名とパスワードを入力します。
 - b. [Data Source Name (JNDI)] フィールドに WLAI_DataSource と入力します。
 - c. [Continue] をクリックします。[Application View Administration] ページが表示されます。

[Application View Administration] ページに、接続条件の概要が表示されます。イベントおよびサービスを定義したときに、スキーマを表示してこのページからイベントやサービスを概略化したり削除したりできます。

アプリケーション ビューの作成が完了した際、サービスをアプリケーション ビューに追加できます。

図 E-7 [Application View Administration for AppViewTest] ページ



6. 新しいアプリケーションビューにサービスを追加するには、サービスの名前、説明および SQL 文を設定する必要があります。

参照リンクを使用して DBMS サンプルアダプタ データベーススキーマとテーブルを参照し、データベーステーブル CUSTOMER_TABLE を指定します。

サービスを追加する手順は次のとおりです。

- a. [Application View Administration] ページで [Service] グループ内の [Add] をクリックします。[Add Service] ページが表示されます。

図 E-8 [Add Service] ページ



- b. [Unique Service Name] フィールドに **InsertCustomer** と入力します。
- c. [Description] フィールドに、サービスの説明を入力します。
- d. [Browse DBMS] をクリックして、データベースのテーブルおよびカラム構成を表示します。複雑なクエリを記述する場合、[Browse] ウィンドウを開いたままで、後でテーブルやカラム名を切り取ってクエリに貼り付けることができます。

図 E-9 [Browse DBMS] ページ



e. [DBMS Schemas for Catalog] ページで、[APP] をクリックします。

図 E-10 [Browse DBMS Table Types] ページ



f. [DBMS Table Types] ページで [TABLE] をクリックします。

図 E-11 [DBMS Browse Tables] ページ



- g. [Tables For: .APP] ページで [CUSTOMER_TABLE] をクリックします。
[Browse] ウィンドウにカラムの名前とタイプが表示されます。

図 E-12 [Browse DBMS for Table] ページ



DBMS Columns For Table: CUSTOMER_TABLE

ColumnName:	ColumnType:	ColumnSize:
FIRSTNAME	VARCHAR	32
LASTNAME	VARCHAR	32
MIDDLENAME	VARCHAR	32
DOB	DATE	10
ADDRESS1	VARCHAR	32
ADDRESS2	VARCHAR	32
ADDRESS3	VARCHAR	32
POSTALCODE	VARCHAR	11
CITY	VARCHAR	32
STATE	VARCHAR	32
COUNTRY	VARCHAR	32
PHONE	VARCHAR	15
FAX	VARCHAR	15
EMAIL	VARCHAR	64

Close Window

- h. [Close Window] をクリックしてウィンドウを閉じ、[Add Service] ページに戻ります。

このウィンドウはアダプタで使用できる機能を紹介するためにツアーの中に組み込まれています。この演習ではテキストを選択する必要はありません。

- i. [Service] ページで以下の情報を [SQL Statement] フィールドに入力します。

```
Insert into APP.CUSTOMER_TABLE (FIRSTNAME, LASTNAME, DOB)
VALUES ([FIRSTNAME VARCHAR], [LASTNAME VARCHAR], [DOB
DATE])
```

- j. [Add] をクリックします。[Application View Administration] ページが表示されます。

サービスの追加に関する詳細については、『*Application Integration ユーザーズガイド*』の「アプリケーションビューの定義」を参照してください。

7. 該当するアプリケーションビューにイベントを追加します。追加するには、イベントにユニークな名前と説明を指定する必要があります。次に、イベントのトリガを追加するデータベース テーブルを指定します。また、追加するイベントが挿入、更新、削除のどれであるかも指定する必要があります。

[Browse DBMS] リンクを使用して、DBMS データベース スキーマとテーブルを参照し、データベース テーブルを指定します。これにより、選択したテーブル名をフィールドに自動的に入力できます。

イベントを追加する手順は次のとおりです。

- a. [Application View Administration] ページで [Event] グループ内の [Add] をクリックします。[Add Event] ページが表示されます。

図 E-13 [] ページ

- b. [Unique Event Name] フィールドに CustomerInserted と入力します。
- c. [Description] フィールドにイベントの説明を入力します。
- d. [Browse DBMS] リンクをクリックして、データベースのテーブルおよびカラム構成を表示します。

図 E-14 [Browse DBMS Tables] ページ

<u>BROWSE</u>	☐
ACTINGCOLLABORATOR	☐
ACTINGCOMDISE	☐
ACTINGCONVERSATION	☐
ACTINGCONTACT	☐
ACTINGCPA	☐
ACTINGCRAGE	☐
ACTINGMIS	☐
ACTINGMESSAGE	☐
ACTINGMESSAGEDATA	☐
ACTINGMESSAGEENVELOPE	☐
ACTINGMESSAGESTORE	☐
ACTINGMESSAGESTORE	☐
ACTINGPAYLOAD	☐
ACTINGPROCESS	☐
ACTINGROLE	☐
ACTINGROLEDEF	☐
ACTINGWFINSTANCE	☐
ACTINGWALCD	☐
ADDRESSMESSAGE	☐
BUSINESSCALENDAR	☐
BUSINESSCOMPARISON	☐
BUSINESS_PROCESS	☐
BUSINESS_PROTOCOL_DEFINITION	☐
CA	☐
CERTIFICATE	☐
CONTAINED_OBJECT	☐
CPACOLLABORATORMAP	☐
CPACONVERSATIONMAP	☐
CPACRAGEMAP	☐
CUSTOMER_TABLE	<input checked="" type="checkbox"/>
CPA_STRING	☐
CPA_STRING	☐

- e. [CUSTOMER TABLE] オプションを選択します。[Fill] をクリックすると選択したテーブル名が自動的に入力されます。

図 E-15 [Add Event] ページ

On this page, you add events to your application view

Unique Event Name:

Description:

Table Name: [Syntax Help](#)

Syntax Help... CLOUDSCAPE: APP.TABLENAME, ORACLE: SCHEMA.TABLENAME, MS SQLSERVER: catalog.schema.tablename, SYBASE: catalog.schema.tablename

Please Select The Type Of Event To Create:

Insert Event

Update Event

Delete Event

- f. [Insert Event] オプションを選択します。
- g. [Add] をクリックします。[Application View Administration] ページが表示されます。

図 E-16 [Application View Administration for AppViewTest] ページ



8. アプリケーションビューをデプロイするための準備を行います。
[Application View Administration] ページには 1 つの格納場所が表示されます。ここで、アプリケーションビューを保存またはデプロイする前にコンテンツを確認できます。このページで、次のことを行うことができます。

- アプリケーションビューの説明の確認または編集
- アプリケーションビューの接続条件の確認または再コンフィグレーション
- サービスおよびイベントの削除
- アプリケーションビューの保存またはサーバへのアプリケーションビューのデプロイ

アプリケーションビューのパラメータを確認し、[Continue] をクリックします。
[Deploy Application View to Server] ページが表示されます。

9. アプリケーションビューをデプロイします。デプロイするには、**enable asynchronous service invocation**、イベント ルータ URL および接続プールパラメータを含むいくつかのパラメータを定義する必要があります。

図 E-17 [Display Application View to Server] ページ



アプリケーションビューをデプロイする手順は次のとおりです。

- a. [Enable Asynchronous Service Invocation] チェックボックスを必ず選択します。
- b. [Event Router URL] フィールドに次のように URL を入力します。
`http://localhost:7001/DbmsEventRouter/EventRouter`
- c. [Connection Pool Parameters] では、デフォルト値をそのまま使用します。
[Minimum Pool Size] – 1
[Maximum Pool Size] – 10
[Target Fraction of Maximum Pool Size] – 0.7
[Allow Pool to Shrink] (選択)

- d. [Log Configuration] フィールドで [Log warnings, errors, and audit messages] を選択します。
 - e. [Deploy persistently?] を必ずチェックします。
 - f. [Restrict Access] リンクをクリックします。[Application View Security] 画面が表示されます。
10. アプリケーションビューのパーミッションを設定します。あらゆるユーザまたはグループに対して、読み込みおよび書き込みアクセスを許可または無効にできます。

