iPortal Application Server
開発者ガイド

日本アイオナテクノロジーズ
2001 年 9 月

iPortal Application Server is a Trademark of IONA Technologies PLC>
Orbix 2000 is a Trademark of IONA Technologies PLC
Orbix is a Registered Trademark of IONA Technologies PLC.

While the information in this publication is believed to be accurate, IONA Technologies PLC makes no warranty of any kind to this material including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. IONA Technologies PLC shall not be liable for errors contained herein, or for incidental or consequential damages in connection with the furnishing, performance or use of this material.
Java is a trademark of Sun Microsystems, Inc.

COPYRIGHT NOTICE

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise, without prior written consent of IONA Technologies PLC. No third party intellectual property right liability is assumed with respect to the use of the information contained herein. IONA Technologies PLC assumes no responsibility for errors or omissions contained in this book. This publication and features described herein are subject to change without notice.

Copyright © 2000 IONA Technologies PLC. All rights reserved.

All products or services mentioned in this manual are covered by the trademarks, service marks, or product names as designated by the companies who market those products.

書面による許可なしでの一切の複製を禁ずる。

iPortal Application Server 開発者ガイド日本語版

2001 年 9 月 1 日

M2708

日本アイオナテクノロジーズ株式会社

目次

はじめに

The IONA iPortal Application Server は J2EE（Java 2 Platform, Enterprise Edition）アプリケーションの開発、アセンブリ、そしてデプロイメントを行うためにデザインされた統合的なシステム環境です。本書では、iPortal Application Server 実行環境の各コンポーネントの設定ならびに管理方法に関する説明、また iPortal Application Server を使用した J2EE アプリケーションの開発、デプロイメント、ならびに管理方法についての説明が記述されています。

本章は、次のセクションで構成されます。

対象となる読者

本書は iPortal Application Server を使用した J2EE アプリケーション用コンポーネントの提供、J2EE アプリケーションのアセンブリ、ならびに J2EE アプリケーションのランタイム環境へのデプロイメントを行う開発者を対象としています。

必須事項

本書は読者の方々が Sun Microsystems による EJB 1.1 仕様に準拠した EJB（Enterprise JavaBeans）の開発方法を既に理解していることを前提として著されています。そのため、本書では EJB のプログラミングに関する解説は行われていません。

また、本書は読者の方々が XML（自己拡張可能マークアップ言語）の基本的な概念を理解していることを前提としています。

このガイドの構成

本書は、次の 6 つのパートならびに付録によって構成されています。

パート 1 :「iPortal Application Server の概要」

パート 1 には iPortal Application Server 環境の概要、ならびにその動作の仕組みを理解するために必要な基本的概念に関する説明がまとめられています。また、J2EE アプリケーション作成、デプロイ、そして管理のために使用する一連の開発ツール群、ならびに iPortal Application Server と相互運用することが可能なサードパーティ提供のツールに関する説明が記述されています。

パート 2 :「iPortal Application Server 準備と設定」

パート 2 には iPortal Application Server コンポーネントが正常に機能するための Java 環境の設定方法、ならびに異なったシステムに対応するための iPortal Application Server のデフォルト設定の変更に関する説明が記述されています。

パート 3 :「エンタープライズ・アプリケーションの開発」

パート 3 には J2EE エンタープライズ・アプリケーションの開発方法に関する説明が記述されています。開発ツール、アプリケーション・コンポーネント、アプリケーション・サービス、Java クライアント・プログラミング、トランザクション・サポート、リソース割当て、ならびにインターオペラビリティについて解説します。

パート 4 :「エンタープライズ・アプリケーションの設定」

パート 4 にはエンタープライズ・アプリケーションの各コンポーネントの必須条件に対応するための EJB コンテナの設定方法に関する説明が記述されています。EJB コンテナはエンタープライズ・アプリケーションのライフ・サイクル、セキュリティ、デプロイメント、およびランタイム・サービスの管理のために使用されます。

パート 5 :「J2EE アプリケーションの管理」

パート 5 には J2EE アプリケーションの管理方法に関する説明が記述されています。Java アプリケーションの管理を可能とするためのツール（IONA の iPortal Administrator 管理ツール、ならびに Sun の Java Management Extensions (JMX) API）に関する説明、iPortal Administrator によるアプリケーション管理を可能とする JMX API の使用方法、ならびにイベント～ Orbix 間での通信を行うための JMX ノーティフィケーションの使用方法に関する説明が記述されています。また、最新のリリースの iPortal Application Server によってサポートされている、更に高度な機能（XML、分散 HTTP サービス、ならびに Web サービスなど）に関する説明が記述されています。

パート 6 :「ユーザーズ・リファレンス」

パート 6 は、iPortal Application Server の使用に際して必要とされる情報を手軽に参照できるリファレンス・ガイドです。Java ユーティリティのコマンドライン・オプション、ならびに EARSCO（Enterprise ARchive Source Code Organization）のディレクトリ構造に関する情報がまとめてあります。

パート 7 :「付録」

付録 A :「コンテナ・コンフィギュレーション DTD」

コンテナ・コンフィギュレーション・ドキュメントの DTD（document type definition）に関する説明が記述されています。

付録 B :「トラブルシューティング」

iPortal Application Server 使用時に発生し得る問題および対処方法に関する説明が記述されています。

関連したドキュメンテーション

iPortal Application Server ドキュメンテーション

IONA Technologies 社は iPortal Application Server に関連する次のようなドキュメンテーションを用意しています。

- 『iPortal Application Server インストール・ガイド』
- 『iPortal Application Server リリースノート』
- 『iPortal Administrator ユーザーズ・ガイド』

IONA Technologies 社が発行するその他の英文ドキュメンテーションに関する情報は本社英文 Web サイト（**http://www.iona.com/**）、また和訳されたドキュメンテーションについての情報は日本アイオナテクノロジーズ Web サイト（**http://www.iona.com/**）をご覧ください。

データベース・ドライバ・ドキュメンテーション

iPortal Application Server には、一部のデータベースとの接続を可能とするために MERANT 社の SequeLink データベース・ドライバが含まれています。MERANT 社による SequeLink に関するドキュメンテーションは SequeLink と共にシステム上にインストールされています。

日本アイオナテクノロジーズによるその他の英文ドキュメンテーションに関する情報は本社英文 Web サイト（**http://www.iona.com/**）、また和訳されたドキュメンテーションについての情報は日本アイオナテクノロジーズ Web サイト（**http://www.iona.com/**）をご覧ください。

その他の関連情報

IONA アップデート・センター（英語）からは本製品およびその他全ての IONA Technologies 社製品の最新リリース版ならびにパッチを入手することができます。

http://www.iona.com/support/update/

Sun Microsystems 社 Web サイトからは J2EE ならびに Java 技術に関するその他の最新情報を入手することができます。

http://www.sun.co.jp/

J2EE では XML（自己拡張可能マークアップ言語）を利用しています。XML に関する公式の情報は W3C（World Wide Web コンソーシアム）Web サイトから入手することができます。

http://www.w3c.org/

Organization for the Advancement of Structured Information Standards (OASIS) による Web サイトからは XML に関する更に詳しい情報を入手することができます（英語）。

http://www.oasis-open.org/cover/sgml-xml.html

ドキュメンテーションの入手方法と製品サポート窓口

Orbix 関連のドキュメンテーションは定期的にアップデートされています。最新版の英文ドキュメンテーションは次の URL よりダウンロードできます。

http://www.iona.com/docs/

本製品およびその他全ての IONA Technologies 社製品のサポートに関するお問合わせ、また本書およびその他全ての IONA Technologies 社ドキュメンテーションに関するご意見ご希望は、次のアドレスまで電子メールにてお問い合わせください（日本語で受け付けています）。

support.japan@iona.com

また、日本アイオナテクノロジーズ社ウェブサイトでは関連技術、アイオナテクノロジーズ社製品、および技術セミナーなどの関連サービスに関する最新の情報を日本語で提供しています。

http://www.iona.co.jp/

表記規則と表記記号

表記規則

本書では次の表記規則を使用しています。

等幅クーリエ体 (Abcdef)	本文中の等幅クーリエ体文字による記述はコードの一部、および記述されたとおりに入力される必要のあるクラス、関数、変数、コマンドなどを表します。例えば、コマンドラインで入力するコマンドは次のように記述されます。 `java iportal.admin`
ゴシック体	本文中のゴシック体による記述は製品名、ドキュメント名、新規用語、あるいはユーザ・インターフェイスによって表示されるメッセージを表します。 ・製品名：Orbix ・ドキュメント名：『iPortal Application Server 開発者ガイド』 ・ユーザ・インターフェイス： スタートボタンをクリックし、プログラム→ IONA iPortal Application Server → iPortal Application Server Tools の順に選択します。
等幅クーリエ斜体 *(Abcdef)* **等幅クーリエ太字** **(Abcdef)**	本文中およびシンタックス中の等幅クーリエ斜体文字あるいは等幅クーリエ太字による記述はユーザが指定入力する必要のある変数（コマンドの引数あるいはパスなど）を表します。 ***install-dir*/etc/domains** **install-dir**/etc/domains

表記記号　本書では次の表記記号を使用しています。

プロンプト無し	コマンド・シェルにプロンプトが記載されていない場合、コマンドのフォーマットが複数のプラットフォームに共通であることを表します。
%	% 記号は UNIX のコマンド・シェル・プロンプトのうち、ROOT 権限を必要としないものを表します。
#	# 記号は UNIX のコマンド・シェル・プロンプトのうち、ROOT 権限を必要とするものを表します。
>	> 記号は DOS、および Windows 系 OS のコマンド・プロンプトを表します。
・・・ ・ ・ ・	フォーマットおよびシンタックスの記述中に水平あるいは垂直に「...」と記述されている場合、記述の簡略化のためにフォーマットおよびシンタックスの一部が省略されていることを表します。
[ ]	フォーマットおよびシンタックスの記述中にある角括弧は任意で選択可能な要素を囲みます。
{ }	フォーマットおよびシンタックスの記述中にある波括弧は選択肢を囲みます。
|	フォーマットおよびシンタックスの記述中にある波括弧内の垂直線 | は各選択肢を区切ります。

技術用語

本書で使用される次の用語の定義です。

Beans	エンタープライズ Java Bean の略気。 (EJB:Enterprise Java Beans)　JAR あるいは EAR ファイルに含まれる EJB はアプリケーションのビジネス・ロジックを含みます。
Web コンポーネント	エンタープライズ・アプリケーションのプレゼンテーション・ロジックを含むサーブレット、JavaServer Page、ならびに HTML ファイル。WAR ファイル内で提供され、EAR ファイルに組み込まれる。
コンポーネント	エンタープライズ・アプリケーションの EAR ファイルに組み込まれた EJB、サーブレット、JSP、ならびにその他の関連ファイル。

パート 1

iPortal Application Server の概要

パート 1 は次の章で構成されます。

第 1 章

iPortal ApplicationServer 製品の概要

iPortal Application Server はコンポーネント・ベースのエンタープライズ・アプリケーションを開発、デプロイ、実行するためのアプリケーション・サーバです。iPortal Application Server には J2EE（Java 2 Platform Enterprise Edition）アプリケーションのアセンブリをグラフィカルに行う開発ツールと、開発したアプリケーションをデプロイし実行するための標準ランタイム環境が用意されています。

本章は、次のセクションで構成されます。

J2EE（Java 2 Platform, Enterprise Edition）

概要

J2EE（Java 2 プラットフォーム・エンタープライズ・エディション）は、多層式のエンタープライズ・アプリケーションをサポートするテクノロジーを提供します。J2EE テクノロジーを使用することでアプリケーション開発の作業が容易になり、ロバストでスケーラビリティに優れたアプリケーションを開発できます。J2EE はオープン標準によるコンポーネントをベースとしたアプリケーション開発をサポートし、エンタープライズ・アプリケーションに必要なランタイム・サービスを実装しています。J2EE では一般的なアプリケーション・タスクが自動的に処理されるので、あまり複雑なコードを記述する必要はありません。

J2EE のアプリケーション・モデル

J2EE のコンポーネント・テクノロジーには EJB（Enterprise Java Bean）、サーブレット、および JSP（JavaServer Page）が含まれます。また J2EE サービスは、標準ネットワーク・プロトコル、データベース・システム、およびメッセージング・システムへのアクセスを提供します。J2EE コンポーネントは、Web コンテナまたは EJB コンテナ内で実行されます。**コンテナ**とは、各コンポーネントに必要なセキュリティ、デプロイメント、スレッドなどの低レベルのサービスを提供する環境を指します。

EJB にはアプリケーションのビジネス・ロジックが含まれています。またサーブレットや JSP などの Web コンポーネントには、アプリケーションのプレゼンテーション・ロジックが含まれています。

19 ページの図 1 に、J2EE のアプリケーション・モデルを示します。

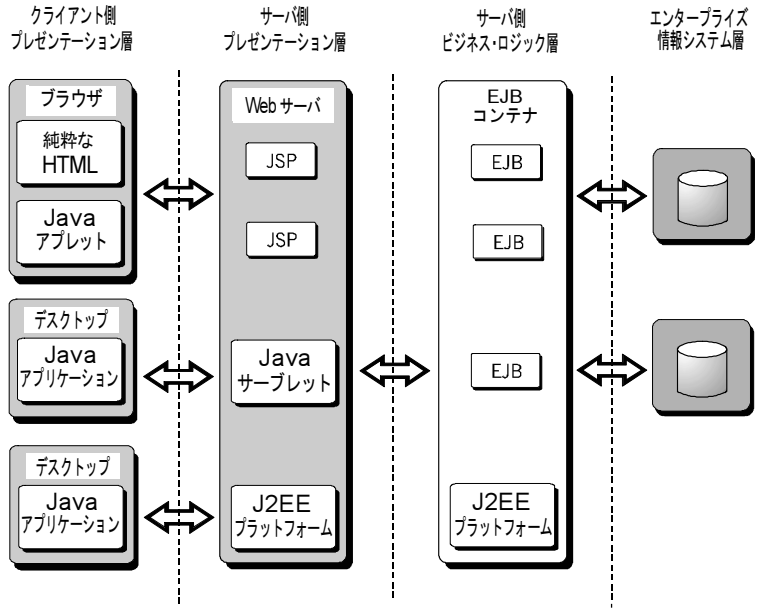


図 1:　J2EE アプリケーション・モデル

アプリケーション・モデルには次のアプリケーション層が含まれます。

- クライアント側のプレゼンテーション層
- サーバ側のプレゼンテーション層
- サーバ側のビジネス・ロジック層
- エンタープライズ情報システム（EIS：Enterprise Information System）層

クライアント側プレゼンテーション層

この層には、エンドユーザ・クライアントが含まれています。これらのクライアントがアクセスするサービスは、サーバ側コンポーネントから提供されます。クライアントには HTML ページやブラウザで実行されるアプレット、Java アプリケーション、そして Java と相互運用するその他のアプリケーションが含まれます。

サーバ側プレゼンテーション層

この層は動的な情報をサーバ側からクライアントへ提供します。ここには JSP や Java サーブレットなどのコンポーネントが含まれます。これらのコンポーネントは J2EE に準拠している Web サーバで動作します。

サーバ側ビジネス・ロジック層

この層はエンタープライズ・アプリケーションの中核となるビジネス・ロジックを提供します。JSP やサーブレットは、クライアントからのリクエストを処理するためにこの層が提供するサービスにアクセスします。ビジネス・ロジック層のコンポーネントには EJB が含まれ、これらは J2EE 準拠の EJB コンテナで動作します。

iPortal Application Server のコンテナは、エンタープライズ Bean のホストとしてライフ・サイクル管理や Bean が必要とするセキュリティ、デプロイメント、トランザクション管理など低レベルのサービスを提供します。EJB には次の 2 つのタイプがあります。

- **エンティティ Bean**

  エンティティ Bean はバックエンド・データベースに記録されている 1 つの永続データ・アイテムを表し、クライアントがデータにアクセスしてこれを処理するためのオペレーションを提供します。したがって永続データにアクセスするにはエンティティ Bean が必要です。例えばエンティティ Bean は、データベースに記録された 1 つのデータ・アイテムを表します。複数のクライアントが 1 つのエンティティ Bean に同時アクセスすることも可能です。データへのアクセスはコンテナがトランザクション機能を使用して制御します。

- **セッション Bean**

  セッション Bean はクライアントのためのサーバ側エージェントとして機能する一時オブジェクトで、クライアントの代わりにアクションを実行します。通常セッション Bean は、クライアントとサーバ側アプリケーション・ロジックとの間の単一セッション中のクライアントを表します。セッション Bean には次の 2 つのタイプあります。

  - **ステートフルなセッション Bean**

    ステートフルなセッション Bean には、クライアントとアプリケーション間の通信状態に関する情報が格納されていて、通常はクライアントに使用されている間のみ存在します。ただし EJB サーバは、タイムアウト期限が経過した場合などに**ステートフルなセッション Bean** を非アクティブまたは無効にすることができます。

  - **ステートレスなセッション Bean**

ステートレスなセッション Bean は呼出し間の状態は保管されません。

エンタープライズ情報システム層

この層にはビジネス・タスクを実装するために EJB コンポーネントがアクセスする、データベースおよびその他の情報システムが含まれています。

Web サーバや EJB コンテナは J2EE のいわば心臓部で、エンタープライズ・アプリケーションのサーバ側コンポーネントを実行する標準プラットフォームを提供します。

アプリケーション・サービス

概要

iPortal Application Server は基本的なサービスを提供するので、各コンポーネントは低レベルの実装の詳細に関与する必要がなくなり、ビジネス・ロジックの処理に集中できます。アプリケーション・サービスはネットワーキング、認証、永続性、および EJB とサーブレットのリモート・オブジェクト・アクセスのサービスを行います。アプリケーションが使用するその他のサービスへのポータブル・アクセスは、標準の Java API（Application Programming Interface）を介して提供されます。

データ・サービスおよびアクセス・サービス

iPortal Application Server はアプリケーションやコンポーネントにデータおよびアクセス・サービスを提供するため標準の J2EE 技術を実装しています。これらのサービスには次が含まれます。

- JNDI（Java Naming and Directory Interface）
- JDBC（Java Database Connectivity）
- JTA（Java Transaction API）

JNDI は、Java アプリケーションにネーミング・サービスを提供するアプリケーション・プログラミング・インターフェイス（API）です。iPortal Application Server では CORBA ネーミング・サービスを使用して J2EE アプリケーションをサポートします。JNDI は特定のネーミング・サービスやディレクトリ・サービスに依存せず、ファイルシステムなど各種システムへのアクセスに使用できます。

JDBC は標準の SQL データベース・アクセス・インターフェイスで、これによりさまざまなリレーショナル・データベースにアクセスできます。JDBC は、高レベルのツールやインターフェイスを構築できる共通のプラットフォームも提供します。

JTA は、アプリケーションとアプリケーション・サーバがトランザクションを使用できるようにする API です。トランザクション処理は、エンタープライズ・アプリケーションを開発するには重要な手法です。iPortal Application Server では、エンタープライズ・アプリケーションでのトランザクション使用を完全にサポートしています。

メッセージング・サービス

iPortal Application Server には次のメッセージング・サービスが用意されています。

- JMS（Java Message Service）
- JavaMail

JMS

JMS API は、JMS 対応の全メッセージング・システムでサポートされる共通のメッセージング概念とプログラミング手法を定義しています。JMS を使用すると、各アプリケーションがメッセージをやり取りして互いに通信することができます。

JavaMail

JavaMail API は標準の Java API を提供し、これにより標準のインターネット・メール・プロバイダとの通信が可能になります。

iPortal Application Server

概要

iPortal Application Server は、複数のコンポーネントから構成されています。サーバは J2EE に準拠したプラットフォームであり、アプリケーションを実行します。トランザクション処理、セキュリティ管理、データベース・アクセスなどの、エンタープライズ・アプリケーションが必要とするエンタープライズ・サービスの処理は、サーバが担当します。

サーバは CORBA（Common Object Request Broker Architecture）アーキテクチャに基づいて Orbix 2000 を使用して構築されています。Bean は標準の Java API を使用しており、Orbix には直接アクセスしません。Orbix はシステムに必要なロバストでスケーラビリティに優れた基幹を提供します。

1 つのシステムで実行できる iPortal Application Server の数に制限はありません。サーバの実行は、各種ランタイム・サービスによってサポートされます。例えばネーミング・サービスにより、Bean の登録時にサーバが JNDI を使用でき、トラ

ンザクション・サービスは分散トランザクション処理を可能にします。またロケーション・サービスは、コンテナや Bean がクライアントの操作に影響を与えることなくサーバ間を移動できるようにします。

iPortal Application Server のアーキテクチャ

iPortal Application Server には EJB サーバと Web サーバが含まれています。23 ページの図 2 に、各サーバの相互関係を示します。iPortal Application Server は J2EE に準拠するため次の仕様を実装しています。

- J2SE（Java 2 Platform Standard Edition）
- EJB（Enterprise JavaBean）1.1
- JSP（JavaServer Page）1.1
- Servlet API 2.2
- JDBC（Java Database Connectivity）2.0
- JNDI（Java Naming and Directory）1.2
- JTS、JTA（Java Transaction Service and API）1.0
- JMS（Java Message Service）1.0.2
- JavaMail 1.1
- JAF（Java Activation Framework）1.0

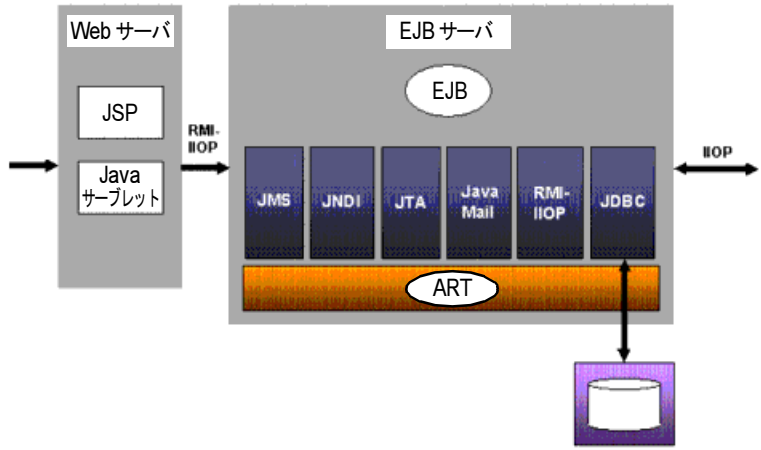


図 2:　iPortal Application Server のアーキテクチャ

アプリケーション・サーバ

iPortal Application Server は J2EE 認定のアプリケーション・サーバです。iPortal Application Server を使用してコンポーネント・ベースのエンタープライズ・アプリケーションを開発、デプロイ、実行することができます。

Web サーバ

Web サーバはサーブレット、JSP、HTML ファイル、およびその他の Web リソースの集まりで構成されます。Web リソースにはイメージ・ファイル、サウンド・ファイル、その他のデータが含まれます。Web コンポーネントにはアプリケーションのプレゼンテーション・ロジックがカプセル化されています。

EJB サーバ

EJB サーバは 1 つ以上の Bean を含み、これを実行します。Bean にはビジネス・ロジックと、各 EJB クライアントが共用するデータがカプセル化されています。Bean が EJB サーバと直接通信することはなく、EJB コンテナが Bean と EJB サーバ間のインターフェイスとして機能します。EJB コンテナを介して EJB サーバと Bean の通信を行うことで、EJB サーバによる Bean の管理がしやすくなります。

EJB コンテナは、EJB サーバがセキュリティ、トランザクション、および分散オブジェクト通信を管理するための支援をします。またエンティティBean の場合は EJB コンテナが Bean の永続性も管理します。

アプリケーション・デプロイメント

アプリケーションのデプロイメント・デスクリプタにより、アプリケーションの実行時に必要な諸条件を指定します。iPortal Application Server は標準の EJB 1.1 コンテナをホストしており、このコンテナにより、コンテナ管理永続性、JDBC を使用したデータ・アクセス、セキュリティ管理、およびトランザクション管理など、アプリケーションに必要な全てのアプリケーション・サービスを提供します。

エンタープライズ・アプリケーションをデプロイするには、iPortal Application Server に付属しているツールを使用してアプリケーションの必要条件をターゲットのオペレーティング環境にマッピングします。次の操作を行うツールが用意されています。

- アプリケーションのデプロイメント・デスクリプタで指定されているデータソース・リファレンスをオペレーティング環境の実際のデータソースにマッピングする。アプリケーションをデプロイする前に、正しいタイプのデータソースが使用できるか確認する必要があります。
- エンティティ Bean のコンテナ管理永続性を使用するフィールドを、データソースでこれに対応するフィールドにマッピングする。

- アプリケーション・アセンブラが指定したセキュリティ・ロールを、オペレーティング環境で使用されるセキュリティ・メカニズムまたはポリシー（ユーザとユーザ・グループなど）にマッピングする。
- アプリケーションで使用される Bean や Web コンポーネントの環境エントリを設定する。

iPortal Application Server がホストする各 EJB コンテナには、コンテナ・コンフィギュレーション・ファイルが関連付けられています（25 ページの図 3 を参照）。このファイルは XML（eXtensible Markup Language：拡張マークアップ言語）ドキュメントで、コンテナの各コンポーネントがオペレーティング環境を使用する方法が記述されています。

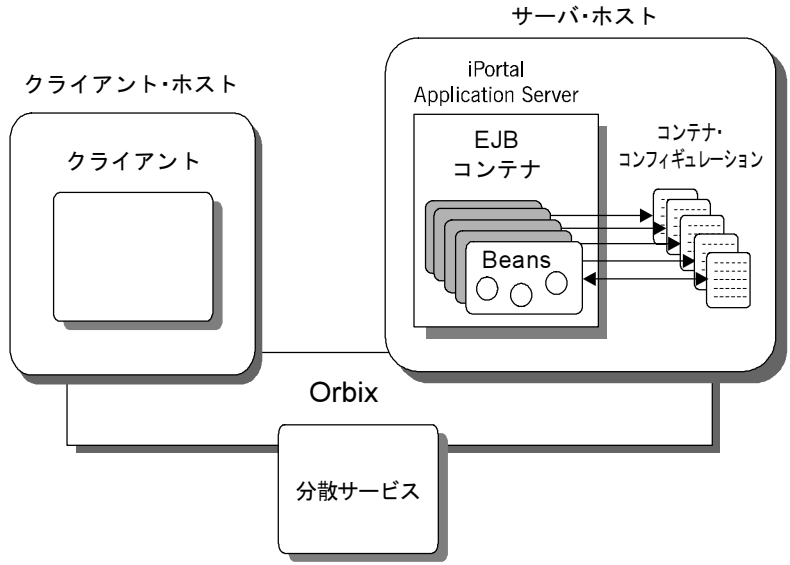


図 3:　EJB コンテナとそれに関連するコンフィギュレーション・ファイル

各 EJB コンテナのコンフィギュレーション・ファイルを作成し、新しいアプリケーションをインストールするたびにこれらのドキュメントを更新するのは、アプリケーション・デプロイヤの責任です。

アプリケーションの必須条件の把握

特定のアプリケーション用にコンテナを構成するには、そのアプリケーションの必須条件を理解しておく必要があります。アプリケーションはこれに必要な Bean と Web コンポーネントを含む EAR（Enterprise ARchive）ファイルとしてパッケージ化されています。このファイルにはデプロイメント・デスクリプタも含まれてい

ます。デプロイメント・デスクリプタは、各コンポーネントの必須条件を記述する XML ドキュメントです。アプリケーションのデプロイメント・デスクリプタの内容を確認するには次の 2 つの方法があります。

- 開発者用ツールキットを使用してアプリケーションを表示する。
- デプロイメント・デスクリプタを直接表示する。

iPortal Administrator

概要 iPortal Administrator は、コンポーネント・ベースの分散エンタープライズ・アプリケーションのデプロイ、構成、監視、および制御を可能にする、IONA 提供の一連のコンポーネントおよびアプリケーションです。iPortal Administrator は IONA の ART（Adaptive Runtime Technology）に統合されているので、IONA スイートに含まれる全製品、およびこれらの製品を使用して開発した全アプリケーションの管理をシームレスに行うことができます。

コンポーネント iPortal Administrator には、次の主要なコンポーネントが含まれています。

- iPortal Administrator Console
- iPortal Administrator Web Console
- iPortal Administrator 管理サービス

27 ページの図 4 に、iPortal Administrator で使用できるツールと、これらのツールが提供する管理機能を示します。

詳しくは『iPortal Administrator ユーザーズ・ガイド』を参照してください。

J2EE アプリケーション

概要 J2EE プラットフォーム仕様は、多層式のエンタープライズ・アプリケーションをサポートするテクノロジーを提供します。エンタープライズ・アプリケーションのビジネス・ロジックは、ユーザ・インターフェイスとシステム・サービスの間にあ

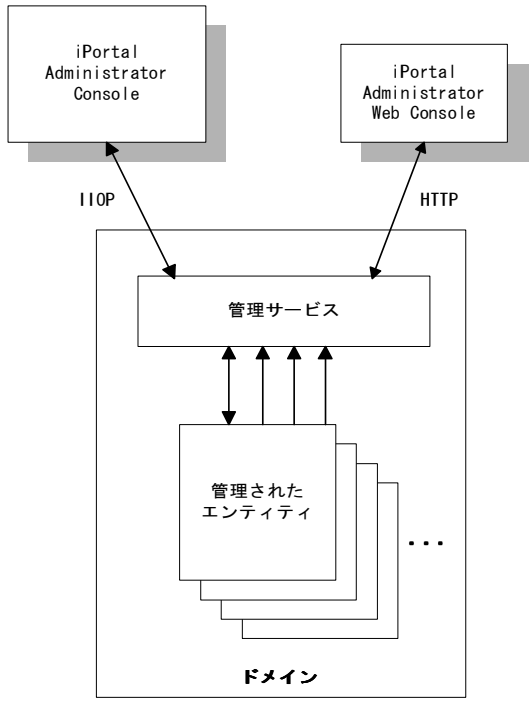


図 4: iPortal Administrator のコンポーネント

る中層に位置しています。ビジネス・ロジックとユーザ・インターフェイス・ロジックは開発者が実装しますが、標準のシステム・サービスは J2EE プラットフォームにより提供されます。

コンポーネント

エンタープライズ・アプリケーションは EJB（Enterprise Java Bean）と Web コンポーネントで構成されます。iPortal Application Server に用意されている開発者向けのツール一式を使用して、J2EE アプリケーションの作成、デプロイ、および管理を行うことができます。

EAR ファイル

エンタープライズ・アプリケーションは EAR ファイルというエンタープライズ・アーカイブ・ファイルにパッケージ化されています。このファイルには、アプリケーションで使用される各 Bean のコンパイル済みインターフェイスとクラス、各 Bean に関する情報を含むデプロイメント・デスクリプタ、アプリケーションで必

要な全 Web コンポーネント、その各 Web コンポーネントの情報を含むデプロイメント・デスクリプタ、各サーブレットのコンパイル済みクラス、およびアプリケーション全体に関するその他の情報がそれぞれ含まれています。

EARSCO フレームワーク

iPortal Application Server は、EARSCO（Enterprise ARchive Source Code Organization）フレームワークを使用します。これは、よりクリーンで管理しやすい J2EE アプリケーションの開発を目的とするフレームワークです。

iPortal Application Server にはエンタープライズ・アプリケーションの開発を支援するツールが多数用意されていますが、これらのツールは全て EARSCO 形式を認識します。開発者用ツールキットについて詳しくは、51 ページの「開発者用ツールキット」を参照してください。

エンタープライズ・アプリケーションのデプロイメント

J2EE アプリケーションのデプロイヤは、ターゲットのオペレーティング環境に合わせてデプロイメントを行う必要があります。また、オペレーティング環境にアプリケーション・コンポーネントで使用される全リソースが揃っているか確認する必要もあります。

各エンタープライズ・コンテナには、デプロイヤがアプリケーションのインストールと構成を行い、アプリケーション・コンポーネントとリソース間のリンクを確立するためのツールが用意されています。

開発者用ツールキット

概要

iPortal Application Server の本リリースには、J2EE アプリケーションの作成、デプロイ、管理を行うための開発者用ツールが一式用意されています。これら各作業を行う場合、まず `iportal.central` というツールを起動します。

`iportal.central` は、iPortal Application Server で使用できる全ツールへのショートカットを提供する便利なツールバーです。

使用できるツール

iPortal Application Server では次のツールを使用できます。

`iportal.appadmin` は、iPortal Application Server でエンタープライズ・アプリケーションの管理を行うために使用します。

iportal.assembler は、EAR ファイルの EJB や Web コンポーネントのデプロイメント・デスクリプタを更新するために使用します。

iportal.build は、EARSCO プロジェクトを EAR ファイルとしてビルドするために使用します。

iportal.client は、iPortal Application Server のシステム・サービスを提供するクライアント・コンテナです。

iportal.datasource は、iPortal Application Server でデータを構成するために使用します。

iportal.deploy は、iPortal Application Server にアプリケーションをデプロイするために使用します。

iportal.earsco は、エンタープライズ・アプリケーション・プロジェクト用に空の EARSCO ディレクトリ構造を作成するために使用します。

iportal.earscoify は、EARSCO フレームワークに EAR ファイルをインポートするために使用します。

iportal.server は、iPortal Application Server を起動するために使用します。

iportal.shutdown は、iPortal Application Server をシャットダウンするために使用します。

iportal.undeploy は、iPortal Application Server からアプリケーションをアンデプロイ（回収）するために使用します。

開発者用ツールキットについて詳しくは、51 ページの「開発者用ツールキット」および 239 ページの「コマンド・リファレンス」を参照してください。

第 2 章

サードパーティー提供のツール

本章では iPortal Application Server と一緒に使用できるサードパーティー提供のツールについて説明します。

本章は、次のセクションで構成されます。

JDK

概要

Java JDK は、Java プラットフォーム上で動作するアプリケーション、アプレット、およびコンポーネントを構築するための開発環境です。JDK には Java プログラミング言語で記述されていて Java プラットフォームで動作する、プログラムの開発とテストを行うための便利なツールが付属しています。これらのコマンドラインから起動します。

バージョン

iPortal Application Server を使用するにはバージョン 1.3 が必要です。それ以前のバージョンでは動作しません。

詳細情報

詳しくは、Sun のウェブサイト（**http://www.sun.com/**）を参照してください。

Forte for Java

概要

Forte for Java は Sun が提供する Java 開発者のための拡張可能な IDE（Integrated Development Environment：統合開発環境）です。NetBeans Tools Platform に基づく技術で、Sun ONE（Sun Open Net Environment）に統合されています。

アプリケーション開発とデバッグ作業の効率化

初めてのユーザでもアプリケーション・ブラウザ、Java 言語ソース・エディタ、デバッガ、コンパイラ、オンライン・ヘルプなど各種の開発者向けツールにアクセスするためのインターフェイスを簡単に使用できるよう、Forte for Java では統合された拡張可能な次の機能が用意されています。

- アプリケーションとアプレットの開発作業を効率化する各種ウィザード
- プロジェクト、クラス、ソースコードの統合管理機能

クラス・ブラウザを使用すると、ソースなしでクラス・ファイルの構造を表示できるので作業時間を短縮できます。また、シリアル化された JavaBeans の認識や新たなシリアル JavaBeans の作成や、クラスおよびシリアル化されたファイルの操作を

行うこともできます。例えば JAR ファイルに記録されているクラス・ライブラリを調べたり、ソースのないクラスからメソッドやフィールドなどをコピーして Java ソースにシグニチャを貼り付けることができます。

HTML ブラウザは、ブラウジング・インターフェイス、オンライン・ヘルプ、およびチュートリアルにアクセスするために使用します。Java 言語のみで記述されているこの組込みブラウザにより、プロジェクトのテストを行えるだけでなく、IDE を終了しなくても Forte for Java チュートリアルやユーザ・ドキュメンテーションなどオンライン・ヘルプのソースにも素早くアクセスできます。

デバッガには、条件付きブレークポイント、ブレークポイント・ログ、エバリュエータ、コードのステップ実行の各機能が備わっているので、バグの発見と修正を効率よく行えます。また、マルチプロセスの同時デバッグ機能により、分化されたアプリケーションの開発も簡単に行えます。さらにリモート・デバッガを使用すれば複雑な分散アプリケーションのデバッグが可能となるほか、マルチプラットフォームのデバッグ機能を使用して複数のプラットフォームにわたる Java アプリケーションのデバッグを行うこともできます。

iPortal Application Server の機能拡張

Forte for Java を開発プラットフォームとしてサポートするには、専用の拡張する機能をインストールして IDE を iPortal Application Server に統合する必要があります。この機能のダウンロードについては **http://www.iona.com/docs/ffj/update.html** をご覧ください。

詳細情報

詳しくは、Sun のウェブサイト（**http://www.sun.com/**）を参照してください。

JDBC ドライバ

概要

Oracle 8i、Merant Sequelink、および Informix Cloudscape は、JDBC 2.0 ドライバです。

データソース

JDBC 2.0 の拡張 API では、新たにデータソースという概念が導入されました。これは使用するデータベースやその他のリソースを指定するための標準の汎用オブジェクトを指します。データソースは JNDI（Java Naming and Directory Interface）のエンティティにバインドすることもでき、その場合は論理名によりデータベースにアクセスできるので、より便利でポータビリティの高いアプリケーション開発が可能となります。

データソースは、以前の JDBC DriverManager の機能と比べて一貫性かつ柔軟性に優れています。データソースと JNDI についての詳しい情報は、Sun Microsystems の JDBC 2.0 Optional Package 仕様を参照してください。

コネクション・プーリング

JDBC コネクションを始めとするデータベース・アクセス・メソッドは、特に中層のサーバ環境で JDBC API が使用されている場合に大量の処理能力を消費します。このような環境では、要求されるごとに繰り返しコネクションを確立するのではなく、既存のコネクションを解除せずに再利用することによりパフォーマンスが大幅に向上します。コネクションを再利用するには、データベース・コネクションのメモリ・キャッシュをコネクション・プールとして使用し、標準の JDBC ドライバの上の層として、コネクション・プーリング・モジュールによりこのコネクション・プールを管理します。

SequeLink Server と iPortal Application Server の使用

概要

iPortal Application Server には、Cloudscape データベースが同梱されています。このデータベースは iPortal Application Server と同時にインストールされますが、Cloudscape テーブルは本製品の付属例に合わせて設定されています。その他のデータベース・サポートは Sequelink 5.1 が提供しています。JDBC 2.0 との互換性が確認されている各種データベースやプラットフォーム用のドライバは、Sequelink から入手可能です。

必須条件

必要なものは次のとおりです。

- Sequelink Server 5.1
- Sequelink JDBC クライアント JAR（`sljc_brand.jar`、`sljcx.jar`、および `spy.jar`）。これらは iPortal Application Server インストール・ディレクトリ内の `lib` ディレクトリに保存されます。
- iPortal Application Server 3.0

注： iPortal Application Server は、データソースを使用する際これを独自のデータソースに「ラッピング」します。register datasource ウィザードを使用すると、指定したデータソースが iPortal Application Server のデータソースにより自動的にラッピングされます。Sequelink は iPortal Application Server によってのみ使用でき、Sequelink データソースを使用するにはこれをラッピングする必要があります。ラッピングを行う方法については iPortal Application Server インストール・ディレクトリ内の `util` ディレクトリにある `RegisterDataSource.java` を参照してください。

手順 register datasource ウィザードを使用して SequeLink データソースのラッピングを行うには、次の手順を行います。

1. Sequelink Server 5.1 をインストールする。詳細は iPortal Application Server Installation Guide を参照。

2. クラスパスに Sequelink JDBC ドライバを追加する。

3. `java iportal.datasource` ツールを使用してデータソースを登録する。このツールの使用方法は 63 ページの「データソースの登録」を参照。register datasource ウィザードの手順 3 で Sequelink JDBC データベース・ドライバ `com.merant.sequelink.jdbcx.datasource.SequeLinkDataSource` を選択する。

4. データソースをコンテナ・コンフィギュレーションに適用する。データソースの登録時に指定した JNDI プロバイダ、JNDI URL およびデータソース名を使用するよう `cc.xml` ファイルを編集する。この操作が必要なのは、コンテナ・コンフィギュレーションを register datasource ウィザードで変更しなかった場合、または新しいコンテナ・コンフィギュレーション・ファイルを作成している場合のみ。