図 E-18 [Application View Security] ページ

アプリケーションビューのパーミッションを設定する手順は次のとおりです。

- a. [Choose an Action] で [Revoke] オプションを選択します。
- b. [Specify a User or Group] に `jdbc` と入力します。
- c. [Permission] で [Write (Deploy/Undeploy/Edit App View)] オプションを選択します。
- d. [Done] をクリックします。[Deploy Application View] 画面が表示されます。

- e. [Deploy] をクリックします。
11. アプリケーション ビューのデプロイが完了すると、デプロイされたアプリケーション ビューに関するすべての情報が [Summary for Application View] ページに表示されます。このページではスキーマ、イベント概要、サービス概要の表示、サービスとイベントのテスト、およびアプリケーション ビューのアンデプロイを実行できます。

☒ E-19 [Summary for Application View] ページ



12. イベントをテストします。アプリケーション ビューが正常に機能するかを確認するため、アプリケーション ビューのイベントおよびサービスをテストできます。イベントをテストするには、サービスを呼び出すか、手動でイベントを作成します。また、アプリケーションがイベントを受け取るまでの待機時間を指定することもできます。
- a. [Event] グループの [CustomerInserted] ライン上で [Test] をクリックします。[Test Event] 画面が表示されます。

図 E-20 [Test Event] ページ

Test Event: Customerportal

Application View Console ViewLog Console [Home](#) [Logout](#)

Summary

This page allows you to test an event. You may create the event by invoking a service, or by manually creating the event.

If you want to use a service invocation to create an event, select the Service option below, and select the service to invoke. Optionally, you can create the event manually using any tools your EIG provides (for example an interactive SQL tool for the DBMS adapter used to insert a new row to create an insert event).

How do you want to create the event?

Service:

Manual

How long should we wait to receive the event?

Time (in milliseconds):

- b. [Test Event] ページで [Service] オプションを選択し、次にサービスメニューから [InsertCustomer] を選択します。
- c. [How long should we wait to receive the event?] フィールドに 6000 と入力します。
- d. [Test] をクリックします。[Test Service] 画面が表示されます。

図 E-21 [Test Service] ページ

Test Service: insertCustomer

Application View: create PMS: Log: create Home Log

Customer

Please fill in any inputs to the service query and click Test.

Test Service: insertCustomer on application view 'AppViewTest'
insert into APP_CUSTOMER_TABLE (FIRSTNAME, LASTNAME, DOB) VALUES (:FIRSTNAME VARCHAR, :LASTNAME VARCHAR, :DOB DATE);

Input

FIRSTNAME text

LASTNAME text

DOB date (yyyy-MM-dd hh:mm:ss), e.g. 2002-12-01 04:52:29-05:00

Test

- e. [FIRSTNAME] フィールドに、名前を入力します。
- f. [LASTNAME] フィールドに、苗字を入力します。
- g. [DOB] フィールドに、生年月日を入力します。[DOB] フィールドの右側に表示されるフォーマットに従ってください。
- h. [Test] をクリックします。[Test Result] ページが表示されます。このページに、生成したイベント、およびアプリケーション サーバによって生成された応答を表す XML ドキュメントのコンテンツが表示されます。

図 E-22 [Test Result] ページ



The screenshot displays the 'Test Result for CustomerInserted' page. It features a 'Summary' sidebar on the left and a main content area with three sections of XML data. The top section shows the generated event XML, the middle section shows the input XML for the 'InsertCustomer' service, and the bottom section shows the output XML from the service. The page includes a 'bea' logo in the top right corner and navigation links for 'Overview' and 'Logoff'.

Test Result for CustomerInserted

Application View Console PMLLogic Console

Summary

This page shows the results from testing a event.

Generated event of type CustomerInserted on application view AppViewTest

```
<?xml version="1.0"?>
<!DOCTYPE CUSTOMER_TABLE.insert>
<CUSTOMER_TABLE.insert>
  <ADDRESS1></ADDRESS1>
  <ADDRESS2></ADDRESS2>
  <ADDRESS3></ADDRESS3>
  <CITY></CITY>
  <COUNTRY></COUNTRY>
  <DOB>2001-09-12 06:07:19</DOB>
  <EMAIL></EMAIL>
  <FA2></FA2>
  <FIRSTNAME>Jane</FIRSTNAME>
  <LASTNAME>Doe</LASTNAME>
  <MIDDLENAME></MIDDLENAME>
  <PHONE></PHONE>
</CUSTOMER_TABLE.insert>
```

Input to service InsertCustomer on application view AppViewTest

```
<?xml version="1.0"?>
<!DOCTYPE Input>
<Input>
  <FIRSTNAME>Jane</FIRSTNAME>
  <LASTNAME>Doe</LASTNAME>
  <DOB>2001-10-08 04:27:24-05:00</DOB>
</Input>
```

Output from service InsertCustomer on application view AppViewTest

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


DBMS サンプル アダプタの開発工程

この節では DBMS サンプル アダプタを開発する際に使用するインタフェースについて説明します。ADK では Java コネクタ アーキテクチャに準拠したアダプタに必要な実装の多くがありますが、EIS およびその環境を定義するまで完全に実装されないインタフェースもあります。このため、ADK で提供される抽象クラスの具象実装として、DBMS サンプル アダプタが作成されました。

DBMS サンプル アダプタの作成手順は以下のとおりです。

- 手順 1 : DBMS サンプル アダプタについて
- 手順 2 : 環境の定義
- 手順 3 : Server Provider Interface パッケージの実装
- 手順 4 : Common Client Interface パッケージの実装
- 手順 5 : イベント パッケージの実装
- 手順 6 : DBMS サンプル アダプタのデプロイ

手順 1 : DBMS サンプル アダプタについて

ADK で提供される実装が DBMS サンプル アダプタでどのように使用されるかについては、Javadoc およびこの節で後述するメソッドのコードを参照してください。

- Javadoc については、以下を参照してください。

`WLI_HOME/adapters/dbms/docs/api/index.html`

- このパッケージのコード リストについては、以下を参照してください。

`WLI_HOME/adapters/dbms/src/com/bean/adapters/dbms/spi`

注意： `WLI_HOME` は、WebLogic Integration がインストールされているドライブまたはディレクトリです。

手順 2 : 環境の定義

アダプタの開発者は、コーディングを開始する前に、アダプタ設定ワークシート（附録 D 「アダプタ設定ワークシート」を参照）を使用して、開発対象のアダプタに関する重要情報を確認および収集できます。DBMS サンプル アダプタの場合、ワークシートの質問に対する答えは以下のようになります。

注意： 先頭にアスタリスク (*) が付いた質問は、Generate Adapter Template ユーティリティを使用する必要があります。

1. * アダプタの開発に使用する EIS の名前は何ですか？

PointBase、SQLServer、Oracle、または Sybase データベース。

2. * 使用する EIS のバージョンは何ですか？

PointBase 4.0、MSSQLServer 7.0、Oracle 8.1.6 または Sybase 11.9.2。

3. * 使用する EIS のタイプ (DBMS または ERP など) は何ですか？

DBMS です。

4. * アダプタのベンダ名は何ですか？

BEA です。

5. * アダプタのバージョン番号は何ですか？

ありません (サンプル用)。

6. * アダプタの論理名は何ですか？

BEA_WLS_DBMS_ADK です。

7. アダプタで EIS 内部の機能呼び出す必要がありますか？

はい。

その必要がある場合、アダプタはサービスをサポートします。

はい。

8. 外部プログラムから EIS の機能呼び出せるようにするために、EIS ではどのようなメカニズムまたは API を提供していますか？

JDBC

9. このメカニズムで *EIS* に対するセッションまたは接続を設定するのにどのような情報が必要ですか？

データベース URL、ドライバクラス、ユーザ名、パスワードが必要です。

10. 特定のサービスについて *EIS* で呼び出す機能を特定するときに、どのような情報が必要ですか？

機能名、executeUpdate、executeQuery が必要です。

11. *EIS* では特定の機能に対する入出力の要件のために、*EIS* をクエリすることができますか？

はい。データ構造を参照できます。

可能な場合、サービスの入力要件を特定するのにどのような情報が必要になりますか？

SQL です。

12. 入力要件のうち、どの要件にも出てくる静的な情報は何ですか？アダプタでは、このような静的な情報を *InteractionSpec* オブジェクトにエンコーディングする必要にあります。

SQL です。

13. 入力要件のうち、各要件に対する動的な情報は何ですか？お使いのアダプタはサービスに必要な入力パラメータを示す XML スキーマを要求ごとに指定する必要があります。