   Sequelink データソースを `account` デモで使用するには、コンテナ・コンフィギュレーションにエンティティを追加する必要がある。次は `SimpleDemos` にあるコンテナ・コンフィギュレーションの場合の例。

```
<resource-name>JDBCAccounts</resource-name>
 <jndi-name>MySqlinkDS</jndi-name>
 <property>
    <prop-name>java.naming.factory.initial</prop-name>
<prop-value>com.sun.jndi.fscontext.RefFSContextFactory</prop-val
   ue>
 </property>
 <property>
```

```
  <prop-name>java.naming.provider.url</prop-name>
  <prop-value>file:/c:/ProgramFiles/iportal/etc/jndi-
   bindings</prop-value>
 </property>
```

5. アプリケーションでデータソースを使用する。**ejb-jar.xml** ファイルでエンティティ Bean について次のエンティティを宣言する。

```
<enterprise-beans>
  <entity>
    <resource-ref>
        <res-ref-name>jdbc/Accounts</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>
  </entity>
</enterprise-beans>
```

これでデータソースの検索と使用が可能となる。

```
//Java
InitialContext ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("java:comp/env/"+
   jdbc/Accounts);
Connection con = ds.getConnection();
// Do some database activities using the connection...
con.close();
```

Cloudview を使用したデータベースの管理

Cloudview を使用してデータベースを管理するには、次の手順を行います。

1. クラスパスに Cloudscape の JAR（**cloudscape.jar** および **cloudtools.jar**）を追加する。

2. **java COM.cloudscape.tools.cview** を実行する。

SonicMQ

概要

SonicMQ 3.0 は、インターネット上でビジネスに不可欠なデータのやり取りを確実に行うための、e ビジネス・メッセージング・サーバです。SonicMQ は従来のメッセージング製品とは一線を画す DRA（Dynamic Routing Architecture）という技術を導入し、今日の e ビジネス・ニーズに応える基盤を提供します。

DRA を使用すると、e ビジネスのグローバルなメッセージングを 1 つのエントリ・ポイントから一括して管理できます。エンタープライズや B2B など新たな部門がインターネット事業に参入した場合、SonicMQ は新しい配信先を動的に検知し最適な転送パスを介してメッセージの送受信を行います。また、マルチドメイン転送により複数のドメインへのメッセージ送信が可能なので、e ビジネスの拡張に合わせてアプリケーションを変更する必要がありません。

詳細情報

詳しくは Progress SonicMQ のウェブサイト（**http://www.progress.com/sonicmq/**）を参照してください。

パート 2

iPortal Application Server 準備と設定

パート 2 は次の章で構成されます。

第 3 章

開発環境の設定

本章では、iPortal Application Server の実行に必要な Java 環境の設定、および iPortal Application Server での使用に適した開発環境の構成方法について説明します。iPortal Application Server の配布メディアには Oracle 8i、Merant Sequelink、および Informix Cloudscape データベース用の JDBC 2.0 データベース・ドライバが収録されています。これらのデータベース・ドライバをインストールおよび設定する方法について詳しくは、ドライバと共にインストールされるドキュメンテーションを参照してください。

本章は、次のセクションで構成されます。

Java 環境の設定

概要

iPortal Application Server の各コンポーネントが正しく動作するためには、これに適した Java 環境を設定する必要があります。このセクションでは、各コンポーネントをコマンドラインで実行するための Java 環境の設定方法について説明します。

Windows NT

Windows NT では次のコマンドを使用します。

```
setenvs.bat
```

setenvs.bat は iPortal Application Server のインストール時に作成されるバッチ・ファイルで、全ての JAR を使用可能にして構成変数を設定します。**setenvs.bat** ファイルは iPortal Application Server のインストール・ディレクトリに作成されます。このファイルは、クラスパスに ***install-dir*\etc\domains** というディレクトリを追加します。

UNIX

UNIX では次のコマンドを使用します。

```
setenvs.sh
```

setenvs.sh は iPortal Application Server のインストール時に作成されるスクリプトで、全ての JAR を使用可能にして構成変数を設定します。**setenvs.sh** スクリプトは iPortal Application Server のインストール・ディレクトリに作成されます。このファイルは、クラスパスに ***install_dir*/etc/domains** というディレクトリを追加します。

CLASSPATH の構成

iPortal Application Server は、**iportal.jar** などの Java を拡張する機能 JAR パッケージにアクセスする際に J2EE のさまざまな機能を使用します。また、J2EE アプリケーションでサードパーティーによる JAR ファイルの実装クラスを使用する場合には、これらのファイルもクラスパスに含める必要があります。

iPortal Application Server の起動

サーバを起動するには **java iportal.server** というコマンドを使用します。デフォルト設定を使用して実行するには、次のコマンドを入力します。

```
java iportal.server
```

各 iPortal Administrator インスタンスには、固有のサーバ名が付いています。サーバ名を指定しない場合、**iPAS.Server.Default** という名前が使用されます。

サーバ名を指定するには、iPortal Application Server のコンフィギュレーションを変更する必要があります。したがって、サーバの構成方法がわからない場合は、デフォルトのサーバ名を使用してください。

サーバ名は、**java iportal.server** コマンドへのサーバ名引数として指定します。例えば **iPAS.Server.DemoServer** というサーバ名を指定するには、次のように入力します。

```
java iportal.server -n iPAS.Server.DemoServer
```

iPortal Application Server の開発環境

概要

iPortal Application Server developer edition は、ローカルの開発ホストにおけるイテレーションのビルドとデプロイメントを迅速に行えるよう最適化されています。

デフォルト・コンフィギュレーション

インストール時のデフォルト・コンフィギュレーションは、**install-dir\ipas3\etc\domains** ディレクトリに保存されています。これには次の各ファイルが含まれます。

default-domain.cfg	コントロール・コンフィギュレーション・ファイル
embedded-services.cfg	組込みサービスのコンフィギュレーション
ipas-defaults.cfg	iPortal Application Server のコンテナ用コンフィギュレーション
orbix2000.cfg	インフラストラクチャ用コンフィギュレーション
ipa.cfg	iPortal Administrator 用コンフィギュレーション

iPortal Application Server のコンフィギュレーションをネットワークに接続されていない独立したサーバからネットワーク・ホストに変更してクライアント／サーバ向けに設定するには、そのホストにある **default-domain.cfg** ファイルを変更します。

default-domain.cfg ファイルには、環境変数とインクルード文が含まれています。環境変数は使用するサービスを決定します。この構成は次のとおりです。

```
IT_LOCATOR_HOST
IT_LOCATOR_PORT
IT_NAMING_HOST
IT_NAMING_PORT
```

通常の場合、**IT_LOCATOR_HOST** と **IT_NAMING_HOST** の値は同じで、デフォルトのポート番号はロケータが 3075、ネーミング・サービスが 3074 です。

注： **コンフィギュレーションの変更内容を有効にするには、iPortal Application Server を再起動する必要があります。**

スタンドアロン・ホスト

デフォルトのコンフィギュレーションは、ネットワークに接続されていないスタンドアロン・ホスト用に設定されています。このコンフィギュレーションではネットワーク上のマシンであるかどうかに関係なく、iPortal Application Server の全コンポーネントを標準設定のまま使用することができます。デフォルト・コンフィギュレーションでは、**IT_LOCATOR_HOST** と **IT_NAMING_HOST** はそれぞれ **localhost** に設定されています。

ネットワーク・ホスト

ネットワークに接続されているクライアントやブラウザがホスト上にある iPortal Application Server を使用するには、**IT_LOCATOR_HOST** および **IT_NAMING_HOST** の値をネットワーク上のホスト名に変更する必要があります。

クライアント／サーバ環境

複数のホスト上でアプリケーションまたはクライアントをデプロイする必要がある場合、**default-domain.cfg** コンフィギュレーション・ファイルをこれら全てのホスト上で変更してください。

組込みロケータ、ネーミング・サービス、および管理サービスを実行している各サーバの **hostname** と **port** は、これらの共有サービスを含む **host** に変更する必要があります。次に例を示します。

```
IT_LOCATOR_HOST = "aplha";
IT_LOCATOR_PORT = "3075";
IT_NAMING_HOST = "alpha";
IT_NAMING_PORT = "3074"
```

iPortal Application Server のコンフィギュレーションは、クラスパスに含まれている特定の **default-domain.cfg** ファイルに基づいて設定されます。**install-dir\ipas3\etc\domains** ディレクトリ以外の場所にあるコンフィギュレーションを使用してサーバを起動するには、そのコンフィギュレーションが保存されているディレクトリをクラス・パスに追加してからサーバを再起動します。

iPortal Application Server コンフィギュレーションの変更

iPortal Application Server の主要なコンフィギュレーション・ファイルは **ipas-defaults.cfg** です。このファイルではポート番号、 ロギング、管理、および永続デプロイメントの構成と、ホスト上にある iPortal Application Server の複数インスタンスの構成を設定します。

ロギング

デフォルトでは iPortal Application Server のロギングはオンになっていて、重大なエラー、その他のエラー、警告メッセージの全てを記録するよう設定されています。

ロギングのレベルを変更するには、次のシンタックスを使用して **event_log:filters** コンフィギュレーション・アイテムを変更します。

```
event_log:filters = 'filter_spec1`,'filter_spec2`,'...
filter_spec = {' (subsystem_id | `*') `=' priority_list `}'
priority_list = priority1`,'priority2,... | `*' )
priority =  "INFO_ALL" | "INFO_LO[W]" | "INFO_HI[GH]" |
"INFO_MED[IUM]" | "WARN[ING]" | "ERR[OR]" |
"FATAL[_ERROR]" | "ALL_EVENTS" | "NO_EVENTS"
```

例えば、

```
 "{IT_POA=ERROR,FATAL},{IT_CORE=*},{*=WARN,ERR,FATAL}"
```

というエントリは、POA フィルタを **LOG_ERROR** と **LOG_FATAL_ERROR** に、ORB コア・フィルタを **LOG_ALL_EVENTS** に、また、デフォルト・フィルタを **LOG_WARNING**、**LOG_ERROR** および **LOG_FATAL_ERROR** にそれぞれ設定します。

ロギングをオフに切り替える

ロギングをオフにするには、ロギング・レベルを **NO_EVENTS** に変更するか、iPortal Application Server のメイン・コンフィギュレーション・スコープおよびその他全てのスコープから、**orb_plugins** 参照にある **local_log_stream** というエントリを削除します。

管理

デフォルトでは iPortal Application Server の管理はオンに設定されていて、iPortal Administrator が全ての iPortal Application Server インスタンスを監視できるようになっています。この機能をオフにするには、**ipas-defaults.cfg** ファイルにある

```
ipas:management:enabled="true";
```

というエントリを

```
ipas:management:enabled="false";
```

に変更し、 **orb_plugins** 参照から **it_mgmt** への全ての参照を削除します。

永続デプロイメント・リポジトリ

デフォルトでは iPortal Application Server の永続デプロイメント（リポジトリ・ベースのデプロイメント）がオンに設定されていて、iPortal Application Server の起動時にサーバにアプリケーションをロードできるようになっています。この機能をオフにするには、**ipas-defaults.cfg** ファイルにある

```
ipas:repository:enabled="true";
```

というエントリを

```
ipas:repository:enabled="false";
```

に変更します。

ポート番号

iPortal Application Server のデフォルトの **httpd port** 番号は 9000 です。これは次のエントリによって変更できます。

```
ipas:httpd:port="9000";
```

iPortal Application Server の複数インスタンス

ホスト上で iPortal Application Server の複数インスタンスを実行するには、コンフィギュレーション・ファイルで複数 orb コンフィギュレーション・スコープを設定し、iPortal Application Server の各インスタンスに **ORBname** パラメータを指定する必要があります。

例えば、**ipas-defaults.cfg** ファイルでは **iPAS.Server.Default** のスコープ定義と、**Server1** のプレースホルダが指定されています。複数インスタンスを作成するには **Server1** のセクションをコピーし、各インスタンス用に固有の名前を付け

てペーストします。各インスタンスのスコープ内で定義されているコンフィギュレーション・アイテムは、起動時にそのインスタンスにより使用されます。**orbname Server1** を使用して新しいインスタンスを起動するには、次を使用します。

```
java iportal.server –n iPAS.Server.Server1
```

注： **複数のサーバを使用する場合、競合を避けるには全てのインスタンスがそれぞれ http、jmx、naming、および locator で別々のポートを使用している必要があります。また、各インスタンスが別々の永続デプロイメント・リポジトリを使用していない場合、複数のサーバを同時に起動すると競合が起きることがあります。**

iPortal Administrator の Web コンソール・ポートの変更

iPortal Administrator の Web コンソールのポート番号を変更するには、**ipa.cfg** ファイルの次の行を変更します。

```
web_server:port_number = "3085";
```

JMX Reference Implementation の HTTP Adaptor をオンに切り替える

JMX RI（Reference Implementation）の HTTP Adaptor は、開発中の MBean のデバッグに使用されます。JMX RI の HTTP Adaptor をオンにして iPortal Application Server にロードできるようにするには、**ipas-defaults.cfg** ファイルで次の行を設定します。

```
ipas:jmx:httpd:enabled="true";
ipas:jmx:httpd:port="8083";
```

パート 3

エンタープライズ・アプリケーションの開発

パート 3 は次の章で構成されます。

第 4 章

開発者用ツールキット

iPortal Application Server の本リリースには、エンタープライズ・アプリケーションの開発作業を支援する一連のツールが用意されています。これらのツールは、J2EE アプリケーションの作成、デプロイメント、および管理に役立ちます。本章では開発者ツールキットに含まれている各ツールについて説明します。

本章は、次のセクションで構成されます。

開発者ツールキット

概要

iPortal Application Server の本リリースにはエンタープライズ・アプリケーションの開発作業を支援する一連のツールが用意されています。これらのツールは J2EE アプリケーションの作成、デプロイメント、および管理に役立ちます。本セクションでは各ツールの機能と使用方法について説明します。

iportal.central

iportal.central は、iPortal Application Server で使用できる全ツールへのショートカットが含まれているツールバーです。**iportal.central** を起動するには、コマンドラインで、

```
java iportal.central
```

と入力するか、スタート →プログラム→ IONA iPortal Application Server → iPortal Tools を選択します。**iportal.central** を初めて起動した場合、ツールバーは図 5 のように表示されます。

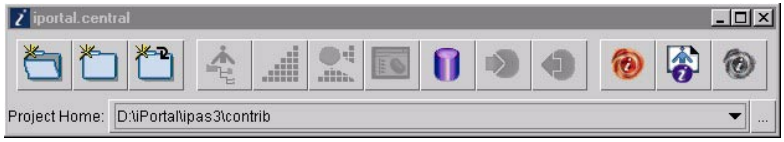


図 5: iportal.central ツールバー

アプリケーション開発のワークフロー

エンタープライズ・アプリケーションを開発するには、53 ページの図 6 にあるような各種のタスクを完了する必要があります。

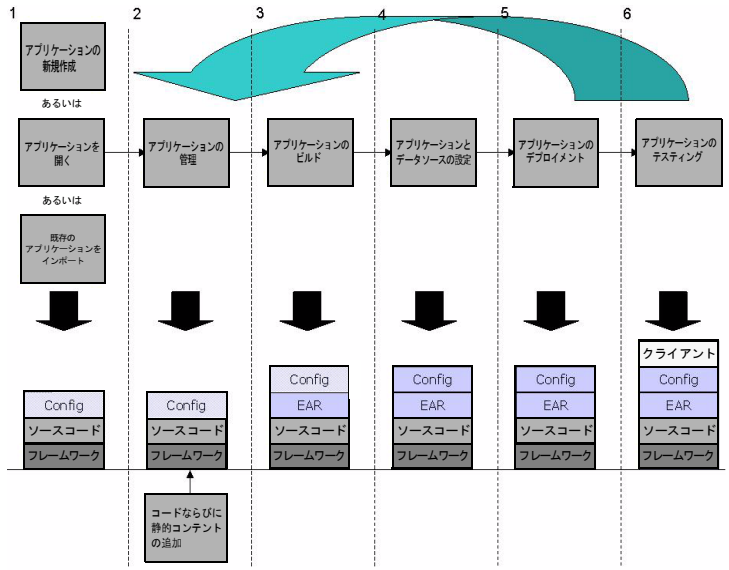


図 6: アプリケーション開発のワークフロー

必要なタスクには次のものが含まれます。

1. 次のいずれかの作業を行う。
   i. 既存の J2EE アプリケーションを開く。
   ii. 新しいアプリケーション用に EARSCO フレームワークを新規作成する。
   iii. 既存のアプリケーションを EARSCO フレームワークにインポートする。
2. アプリケーションの各コンポーネントを管理する。
3. アプリケーションをビルドする。
4. アプリケーションを構成しデータソースを登録する。
5. アプリケーションをデプロイする。
6. アプリケーションをテストする。

次のセクションではこれらの各タスクについてより詳しく説明します。

既存のアプリケーションを開く

iportal.central ツールを使用して既存の J2EE アプリケーションを EARSCO フレームワークで開くには、次の手順を行います。

1. iportal.central ツールバーにある次の Open Project アイコンをクリックする。Open ダイアログボックスが開く。

2. EAR ファイルが入っているディレクトリを選択する。Open をクリックし、EARSCO フレームワークでこのファイルを開く。

EARSCO フレームワークの作成

概要

iportal.earsco というツールを使用して、エンタープライズ・アプリケーション・プロジェクトに使用する EARSCO フレームワークの骨組みを作成します。

iportal.earsco の使用方法

iportal.earsco を使用するには 2 つの方法があります。

- iportal.central ツールバーのショートカットを使用する方法
- コマンドラインを使用する方法

ショートカット

ショートカットを使用すると earsco creation ウィザードが起動されます。このウィザードを使用して iPortal Application Server へのデプロイメントに使用する EAR ファイルをビルドするフレームワークを作成します。

1. iportal.central ツールバーにある iportal.earsco アイコンをクリックし、earsco creation ウィザードを起動する。

デフォルト設定の **D:\iPortal\ipas3\contrib** 以外のディレクトリにアプリケーションを作成するには、ツールをそのディレクトリから実行するか、**-d** オプションを指定して実行する必要があります。

2. Next をクリックし次のウィンドウに進み、作成するアプリケーションの名前を入力する。

3. Next をクリックして次のウィンドウに進み、必要に応じて Web コンポーネントの名前を入力する。

4. Next をクリックして次のウィンドウに進み、必要に応じて EJB の名前を入力する。Finish をクリックしてウィザードを終了する。

   EAR ファイルには 1 つ以上の EJB または Web コンポーネントが必要です。

コマンドライン

コマンドラインを使用する場合、次のコマンドを入力します。

```
java iportal.earsco -d  'dir_name'
```

dir_name には EARSCO ディレクトリ構造を作成するディレクトリの名前を指定します。

次に Windows と Unix の各プラットフォームでのコマンドを示します。

Windows	java iportal.earsco -d C:\iportal\sample EARSCO フレームワークを **C:\iportal\sample** ディレクトリに作成します。
UNIX	java iportal.earsco -d /**iportal/sample** EARSCO フレームワークを **/iportal/sample** ディレクトリに作成します。

これらのコマンドは earsco creation ウィザードを起動します。ウィザードの操作手順はショートカットを使用した場合と同じです。

EAR ファイルの EARSCO フレームワークへのインポート

概要

`iportal.earscoify` というツールを使用して、EAR ファイルを EARSCO フレームワークにインポートします。

`iportal.earscoify` は、次のいずれかの処理を行います。

- 特定の EAR ファイルをインポートし、クラス・ファイルを抽出して EARSCO フレームワークにコピーする。ソース・ファイルが使用可能かどうかに関わらず、ソース・ファイル用のディレクトリを作成する。
- 特定の EAR ファイルとソース・パスをインポートし、Java ソース・ファイルとクラス・ファイルを全て抽出して EARSCO フレームワークにコピーする。

iportal.earscoify の使用方法

`iportal.earscoify` を使用するには、2 つの方法があります。

- iportal.central ツールバーのショートカットを使用する方法
- コマンドラインを使用する方法

ショートカット

ショートカットを使用すると earscoify ウィザードが起動されます。このウィザードを使用して EARSCO 形式に変換する EAR ファイルを指定し、特定ディレクトリで適切な Java ソース・ファイルを検索します。