サービスの SQL 式により、入力要件は異なります。

14. サービスの出力要件を特定するのに必要な情報は何ですか？

なし

15. *EIS* ではアダプタから呼び出せる機能のカタログを参照するメカニズムがありますか？その場合、アダプタでサービスをサポートします。

はい。

16. *EIS* の内部で起きる変更に対する通知を受け取る必要がありますか？その必要がある場合、アダプタはイベントをサポートします。

はい。

17. 外部プログラムから *EIS* のイベント通知を受け取るために、*EIS* ではどのようなメカニズムまたは *API* が提供されていますか？ この質問の回答によって、プル型とプッシュ型のいずれのメカニズムを開発すべきかを定めることができます。

なし。DBMS サンプルアダプタは、プルメカニズムを使用して、WebLogic Integration イベントジェネレータ上に構築されています。

18. *EIS* では、お使いのアダプタがサポートできるイベントを特定できますか？

はい。

19. *EIS* では特定のイベントに対するメタデータをクエリできますか？

はい。

20. お使いのアダプタでサポート可能な（言語および国によって定義される）ロケールは何ですか？

複数のロケールのサポートが必要です。

手順 3 : Server Provider Interface パッケージの実装

DBMS サンプルアダプタの Server Provider Interface (SPI) を実装し、J2EE 準拠の SPI 要件を満たすため、ADK のクラスが拡張され、以下の具象クラスが作成されました。

表 E-1 SPI クラスの拡張

具象クラス	拡張された ADK クラス
ManagedConnectionFactoryImpl	AbstractManagedConnectionFactory
ManagedConnectionImpl	AbstractManagedConnection
ConnectionMetaDataImpl	AbstractConnectionMetaData
LocalTransactionImpl	AbstractLocalTransaction

これらのクラスにより、EIS への接続性が提供されるため、イベント リストや要求送信用のフレームワークの構築、およびトランザクションの境界設定を行い、該当する EIS を管理できます。

ManagedConnectionFactoryImpl

DBMS サンプルアダプタ用の SPI を実装するためにまず ManagedConnectionFactory インタフェースを実装しました。ManagedConnectionFactory は ManagedConnection インスタンスの照合および作成メソッドを提供し、接続プーリングをサポートします。

基本実装

ADK で提供される `com.bea.adapter.spi.AbstractManagedConnectionFactory` は、Java コネクタ アーキテクチャ インタフェース `javax.resource.spi.ManagedConnectionFactory` を実装したものです。DBMS サンプルアダプタでは `com.bea.adapter.dbms.spi.ManagedConnectionFactoryImpl` でこのクラスが拡張されています。コード リスト E-1 は `ManagedConnectionFactoryImpl` の派生ツリーを示しています。

コードリスト E-1 `com.bea.adapter.dbms.spi.ManagedConnectionFactoryImpl`

```
javax.resource.spi.ManagedConnectionFactory
|
|-->com.bea.adapter.spi.AbstractManagedConnectionFactory
|
|   |-->com.bea.adapter.dbms.spi.ManagedConnectionFactoryImpl
```

開発者のコメント

ManagedConnectionFactory は Java コネクタ アーキテクチャ SPI パッケージの中心クラスです。ADK の AbstractManagedConnectionFactory は Sun Microsystems のインタフェースで宣言されているメソッドに必要な実装の多くを提供しています。この ADK の AbstractManagedConnectionFactory を DBMS サンプルアダプタ向けに拡張するため、キーメソッドである

`createConnectionFactory()` および `createManagedConnection()` が実装されました。また、`equals()`、`hashCode()`、`checkState()` のオーバーライドも記述され、DBMS サンプル アダプタによってデータベース固有の動作もサポートされるようになりました。

スーパークラスが認識できないプライベート属性があります。アダプタを作成する際、そのような属性に対してセッター/ゲッター メソッドを指定する必要があります。抽象メソッドの `createConnectionFactory()` が実装されているため、入力パラメータを使用して EIS 固有の `ConnectionFactory` を提供できます。

また、`createManagedConnection()` は、クラスの主要なファクトリ メソッドです。このメソッドにより、アダプタが正しくコンフィグレーションされているかどうか最初にチェックされます。次に、スーパークラスのメソッドが実装され、接続およびパスワード資格オブジェクトが取得されます。この処理が正常に実行されると、物理データベース接続が開始され、インスタント化の後、物理接続に割り当てられている `ManagedConnectionImpl` (`ManagedConnection` の DBMS サンプル アダプタ実装) が返されます。

その他のメソッドは、メンバー属性用のゲッター/セッター メソッドです。

ManagedConnectionImpl

`ManagedConnection` インスタンスは管理環境のベースになる EIS への物理接続を表します。`ManagedConnection` オブジェクトは、アプリケーション サーバによってプールされます。詳細については、6-34 ページの「`ManagedConnection`」で、ADK による `AbstractManagedConnection` インスタンスの実装方法に関する箇所を参照してください。

基本実装

ADK で提供される `com.bea.adapter.spi.AbstractManagedConnection` は、J2EE インタフェース `javax.resource.spi.ManagedConnection` を実装したものです。DBMS サンプル アダプタでは `com.bea.adapter.dbms.spi.ManagedConnectionImpl` でこのクラスが拡張されています。コード リスト E-2 は `ManagedConnectionImpl` の派生ツリーを示しています。

コードリスト E-2 com.bea.adapter.dbms.spi.ManagedConnectionImpl

```

javax.resource.spi.ManagedConnection
|
|-->com.bea.adapter.spi.AbstractManagedConnection
|
|   |-->com.bea.adapter.dbms.spi.ManagedConnectionImpl

```

開発者のコメント

ManagedConnection は Java コネクタアーキテクチャ SPI 仕様の重要箇所であるため、このクラスの詳細は Javadoc コメント内に記載されています。以下のメソッドの実装を重点的に参照してください。

- `java.lang.Object.getConnection(javax.security.auth.Subject subject, javax.resource.spi.ConnectionRequestInfo connectionRequestInfo)`
- `protected void destroyPhysicalConnection(java.lang.Object objPhysicalConnection)`
- `protected void destroyConnectionHandle(java.lang.Object objHandle)`
- `boolean compareCredentials(javax.security.auth.Subject subject, javax.resource.spi.ConnectionRequestInfo info)`

`ping()` メソッドを使用すると、物理データベース接続（`cci` 以外の `Connection`）が有効かどうかチェックできます。例外が発生すると、`ping()` によって、例外のタイプが詳しくチェックされるため、接続が必要以上に切断されることはありません。

その他のメソッドは、EIS 固有のメソッド、または単にセッター/ゲッターを必要とするメソッドです。

ConnectionMetaDataImpl

ManagedConnectionMetaData インタフェースは、ManagedConnection インスタンスに関連付けられている基本となる EIS インスタンスに関する情報を提供します。アプリケーションサーバでは、この情報に基づいて、接続対象の EIS イン

スタンスに関する実行時情報が取得されます。詳細については 6-34 ページの「ManagedConnection」で、ADK による `AbstractConnectionMetaData` インスタンスの実装方法に関する説明を参照してください。

基本実装

ADK で提供される `com.bea.adapter.spi.AbstractConnectionMetaData` は J2EE インタフェース `javax.resource.spi.ManagedConnectionMetaData` を実装したものです。DBMS サンプルアダプタでは、`com.bea.adapter.dbms.spi.ConnectionMetaDataImpl` で、このクラスが拡張されています。コード リスト E-3 は `ConnectionMetaDataImpl` の派生ツリーを示しています。

コードリスト E-3 `com.bea.adapter.dbms.spi.ConnectionMetaDataImpl`

```
javax.resource.spi.ManagedConnectionMetaData
|
|-->com.bea.adapter.spi.AbstractConnectionMetaData
|
|   |-->com.bea.adapter.dbms.spi.ConnectionMetaDataImpl
```

開発者のコメント

ADK の `AbstractConnectionMetaData` は以下のクラスを実装しています。

- `javax.resource.cci.ConnectionMetaData`
- `javax.resource.spi.ManagedConnectionMetaData`

`ConnectionMetaData` クラスの実装には、`DatabaseMetaData` オブジェクトが使用されます。ADK の抽象的実装が使用されたため、このクラスに抽象メソッドを実装する場合 EIS 固有の情報を指定する必要があります。

LocalTransactionImpl

LocalTransaction インタフェースは、EIS リソース マネージャの内部で管理されているトランザクションのサポートを提供します（つまり、外部トランザクション マネージャを必要としないトランザクション）。詳細については、6-37 ページの「LocalTransaction」で、ADK による AbstractLocalTransaction インスタンスの実装方法に関する説明を参照してください。

基本実装

ADK で提供される `com.bea.adapter.spi.AbstractLocalTransaction` は J2EE インタフェース `javax.resource.spi.LocalTransaction` を実装したものです。DBMS サンプルアダプタでは、`com.bea.adapter.dbms.spi.LocalTransactionImpl` でこのクラスが拡張されています。コード リスト E-4 は `LocalTransactionImpl` の派生ツリーを示しています。

コードリスト E-4 `com.bea.adapter.dbms.spi.LocalTransactionImpl`

```
javax.resource.spi.LocalTransaction
|
|-->com.bea.adapter.spi.AbstractLocalTransaction
|
|-->com.bea.adapter.dbms.spi.LocalTransactionImpl
```

開発者のコメント

このクラスでは、ロギングおよびイベント通知を提供する ADK の抽象スーパークラスが使用されます。このスーパークラスは、Sun が提供する CCI と SPI LocalTransaction インタフェースの両方を実装しています。DBMS サンプルアダプタの具象クラスは、以下に示すスーパークラスの 3 つの抽象メソッドを実装しています。

- `doBeginTx()`
- `doCommitTx()`
- `doRollbackTx()`

手順 4 : Common Client Interface パッケージの実装

DBMS サンプル アダプタの Common Client Interface (CCI) を実装し、J2EE 準拠の CCI 要件を満たすため、ADK のクラスが拡張され、以下の具象クラスが作成されました。

表 E-2 CCI クラスの拡張

具象クラス	拡張された ADK クラス
ConnectionImpl	AbstractConnection
InteractionImpl	AbstractInteraction
InteractionSpecImpl	InteractionSpecImpl

これらのクラスにより、バックエンド システムにアクセスできます。クライアント インタフェースは、EIS との対話における要求および応答レコードのフォーマットを指定します。

注意： *Java コネクタ アーキテクチャ 1.0* の仕様では CCI の実装は省略可能になっていますが、将来的には必須になる予定です。参考として DBMS サンプル アダプタは詳細な実装例を提供しています。

ConnectionImpl

Connection は、クライアントが基盤となる物理接続にアクセスするときに使用する、アプリケーションレベルのハンドルを表します。Connection インスタンスに割り当てられる実際の物理接続は ManagedConnection インスタンスで表されます。詳細については 6-39 ページの「Connection」で ADK による AbstractConnection インスタンスの実装方法に関する説明を参照してください。

基本実装

ADK で提供される `com.bea.adapter.cci.AbstractConnection` は、J2EE インタフェース `javax.resource.cci.Connection` を実装したものです。DBMS サンプルアダプタでは `com.bea.adapter.dbms.cci.ConnectionImpl` でこのクラスが拡張されています。コード リスト E-5 は `ConnectionImpl` の派生ツリーを示しています。

コードリスト E-5 `com.bea.adapter.dbms.cci.ConnectionImpl`

```
javax.resource.cci.Connection
|
|-->com.bea.adapter.cci.AbstractConnection
|
|-->com.bea.adapter.dbms.cci.ConnectionImpl
```

開発者のコメント

`ConnectionImpl` クラスは `javax.resource.cci.Connection` interface の DBMS サンプルアダプタの具体的な実装です。このクラスにより ADK の `AbstractConnection` クラスが拡張されます。接続インスタンスに割り当てられる実際の物理接続は `ManagedConnection` インスタンスで表されます。

クライアントでは、`ConnectionFactory` インスタンス上で `getConnection()` メソッドを使用して接続インスタンスが取得されます。接続はゼロまたは 1 つ以上の対話インスタンスと関連付けることができます。この具象クラスが単純であるため、ADK の基本クラスの拡張が可能になります。

InteractionImpl

`Interaction` インスタンスにより、コンポーネントから EIS 機能を実行できます。対話インスタンスが接続に基づいて作成され、`Connection` インスタンスとの関係が維持されます。詳細については 6-40 ページの「Interaction」で、ADK による `AbstractInteraction` インスタンスの実装方法に関する説明を参照してください。

基本実装

ADK で提供する `com.bea.adapter.cci.AbstractInteraction` は J2EE インタフェース `javax.resource.cci.Interaction` を実装したものです。DBMS サンプルアダプタでは `com.bea.adapter.dbms.cci.InteractionImpl` でこのクラスが拡張されています。コード リスト E-6 は `InteractionImpl` の派生ツリーを示しています。

コードリスト E-6 `com.bea.adapter.dbms.cci.InteractionImpl`

```
javax.resource.cci.Interaction
|
|-->com.bea.adapter.cci.AbstractInteraction
|
|-->com.bea.adapter.dbms.cci.InteractionImpl
```

開発者のコメント

`InteractionImpl` クラスは、ADK の対話オブジェクトの具象実装です。このメソッドは、Java コネクタ アーキテクチャおよび ADK で必要とされるメソッドを EIS 独自の方法で実装したものです。

Java コネクタ アーキテクチャの `javax.resource.cci.InteractionExecute()` メソッド（このクラスを中心メソッド）の 2 つのバージョンが DBMS サンプルアダプタに対し実装されました。`execute()` メソッドのメイン ロジックでは、次のシグネチャが使用されます。

```
public Record execute(InteractionSpec ispec, Record input)
```

このメソッドは対話からの実際の出力を返すため、他のメソッドに比べより頻繁に呼び出されます。

第 2 の実装は、便利なメソッドとして提供されます。`execute()` のフォームでは、次のシグネチャが使用されます。

```
public boolean execute(InteractionSpec ispec, Record input, Record output)
```

第 2 の実装のロジックでは、対話の成否のみを示すブールが返されます。

InteractionSpecImpl

InteractionSpecImpl には EIS インスタンスとの対話を実行するためのプロパティが保存されます。InteractionSpec は基本の EIS で特定の機能を実行するために使用されます。

CCI 仕様では InteractionSpec の標準プロパティが定義されていますが、そのプロパティが基本の EIS に適応しない場合、InteractionSpec を実装して標準プロパティをサポートする必要はありません。

InteractionSpec 実装クラスは、サポートされる各プロパティに対してゲッターメソッドおよびセッターメソッドを提供する必要があります。ゲッターおよびセッターメソッドで採用される規則は Java Beans 設計パターンに従う必要があります。詳細については 6-52 ページの「InteractionSpec」で、ADK による InteractionSpecImpl インスタンスの実装方法に関する説明を参照してください。

基本実装

ADK で提供される com.bea.adapter.cci.InteractionSpecImpl は J2EE インタフェース javax.resource.cci.InteractionSpec を実装したものです。

DBMS サンプルアダプタでは com.bea.adapter.dbms.

cci.InteractionSpecImpl でこのクラスが拡張されています。コードリスト E-7 は InteractionSpecImpl の派生ツリーを示しています。

コードリスト E-7 com.bea.adapter.dbms.cci.InteractionSpecImpl

```
javax.resource.cci.InteractionSpec
|
|-->com.bea.adapter.cci.InteractionSpecImpl
|
|-->com.bea.adapter.dbms.cci.InteractionSpecImpl
```

開発者のコメント

InteractionSpec インタフェースの実装クラスには java.io.Serializable インタフェースの実装が必要です。InteractionSpec により ADK InteractionSpec が拡張され、文字列属性 m_sql に対するゲッターおよびセッ

ター メソッドが追加されます。ゲッターおよびセッター メソッドは *Java* コネクタアーキテクチャ 1.0 の仕様で指定されている *Java Beans* 設計パターンに従う必要があります。

手順 5 : イベント パッケージの実装

このパッケージには、DBMS サンプルアダプタの `EventGeneratorWorker` クラスのみが収録されています。このクラスは、DBMS サンプルアダプタのイベントジェネレータに対して機能します。