1. iportal.central ツールバーにある次の iportal.earscoify アイコンをクリックし、earscoify ウィザードを起動する。。

2. Next をクリックして次のウィンドウに進み、テキスト・フィールドに EAR ファイルの完全パスを入力するか、... を選択してファイル・システムで EAR ファイルを検索する。

3. Next をクリックして次のウィンドウに進み、ソース・ファイル・ディレクトリの名前を入力する。Finish をクリックしてウィザードを終了する。

コマンドライン コマンドラインを使用する場合、次のコマンドを入力します。

```
java iportal.earscoify -e 'ear_name'
```

ear_name には、EARSCO フレームワークに変換する EAR の名前を指定します。ここでは拡張子 **.ear** の指定は任意です。

```
java iportal.earscoify -e 'ear_name' -srcpath 'source_path'
```

source_path には、適切な Java ソース・ファイルを検索するパスを指定します。複数のディレクトリを指定する場合、各ディレクトリ名の間を UNIX の場合はコロン（:）、Windows の場合はセミコロン（;）で区切ります。同名の Java ソース・ファイルが複数見つかった場合、これらのファイルはコピーされず画面にエラー・メッセージと問題のあるソース・ファイルが一覧表示されます。その場合は、正しい Java ソース・ファイルを適切なディレクトリに手動でコピーする必要があります。

注： **ショートカットを使用した場合と異なり、コマンドラインを使用すると Java ソース・ファイルのディレクトリを複数指定できます。**

アプリケーションの管理

概要 **iportal.appadmin** というツールを使用して、エンタープライズ・アプリケーションの管理を行います。

iportal.appadmin により、次の操作を行えます。

- EJB モジュールの新規作成
- Web モジュールの新規作成
- アプリケーション・クライアント・モジュールの新規作成
- モジュールの削除
- モジュールの更新

アプリケーションの表示方法は、次の 3 つのオプションから選択できます。

- ツリー表示。アプリケーション・コンポーネントをツリー構造で表示する。

- XML 表示。アプリケーションのコンテナ・コンフィギュレーション・ファイル `cc.xml` ファイルを表示する。
- ファイル表示。アプリケーションで使用する JAR ファイルと WAR ファイルを全て表示する。

iportal.appadmin の使用方法

`iportal.appadmin` を使用するには 2 つの方法があります。

- iportal.central ツールバーのショートカットを使用する方法
- コマンドラインを使用する方法

ショートカット

ショートカットを使用するとアプリケーション・アドミニストレータが起動されます。このツールによりアプリケーションの更新と管理を行います。ここでは EJB モジュールを例に挙げて操作手順を説明します。

1. iportal.central ツールバーにある次の iportal.appadmin アイコンをクリックし、アプリケーション・アドミニストレータを起動する。

2. アプリケーションに新しい EJB モジュールを追加するには、メニュー・バーで Edit → New EJB Module を選択するか、キーボード・ショートカット Alt-B を使用する。

3. テキスト・フィールドに EJB モジュールの名前を入力してコードとスタティックなコンテンツを追加する。システム上にインクルードする JAR ファイルがある場合、Do you have a pre-compiled jar? チェックボックスをオンにしてファイルシステムにある JAR ファイルを参照して指定する。Enter をクリックして新しい EJB モジュールを作成する。EJB モジュールの作成に成功すると、その旨を伝えるメッセージが表示される。EJB モジュールのフォルダ・アイコンの下にあるツリー構造に EJB モジュールの名前が表示され、その詳細が `application.xml` ファイルに更新される。

4. アプリケーションからモジュールを削除するには、モジュールをハイライトしてメニュー・バーから Edit → Remove Module を選択するか、キーボード・ショートカット Alt-R を使用する。

コマンドライン

1. コマンドラインを使用する場合、次のコマンドを入力する。

```
java iportal.appadmin -d 'project_start_directory'
```

2. **project_start_directory** にはプロジェクトを保存するディレクトリの名前を指定する。

このアプリケーションからのみモジュールを削除するには、Do you wish to remove the module from the EARSCO directory structure? ダイアログボックスで No を選択します。EARSCO ディレクトリ構造からモジュールを削除するには、ここで Yes を選択します。モジュールの削除に成功すると、その旨を伝えるメッセージが表示されます。

アプリケーションのビルド

概要

iportal.build というツールを使用して、EARSCO プロジェクトをビルドし、EAR ファイルを作成します。

iportal.build の使用方法

iportal.build を使用するには次の 2 つの方法があります。

- iportal.central ツールバーのショートカットを使用する方法
- コマンドラインを使用する方法

ショートカット

ショートカットを使用する場合、次の手順を行います。

iportal.central ツールバーにある次の iportal.build アイコンをクリックし、iportal.build ウィンドウを開く。

例えば **simple.ear** を使用した場合、ウィンドウには次の出力が表示される。

```
Building ear simple.ear
Validating bankaccount.jar
Validating converter.jar
Validating grid.jar
Validating person.jar
```

```
Validating scorecard.jar
Validating carshop.jar
BUILD SUCCESSFUL
```

3. **iportal.build** の処理が終了したら Close をクリックしてウィンドウを閉じる。

コマンドライン

コマンドラインを使用する場合、次のコマンドを入力します。

```
java iportal.build -d 'eardir' -compilerargs
   'java_compile_arguments'
```

eardir には EAR ファイルを保存するディレクトリの名前を、**java_compiler_arguments** にはコンパイル時に **javac** に渡される引数をそれぞれ指定します。

古いビルドの削除

概要

iportal.build -clean というツールを使用して、ビルドした EAR ファイルおよび EARSCO フレームワークから生成された全ての中間ファイルを削除します。

iportal.build -clean の使用方法

iportal.build -clean を使用するには、次の 2 つの方法があります。

- iportal.central ツールバーのショートカットを使用する方法
- コマンドラインを使用する方法

ショートカット

ショートカットを使用する場合、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.build -clean アイコンをクリックし、iportal.build ウィンドウを開く。

次の出力が表示される。

```
IONA iPortal Application Server - Version 3.0 Iteration 0910
   [20010514-1534]
Copyright (c) IONA Technologies plc., 1999-2001. All Rights
   Reserved.
Java 1.3.0 on Windows NT 4.0 (x86

BUILD CLEANED
```

2. **iportal.build -clean** の処理が終了したら、Close をクリックしてウィンドウを閉じる。

コマンドライン

コマンドラインを使用する場合、次のコマンドを入力する。

```
java iportal.build -clean
```

アプリケーションの構成

概要

iportal.assembler というツールを使用して、グラフィカル・アプリケーション・ビルダによりアプリケーションを構成します。グラフィカル・アプリケーション・ビルダは、JAR ファイルおよび WAR ファイルにある Bean や Web コンポーネントのデプロイメント・デスクリプタを更新するために使用します。

iportal.assember の使用方法

iportal.assembler を使用するには、次の 2 つの方法があります。

- iportal.central ツールバーのショートカットを使用する方法
- コマンドラインを使用する方法

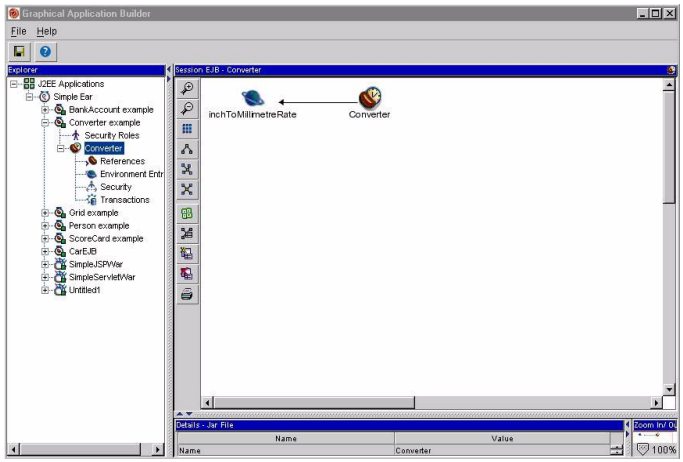
ショートカット

ショートカットを使用する場合、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.assembler アイコンをクリックし、グラフィカル・アプリケーション・ビルダを起動する。

2. 例えば **Converter.jar** の場合、File → Open Archive を選択してグラフィカル・アプリケーション・ビルダで **simple.ear** を開く。**simple.ear** をダブルクリックして Bean と Web コンポーネントを表示する。

3. **Converter.jar** をダブルクリックし、そのコンポーネントを次のように表示する。



4. Bean の環境エントリを更新するか、新規作成するには Environment Entries オプションを選択する。

5. References オプションを選択する。Bean の EJB 参照の更新、あるいは新規作成には EJBs タブを選択する。Bean のリソース参照を更新するか、新規作成するには Resources タブを選択する。Bean のセキュリティ・ロール参照を更新するか、新規作成するには Roles タブを選択する。

6. Bean のトランザクション情報を更新するには Transactions オプションを選択する。

7. メニュー・バーで File → Save を選択する。グラフィカル・アプリケーション・ビルダが更新された Bean を保存する。

コマンドライン

コマンドラインを使用する場合、次のコマンドを入力します。

```
java iportal.assembler
```

ショートカットを使用した場合と同じく、グラフィカル・アプリケーション・ビルダが起動されます。

データソースの登録

概要

`iportal.datasource` はデータソースの構成に使用します。このツールは register datasource ウィザードを使用します。`iportal.datasource` を使用して iPortal Application Server 用の JNDI データソースを作成し、新しいデータソースを追加した場合に `cc.xml` の更新を行います。

iportal.datasource の使用方法

`iportal.datasource` を使用するには、次の 2 つの方法があります。

- iportal.central ツールバーのショートカットを使用する方法
- コマンドラインを使用する方法

ショートカット

ショートカットを使用する場合、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.datasource アイコンをクリックし、register datasource ウィザードを起動する。

2. Next をクリックして JNDI サービス・プロバイダを指定する。JNDI データソース情報を保存する URL ロケーションを指定し、Next をクリックする。
3. データソースの名前を指定して Next をクリックする。
4. ドロップダウン・リストから使用する JDBC データソースを選択する。分散トランザクション環境で使用するデータソースの場合は XA? チェックボックスをオンにする。それ以外の場合はテキスト・フィールドに代替データソースを指定し、Next をクリックする。
5. ご使用の環境に適した JDBC コネクション・プロパティを指定し、Next をクリックする。
6. Oracle ドライバを使用する場合、使用するドライバのタイプを指定して Next をクリックする。
7. データソース・ラッパ用の JNDI 名を指定する。この名前はコンテナ・コンフィギュレーション・ファイル `cc.xml` で使用される。Next をクリックする。

8. コンフィギュレーション・ファイルの名前を指定する。（この操作手順はオプション）Finish をクリックしてウィザードを終了する。

コマンドライン

コマンドラインを使用する場合、次のコマンドを入力します。

```
java iportal.datasource
```

ショートカットを使用した場合と同じく、register datasource ウィザードが起動されます。

iPortal Application Server の起動

概要

iportal.server というツールを使用して、iPortal Application Server を起動します。iPortal Application Server の各インスタンスには固有のサーバ名を割り当てます。サーバ名を指定しない場合、**iPAS.Server.Default** というサーバ名が使用されます。

iportal.server の使用方法

iportal.server を使用するには 2 つの方法があります。

- iportal.central ツールバーのショートカットを使用する方法
- コマンドラインを使用する方法

ショートカット

ショートカットを使用する場合、iportal.central ツールバーにある次の iportal.server アイコンをクリックします。

iportal.server ウィンドウが開き、サーバが正常に起動されると次のような出力が表示されます。

```
IONA iPortal Application Server - Version 3.0 Copyright (c) IONA
   Technologies plc., 1999-2001. All Rights Reserved.
Java 1.3.0 on Windows NT 4.0 (x86)
Starting Embedded Service: Locator...done
Starting Embedded Service: NameService...done
Starting Embedded Service: iPortal Administrator Management
   Service...done
```

```
iPortal Administrator's Web Adaptor available on
   "http://localhost:3085"
Starting HTTP Listener on port 9000 ... done
Connecting to iPortal Administrator's Management Server...done
Ready...
```

コマンドライン

コマンドラインを使用する場合、次のコマンドを入力します。

```
java iportal.server
```

ショートカットを使用した場合と同じく iportal.server ウィンドウが開きます。

iPortal Administrator Console の実行

iPortal Administrator Console（**iportal.admin**）は、iPortal Administrator に Java グラフィカル・ユーザ・インターフェイス（GUI）を提供します。このコンソールを使用してアプリケーション、コンフィギュレーションの設定、およびイベント・ロギングを管理します。このコンソールは IIOP（Internet Inter-ORB Protocol）により管理サービスと通信を行います。

iportal.admin の使用方法

iportal.admin を起動するには iportal.central ツールバーを使用します。iportal.central ツールバーにある次の iportal.admin アイコンをクリックします。

iPortal Administrator ツールが起動されます。このツールの使用方法について詳しくは『iPortal Administrator ユーザーズガイド』を参照してください。

iPortal Application Server の終了

概要

iportal.shutdown というツールを使用して、iPortal Application Server をシャットダウンします。

iportal.shutdown の使用方法

iportal.shutdown を使用するには、次の 2 つの方法があります。

- iportal.central ツールバーのショートカットを使用する方法
- コマンドラインを使用する方法

ショートカット

ショートカットを使用する場合、iportal.central ツールバーにある次の iportal.shutdown アイコンをクリックします。

iportal.shutdown ウィンドウが開き、サーバが正常にシャットダウンされると、次の出力が表示されます。

```
IONA iPortal Application Server - Version 3.0 Copyright (c) IONA
   Technologies plc., 1999-2001. All Rights Reserved.
Java 1.3.0 on Windows NT 4.0 (x86)
Shutting down iPortal Application Server: iPAS.Server.Default ...
iPortal Application Server (iPAS.Server.Default) successfully
   shut down
```

コマンドライン

コマンドラインを使用する場合、次のコマンドを入力します。

```
java iportal.shutdown
```

デフォルト・サーバの **iPAS.Server.Default** がシャットダウンされます。シャットダウンするサーバを指定するには、次のコマンドを使用します。

```
java iportal.shutdown -n server_name
```

アプリケーションのデプロイメント

概要

iportal.deploy というツールを使用して、アプリケーションを iPortal Application Server にデプロイします。J2EE アプリケーションを iPortal Application Server にデプロイするには、archive deployer wizard ウィザードを使用します。

iportal.deploy の使用方法

iportal.deploy を使用するには、次の 2 つの方法があります。

- iportal.central ツールバーのショートカットを使用する方法
- コマンドラインを使用する方法

ショートカット

ショートカットを使用する場合、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.deploy アイコンをクリックし、archive deployer wizard ウィザードを起動します

2. デプロイするファイルまたはディレクトリの名前を指定して Next をクリックする。
3. コンテナ・コンフィギュレーション・ファイルの名前を指定して Next をクリックする。
4. デプロイを行う前に JAR ファイルを検証する。これを行うためには Validate と Only deploy if valid の各チェックボックスをオンにする。Next をクリックする。
5. Finish をクリックしてアプリケーションをデプロイする。

コマンドライン

コマンドラインを使用する場合、次のコマンドを入力します。

```
java iportal.deploy
```

ショートカットを使用した場合と同じく、archive deployer wizard ウィザードが起動されます。

アプリケーションのアンデプロイ

概要

`iportal.undeploy` というツールを使用して、iPortal Application Server にデプロイされたアプリケーションを回収します。

iportal.undeploy の使用方法

`iportal.undeploy` を使用するには 2 つの方法があります。

- iportal.central ツールバーのショートカットを使用する方法
- コマンドラインを使用する方法

ショートカット

ショートカットを使用する場合、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.undeploy アイコンをクリックし、archive undeployer ウィザードを起動する。

2. アンデプロイする EAR ファイルの名前がドロップダウン・リストに表示される。必要なファイルを選択し、Next をクリックする。

3. Finish をクリックしてアプリケーションをアンデプロイする。

コマンドライン

コマンドラインを使用する場合、次のコマンドを入力します。

```
java iportal.undeploy
```

ショートカットを使用した場合と同じく archive undeployer ウィザードが起動されます。

アプリケーションのテスト

概要

iportal.client というツールを使用して、J2EE アプリケーションをテストします。

iportal.client はコマンドラインから使用します。シンタックスは次のとおりです。

```
java iportal.client [ -u,-usage ] [ -s,-silent ] [ -m,-module
   "appClient module" ] [ -e, -ear "ear archive" ] [ -c,-config
   "configuration file" ] [ -o,-ORBname "configuration scope selector" ]
   client-arguments...
[The option -module must be specified]
```

java iportal.client コマンドは、アプリケーション・クライアントを iPortal Application Server で実行します。

-u usage	**iportal.client** コマンドで使用できるオプションを表示する。
-e ear archive	使用する EAR ファイルの名前を指定する。
-c config configuration file	使用するコンテナ・コンフィギュレーション・ファイルの名前を指定する。
-m appClient module	使用するモジュールの名前を指定する。このオプションの指定は必須。
-s silent	**iportal.client** をサイレント・モードで実行する。
-o "configuration scope selector	使用する Orb の名前を指定する。

関連情報

アプリケーション・クライアントの **main()** メソッドが戻った時点で **iportal.client** は Java VM を終了します。Swing GUI アプリケーションのように Java VM を終了する必要がない場合には、**main()** メソッドの最後に次のコードを追加するとこの動作を避けることができます。

```
while (true)
{
  try
  {
    Thread.sleep(10000);
  }
  catch(Exception ex){}
}
```

Web アプリケーションをテストするには、ブラウザで **http://localhost:9000/context_root/warname** というロケーションを指定します。

第 5 章

Enterprise JavaBeans（EJB）

本章では、EJB（Enterprise JavaBeans）の概要および、エンタープライズ・アプリケーションの開発過程における EJB の役割について説明します。EJB アプリケーションは Bean と呼ばれるコンポーネントから構成されます。各 Bean は Java プログラミング言語で記述されており、それ自体で独立した機能性を持ちます。したがって、個々の Bean は複数のアプリケーションで使用することができ、EJB アプリケーションでは新しく開発された Bean を既存の Bean と組み合わせて使用することもあります。

本章は、次のセクションで構成されます。

EJB の基本概念

概要

Sun Microsystems, Inc. による Enterprise JavaBeans Specification 1.1 には、Java プログラミング言語を使用してオブジェクト指向型の分散ビジネス・アプリケーションを開発するためのコンポーネントを基本としたアーキテクチャが定義されています。Enterprise JavaBeans アーキテクチャは、エンタープライズ・アプリケーションの開発、デプロイメント、そしてランタイムの各過程に関与しています。

EJB では、セキュリティ、トランザクション、状態管理の詳細を管理するコンポーネント・フレームワーク内にビジネス・ロジックがカプセル化されている一方、リソース・プーリング、スケーリング、スレッディングなど低レベルの細かい部分は EJB コンテナにより処理されます。したがって、低レベルの処理のサポート機能が組み込まれており、開発者はビジネス本来の問題解決に集中して取り組むことができます。

コンテナ

Enterprise JavaBean アプリケーションは、**コンテナ**内で動作します。コンテナとはアプリケーション・コンポーネントに必要な低レベルのサービスを適する環境です。例えばコンテナは、分散コンポーネント間のネットワークを介した直接通信、通信の際のセキュリティ確保、また分散トランザクション処理の制御などを行うために必要な諸機能を提供します。

デプロイメント・デスクリプタ

EJB アプリケーションを開発する場合、コンポーネントのコードでこれらの低レベルのサービスを使用する必要はありません。各アプリケーションには**デプロイメント・デスクリプタ**という XML（extensible markup language：拡張マークアップ言語）ドキュメントが関連付けられていて、必要なサービスはこのドキュメントに記述されています。デプロイを行う時点でコンテナがデプロイメント・デスクリプタを読み込み、必要なサービスを自動的に提供します。したがって、開発者は分散プログラミングの細かい部分に気を取られることなく、アプリケーションのビジネス・ロジック開発に集中することができます。

EJB のタイプ

概要

EJB には、**セッション Bean** と**エンティティ Bean** という 2 つの基本的なタイプがあります。

エンティティ Bean

エンティティBean は、バックエンド・データベースに記録されている永続データの 1 アイテムを表し、クライアントがこのデータにアクセスして操作するのに必要なオペレーションを提供します。永続データへのアクセスにはエンティティBean を使用します。エンティティBean には複数のクライアントが同時にアクセスできます。コンテナはトランザクションを使用してデータへのアクセスを制御します。

セッション Bean

セッション Bean は、クライアントの代わりにアクションを実行する、クライアント用のサーバ側エージェントとして機能する一時オブジェクトです。通常セッション Bean は、クライアント側とサーバ側のアプリケーション・ロジック間の 1 セッションにおける 1 クライアントを表します。セッション Bean には、**ステートフル**なセッション Bean と**ステートレス**なセッション Bean の 2 種類があります。

ステートフルなセッション Bean には、クライアントとアプリケーション間の通信状態に関する情報が格納されていて、通常はクライアントに使用されている間のみ存在します。ただし、タイムアウト期限が経過した場合などには、EJB サーバはステートフルなセッション Bean を非アクティブまたは無効にすることができます。

ステートレスなセッション Bean には、呼出し間の状態は格納されません。

データソースにアクセスする EJB

概要

エンティティBean の状態を元のデータソースに転送することを、Bean 永続性といいます。エンティティBean が使用できる永続性には、**Bean 管理永続性**と**コンテナ管理永続性**の 2 種類があります。

Bean 管理永続性

Bean 管理永続性では、Bean の実装により状態転送が行われ、データソースにアクセスして状態を転送するためのコードが Bean に含まれています。セッション Bean がデータソースにアクセスする場合や、エンティティBean が Bean 管理永続性を使用する場合、Bean の実装クラスにデータ・アクセス用のコードを含める必要があります。

このコードの内容は、Bean がアクセスするデータソースのタイプによって異なります。例えば、JDBC（Java Database Connectivity）を介してリレーショナル・データベースにアクセスする場合や、専用のクラスを介して古いアプリケーションにより管理されているデータにアクセスする場合などがあります。

コンテナ管理永続性

コンテナ管理永続性では、元のデータソースとの状態に関する情報のやり取りをコンテナが管理します。この場合、Bean に関連するフィールドを Bean のデプロイメント・デスクリプタが記述し、これらフィールドの永続性を制御する実装コードはコンテナが生成します。

CMP プラグイン

上級ユーザの場合、iPortal Application Server では、コンテナ管理永続性（CMP）をカスタマイズして独自の実装を**プラグイン**できるので、サードパーティーが提供する永続性ツールを利用することも可能です。

プラグイン機能の例は、iPortal Application Server のインストール・ディレクトリ内の `ipas3\demos\CMPPlugin` ディレクトリにある `CMPPlugin` というデモをご覧ください。

EJB を使用した作業

概要

J2EE プラットフォーム仕様によって、分散アプリケーション開発にかかる手間を省くことができます。このプラットフォームでは、アプリケーションの作成に関わる各担当者にタスクを割り当てます。一般的なタスクは EJB コンテナおよびツールのベンダーが行うので、アプリケーション開発者の責任が軽減されます。

関連のあるロール

EJB アプリケーションの作成には、次のようなロールが関連しています。

- Bean プロバイダ
- アプリケーション・アセンブラ
- デプロイヤ
- EJB サーバおよびコンテナのプロバイダ
- システム管理者

Bean プロバイダ

Bean プロバイダは個々の Bean を作成し、Bean 内でビジネス・ロジックを実装します。汎用の Bean を扱うベンダーや、特定の組織向けに専用の Bean を作成する開発者などがこれに該当します。

アプリケーション・アセンブラ

アプリケーション・アセンブラは、Bean を組み合わせ、デプロイ可能な独立したアプリケーションを作成します。アプリケーション・アセンブラはアプリケーションに Bean 以外のコンポーネントを追加する場合もあります。

デプロイヤ

デプロイヤは、Bean を組み合わせて作成したアプリケーションを特定のオペレーティング環境にインストールおよび構成します。

EJB サーバおよび EJB コンテナのプロバイダ

サーバおよびコンテナのプロバイダは、EJB コンポーネントが稼動するプラットフォームの実装を行います。このプラットフォームは、各コンポーネントが必要とするセキュリティ、トランザクション、永続性などの分散サービスを提供します。コンポーネントとコンテナ間では標準のインターフェイスを使用するので、各コンポーネントは全ての標準コンテナで実行できます。EJB サーバとは、EJB コンテナをホストするシステムを指します。このようなプラットフォームは iPortal Application Server により提供されます。

システム管理者

システム管理者は、アプリケーションを監視し、その動作を管理します。システム管理者の仕事としては、アプリケーションが異常な動作をした場合の対処などが含まれます。

EJB の開発

概要

各 Bean は、JAR ファイルという標準の Java アーカイブ・ファイルにパッケージ化されています。JAR ファイルには、1 つ以上の Bean が含まれています。

JAR ファイル

JAR ファイルには各 Bean につきインターフェイスとクラスが含まれています。これらは Bean プロバイダがコンパイルしてから JAR ファイルに含めます。JAR ファイルには、ファイル中の各 Bean に関する情報を記述するデプロイメント・デスクリプタも 1 つ含まれています。

さらに、JAR ファイルにはその他のファイルを含めることもできます。例えば、JAR ファイルには、含まれている Bean を識別する全体的な manifest ファイルも含まれています。多くの Bean プロバイダでは Java の IDE（Integrated Development

Environment：総合開発環境）を使用して Bean を作成します。通常、IDE は Bean に関連するコードの大半を生成し、Bean を JAR ファイルとしてパッケージ化します。

Bean の必要条件 Bean プロバイダが Bean を開発する場合、各 Bean につき**ホーム・インターフェイス**と**リモート・インターフェイス**という 2 つのインターフェイスと、**Bean クラス**という 1 つのクラスを定義する必要があります。

ホーム・インターフェイス ホーム・インターフェイスには、クライアントが Bean のインスタンスを作成、検索、削除するのに必要なメソッドが含まれています。

リモート・インターフェイス リモート・インターフェイスには、Bean のビジネス・ロジックにアクセスするためにクライアントが呼び出すことのできるメソッドが含まれています。

Bean クラス Bean クラスはホーム・インターフェイスとリモート・インターフェイスで定義されているメソッドを実装します。

EJB におけるデプロイメント・デスクリプタの更新

概要 J2EE アプリケーションの各コンポーネントを適切な JAR ファイルと WAR ファイルにバンドルしたら、これを完全な J2EE アプリケーション EAR ファイルにアセンブリすることができます。iPortal Application Server には、J2EE アプリケーションの作成を支援する開発者用のツールキットが用意されています。JAR ファイルにある Bean コンポーネント用にデプロイメント・デスクリプタの設定を更新するには、`iportal.assembler` というツールを使用します。既存の Bean 用に設定されているデプロイメント・デスクリプタの値を変更することもできます。

必要な環境エントリの指定

概要 Bean は、ランタイムにその環境にアクセスして、アプリケーション・デプロイヤにより設定された値を取得できます。環境には名前と値のペアが含まれています。Bean のソースコードは環境を読み込んで指定された環境エントリの値を取得し、これをビジネス・ロジックで使用することができます。

iportal.assembler を使用して、Bean で値が必要な環境エントリを指定し、これらのエントリのデフォルト値を設定できます。

その後、アプリケーション・アセンブラおよびアプリケーション・デプロイヤは必要に応じてこれらのエントリの値を更新できます。

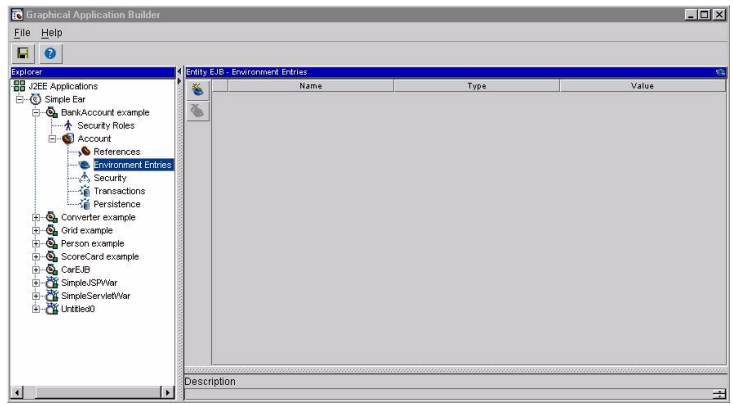
環境エントリの宣言

Bean の環境エントリを宣言するには、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.assembler アイコンをクリックするか、コマンドラインに **java iportal.assembler** と入力し、グラフィカル・アプリケーション・ビルダを起動する。

2. メニューバーから File → Open Archive を選択し、Bean が含まれている JAR ファイルを展開表示する。

3. Bean を選択して Explorer パネルのリストから Environment Entries オプションを選択する。Bean の環境エントリが次のように表示される。



4. ワークスペース・パネルで新しい環境エントリ・ボタンを選択する。Environment Entry Creation ウィザードの Create New Environment Entry ウィンドウが開く。

5. Next を選択して Enter Environment Entry Name and Description ウィンドウを表示し、Name フィールドに環境エントリに名前を、またオプションの Description フィールドに環境エントリの説明を入力する。

6. Next を選択して Enter the Type and Value of the Environment Entry ウィンドウを表示し、ドロップダウン・リストから Environment Entry Type を選択する。

7. Next を選択して Confirm Details ウィンドウを表示し内容を確認した後、Finish を選択してウィザードを終了する。各環境エントリにつき手順 4 ～ 7 を繰り返す。

8. メニュー・バーから File → Save を選択する。グラフィカル・アプリケーション・ビルダが JAR ファイルを保存する。

Bean に必要な環境エントリの値は、エントリを作成した後でも必要に応じて変更できます。既存のエントリを編集するには Environment Entries タブでエントリを選択してテキスト・フィールドの値を変更し、File → Save を選択します。

Bean 間のリファレンスの指定

概要 Bean は他の Bean のリモート・インターフェイス上のメソッドを呼び出すことができます。このようなメソッド呼出しを行うには、呼出し元の Bean が次の条件を満たしている必要があります。

1. JNDI を使用してターゲット Bean のホーム・インターフェイスへのリファレンスを取得する。次はその例。

```
//Java
public class EmployeeServiceBean implements SessionBean {
   public void changePhoneNumber(...) {
      ...
      // Obtain the default initial JNDI context.
      Context initCtx = new InitialContext();

      // Look up the home interface of the EmployeeRecord
      // enterprise bean in the environment.
      Object result =
      initCtx.lookup("java:comp/env/ejb/EmplRecord");

     // Convert the result to the proper type.
     EmployeeRecordHome emplRecordHome =
    (EmployeeRecordHome)javax.rmi.PortableRemoteObject.narrow
    (result, EmployeeRecordHome.class);
```

```
    ...
```

2. ホーム・インターフェイスを使用してリモート・インターフェイス・インスタンスへのリファレンスを取得する。次はその例。

```
try {
     EmployeeRecord empl =
                  emplRecordHome.findEmployee ("Paul");
    }
catch (java.rmi.RemoteException) {
   ...
}
```

3. ターゲット Bean のリモート・インターフェイスで定義されているビジネス・メソッドを呼び出す。

Bean のソースコードを開発したら、その Bean が参照する他の Bean を宣言する必要があります。Bean 内の各リファレンスにつき、そのデプロイメント・デスクリプタで他の Bean のホーム・インターフェイス、ホーム・インターフェイスのタイプ、およびリモート・インターフェイスのタイプを検索する際に使用する名前を指定する必要があります。

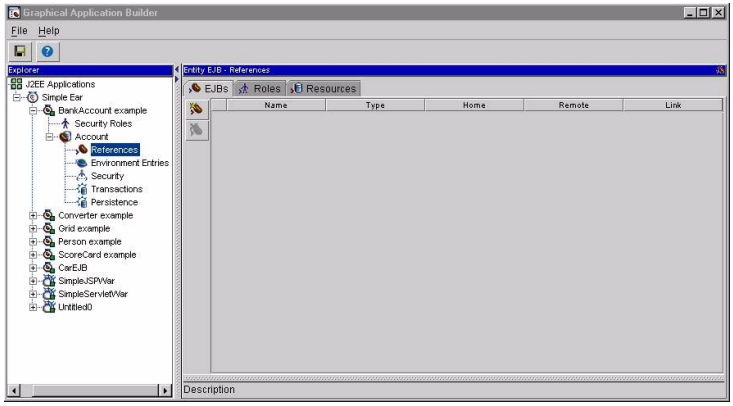
Bean へのリファレンスの宣言

Bean に関連するリファレンスを宣言するには、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.assembler アイコンをクリックするか、コマンドラインで **`java iportal.assembler`** と入力し、グラフィカル・アプリケーション・ビルダを起動する。。

2. メニューバーから File → Open Archive を選択し、Bean が含まれている JAR ファイルを展開表示する。

3. Bean を選択して Explorer パネルのリストから References オプションを選択する。ワークスペース・パネルで EJBs タブを選択する。次のような Bean リファレンスが表示される。



4. 新規リファレンス・ボタンを選択する。EJB Reference Creation ウィザードの Create New EJB Reference ウィンドウが表示される。

5. Next を選択して Enter EJB Name and Description ウィンドウを表示する。Name フィールドにリファレンスの名前を入力する。ここでは例えば `ejb/EmplRecord` のように Bean の検索コードで使用されているエントリ名を指定する。オプションの Description フィールドにリファレンスの説明テキストを入力する。

6. Next を選択して Choose the Bean Type ウィンドウを表示する。予期される Bean の種類に応じて Session あるいは Entity のいずれかのラジオ・ボタンが自動的に選択される。

7. Next をクリックして Choose the Home and Remote Interfaces ウィンドウを表示する。ホーム・インターフェイスとリモート・インターフェイスの情報は、予期される Bean の詳細に基づいて自動的に入力される。

8. Next を選択して Confirm Details を表示し、Finish をクリックしてウィザードを終了する。Bean に関連するリファレンスの詳細が表示されル。Bean の各リファレンスにつき手順 4 ～ 9 を繰り返す。

9. メニューバーから File → Save を選択する。グラフィカル・アプリケーション・ビルダが JAR ファイルを保存する。

Bean に関連するリファレンスは、宣言した後でも必要に応じて変更できます。既存のリファレンスを編集するには References オプションを選択して EJBs タブを選択し、テキスト・フィールドの値を変更してから File → Save を選択します。

リソースへのリファレンスの指定

概要

Bean では、ファイルやデータベースなどの外部リソースへのアクセスが必要になることがあります。各リソースには**リソース・マネージャ**が関連付けられていて、Bean はこのマネージャを介してリソースにアクセスします。例えば、DBMS はデータベースへのアクセスを提供します。

Bean がリソース・マネージャと通信するためには、まず、**リソース・マネージャ・コネクション**を取得する必要があります。リソース・マネージャ・コネクションは、Bean とリソース・マネージャ間の通信を可能にするオブジェクトです。

リソース・マネージャ・コネクションを作成するには、まず Bean が**リソース・マネージャ・コネクション・ファクトリ**へのリファレンスを取得する必要があります。このファクトリは、コネクションを提供するオブジェクトです。リファレンスを取得した後、Bean はこのオブジェクトを使用してコネクションを取得しリソースにアクセスします。

例えば、Bean は次のような手順で JDBC データベースにアクセスします。

```
//Java
  private java.sql.Connection getConnection() {
     try {
        Context initCtx = new InitialContext();
        DataSource dataSource =
                initCtx.lookup("java:comp/env/jdbc/Accounts");

        return dataSource.getConnection();
         } catch (Exception ex) {
           throw new EJBException(ex);
         }
  }
```

1. データベースのリソース・マネージャ・コネクション・ファクトリを取得する。

ファクトリからコネクションを返す。

```
DataSource dataSource =
                initCtx.lookup("java:comp/env/jdbc/Accounts");

 return dataSource.getConnection();
```

2. コネクション・オブジェクトを使用してデータベースに SQL コマンドを送る。

Bean を提供する際は、リソース・マネージャ・ファクトリを取得するためその Bean が使用する各 JNDI の名前を宣言する必要があります。また Bean が受け入れるファクトリ・オブジェクトの型も宣言します。

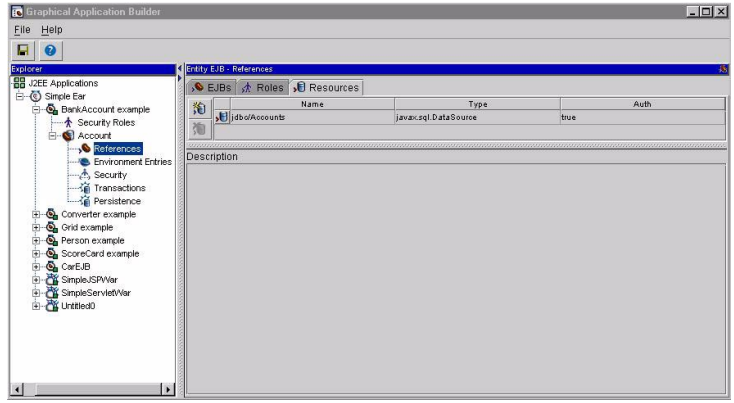
リソース・リファレンスの宣言

Bean に関連するリソース・リファレンスを宣言するには、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.assembler アイコンをクリックするか、コマンドラインで **`java iportal.assembler`** と入力し、グラフィカル・アプリケーション・ビルダを起動する。

2. メニューバーから File → Open Archive を選択し、Bean が含まれている JAR ファイルを展開表示する。

3. Bean を選択して Explorer パネルのリストから References オプションを選択する。ワークスペース・パネルで Resources タブを選択すると、次のようなリソース・リファレンスが表示される。



4. 新規リソース・リファレンス・ボタンを選択する。Resource Reference Creation ウィザードの Create New Resource Reference ウィンドウが表示される。

5. Next を選択して Enter Datasource Name and Type ウィンドウを表示する。ドロップダウン・リストから Datasource を選択する。Reference Name フィールドに Bean がリソース・マネージャ・コネクション・ファクトリへのリファレンス取得に使用する JNDI 名を入力する。Resource Factory Type フィールドの値は選択したデータソースに応じて自動的に入力される。

6. Next を選択して Is the Datasource Container Authenticated? ウィンドウを表示し、コンテナ認証を指定するドロップダウン・リストから Yes または No を選択する。オプションの Description フィールドにリファレンスの説明テキストを入力する。

7. Next を選択して Confirm Details ウィンドウを表示し、Finish を選択してウィザードを終了する。Bean に関連するリソース・リファレンスの詳細が表示される。各リソース・リファレンスにつき手順 4 ～ 8 を繰り返す。

8. メニューバーから File → Save を選択する。グラフィカル・アプリケーション・ビルダが JAR ファイルを保存する。

Bean に関連するリソース・リファレンスは、宣言した後でも必要に応じて変更できます。既存のリファレンスを編集するには References オプションを選択して Resources タブを選択し、テキスト・フィールドの値を変更してから File → Save を選択します。

セキュリティ・ロールへのリファレンスの指定

セッション Bean またはエンティティBean を開発する場合、Bean のソースコードにセキュリティを追加する必要はありません。Bean がセキュアな環境で動作する場合、Bean にアクセスするユーザのタイプを判断するには、宣言されているメソッドの使用許可を使用します。

ただし状況によっては、Bean プロバイダがセキュリティ・コードを記述する必要もあります。この場合、デプロイメント・デスクリプタで宣言済みのメソッド使用許可を使用してセキュリティ・チェックを宣言することはできません。これをコードで指定するには EJB コンテキスト・メソッドの **`isCallerInRole()`** を使用します。

次に例を示します。

```
//Java
public class PayrollBean ... {
   EntityContext ejbContext;
   public void updateEmployeeInfo(EmplInfo info) {
      oldInfo = ... read from database;
      // The salary field can be changed only by caller's
      // who have the security role "Payroll"
      if (info.salary != oldInfo.salary &&
      !ejbContext.isCallerInRole("Payroll")) {
         throw new SecurityException(...);
         }
      ...
      }
  ...
}
```