EventGenerator

`EventGenerator` クラスには以下のインタフェースが実装されています。

- `com.bea.wlai.event.IEventGenerator`
- `java.lang.Runnable`

基本実装

DBMS サンプルアダプタのイベントジェネレータにより ADK の `AbstractPullEventGenerator` が拡張されます。これは、データベースがイベントジェネレータに対して情報をプッシュできないためです。このため、通知の対象とする変更点をプルするか、実際にポーリングする必要があります。コードリスト E-8 は `EventGenerator` の派生ツリーを示したものです。

コードリスト E-8 EventGenerator

```
com.bea.adapter.event.AbstractEventGenerator
|
|-->com.bea.adapter.event.AbstractPullEventGenerator
|
|-->com.bea.adapter.dbms.event.DbmsEventGeneratorWorker
```

開発者のコメント

この ADK の `AbstractPullEventGenerator` を実装すると以下の抽象メソッドが実装されます。

- `protected abstract void postEvents(IEventRouter router) throws Exception`
- `protected abstract void setupNewTypes(List listOfNewTypes)`
- `protected abstract void removeDeadTypes(List listOfDeadTypes)`

また、以下のメソッドがオーバーライドされます。

- `void doInit(Map map)`
- `void doCleanUpOnQuit()`

これらのメソッドは EIS 固有で、データベースとの対話中に EIS のコンテンツ内のイベントを識別するために使用され、イベント定義やイベントが作成および削除されます。また、エラーが発生した場合、これらのメソッドを使用してデータベースで実行されるトリガを作成および削除できます。

このクラスのキーメソッドは `postEvents()` です。このメソッドはデータベースの **EVENT** テーブル内の列から読み込まれたデータの `IEvent` オブジェクトを作成します。このメソッドでは `IEventRouter` が引数として使用されます。`IEventDefinition` オブジェクトの `createDefaultEvent()` メソッドを使用して `IEvent` が作成されると、`postEvents()` がイベントデータを設定し、`router.postEvent(event)` を呼び出してイベントがルータに伝えられます。イベントがルータに送られるとデータベースからイベントデータの該当する列が削除されます。

メソッド `setupNewTypes()` はデータベースに対して適切なトリガが作成されていることを確認して新しいイベント定義を作成します。このメソッドは各イベント定義ごとにトリガ情報オブジェクトを作成します。このオブジェクトにはイベント定義で表されるカタログ、スキーマ、テーブル、トリガタイプ、およびトリガキーが記述されます。トリガキーのマップが保存されるため、トリガが重複してデータベースに追加されることがありません。マップ内に新しいキーが存在しない場合はデータベースに対するトリガテキストが生成されます。

`removeDeadTypes()` メソッドもトリガ情報オブジェクトを作成しますが、このオブジェクトは、1 つ以上のイベントタイプが一致しているかどうかの検査も行います。このトリガと一致するすべてのイベント定義は、マップから削除され、次にデータベースからトリガ自身が削除されます。

手順 6 : DBMS サンプル アダプタのデプロイ

SPI、CCI、イベント インタフェースを実装したときに、以下の手順に従ってアダプタをデプロイします。

- 手順 6a : 環境のセットアップ
- 手順 6b : ra.xml ファイルの更新
- 手順 6c : RAR ファイルの作成
- 手順 6d : JAR および EAR ファイルの構築
- 手順 6e : EAR ファイルの作成とデプロイ

手順 6a : 環境のセットアップ

アダプタを WebLogic Integration 環境にデプロイする前に、ご使用のコンピュータでのアダプタの場所を決定します。アダプタは、`WLI_HOME/adapters/dbms`にあります。`WLI_HOME`を、WebLogic Integration がインストールされているディレクトリのパス名に置き換えます。以後、この格納場所を `ADAPTER_ROOT` と呼びます。

手順 6b : ra.xml ファイルの更新

DBMS サンプル アダプタでは、アダプタの RAR ファイル (`META-INF/ra.xml`) の中に、`ra.xml` ファイルが組み込まれています。DBMS サンプル アダプタでは、`AbstractManagedConnectionFactory` クラスが拡張されているため、以下のプロパティが `ra.xml` ファイルで定義されます。

- `LogLevel`
- `LanguageCode`
- `CountryCode`
- `MessageBundleBase`
- `LogConfigFile`
- `RootLogContext`
- `AdditionalLogContext`

DBMS サンプルアダプタには、以下の表に示す宣言も必要です。

表 E-3 ra.xml のプロパティ

プロパティ	例
UserName	DBMS サンプルアダプタのログインユーザ名
Password	ユーザ名に対するパスワード
DataSourceName	JDBC 接続プールの名前

DBMS サンプルアダプタの詳細な ra.xml ファイルは、次に示すディレクトリで参照できます。

`WLI_HOME/adapters/dbms/src/rar/META-INF/`

手順 6c : RAR ファイルの作成

クラスファイル、ロギング コンフィグレーション情報、メッセージバンドルは JAR ファイルに収集される必要があります。次にこのファイルと `META-INF/ra.xml` を RAR ファイルに統合します。Ant `build.xml` ファイルは、この RAR ファイルの適切な構成を示しています。

手順 6d : JAR および EAR ファイルの構築

JAR および EAR ファイルを構築するには、次の手順を実行します。

1. テキスト エディタで、`WLI_HOME/adapters/utils` にある `antEnv.cmd` (Windows) または `antEnv.sh` (UNIX) を編集します。以下の変数に有効なパス名を割り当てます。
 - `BEA_HOME` - BEA 製品のトップレベルディレクトリ
 - `WLI_HOME` - Application Integration ディレクトリの場所
 - `JAVA_HOME` - Java Development Kit の場所
 - `WL_HOME` - WebLogic Server ディレクトリの場所
 - `ANT_HOME` - `WLI_HOME/adapters/utils` といった Ant インストールの場所

2. コマンドラインから `antEnv` を実行し、必要な環境変数の新しい値を有効にします。
3. `WLI_HOME/adapters/dbms/project` に移動します。
4. `WLI_HOME/adapters/dbms/project` ディレクトリから `ant release` を実行して、アダプタを構築します。

手順 6e : EAR ファイルの作成とデプロイ

DBMS サンプル アダプタは、EAR ファイルを作成後、デプロイを実行して表示できます。手順は、次のとおりです。

1. 以下のリストで示すように、アダプタの EAR ファイルをドメインの `config.xml` ファイルで宣言します。

コードリスト E-9 DBMS サンプル アダプタの EAR ファイルの宣言

```
<!-- EAR ファイルのデプロイ -->
<Application Deployed="true" Name="BEA_WLS_DBMS_ADK"
Path="WLI_HOME/adapters/dbms/lib/BEA_WLS_DBMS_ADK.ear">
    <ConnectorComponent Name="BEA_WLS_DBMS_ADK" Targets="myserver"
        URI="BEA_WLS_DBMS_ADK.rar"/>
    <WebAppComponent Name="DbmsEventRouter" Targets="myserver"
        URI="BEA_WLS_DBMS_ADK_EventRouter.war"/>
    <WebAppComponent Name="BEA_WLS_DBMS_ADK_Web" Targets="myserver"
        URI="BEA_WLS_DBMS_ADK_Web.war"/>
</Application>
```

注意： `WLI_HOME` は、使用する環境の **WebLogic Integration** のインストールディレクトリのパス名に置き換えます。

2. 以下のように入力して、**WebLogic Server Administration Console** を起動します。

```
http://host:port/console
```

この URL で、`host` はサーバの名前、`port` はリスン ポートをそれぞれ表します。例：

`http://localhost:7001/console`

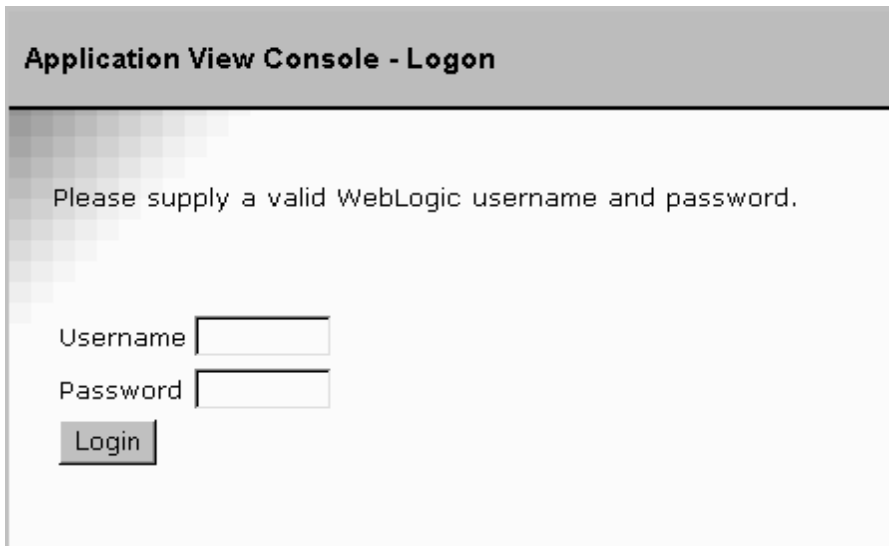
3. WebLogic Server Administration Console で、以下を行います。
 - a. アダプタ グループをデフォルトの WebLogic Server セキュリティレルムに追加します。
 - b. ユーザをアダプタ グループに追加します。
 - c. 変更を保存します。
4. アプリケーション ビューをコンフィグレーション、およびデプロイするには、次の場所に移動します。