Bean プロバイダは、Bean コードで使用される各ロールにつき、1 つのセキュリティ・ロール・リファレンスを作成する必要があります。

アプリケーション・アセンブリの時点でアセンブラがアプリケーションのセキュリティ・ロールを作成し、個々の Bean のセキュリティ・ロール・リファレンスをそのアプリケーション用に定義されているロールにリンクすることにより解決します。

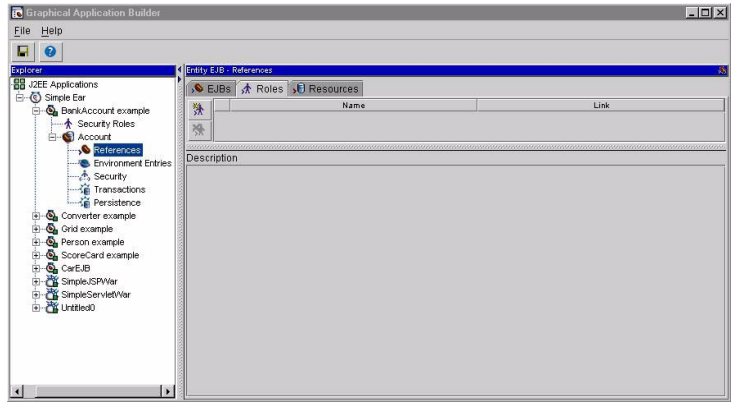
セキュリティ・ロールの宣言

Bean に関連するセキュリティ・ロール・リファレンスを作成するには、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.assembler アイコンをクリックするか、コマンドラインで **`java iportal.assembler`** と入力し、グラフィカル・アプリケーション・ビルダを起動する。。

2. メニューバーから File → Open Archive を選択し、Bean が含まれている JAR ファイルを展開表示する。

3. Bean を選択して Explorer パネルのリストから References オプションを選択する。ワークスペース・パネルで Roles タブを選択する。次のようなセキュリティ・ロール・リファレンスが表示される。



4. 新規セキュリティ・ロール・リファレンス・ボタンを選択する。Security Role Reference Creation ウィザードの Create New Security Role Reference ウィンドウが開く。

5. Next を選択して Enter Security Role Reference Name and Description ウィンドウを表示する。Name フィールドにセキュリティ・ロール・リファレンスの名前を入力し、オプションの Description フィールドにリファレンスの説明テキストを入力する。

6. Next を選択して Enter Name of Link ウィンドウを表示する。ここでは Link フィールドに値を入力しないようにする。このフィールドはアセンブリ・エリアのみに該当する。

7. Next を選択して Confirm Details ウィンドウを表示し、Finish を選択してウィザードを終了します。Bean に関連するセキュリティ・ロール・リファレンスの詳細が表示されます。新しいセキュリティ・ロール・リファレンスそれぞれにつき、手順 4 ～ 8 を繰り返します。

8. メニューバーから File → Save を選択します。グラフィカル・アプリケーション・ビルダが JAR ファイルを保存します。

Bean に関連するセキュリティ・ロール・リファレンスは、作成した後でも必要に応じて変更できます。既存のリファレンスを編集するには References オプションを選択して Roles タブを選択し、テキスト・フィールドの値を変更してから File → Save を選択します。

リエントラントな呼出しのサポートの指定

概要 通常、エンティティBean は、これに関連するデータに矛盾が生じないようトランザクション・コンテキスト内で実行され、またエンティティBean によってはトランザクション・コンテキストでのループバックをサポートすることもあります。ループバックは、例えばクライアントがエンティティBean のメソッドを呼び出した後でその Bean がトランザクション・コンテキスト内で 2 番目の Bean のメソッドを呼び出し、この 2 番目の Bean が同じコンテキスト内で最初の Bean にコールバックを行った場合などに生じます。

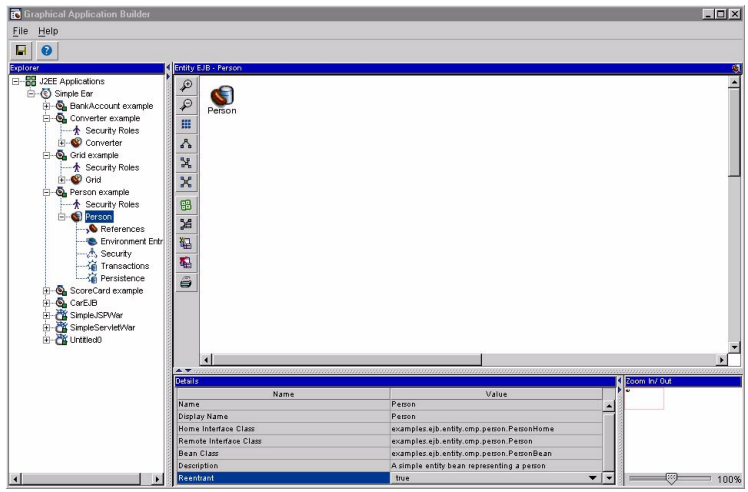
リエントラントな呼出しをサポートする Bean のプログラミングは複雑です。エンティティBean を提供する場合、Bean がリエントラントな呼出しをサポートするかどうかを指定できます。サポートする場合にはコンテナがループバックを許可します。リエントラントな呼出しをサポートしない場合、リエントラントな呼出しを行わないコードを使用して Bean を開発する必要があります。

リエントラントな呼出しのサポートを指定するには、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.assembler アイコンをクリックするか、コマンドラインで `java iportal.assembler` と入力し、グラフィカル・アプリケーション・ビルダを起動する。

2. メニューバーから File → Open Archive を選択し、Bean が含まれている JAR ファイルを展開表示する。

3. エンティティ Bean を選択して次のように Reentrant フィールドで True を選択し、リエントラントな呼出しのサポートを許可する。



4. メニューバーから File → Save を選択する。グラフィカル・アプリケーション・ビルダが JAR ファイルを保存する。

XML のエクスポート

JAR ファイルのデプロイメント・デスクリプタの更新が終了したら、XML Viewer を使用して `xml` ファイルをエクスポートします。このビューワにはグラフィカル・アプリケーション・ビルダのツールバーからアクセスします。`xml` ファイルをエクスポートするには次の手順を行います。

1. アプリケーション全体用の `application.xml` をエクスポートする。このファイルは EARSCO フレームワークのトップレベルにある `/etc` ディレクトリにコピーする。

2. アプリケーションで使用される各 JAR ファイルにつき、`ejb-jar.xml` をエクスポートする。これらのファイルは EARSCO フレームワーク内のそれぞれ適切な `<JARfilename>.jar/etc` ディレクトリにコピーする。

3. グラフィカル・アプリケーション・ビルダで EAR ファイルを閉じる。

4. **iportal.build** ツールを再実行して更新された **xml** を実行可能な EAR ファイルに反映させ、ソースコードと実行可能コードを同期させる。EAR ファイルを再ビルドするには次のコマンドを使用する。

```
java iportal.build
```

更新されたアイテムのみが再ビルドされます。

iPortal Application Server の起動

概要 EAR ファイルをデプロイする前に、まず iPortal Application Server を起動する必要があります。

iportal.central ツールバーにある iportal.server アイコンをクリックするか、コマンドラインで次のように入力します。

```
java iportal.server
```

iPortal Application Server が起動されます。iPortal Application Server の起動についての詳細は、64 ページの「iPortal Application Server の起動」を参照してください。

EAR ファイルのデプロイメント

概要 **iportal.deploy** を使用して、iPortal Application Server に EAR ファイルを簡単にデプロイできます。

iportal.central ツールバーで iportal.deploy アイコンをクリックするか、コマンドラインで次のように入力します。

```
java iportal.deploy
```

iPortal Application Server に EAR ファイルをデプロイする方法について詳しくは、66 ページの「アプリケーションのデプロイメント」を参照してください。

アプリケーションのテスト

概要 アプリケーション・クライアントを使用してアプリケーションをテストするには、次のコマンドを使用します。

```
java iportal.client -m module_name -e earfile -c config
```

アプリケーション・クライアントと **iportal.client** コマンドについて詳しくは、103 ページの「アプリケーション・ クライアントの プログラミング」および 242 ページの「java iportal.client」を参照してください。

第 6 章

Web コンポーネント

本章では、Web コンポーネントの概要を説明します。Web コンポーネントには、サーブレット、JSP（JavaServer Page）、アプレット、Web デプロイメント・デスクリプタが含まれます。エンタープライズ・アプリケーションは Web コンポーネントと EJB で構成されていて、Web コンポーネントは Web アプリケーションの一部としてデプロイされます。Web アプリケーションは WAR ファイルという 1 つの Web アプリケーション・アーカイブ・ファイルに保存され、この WAR ファイルが iPortal Application Server の Web サーバへデプロイされます。

本章は、次のセクションで構成されます。

Web コンポーネントの概要

概要

Web コンポーネントには、サーブレット、JSP（JavaServer Page）、および HTML、画像、サウンド・ファイルなどのスタティック・ドキュメントが含まれます。Web コンポーネントはそのデプロイメント・デスクリプタと共に WAR ファイルという標準の Web アーカイブ・ファイルにパッケージ化され、サーブレットの場合はコンパイル済みのクラス・ファイルも含まれます。WAR ファイルは、エンタープライズ・アプリケーションを作成する際のアプリケーション・アセンブリで JAR ファイルと共に使用されます。

サーブレット

サーブレットは独立した小さな Java クラスで、Web サーバで動的にロードし実行できるコードにコンパイルされます。サーブレットでは CGI スクリプトを使用するより処理が高速に行え、また通常の Web サーバでサポートされる標準の API を使用しています。サーブレットにより Java プログラミング言語の利点を活かし、Java プラットフォームで使用できる API の大部分にアクセスすることができます。

JavaServer Page

JSP（JavaServer Pages）を使用すると、動的な Web ページのデザイン、開発、および管理を簡単に行うことができます。JSP は Java と HTML の組合せにより Web ページ用の動的コンテンツを提供します。JSP ではユーザ・インターフェイスとコンテンツ生成が分かれているので、Web デザイナーは動的コンテンツ自体を変更せずにページ全体のレイアウトを変えることができます。JSP は、使用する前にまずサーブレット・コードにコンパイルされます。JSP は J2EE 標準の一部なので、多様なプラットフォームでデプロイすることが可能です。

Web デプロイメント・デスクリプタ

EJB アプリケーションを開発する場合、コンポーネントのコードで低レベルのサービスを使用する必要はありません。各アプリケーションには**デプロイメント・デスクリプタ**という XML（extensible markup language：拡張マークアップ言語）ドキュメントが関連付けられていて、必要なサービスはここに記述されています。デプロイを行う時点でコンテナがデプロイメント・デスクリプタを読み込み、必要なサービスを自動的に提供します。

Web コンポーネントを使用した エンタープライズ・アプリケーションの開発

概要 エンタープライズ・アプリケーションの各コンポーネントを適切な JAR ファイルと WAR ファイルにバンドルしたら、次にこれらのファイルを J2EE アプリケーション EAR ファイルとしてアセンブリする作業を行います。iPortal Application Server には、エンタープライズ・アプリケーションの開発を支援する各種のツールが用意されています。

セクションの内容 本セクションは次のトピックで構成されています。

デプロイメント・デスクリプタの更新

概要 J2EE アプリケーションの各コンポーネントを適切な JAR ファイルと WAR ファイルにバンドルしたら、次にこれらのファイルを J2EE アプリケーション EAR ファイルとしてアセンブリする作業を行います。iPortal Application Server には、J2EE アプリケーションの作成を支援する各種の開発者向けツールが用意されています。WAR ファイルに含まれる Web コンポーネントのデプロイメント・デスクリプタ

設定を更新するには、**iportal.assembler** というツールを使用します。このツールでは既存の Web コンポーネントのデプロイメント・デスクリプタ値を変更したり、新しく作成した Web コンポーネントに必要な値を設定することができます。

Welcome ファイルリストの指定

概要 Welcome ファイルのリストは、順序付けされたファイルのリストをサーバに提供します。このファイル・リストによって、特定ファイルを指定せずに Web サイトにアクセスした場合にリクエストの処理方法を指示します。

リストの冒頭にあるファイルがある場合、サーバはこのファイルを使用しますが、ファイルがない場合にはリストの順序に従って最初に見つかったファイルを使用します。通常これらのファイルは **index.jsp**、**index.html**、**index.htm** といった名前が付いています。

WAR ファイルに関連する Welcome ファイルのリストを指定するには、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.assembler アイコンをクリックするか、コマンドラインに **java iportal.assembler** と入力して、グラフィカル・アプリケーション・ビルダを起動する。

2. メニューバーから File → Open Archive を選択し、Web コンポーネントが含まれている WAR ファイルを展開表示する。WAR ファイルの詳細が表示される。

3. Welcome Files フィールドを選択して Welcome Files ダイアログボックスを表示し、指定する Welcome ファイルを選択する。

4. Add をクリックして Welcome ファイルを追加し、OK をクリックしてダイアログボックスを終了する。Welcome ファイルの詳細が Web アーカイブ・ファイルに表示される。

5. メニューバーから File → Save を選択する。グラフィカル・アプリケーション・ビルダが WAR ファイルを保存する。

サーブレット・コンテキスト・パラメータの指定

概要 サーブレット・コンテキスト・パラメータを使用してアプリケーションで使用する URI と名前のマッピングを定義し、アプリケーションのサーブレットが共有情報にアクセスできるようにします。

コンテキスト・パラメータ・エレメントは、その Web アプリケーション・サーブレットのコンテキスト内にある全てのサーブレットによりアクセスできるパラメータを指定します。コンテキスト・パラメータを使用して、Web アプリケーションのカスタム部分をデプロイメント時に構成することができます。

サーブレットのコンテキスト・パラメータを指定するには、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.assembler アイコンをクリックするか、コマンドラインに `java iportal.assembler` と入力してグラフィカル・アプリケーション・ビルダを起動する。

2. メニューバーから File → Open Archive を選択し、Web コンポーネントが含まれている WAR ファイルを展開表示します。WAR ファイルの詳細が表示されます。

3. Context Parameters フィールドを選択して Context Parameter Properties ダイアログボックスを表示します。ここにサーブレットのコンテキスト・パラメータを入力します。

4. **Add** を選択してコンテキスト・パラメータを入力し、**OK** を選択してダイアログボックスを終了します。コンテキスト・パラメータの詳細が Web アーカイブ・ファイルに表示されます。

5. メニューバーから File → Save を選択します。グラフィカル・アプリケーション・ビルダが WAR ファイルを保存します。

URL パターンとサーブレットの関連付け

概要 サーブレット・マッピング・エレメントは、サーブレットと URL パターンのマッピングを定義します。URL パターン・エレメントにはマッピングの URL パターンが含まれています。サーブレット・コンテナがリクエストをサーブレットにマッピングするには、URL パスを使用する必要があります。

URL パターンとサーブレットを関連付けるには、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.assembler アイコンをクリックするか、コマンドラインに **`java iportal.assembler`** と入力して、グラフィカル・アプリケーション・ビルダを起動する。

2. メニューバーから File → Open Archive を選択し、Web コンポーネントが含まれている WAR ファイルを展開表示する。WAR ファイルの詳細が表示される。

3. Servlet Mappings フィールドを選択して Servlet Mapping Properties ダイアログボックスを表示する。ここにサーブレット・マッピングの詳細を入力する。

4. Add を選択してサーブレット・マッピングのプロパティを入力し、OK を選択してダイアログボックスを終了する。サーブレット・マッピングの詳細が Web アーカイブ・ファイルに表示される。

5. メニューバーから File → Save を選択する。グラフィカル・アプリケーション・ビルダが WAR ファイルを保存する。

MIME パターンを使用したサーブレット結果のフィルタリング

概要 Web ブラウザにページ・コンテキストの処理方法を指示するには、MIME タイプを使用します。MIME タイプ・エレメントによって、ファイル拡張子を一般的な MIME タイプにマッピングします。

MIME パターンを使用してサーブレットの結果をフィルタリングするには、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.assembler アイコンをクリックするか、コマンドラインに **`java iportal.assembler`** と入力し、グラフィカル・アプリケーション・ビルダを起動する。

2. メニューバーから File → Open Archive を選択し、Web コンポーネントが含まれている WAR ファイルを展開表示する。WAR ファイルの詳細が表示される。

3. MIME Mappings フィールドを選択して MIME Mapping Properties ダイアログボックスを表示する。ここに MIME マッピングの詳細を入力する。

4. Add を選択して MIME マッピングのプロパティを入力し、OK をクリックしてダイアログボックスを終了する。MIME マッピングの詳細が Web アーカイブ・ファイルに表示される。

5. メニューバーから File → Save を選択する。グラフィカル・アプリケーション・ビルダが WAR ファイルを保存する。

ファイルとエラー・コードの関連付け

概要 エラー・ページ・エレメントには、エラー・コードまたは例外の型から Web アプリケーション内のリソース・パスへのマッピングが含まれています。エラー・コード・エレメントには、例えば「404」などの HTTP エラー・コードが含まれています。さらに例外の型エレメントには Java 例外の型の完全修飾クラス名が、またロケーション・エレメントには Web アプリケーション内のリソースのロケーションが含まれています。

ファイルをサーブレットのエラー・コードに関連付けるには、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.assembler アイコンをクリックするか、コマンドラインに `java iportal.assembler` と入力し、グラフィカル・アプリケーション・ビルダを起動する

2. メニューバーから File → Open Archive を選択し、Web コンポーネントが含まれている WAR ファイルを展開表示する。WAR ファイルの詳細が表示される。

3. Error Pages フィールドを選択して Error Page Properties ダイアログボックスを表示する。ここにエラー・コードの詳細を入力する。

4. Add を選択してエラー・ページのプロパティを入力し、OK をクリックしてダイアログボックスを終了する。エラー・ページの詳細が Web アーカイブ・ファイルに表示される。

5. メニューバーから File → Save を選択する。グラフィカル・アプリケーション・ビルダが WAR ファイルを保存する。

JSP が使用するタグ・ライブラリの指定

概要 タグ・ライブラリ・エレメントを使用して、JSP タグ・ライブラリを記述します。タグ・ライブラリ URI エレメントは web.xml ドキュメントからの URI を記述し、これにより、Web アプリケーションで使用するタグ・ライブラリを指定します。taglib ロケーション・エレメントには、タグ・ライブラリ記述ファイルの検索場所が、Web アプリケーションのルートからの相対的なリソース・ロケーションとして含まれています。

JSP が使用するタグ・ライブラリを指定するには、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.assembler アイコンをクリックするか、コマンドラインに `java iportal.assembler` と入力し、グラフィカル・アプリケーション・ビルダを起動する。

2. メニューバーから File → Open Archive を選択し、Web コンポーネントが含まれている WAR ファイルを展開表示する。WAR ファイルの詳細が表示される。

3. Tag Library フィールドを選択して Tag Library Properties ダイアログボックスを表示する。ここにタグ・ライブラリの詳細を入力する。

4. Add をクリックしてタグ・ライブラリのプロパティを入力し、OK をクリックしてダイアログボックスを終了する。タグ・ライブラリの詳細が Web アーカイブ・ファイルに表示される。

5. メニューバーから File → Save を選択する。グラフィカル・アプリケーション・ビルダが WAR ファイルを保存する。

EJB へのリファレンスの指定

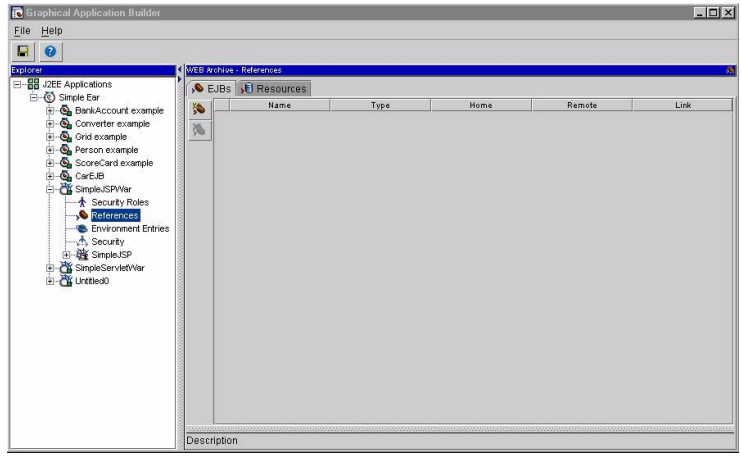
概要 EJB リファレンス・エレメントを使用して、WAR ファイルからエンタープライズ Bean へのリファレンスを宣言します。

エンタープライズ Bean へのリファレンスを作成するには、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.assembler アイコンをクリックするか、コマンドラインに **`java iportal.assembler`** と入力し、グラフィカル・アプリケーション・ビルダを起動する。

2. メニューバーから File → Open Archive を選択し、Web コンポーネントが含まれている WAR ファイルを展開表示する。WAR ファイルの詳細が表示される。

3. References オプションを選択します。ワークスペース・パネルで EJBs タブを選択する。次のようなリファレンスが表示される。



4. 新規リファレンス・ボタンを選択する。新しいリファレンス・エントリが表示される。

5. 各フィールドにリファレンスの詳細を入力する。

6. Name：リファレンスの名前。

7. Type：リファレンス先の Bean のタイプ。ドロップダウン・リストでエンティティ（Entity）またはセッション（Session）を選択する。

8. Home：Bean のホーム・インターフェイスの完全パスと名前を直接入力するか、... ボタンを選択して Archive Manager を起動し、表示されたリストから適切なクラス・ファイルを選択する。

9. Remote Interface：Bean のリモート・インターフェイスの完全パスと名前を直接入力するか、... ボタンを選択して Archive Manager を起動し、表示されたリストから適切なクラス・ファイルを選択する。

10. Description：リファレンスの説明テキスト。

11. WAR ファイルに含まれる各リファレンスにつき手順 4 ～ 6 を繰り返す。

12. メニューバーから File → Save を選択する。グラフィカル・アプリケーション・ビルダが WAR ファイルを保存する。

Bean のリファレンスは、宣言した後でも必要に応じて編集できます。既存のリファレンスを編集するには References オプションを選択して EJBs タブを選択し、テキスト・フィールドの値を変更してから File → Save を選択します。

リソースへのリファレンスの指定

グラフィカル・アプリケーション・ビルダを使用して Web コンポーネントのリソースへのリファレンスを指定できます。この手順は Bean のリソースへのリファレンスを指定する場合と同じです。詳しくは、81 ページの「リソースへのリファレンスの指定」を参照してください。

環境エントリの指定

グラフィカル・アプリケーション・ビルダを使用して Web コンポーネントの環境エントリを指定できます。この手順は Bean の環境エントリを指定する場合と同じです。詳しくは、76 ページの「必要な環境エントリの指定」 を参照してください。

XML のエクスポート

WAR ファイル用のデプロイメント・デスクリプタを更新したら、グラフィカル・アプリケーション・ビルダのツールバーにある XML Viewer を使用して次の手順で `xml` ファイルをエクスポートします。

1. アプリケーション全体用の `application.xml` をエクスポートする。このファイルは EARSCO フレームワークのトップレベルにある /etc というディレクトリにコピーする。

2. アプリケーションで使用される各 WAR ファイルにつき、**web.xml** をエクスポートする。これらのファイルは EARSCO フレームワーク内のそれぞれ適切な **<WARfilename>.war/etc** ディレクトリにコピーする。

3. グラフィカル・アプリケーション・ビルダで EAR ファイルを閉じる。

4. EarBuilder ツールを再実行して更新された **xml** を実行可能な EAR ファイルに反映させ、ソースコードと実行可能コードを同期させる。EAR ファイルを再ビルドするには次のコマンドを使用する。

```
java iportal.build
```

5. 更新されたアイテムのみが再度ビルドされる。

iPortal Application Server の起動

概要 EAR ファイルをデプロイする前に、まず iPortal Application Server を起動する必要があります。

iportal.central ツールバーにある iportal.server アイコンをクリックするか、コマンドラインで次のように入力します。

```
java iportal.server
```

iPortal Application Server が起動されます。iPortal Application Server の起動について詳しくは、64 ページの「iPortal Application Server の起動」を参照してください。

EAR ファイルのデプロイメント

概要 **iportal.deploy** を使用して、iPortal Application Server に EAR ファイルを簡単にデプロイできます。

iportal.central ツールバーで iportal.deploy アイコンをクリックするか、コマンドラインで次のように入力します。

```
java iportal.deploy
```

iPortal Application Server に EAR ファイルをデプロイする方法について詳しくは、66 ページの「アプリケーションのデプロイメント」を参照してください。

アプリケーションのテスト

概要 デプロイメント済みの Web アプリケーションにアクセスするには、ブラウザで次のロケーションを指定します。

```
http://localhost:9000/context_root/warname
```

第 7 章

アプリケーション・クライアントのプログラミング

本章ではアプリケーション・クライアントのプログラミングについて説明します。アプリケーション・クライアントは Java プログラミング言語で記述されたプログラムです。クライアント・プログラムはランタイムに J2EE サーバとは別の仮想マシン（VM）を使用して実行されます。J2EE 仕様では、アプリケーション・クライアントは、自己の Java VM で実行される最上層のクライアント・プログラムです。アプリケーション・クライアントは Java 技術に基づくアプリケーション・モデルを採用しており、`main()` メソッドにより起動され、VM が終了するまで実行されます。アプリケーション・クライアントも他の J2EE アプリケーション・コンポーネントと同様に、システム・サービスの提供に関してはコンテナに依存しています。アプリケーション・クライアントは、その他の J2EE コンテナに比べてごく小さい場合もあります。

本章は、次のセクションで構成されます。

コンテナによるエンタープライズ Bean の提供方法

概要

EJB コンテナの実装方法はさまざまですが、全てのコンテナはエンタープライズ Bean 標準のクライアント側のビューをサポートしています。クライアント・プログラムのコードを記述する場合、Bean にアクセスするにはそのホーム・インターフェイスに対してメソッドを呼び出し、その後リモート・インターフェイスのメソッドを呼び出して Bean のビジネス・ロジックを使用します。このプログラミング・モデルは、エンタープライズ Bean の全クライアントに適用されます。

クライアントのコードを記述するには、EJB コンテナがエンタープライズ Bean を提供する方法について理解しておく必要があります。Bean ホーム・インターフェイスのメソッドを呼び出すには、まずこれらのメソッドをサポートするオブジェクトを取得する必要があります。このためコンテナは EJB ホームと呼ばれるオブジェクトを作成し、このオブジェクトへのリファレンスを JNDI によりパブリッシュします。JNDI は CORBA のネーミング・サービス、フラット・ファイル、および専用のディレクトリ・サービスなどのサービスの上に実装されています。

EJB ホーム

EJB ホームは、ホーム・インターフェイスが定義する全てのメソッドを実装し、コンテナが Bean のライフ・サイクルを管理する際に必要な機能を提供します。ホーム・インターフェイスに対して create メソッドが呼び出されると、EJB ホームが EJB オブジェクトのインスタンスを作成し、このインスタンスが EJB オブジェクトと関連付けられた時点で `ejbCreate()` メソッドが呼び出されます。このメソッドが完了すると、EJB ホームがその EJB オブジェクトのスタブ（リモート・リファレンス）をクライアントに返します。クライアントはスタブを使用してビジネス・メソッドを要求することにより EJB オブジェクトとの通信を開始できるようになり、EJB オブジェクトはこれらのメソッド呼出しを Bean インスタンスに委譲することになります。図 7 は、ホーム・インターフェイスとリモート・インターフェイスをそれぞれ実装する EJB ホームと EJB オブジェクトを表しています。

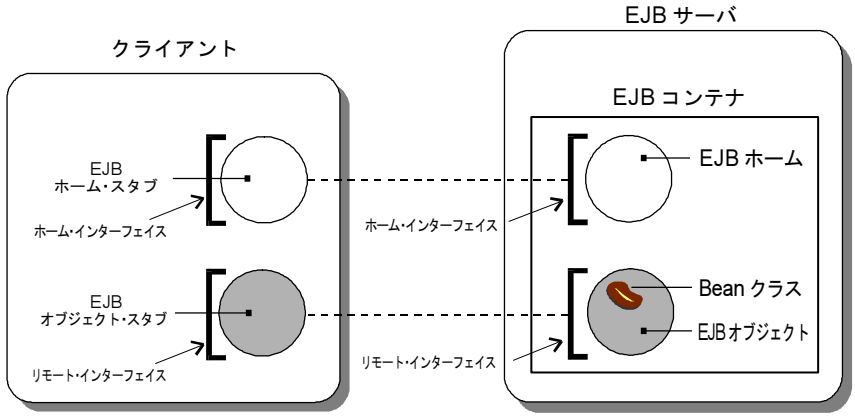


図 7: エンタープライズ Bean のホーム・インターフェイスとリモート・インターフェイスの使用

クライアントのコードには次の処理を含める必要があります。

- JNDI の初期コンテキストを取得する。
- 初期コンテキストを使用して Bean で使用する EJB ホームへのリファレンスを調べる。
- EJB ホームの create メソッドまたは find メソッドを呼び出し、Bean インスタンスへのリファレンスを取得する。
- Bean インスタンスを使用してリモート・インターフェイスで定義されている Bean のビジネス・メソッドを呼び出す。

クライアントをデプロイする時点で、クライアントにより使用されるエンタープライズ Bean 全てがスタブ・クラスを使用できることを確認してください。

Java アプリケーション・クライアントの概要

概要

サーブレット、JSP、エンタープライズ Bean などの Java プログラムを開発する際、アセンブリが済んだアプリケーションでこれらのコンポーネントとエンタープライズ Bean 間でどのようなインタラクションを行うか決定する必要があります。その後クライアント側の API を使用してクライアントから必要な Bean への呼出しを行います。

Bean との通信

クライアントは Bean と直接やり取りを行うのではなく、JNDI（Java Naming and Directory Interface）を介して Bean のホーム・インターフェイスを検索し、そこからホーム・インターフェイスを実装するオブジェクトへのリファレンスを取得します。また、クライアントが Bean のリモート・インターフェイスへのリファレンスを取得するには、そのホーム・インターフェイスのメソッドを呼び出します。

Bean のホーム・インターフェイスは、Bean インスタンスの作成、検索、そして破棄を行うライフ・サイクル・サービスを定義しています。リモート・インターフェイスは、Bean が提供するビジネス・サービスを定義し、クライアントが呼び出すことのできる Bean メソッドやパブリック・インターフェイスを全て指定します。ホーム・インターフェイスとリモート・インターフェイスで定義されている機能の実装は、クラス実装により行います。

iportal.client

iPortal Application Server では、`iportal.client` というクライアント・コンテナが次のようなシステム・サービスを提供します。

- クライアントがアプリケーション・サーバとの通信を確立するための環境を作成する。
- クライアントが `java:name space` を使えるように JNDI ラッパを提供する。
- クライアントを認証する。

EARSCO フレームワークへの アプリケーション・クライアントの追加

概要

アプリケーション・クライアントを使用する J2EE アプリケーションでは、EARSCO フレームワークにアプリケーション・クライアント・ディレクトリを追加する必要があります。**iportal.client** には、GUI ベースのショートカット・バージョンはありません。

アプリケーション・クライアントの追加

EARSCO フレームワークにアプリケーション・クライアントを追加するには、次の手順を行います。

1. **ApplicationEAR\src** ディレクトリに、例えば **AppClient.jar** などのアプリケーション・クライアント・ディレクトリを作成する。

2. **AppClient.jar** ディレクトリに、次の 2 つのディレクトリを作成する。

etc
src

3. etc ディレクトリには次の 2 ファイルを保存する。

Application-client.xml
MANIFEST.MF

   src ディレクトリにはアプリケーション・クライアントのソースコードを保存する。

4. 次の各ファイルに必要なエントリを追加する。

ApplicationEAR\src\application.xml

```
<application>
   <module>
       <java>AppClient.jar</java>
   </module>
</application>
```

ApplicationEAR\src\cc.xml

```
<configuration>
   <application-client>
       <application-client-name>AppClient.jar
```

```
        </application-client-name>
    </application-client>
</configuration>
```

アプリケーション・クライアントの開発

アプリケーション・クライアントは、自己の Java VM で実行される最上層のクライアント・プログラムで、他の J2EE アプリケーション・コンポーネントと同様に、システム・サービスの提供に関してはコンテナに依存しています。アプリケーション・クライアントは JAR ファイルにパッケージ化されていて、デプロイメント・デスクリプタと manifest ファイルを含んでいます。

デプロイメント・デスクリプタ・ファイルはアプリケーションが参照する EJB と外部リソースを記述します。次にこの例を示します。

application-client.xml

```
<application-client>
   <display-name>AccountClient</display-name>
   <description>Client account details</description>
   <ejb-ref>
      <ejb-ref-name>ejb/Account</ejb-ref-name>
      <ejb-ref-type>Entity</ejb-ref-type>
      <home>
         examples.ejb.entity.bmp.bankaccount.AccountHome
      </home>
      <remote>
         examples.ejb.entity.bmp.bankaccount.Account
      </remote>
      <ejb-link>Account</ejb-link>
   <ejb-ref><
</application-client>
```

アプリケーション・クライアントは、その manifest ファイルの**Main-Class** 属性で指定されているクラスの**main()** メソッドで起動されます。次に manifest ファイルの例を示します。

MAINFEST.MF

```
Manifest-Version: 1.0
Main-Class: examples.ejb.entity.bmp.bankaccount.AccountClient
```

関連情報

アプリケーション・クライアントの **main()** メソッドが戻った時点で **iportal.client** は Java VM を終了します。Swing GUI アプリケーションのように Java VM を終了する必要がない場合には、**main()** メソッドの最後に次のコードを追加するとこの動作を避けることができます。

```
while (true)
{
  try
  {
    Thread.sleep(10000);
  }
  catch(Exception ex){}
}
```

スタンドアロン・クライアントからの EJB へのアクセス

概要

状況によっては **iportal.client** コンテナを使わずにスタンドアロンの Java クライアントから iPortal Application Server でデプロイされた EJB にアクセスすることが必要です。その場合プログラマは、クライアントから EJB にアクセスするための環境を提供する必要があります。

iPortal Application Server によってデプロイされた EJB への、スタンドアロン Java クライアントからのアクセス

iPortal Application Server でデプロイされた EJB にスタンドアロン Java クライアントを使用してアクセスするには、次の手順を行います。

1. 次のシステム・プロパティを設定する。

```
System.setProperty("org.omg.CORBA.ORBClass","com.iona.corba.art.
  artimpl.ORBImpl");
System.setProperty
  ("org.omg.CORBA.ORBSingletonClass","com.iona.corba.art.artimp
  l.ORBSingleton");
System.setProperty("javax.rmi.CORBA.PortableRemoteObjectClass","
  com.iona.corba.rmi.PortableRemoteObjectDelegateImpl");
System.setProperty("java.naming.factory.initial","com.sun.jndi.c
  osnaming.CNCtxFactory");
```

これらのプロパティはコマンドラインでも設定できます。

2. コンテナの外でクライアントを実行している場合、**cc.xml** に指定さえている名前を使用して Bean のリファレンスを解決する必要がある。このリファレンス解決には **java: name space** は使用できない。

   例えば **cc.xml** で **Grid** という Bean が **iona/ipas/simple/Grid** という JNDI 名により定義されているので、**java:comp/env/ejb/GridBean** ではなく JNDI 名の方を使用する。これには次の設定を行う。

```
Context ctx = new InitialContext();
GridHome gridHome = (ctx.lookup("iona/ipas/simple/Grid"),
   GridHome.class);
```

3. クラスパスにホーム・インターフェイスとリモート・インターフェイスを含め、さらに **/etc/domains** ディレクトリの **default-domain.cfg** も含める。

スタンドアロン Java クライアントの利点

iportal.client は比較的軽いコンテナですが、その使用が不可能な場合もあります。例えば、通常のアプリケーション・クライアントではないテスト・ツールなどは、**iportal.client** 内で動作しないことがあります。

また、スタンドアロン・クライアントの場合、アプリケーション・クライアントに比べてランタイムに必要な条件が少ないので、iPortal Application Server を実行しているホスト上にないリモート・マシンに EJB クライアントをデプロイする場合などはスタンドアロン・クライアントの使用が適しています。

さらに、アプリケーション・クライアントは、一度パッケージ化されると **applcation-client.xml** に指定されている特定の EJB に関連付けられますが、スタンドアロン・クライアントは EJB の JNDI 名をランタイムに取得できるので、柔軟性に優れています。

スタンドアロン Java クライアント用 ランタイム環境の作成

概要 スタンドアロン Java クライアントを使用してリモートの iPortal Application Server でデプロイされた Bean にアクセスするためのランタイム環境を整えるには、次の手順を行います。

1. **\ipas3\etc\domains** ディレクトリの **default-domain.cfg** ファイルで **IT_USER_HOST_NAME** の値を **localhost** から実際のマシン名に変更する。変更後、iPortal Application Server を再起動する。

2. スタンドアロン・クライアントを実行するマシンに **default-domain.cfg**、**orbix2000.cfg**、**ipas-defaults.cfg**、**embedded-services.cfg** の各ファイルをコピーする。

3. **default-domain.cfg** ファイルに **orbix2000.cfg**、**ipas-defaults.cfg**、および **embedded-services.cfg** の各ファイルへの絶対パスを追加する。

4. 変更後の **default-domain.cfg** は次のようになる。

```
# Setup default settings.
IT_PRODUCT_DIR = "i:\ipash5";
IT_USER_HOST_NAME = "laptop1";
IT_NAMING_PORT = "3074";
IT_LOCATOR_PORT = "3075";
# The basic ART config for this domain.
#include "i:\ipash5\ipas3\etc\domains\orbix2000.cfg";
# The basic iPAS config for this domain.
#include "i:\ipash5\ipas3\etc\domains\ipas-defaults.cfg";
# The embedded services config for this domain.
#include "i:\ipash5\ipas3\etc\domains\embedded-services.cfg";
# The external services config for this domain.
#
#include "d:\ipash5\ipas3\etc\domains\external-services.cfg";
# The iPA config for this domain.
#include "i:\ipash5\ipas3\etc\domains\ipa.cfg";
```

5. クライアントをデプロイするマシンで、**default-domain.cfg** のあるディレクトリをクラスパスに含める。

これらのコンフィギュレーション変更に加えてリモート・インターフェイス、ホーム・インターフェイス、および Bean で定義されている例外の全てをクライアントが使用できるように設定します。さらに、クライアント・マシンのクラスパスに **iportal.jar**、**ipas.jar**、**orbix2000.jar**、**omg.jar**、**spy.jar**、**sljc_brand.jar**、および **sljcx.jar** の各 JAR ファイルを追加します。

クライアント VM での EJB 固有のクラスの設定

概要

EJB にアクセスするには、クライアントの VM のクラスパスに EJB 固有のクラスを設定する必要があります。JNDI に対して **lookup()** を呼び出してホーム・インターフェイスを取得すると、クライアントにスタブがダウンロードされます。このスタブはそのホーム・インターフェイスで宣言されている全メソッドのローカルな実装を提供します。

ホーム・インターフェイスの **create()** メソッドが呼び出されると、EJB ホームが Bean のリモート・インターフェイスを実装するスタブを返します。したがって、各タイプの Bean につきホーム・インターフェイスとリモート・インターフェイスを表す 2 つのスタブが作成されます。これらのスタブはアプリケーション・サーバから自動的にダウンロードされます。クライアントがホーム・インターフェイスとリモート・インターフェイス両方のリモート・リファレンスをナローイングしこれを使用するには、そのクライアントのクラスパスに次の Java クラス・ファイルが必要です。

- ホーム・インターフェイス
- リモート・インターフェイス

Bean クラスには直接アクセスすることはないので、クライアント・アプリケーションがこれらを使用する必要はありません。スタブ・クラスはビルド時に iPortal Application Server により生成され、Bean JAR ファイルにパッケージ化されます。

その他の必要なクラス

アプリケーションによってはクラスパスにその他のクラスを追加する必要があります。具体的には次のようなクラスを追加します。

- リモート・インターフェイスのビジネス・メソッドにより送出されるアプリケーション固有の例外
- ビジネス・メソッドから戻り値として返されるクラス

- リモートのビジネス・インターフェイスにより拡張されたアプリケーション固有のインターフェイス
- iPortal Application Server の **iportal.jar** クラス
- iPortal Application Server のコンフィギュレーション・ファイル