`http://host:port/wlai`

この URL で、*host* はサーバの名前、*port* はリスン ポートをそれぞれ表します。例：

`http://localhost:7001/wlai`

[Application View Console - Logon] ページが表示されます。



Application View Console - Logon

Please supply a valid WebLogic username and password.

Username

Password

5. 該当するフィールドにユーザ名とパスワードを入力し、WebLogic Integration にログオンします。

6. 『*Application Integration ユーザーズ ガイド*』の「アプリケーションビューの定義」に示す手順に従って、アプリケーションビューをコンフィグレーションし、デプロイします。

DBMS サンプル アダプタの設計時 GUI の開発工程

設計時 GUI は、ユーザがアプリケーションビューの作成、サービスとイベントの追加、および WebLogic Integration 環境に搭載されているアダプタのデプロイを実行できるインターフェースです。この節では、DBMS サンプルアダプタに対して設計時 GUI がどのように開発されるかについて説明します。

- 手順 1 : 要件の決定
- 手順 2 : 必要な Java Server Pages の決定
- 手順 3 : メッセージバンドルの作成
- 手順 4 : 設計時 GUI の実装
- 手順 5 : Java Server Pages の記述

手順 1 : 要件の決定

DBMS に対する設計時 GUI の開発を始める前に、以下のパラメータの値を決定する必要があります。

- サポートするデータベース。
- サポートするブラウジングレベルの数。
- DBMS スキーマ生成の決定。
- アダプタがサービスおよびイベントのテストをサポートする必要があるか。

手順 2 : 必要な Java Server Pages の決定

DBMS サンプルアダプタは、設計時 GUI に ADK に付属する Java Server Pages (JSP) を使用します。アダプタ固有の機能を提供するために、追加 JSP が提供されています。追加 JSP を以下の表に示します。

表 E-4 追加の ADK JSP

ファイル名	説明
addevent.jsp	[Add Event] ページでは新しいイベントをアプリケーションビューに追加できる。
addservice.jsp	[Add Service] ページでは新しいサービスをアプリケーションビューに追加できる。
browse.jsp	[Browse] ページではロジックフローが処理され、DBMS サンプルアダプタのブラウズウィンドウに表示される。この機能はこのアダプタ専用が開発されたが設計時インタフェースとベースのアダプタ間の非常に一般的な対話を表す。 [Browse] ページは、ADK の <code>AbstractDesignTimeRequestHandler</code> を拡張する <code>DesignTimeRequestHandler</code> (ハンドラ) を呼び出す。DBMS サンプルアダプタの参照機能を理解するための最適な方法は、アダプタをデプロイし、Web ブラウザを使用して設計時のフレームワークにアクセスすること。
confconn.jsp	[Confirm Connection] ページは、EIS の接続パラメータを指定するためのフォーム。

表 E-4 追加の ADK JSP (続き)

ファイル名	説明
testform.jsp	<p>[Testform] ページは、ADK の testsrcvc.jsp ページに組み込まれている (<jsp:include page='testform.jsp'/>)。この対話には InteractionSpec が使用され、サービスの SQL が表示される。次に、必要なユーザ入力データを集めてサービスをテストするためのフォームが作成される。</p> <p>フォームを作成するために、Testform はハンドラのアプリケーションビューから RequestDocumentDefinition を取得し、それを .jsp Writer と共に com.bea.adapter.dbms.utils.TestFormBuilder と呼ばれるユーティリティクラスに渡す。このクラスが、フォームを作成する。</p>

手順 3 : メッセージバンドルの作成

テキストラベル、メッセージ、および例外のインターナショナルライゼーションをサポートするため、DBMS サンプルアダプタでは、テキストプロパティファイルに基づくメッセージバンドルが使用されます。プロパティファイルには、BEA_WLS_SAMPLE_ADK プロパティファイルからコピーされた名前と値の組み合わせがあり、新しいエントリが DBMS アダプタ固有のプロパティに追加されています。

DBMS サンプルアダプタのメッセージバンドルは、`WLI_HOME/adapters/dbms/src` ディレクトリに格納されています。これは、ADK と一緒にインストールされています。詳細については、同じディレクトリ内の `BEA_WLS_DBMS_ADK.properties` ファイルを参照してください。

メッセージバンドルの作成方法の詳細については、次に示すサイトにある JavaSoft チュートリアルを参照してください。

<http://java.sun.com/docs/books/tutorial/i18n/index.html>

手順 4 : 設計時 GUI の実装

設計時 GUI を実装するには `DesignTimeRequestHandler` クラスを作成する必要があります。このクラスによりフォームからのユーザ入力を受け付けられ、設計時アクションの実行メソッドが提供されます。`DesignTimeRequestHandler` の詳細については、8-31 ページの「手順 4 : 設計時 GUI の実装」を参照してください。

DBMS サンプルアダプタのパブリッククラス `DesignTimeRequestHandler` により `AbstractDesignTimeRequestHandler` が拡張され、次の表に示すメソッドが提供されます。

表 E-5 DBMS サンプルアダプタの設計時 GUI 用メソッド

メソッド	説明
<code>browse(java.lang.String dbtype, com.bea.connector.DocumentRecord input)</code>	<code>addservc.jsp</code> と <code>addservc.jsp</code> ページの参照機能に対するバックエンド動作を処理する。
<code>getAdapterLogicalName()</code>	アダプタの論理名を返し、アプリケーションビューなどのエンティティがデプロイされた場合に親クラスをサポートする。
<code>getManagedConnectionFactoryClass()</code>	EIS に対する CCI 接続を取得するために親クラスで使用されるアダプタの <code>SPI ManagedConnectionFactory</code> 実装クラスを返す。
<code>supportsServiceTest()</code>	このアダプタが設計時にサービスのテストをサポートしていることを示す。
<code>initServiceDescriptor(ActionResult result, IServiceDescriptor sd, HttpServletRequest request)</code>	サービスのリクエストや応答の作成に関するサービス記述子を初期化する。通常、EIS に対する対話を実行し、メタデータを読み込んで、そのメタデータを XML スキーマに変換する。 そのため、アダプタの CCI インタフェースが使用された。このメソッドは、 <code>AbstractDesignTimeRequestHandler</code> の <code>addsrvc</code> メソッドによって呼び出される。

表 E-5 DBMS サンプル アダプタの設計時 GUI 用メソッド (続き)

メソッド	説明
<code>initEventDescriptor(ActionResult result, IEventDescriptor ed, HttpServletRequest request)</code>	<p>イベント記述子を初期化する。イベント記述子ではアプリケーションビューのイベントに関する情報が定義される。サブクラスではこのメソッドの実装方法を定義する必要がある。</p> <p>イベントがサポートされていない場合、UnsupportedOperationException が呼び出される。ただし、イベント名と定義の有効性が検証され、アプリケーションビューに対する既存のイベントでないことが確認されてから、このメソッドが (AbstractDesignTimeRequestHandler によって) 呼び出される。</p>
<code>GetDatabaseType()</code>	<p>使用するデータベース管理システムのタイプを決定する。WebLogic Integration は PointBase、Oracle、Microsoft SQL Server、および Sybase をサポート。</p>

手順 5 : Java Server Pages の記述

開発プロセスに以下のような実践を含めることを検討してください。

- カスタム JSP Tags の使用
- オブジェクトのステートの保存
- WEB-INF/web.xml のデプロイメント記述子の記述

カスタム JSP Tags の使用

Java Server Pages (JSP) は `display.jsp` ページに表示されるため、`display.jsp` は最初に記述する必要のある `.jsp` ファイルです。ADK にはカスタム JSP タグのライブラリが実装されています。これらのタグは ADK および DBMS サンプルアダプタの Java Server Pages 全体を通して使用されます。これらのタグは、検証機能を追加したり、ユーザがクリックしたときにフィールド値を保存するなど、さまざまな機能をサポートしています。

オブジェクトのステートの保存

ADK を使用してアダプタ向けに JSP を記述する際、オブジェクトのステートを保存する必要が生じる場合があります。これにはいくつかの方法があります。AbstractDesignTimeRequestHandler には編集されているアプリケーションビューの ApplicationViewDescriptor が保持されます。このファイルからのハンドラの呼び出しが効率的であるため、このファイルは、しばしばステートを保存するのに最善の場所となります。

別の選択肢として、従来のメソッド (`getApplicationManager()`、`getSchemaManager()` および `getNamespaceManager()`) で

AbstractDesignTimeRequestHandler から **Manager Bean** を要求して、アプリケーションビューに関する情報をリポジトリから検索することもできます。このメソッドには時間がかかりますが、必要になる場合もあります。また JSP などでセッション オブジェクトも使用できます。ただし、セッション内のすべてのオブジェクトを明示的に `java.io.Serializable` インタフェースに実装する必要があります。

WEB-INF/web.xml のデプロイメント記述子の記述

WEB-INF/web.xml のデプロイメント記述子を記述します。通常は、アダプタの web.xml ファイルを開始点として使用し、必要に応じてコンポーネントを変更します。このアダプタの web.xml ファイルを参照するには、次に示すディレクトリに移動してください。

```
WLI_HOME/adapters/dbms/src/war/WEB-INF/web.xml
```

詳細については次に示す BEA WebLogic Server 製品 マニュアルを参照してください。

```
http://edocs.beasys.co.jp/e-docs/index.html
```

索引

A

AbstractConnection 6-40
AbstractConnectionFactory A-2
AbstractConnectionMetaData A-2
AbstractDesignTimeRequestHandler 1-6,
8-31, 8-32, 8-34, 8-38
abstractDesignTimeRequestHandler 8-1
AbstractDocumentRecordInteraction 6-47
AbstractInputTagSupport 8-5
AbstractInteraction 6-41
AbstractLocalTransaction 6-38
AbstractManagedConnection 6-35, 6-40, A-2
AbstractManagedConnectionFactory A-2
AbstractManagedConnectionMetaData 6-35
AbstractPullEventGenerator 7-10, 7-12, 7-14
AbstractPushEventGenerator 7-12
ActionResult 8-4
addevent.jsp 8-39
addservc 8-34
addservc.jsp 8-41
ADK 1-2
adk-eventgenerator.jar 7-10, 7-11
ADK タグ ライブラリ 8-39, 8-41
Ant 3-4, 4-5, 4-6, 6-57, 7-6, 8-29
 使用理由 3-4
antEnv 4-6
antEnv.cmd 4-5
antEnv.sh 4-5
ANT_HOME 4-6, E-41
ant release 4-6
Apache Software Foundation 5-2
Apache プロジェクト 2-5, 5-2, 7-7
Application Integration 1-3
Application View Management Console 1-3
[Application View Summary] ページ
8-32

B

BEA_HOME 4-5, E-41
build.xml 6-57

C

CCI 6-35, 6-38, 6-39, 6-43, 6-45, 6-51, 6-52,
6-54, 6-55, 6-56, 6-58, 6-60, 8-32,
8-35
chmod u+x ant 4-6
com.bea.adapter.cci.Abstract
 DocumentRecordInteraction 6-56
com.bea.adapter.cci.AbstractDocumentReco
 rdInteraction 6-47
com.bea.adapter.cci.AbstractInteraction 6-47
com.bea.adapter.cci.DesignTimeInteractionS
 pecImpl 6-49
com.bea.adapter.cci.DocumentDefinitionRec
 ord 6-47
com.bea.adapter.cci.ServiceInteractionSpecI
 mpl 6-48
com.bea.adapter.event 7-11
com.bea.adapter.spi.AbstractConnectionMet
 aData 6-51
com.bea.adapter.spi.ConnectionEventLogger
 6-36
com.bea.adapter.spi.NonManagedConnectio
 nEventListener 6-36
com.bea.adapter.spi.NonManagedConnectio
 nManager 6-37
com.bea.adapter.test.TestHarness 6-57
com.bea.connector.DocumentRecord 6-45
com.bea.document.IDocument 6-45, B-2
com.bea.web.ActionResult 8-4
com.bea.web.ControllerServlet 8-4
com.bea.web.RequestHandler 8-3

com.bea.web.tag.AbstractInputTagSupport
8-5
com.bea.web.tag.IntegerTagSupport 8-6
com.bea.web.validation.IntegerWord 8-6,
8-8
com.bea.web.validation.Word 8-4, 8-5, 8-6
Common Client Interface、CCI を参照
confconn.jsp 8-32, 8-35
config.xml 4-6
Connection 6-39
ConnectionEventListener 6-36
ConnectionFactory 6-50
ConnectionFactory.getMetaData 6-56
ConnectionFactoryImpl 6-50
ConnectionManager 6-36, 6-37, A-2
ConnectionMetaData 6-51
ConnectionRequestInfo 6-37
ConnectionSpec 6-51, 6-52
ControllerServlet 8-4, 8-6, 8-7, 8-8, 8-35,
8-37, 8-39, 8-42

D

DbmsEventGeneratorWorker.java 7-16
DesignTimeInteractionSpecImpl 6-49
DesignTimeRequestHandler 8-31
display.jsp 8-50
DisplayPage 8-37
DocumentDefinitionRecord 6-47
Document Object Model、DOM を参照
DocumentRecord 6-45, 6-47
DocumentRecordInteraction 6-48
DOM 5-2, 6-45, B-2

E

EAR ファイル 2-11, 2-12
error.jsp 8-50
EventGenerator 7-14, 7-17
EventMetaData 7-12
EventRouter 7-8, 7-15
ExecutionTimeout 6-53

F

FunctionName 6-52

G

GenerateAdapterTemplate 3-2, 3-3, 4-1, 5-2,
7-6, 8-47, A-2
GenerateAdapterTemplate.cmd 4-2
GenerateAdapterTemplate.sh 4-2
GUI 1-1

I

I18N 5-16
IDocument 3-5, 6-45, 7-13, B-2, B-3
IDocumentDefinition 6-48
IEventDefinition 7-9, 7-10, 7-13
ILogger 5-4
IndexedRecord 6-55
Interaction 6-39, 6-40, 6-48
InteractionSpec 6-41, 6-42, 6-48, 6-52
InteractionSpecImpl 6-53
InteractionVerb 6-52
IPushHandler 7-12

J

J2EE コネクタ アーキテクチャの仕様 -xvi
Jakarta プロジェクト 7-7
JAR ファイル 2-12
Java 2-6
JavaBean 6-51, 6-52
Javadoc 3-3, 4-6
JAVA_HOME 4-5, E-41
java.io.Serializable 6-50
Java Reflection 8-9, 8-38
JavaScript ライブラリ 1-6, 2-3
JavaServer Page、JSP を参照
java.util.Map 6-37
javax.resource.cci.ResourceAdapterMetaDat
a 6-56
javax.resource.cci.Connection 6-39

javax.resource.cci.ConnectionFactory 6-50
javax.resource.cci.ConnectionMetaData
6-35, 6-51
javax.resource.cci.ConnectionSpec 6-51
javax.resource.cci.Interaction 6-40, 6-48
javax.resource.cci.InteractionSpec 6-52, 6-53
javax.resource.cci.LocalTransaction 6-54
javax.resource.cci.Record 6-45, 6-54, 6-56
javax.resource.cci.ResourceAdapterMetaDat
a 6-56
javax.resource.ReferenceableInterfaces 6-50
javax.resource.spi 6-24, 6-54
javax.resource.spi.ConnectionEventListener
6-36
javax.resource.spi.ConnectionManager 6-36,
6-37
javax.resource.spi.ConnectionRequestInfo
6-37
javax.resource.spi.LocalTransaction 6-37
javax.resource.spi.ManagedConnection 6-36
javax.resource.spi.ManagedConnectionMeta
Data 6-35
Java パッケージ基本名 4-4
Java 例外 8-2, 8-50
JNDI 6-50
JSP 1-6, 2-3, 8-1, 8-6, 8-7, 8-9, 8-34, 8-36,
8-50, 8-51
JSP テンプレート 1-6, 2-3
JUnit 6-57
junit.framework.TestCase 6-57
junit.framework.TestSuite 6-57

L

L10N 5-16
LocalTransaction 6-37, 6-54
Log4j 2-5, 5-2
log4j 5-2, 7-7
LogConfigFile 8-37

M

main.jsp 8-50

ManagedConnection 6-25, 6-35, 6-36, 6-39
ManagedConnectionFactory 6-25, 6-58,
8-33, 8-35, 8-38
ManagedConnectionImpl 6-35
ManagedConnectionMetaData 6-25, 6-35
ManagedConnectionMetaDataImpl 6-35
manifest ファイル 6-11
MappedRecord 6-55
MessageBundleBase 8-37

N

NDC 5-17
NonManagedScenarioTestCase 6-58

O

overview.html 4-6

P

PatternLayout 5-7
PushEvent 7-12, 7-13

R

RAR ファイル 2-11, 2-12
ra.xml 8-37
Record 6-41, 6-52, 6-54, 6-55
RecordImpl 6-56
RequestHandler 8-3, 8-4, 8-6, 8-7, 8-9, 8-35,
8-37, 8-40
RequestHandlerClass 8-37
ResourceAdapterMetaData 6-56
ResourceAdapterMetaDataImpl 6-56
RootLogContext 8-37

S

sample.client.ApplicationViewClient 6-59
sample.event.EventGenerator 3-3
sample.event.OfflineEventGeneratorTestCas
e 6-59

sample.spi.ConnectionMetaDataImpl 3-3
sample.spi.ManagedConnectionFactoryImpl
3-2
sample.spi.ManagedConnectionImpl 3-3
sample.spi.NonManagedScenarioTestCase
6-58
sample.web.DesignTimeRequestHandler 3-3
Schema Object Model、SOM を参照
Service Provider Interface、SPI を参照
SOM 3-5
SPI 6-35, 6-38, 6-58, 6-60, 8-33, A-2
[Submit] ボタン、フォームの表示 8-38,
8-40, 8-42

T

test.properties 6-57, 6-59
TestSuite 6-57

W

WAR ファイル 2-11
web.xml 2-3, 8-4, 8-37, 8-40
WebLogic 6.0 5-2
WebLogic Integration A-2
WebLogic Server -xvi, 5-6, A-2
WebLogic Server 6.0 A-2
web.xml 7-15, 8-47, 8-50
 セキュリティ制約のコンポーネント
 8-49
 ログイン コンフィグレーションのコン
 ポーネント 8-49
Web アプリケーション 2-3, 3-4, 8-2, 8-37,
8-47
 セキュリティ制約 8-49
Web アプリケーション記述子 2-3
WLAI_HOME E-41
WL_HOME 4-6, E-41
WLI_HOME 4-5
Word 8-4, 8-8

X

XA トランザクション A-2
XCCI 6-43, 6-47
 DocumentRecords 6-43
 サービス 6-43
 設計パターン 6-48
XERCES 5-2
XML 2-3
 スキーマ 1-4, 1-5, 2-3, 2-4, 3-5, 6-48,
 7-1, 7-5, 7-18, B-3
 ドキュメント 6-45, 7-18, B-2
 要求ドキュメント 1-4
XML スキーマの仕様 -xvi
XML ツール 3-5
XPath 6-45, B-2

あ

アサーション チェック 6-40
アダプタ 1-4, 1-6
 イベント 1-5, 1-7
 サービス 1-7
アダプタ設定ワークシート 4-1, 7-5
アダプタ、デプロイ 2-11
アダプタ論理名 2-6, 4-4, 5-2, 7-6, A-2
アプリケーション ビュー 1-5, 1-6, 1-7, 2-3,
8-1, 8-31, 8-32, 8-39
アプリケーション ビュー記述子 8-31
アプリケーション ビュー セキュリティ
8-32
アペンダ 5-6, 5-10

い

イベント アダプタの開発 7-1
イベント ジェネレータ 2-2, 7-8, 7-9, 7-10,
7-12
イベント リスナ 6-34
イベント ルータ 6-59
印刷、製品のマニュアル -xv
インストーラ 4-6
インターナショナルライゼーション 7-7, 8-4

え

エンタープライズ JavaBean (EJB) 1-5
エンタープライズ アダプタ アーカイブ
 ファイル 2-11
エンタープライズ情報システム (EIS) 1-4

お

応答ドキュメント定義 6-44

か

カスタマ サポート情報 -xvi
カスタム開発環境の作成 4-1
カテゴリ
 親 5-3
 子 5-3
 祖先 5-3
 複数アペンドの参照 5-6
 プロパティ 5-3
 命名 5-4
 優先度の割り当て 5-5
 ルート 5-4
関連情報 -xvi
 J2EE コネクタ アーキテクチャの仕様
 -xvi
 XML スキーマの仕様 -xvi

く

クラス
 抽象 3-1

け

検証 8-2

さ

サービス
 同期 1-4
サービス記述子 8-34
サービスプロバイダ インタフェース、SPI

を参照

サポート
 テクニカル -xvi
サンプル アダプタ 3-1, 3-3, 4-1, 6-35

し

実行時 2-1, 2-6, 8-51
実行時エンジン 2-2
出力想定 2-4
状態管理 6-40

す

スキーマ オブジェクト モデル、SOM を参
照

せ

設計時 2-1, 8-33
 GUI 1-6
設計時 GUI 1-1
接続 6-41
接続ファクトリ
 共有 2-8, 2-11

た

対話 6-41
対話設定 2-4
タグ ライブラリ 2-3

ち

抽象基本クラス 2-2

て

テキスト フィールド、サイズ の表示 8-38,
 8-40, 8-42
テスト支援機能 7-20
データ抽出 7-9
データ トランスフォーメーション 7-8,

7-18

デプロイメント記述子 8-47

デプロイメント ヘルパー 1-6, 2-3

と

ドキュメント オブジェクト モデル、DOM
を参照

トランザクション 2-4

トランザクション、XA A-2

トランザクション、ローカル A-2

に

入力要件 2-4

ね

ネームスペース 6-50

は

パッケージ形式 4-4

バリデータ 8-4, 8-5

ふ

フォーム処理 8-2

クラス 8-3

シーケンス 8-6

要件 8-6

プッシュ データ抽出 7-8, 7-10, 7-13

プル データ抽出 7-8, 7-9, 7-10, 7-13

フレームワーク 1-2

実行時 1-2, 2-1, 2-2

設計時 1-2, 1-6, 3-5, 8-1

パッケージ化 1-2, 1-7

ロギング 1-2, 2-2, 2-5, 2-6, 5-1, 5-3,
6-34, 6-36, 6-40, 6-41, 6-50

ま

マニュアル入手先 -xv

め

メタデータ 3-5, 6-35, 6-38, 6-44, B-3
セカンダリ 2-4

メッセージ バンドル 2-6, 6-24, 7-7, 8-37,
8-40

ゆ

優先度 5-4

ユニークなビジネス名 6-43

よ

要求ドキュメント定義 6-43

ら

ラベル、[Form] フィールドの表示 8-38,
8-40, 8-42

り

リソース アダプタ、アダプタを参照

れ

例外処理 6-38

ろ

ローカライゼーション 6-24, 7-7, 8-4, 8-5

ローカル トランザクション 6-54

ロギング 2-5, 5-1, 5-2, 7-7

AUDIT 5-4

DEBUG 5-5

ERROR 5-4

INFO 5-5

WARN 5-4

アペンダ 5-3, 5-6

インターナショナルライゼーション 2-6,
5-1, 5-2, 5-4, 8-4

カテゴリ 5-3, 7-7

コンセプト 5-2

メッセージレイアウト 5-3
優先度 5-3, 5-4
ローカライゼーション 2-5, 2-6, 5-1,
5-4, 8-4
ロギング コンフィグレーション ファイル
5-2
ロギング ツールキット 2-5, 5-2
ログ カテゴリ 2-6