例 次の例はビジネス・インターフェイスを使用しています。

```
public interface ConverterBusiness {
public abstract ConverterData mmToInch(float f) throws
RemoteException, ConversionException;
...
}
```

ここではリモート・インターフェイスが次のようにビジネス・インターフェイスを拡張しています。

```
public interface Converter extends EJBObject, ConverterBusiness {
}
```

アプリケーションの Bean クラスは次のとおりです。

```
public class ConverterBean implements SessionBean,
   ConverterBusiness {
public ConverterData mmToInch(float millimetres) throws
   ConversionException {
                ...}
                ...
                }
```

この例ではクライアントのクラスパスに次の各クラスを追加する必要があります。

```
ConverterBusiness.class
ConversionException.class
ConverterData.class
Converter.class
ConverterHome.class
```

第 8 章

アプリケーション・サービス

本章では iPortal Application Serverp が提供する基本的なサービスについて説明します。これらのサービスにより、各コンポーネントが低レベルの実装ディテールに関わる必要がなくなり、エンタープライズ・アプリケーションのビジネス・ロジックの処理に集中できます。アプリケーション・サービスはネットワーキング、認証、永続性、および EJB とサーブレットのリモート・オブジェクト・アクセスのサービスを行います。アプリケーションが使用するその他のサービスへのポータブル・アクセスは、標準の Java API を介して提供されます。

iPortal Application Server が提供するサービスには、データおよびアクセス・サービスとメッセージング・サービスがあります。

本章は、次のセクションで構成されます。

データおよびアクセス・サービス

概要

iPortal Application Server はアプリケーションやコンポーネントにデータおよびアクセス・サービスを提供するため標準の J2EE 技術を実装しています。これらのサーバには次が含まれます。

- JNDI（Java Naming and Directory Interface）
- JDBC（Java Database Connectivity）
- JTA（Java Transaction API）

セクションの内容

本セクションは次のトピックで構成されます。

JNDI

概要

JNDI は、Java アプリケーションにネーミング・サービスを提供するアプリケーション・プログラミング・インターフェイス（API）です。iPortal Application Server では CORBA ネーミング・サービスを使用して J2EE アプリケーションをサポートします。

ネーミング・サービスの管理

ネーミング・サービスは、クライアントが名前を使用して特定のオブジェクトを検索できるように名前とオブジェクトを関連付けます。**ディレクトリ・サービス**はネーミング・サービスを拡張したもので、オブジェクトに名前だけでなく属性を関連付けることができます。ネーミング・サービスとディレクトリ・サービスを使用して、オブジェクトをその名前や属性により検索することができます。

JNDI は J2EE プラットフォームで重要な役割を果たします。例えば EJB コンテナは、Bean のホーム・インターフェイスとリモート・インターフェイスを提供する際に JNDI 名をパブリッシュし、クライアントはこの JNDI 名を検索することにより Bean へのアクセスを取得します。

JNDI API

JNDI API は、Java で記述されたアプリケーションにネーミング・サービスとディレクトリ・サービスを提供する標準のインターフェイスです。JNDI はネーミング・サービスとディレクトリ・サービスへのインターフェイスを提供しますが、これらのサービスの実装は提供しません。したがって、JNDI を使用するアプリケーションではさまざまな実装が可能です。これらサービスの各実装には、SPI（Service Provider Interface）が関連付けられています。SPI は JNDI API の呼出しをそれに対応する実装メカニズムにマッピングします。

iPortal Application Server は標準の CORBA ネーミング・サービスを使用します。CORBA ネーミング・サービスの定義については OMG（Object Management Group）の INS（Interoperable Naming Specification）を参照してください。

JNDI と iPortal Application Server の概要

概要

iPortal Application Server は JNDI API を使用してオブジェクトにコンテキストを関連付けます。図 8 の例では、JNDI が CORBA ネーミング・サービスの SPI を使用してネーミング情報を格納するよう、サーバが指定します。

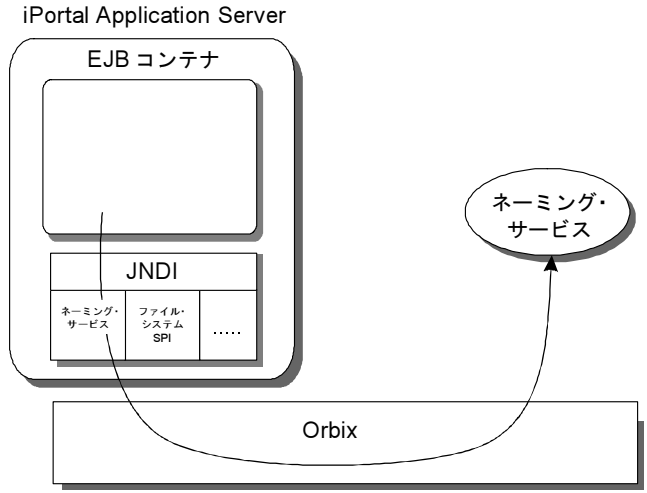


図 8:　JNDI を使用したネーミング・サービスへのアクセス

ネーミング・サービスを管理する際、iPortal Application Server が作成するネーム・バインディングのタイプを把握しておくことが重要です。バインディングのタイプがわかれば、バインディングが正しく作成されたかを監視し、不要になったバインディングを削除することができます。

iPortal Application Server は、次の場合にネーミング・サービスを使用します。

- JNDI コンテキストと EJB ホーム・オブジェクトの関連付け
- 内部オブジェクトの登録

JNDI コンテキストと EJB ホーム・オブジェクトの関連付け

1 つのコンテナで、複数のエンタープライズ Bean をデプロイすることができます。コンテナ内の各 Bean につきそのホーム・インターフェイスを実装する EJB ホーム・オブジェクトが 1 つあります。クライアントは、このインターフェイスを使用して Bean インスタンスの作成および検索を行います。

クライアントは JNDI API を使用して EJB ホーム・オブジェクトへのリファレンスを取得し、EJB ホーム・オブジェクトに関連付けられている JNDI 名を調べます。iPortal Application Server 内のコンテナが、エンティティBean の EJB ホーム・オブジェクトを JNDI ネームスペースで使用できるようにします。

Bean をデプロイする際、デプロイヤがその Bean の EJB ホーム・オブジェクトの JNDI 名を指定します。デプロイヤがコンテキストを指定するには、コンテナ・コンフィギュレーション・ファイルの jndi-name エレメントを使用します。

デプロイヤが Bean の JAR ファイルをコンテナにインストールしてコンテナを構成すると、コンテナが JNDI コンテキストをその Bean の EJB ホーム・オブジェクトにバインドします。このバインディングは JNDI SPI を介して CORBA ネーミング・サービスに格納されます。

EJB ホーム・オブジェクトのネーム・バインディングはデプロイヤがコンテナの構成を終えた時点で、ネーミング・サービスに永続的に格納されます。これらのネーム・バインディングは、iPortal Application Server を終了や再起動しても再作成されません。

内部オブジェクトの登録

iPortal Application Server は EJB ホーム・オブジェクトのネーム・バインディングを格納するだけでなく、内部的に使用するバインディングも作成します。ネーミング・サービスを管理するためには内部オブジェクトのバインディングについて深い知識は不要ですが、こうしたバインディングが存在するということは理解しておいてください。

実行している各 iPortal Application Server が、ネーミング・サービス内でバインディングを 1 つ作成します。このバインディングには、サーバに割り当てられた ORB 名に基づく名前が関連付けられます。例えば、サーバの ORB 名が **iPAS.Server.Default** の場合、内部サーバ・オブジェクトにはこれに対応する JNDI 名、**iPAS/Server/Default** がバインドされます。前向きスラッシュ（**/**）は名前の各コンポーネントの区切り文字です。

サーバの JNDI コンテキストの設定

概要 iPortal Application Server は起動時にそれ自身へのリファレンスを JNDI に登録します。このときデフォルトでは、このオブジェクトに関連付けられる JNDI コンテキストとしてサーバ名が使用されます。JNDI ネームスペースで競合が生じないよう、**ipas_server_jndi_name** という変数を使用してサーバの代替 JNDI 名を指定できます。

例 例えば、デフォルト・サーバに **MyCo.MyServer** という JNDI 名を使用するには、コンフィギュレーション・ファイルで次の値を設定します。

```
iPAS{
      ...
      Server{
          ...
          Default {
             ipas_server_jndi_name = "MyCo/MyServer";
          };
      };
};
```

JDBC

概要 JDBC は、標準の SQL データベース・アクセス・インターフェイスで、これによりバックエンド・データベース・リソースへのアクセスを提供します。JDBC は、高レベルのツールやインターフェイスを構築できる共通のプラットフォームも提供します。アプリケーション・デプロイメントでは、コンポーネントへのリソース割当てが重要なタスクとなります。Bean では、通常リソースとしてデータソースを使用します。したがって、Bean がデータソースを使用する方法および、これらのデータソースへのアクセスを提供する方法を理解しておく必要があります。

EJB アプリケーションは、分散システムの中層で動作してリモート・クライアントに必要なビジネス機能を提供し、これらのクライアントをバックエンド・データソースから隔離するように設計されています。通常、データソースへのアクセスは Bean が担当します。

データソースの使用

Bean によるデータ・アクセスは EJB コンテナが管理します。例えば Bean のソースコードが特定のデータベースに明示的に参照する場合、Bean がそのデータベースとのコネクションを確立する方法をコンテナが提供します。Bean はそのデプロイメント・デスクリプタを介して特定のデータベースの必要条件を指定し、デプロイヤがその条件を満たします。Bean で行われる全てのリファレンスは論理的なもので、アセンブラまたはデプロイヤにより具現化されます。

このようにコンテナが仲介処理を行うことには多数の利点があります。例えばこの方法では、Bean のデータソース・アクセスをデプロイメント時点で特定のデータベースにバインドしたり、コンテナがコネクションの使用状況を最適化したり、またトランザクションに関連するデータ・アクセスをコンテナが制御することが可能です。iPortal Application Server は実際のデータソースをラッパを使用してラッピングし、このラッピングされたデータソースが元のデータソースに委譲されます。データベース・アクセスが必要なアプリケーションをデプロイする場合、その前に Bean で使用できるデータ・アクセスのタイプを把握し、コンテナがデータソースとの通信をどのようにサポートするかを理解しておく必要があります。

データソースについて詳しくは、135 ページの「データソースの使用」を参照してください。

JTA

概要

JTA は、アプリケーションとアプリケーション・サーバがトランザクションを使用できるようにする API です。トランザクション処理は、エンタープライズ・アプリケーション開発では重要な手法です。iPortal Application Server では、エンタープライズ・アプリケーションでのトランザクション使用を完全にサポートしています。

トランザクションとは特定の処理のまとまりで、数個のプログラミング手順を含むこともあります。アプリケーションのデータ整合性を維持するには、トランザクションの実行でその各手順が正常に完了する必要があります。

ロールバックとコミット

トランザクションの 1 手順が失敗した場合、そのトランザクションの全行程を**ロールバック**する必要があります。ロールバックを行うと、トランザクションが変更を試みたデータは元の状態に戻ります。全行程が正常に完了すると、トランザクションが**コミット**され、このトランザクションで行われた変更内容が全て適用されます。

アプリケーションでトランザクションを使用すると、複雑なコードを記述しなくてもエラーからのリカバリやマルチユーザ処理をサポートすることができます。iPortal Application Server では、エンタープライズ・アプリケーションでのトランザクション使用に完全対応しています。

トランザクションについて詳しくは、123 ページの「EJB での トランザクションの使用」を参照してください。

メッセージング・サービス

概要

J2EE のメッセージング技術は標準の API を採用しているので、iPortal Application Server アプリケーションはこれを使用して相互に通信したり、iPortal Application Server 以外のアプリケーションと通信することが可能です。メッセージング・サービスには次が含まれます。

- JMS（Java Message Service）
- JavaMail

JMS

JMS API は、JMS 対応の全メッセージング・システムでサポートされるメッセージング概念とプログラミング手法のセットを定義しています。JMS を使用すると、各アプリケーションがメッセージをやり取りして互いに通信することができます。

JavaMail

JavaMail API は標準の Java API を提供し、これにより標準のインターネット・メール・プロバイダとの通信が可能になります。

JAF JAF は、標準の拡張機能として実装されています。Sun では JAF ソフトウェアのリファレンス実装のバイナリ・ファイルを無償で提供しています。開発者はこれを利用して、JDKTM（Java Development Kit）1.1 ソフトウェア、または JavaTM 2 Standard Edition をサポートする全プラットフォーム用に JAF 技術対応のアプリケーションを作成できます。

第 9 章

EJB での トランザクションの使用

トランザクション処理は、エンタープライズ・アプリケーションを開発する上で大変重要な手法です。本章では iPortal Application Server でのトランザクションのサポートについて述べ、トランザクション処理サポートをエンタープライズ・アプリケーションで使用するための構成方法について説明します。

本章は、次のセクションで構成されます。

トランザクションの特性

概要

トランザクションとは特定な処理のまとまりで、これには数個のプログラミング手順が含まれることもあります。アプリケーションのデータ整合性を維持するには、トランザクションの実行過程においてその各手順が正常に完了する必要があります。

トランザクションの 1 手順が失敗した場合、そのトランザクションの全行程を**ロールバック**する必要があります。ロールバックを行うと、トランザクションが変更を試みたデータは元の状態に戻ります。全行程が正常に完了すると、トランザクションが**コミット**され、このトランザクションで行われた変更内容が全て適用されます。

アプリケーションでトランザクションを使用すると、複雑なコードを記述しなくてもエラー・リカバリやマルチユーザ処理をサポートすることができます。iPortal Application Server では、エンタープライズ・アプリケーションでのトランザクション使用に完全対応しており、本章ではその概要について説明しています。

トランザクションの一般的な使用例としては、バンキング・アプリケーションの資金転送が挙げられます。このトランザクションには送金元口座からの引出しと送金先口座への振込みという 2 つの手順が必要です。これをまとめて 1 つのトランザクションとして処理するには、次のような特性が必要です。

- 引出しと振込みの両方が成功するか、もしくはこの両方が失敗すること。
- システムの合計金額がトランザクションの前後で一致すること。
- 引出しと振込みの間にシステムの一貫性が失われても、この状態がアプリケーションの他の部分には見えないこと。
- トランザクション全体のコミット後の結果が永久に保存されること。

ACID 特性

トランザクションには次の 4 つの特性が必要です。これらをまとめて ACID **特性**と呼びます。

- Atomicity（原子性）
- Consistency（一貫性）
- Isolation（隔離性）
- Durability（耐久性）

Atomicity 原子性。そのトランザクションが成功するには、そこに含まれる全てのオペレーション が成功する必要があります。トランザクション完了時にデータへの変更は全て同時に コミットされるか ロールバックされます。

Consistency 一貫性。トランザクションにより一連の処理を行ったシステムは、一貫性のある特定な状態から、別の状態へ移行します。トランザクションはセマンティクスおよび物理的な意味の両方でデータの整合性を維持します。

Isolation 隔離性。システムにアクセスしている他のエンティティ はトランザクション処理中の途中結果にはアクセスできません。

Durability 耐久性。トランザクションの結果は永続化され、システムあるいはメディアにエラーが発生してもそのリカバリが可能です。

EJB アプリケーションのトランザクション

概要 エンタープライズ・アプリケーション・アーキテクチャにとって、トランザクション処理のサポートは重要な役割を果たします。EJB サーバは J2EE アーキテクチャの一部としてトランザクション処理をサポートしているので、トランザクションを使用したアプリケーションでは各コンポーネントが相互処理を行うことが可能です。

EJB サーバは、トランザクション管理と処理の手間を省きます。EJB サーバにはトランザクション・マネージャが実装されていて、これにより、JDBC データソースとの通信、コンポーネント間でのトランザクションの送受信などが可能になります。

EJB サーバにより、Bean 開発者やアプリケーション・アセンブラはトランザクションの開始と終了の定義に集中できます。トランザクションの開始と終了を定義するには、**コンテナ管理のトランザクション区分**と **Bean 管理のコンテナ区分**の 2 つの方法があります。

コンテナ管理のトランザクション区分

コンテナ管理のトランザクション区分を使用する場合、EJB コンテナが各トランザクションの区分を設定します。コンテナが各トランザクションを開始し、その結果を必要に応じてコミットまたはロールバックします。この方法は、セッション Bean とエンティティBean の両方に使用できます。

このトランザクション区分方法では Bean のソースコードで各トランザクションの区分を明示的に定義する必要がありません。Bean のデプロイメント・デスクリプタに Bean のメソッドの**トランザクション属性**が含まれていて、この属性がコンテナによるトランザクション区分の方法をランタイムに指定します。

通常の場合、コンテナはメソッド実行の直前にトランザクションを開始し、メソッド終了の直前にトランザクションをコミットします。メソッドでは入れ子になったトランザクションや複数のトランザクションは使用できません。トランザクションが失敗した場合（例えばメソッドが処理中にシステム例外を送出した場合）は、コンテナがそのトランザクションをロールバックします。

アプリケーション固有のエラーによりトランザクションが失敗した場合（例えばメソッドがユーザ例外を送出した場合）は、Bean のソースコードが明示的にトランザクションをロールバックできます。これを行うには Bean が `javax.ejb.SessionContext` オブジェクト、または `javax.ejb.EntityContext` オブジェクトの `setRollbackOnly()` メソッドを呼び出します。

Bean のトランザクション属性の設定

トランザクション属性により、メソッドが適切なトランザクション処理を実行するように指定できます。メソッドの実行方法には、トランザクションなしの実行、独自の新規トランザクションのコンテキスト内での実行、そして既存トランザクションのコンテキストでの実行、の 3 つがあります。

既存のトランザクションのコンテキストで実行されたメソッドは、そのメソッドを呼び出したクライアントが開始したトランザクションの一部となります。EJB アプリケーションでは、トランザクションを開始するクライアントが JSP やサーブレットの場合もありますが、別の Bean がトランザクションを開始することもよくあります。

Bean のメソッドへのトランザクション属性の割当ては Bean プロバイダまたはアプリケーション・アセンブラが行います。各メソッドには次のいずれかの属性を設定できます。

Required	このメソッドは常にトランザクションのコンテキストで実行される。呼出し元にトランザクションが関連付けられている場合、そのトランザクション内で実行され、それ以外の場合は新規トランザクション内で実行される。このトランザクションはメソッドが終了する直前にコミットする。
RequiresNew	このメソッドは常に新規トランザクションのコンテキストで実行される。呼出し元にトランザクションが関連付けられている場合、コンテナが現在のスレッドとそのトランザクションとの関連付けをサスペンドしてから、新規トランザクションを実行する。このトランザクションが完了した時点でコンテナはサスペンドしていたトランザクションの実行を再開する。
NotSupported	このメソッドは、トランザクションのコンテキストでは実行されない。呼出し元にトランザクションが関連付けられている場合、コンテナが現在のスレッドとそのトランザクションとの関連付けをサスペンドしてからメソッドを実行する。このメソッドが完了した時点でコンテナはサスペンドしていたトランザクションの実行を再開する。
Supports	呼出し元にトランザクションが関連付けられている場合、このメソッドはそのトランザクションのコンテキストで実行される。トランザクション・コンテキストが関連付けられていない場合は、トランザクションなしで実行される。
Mandatory	呼出し元にトランザクション・コンテキストが関連付けられている場合、このメソッドはそのトランザクションのコンテキストで実行される。トランザクション・コンテキストが関連付けられていない場合、**javax.transaction.TransactionRequiredException** 型の例外を送出する。
Never	このメソッドはトランザクションのコンテキストでは実行されない。呼出し元にトランザクション・コンテキストが関連付けられている場合、コンテナが **java.rmi.RemoteException** 型の例外を送出する。

セッション Bean

セッション Bean の場合、Bean のビジネス・メソッドのみにトランザクション属性を設定できます。

エンティティ Bean

エンティティBean の場合、Bean のビジネス・メソッドと **remove**、**create**、および **find** の各メソッド全てにトランザクション属性を設定できます。

Bean プロバイダやアプリケーション・アセンブラは Bean の各メソッドに別々に属性を割り当てるか、Bean の全メソッドに一括して属性を割り当てることができます。Bean 全体と個々のメソッド両方に属性が割り当てられている場合、個々のメソッドの属性設定が優先されます。

Required は一番よく使用される属性です。Bean プロバイダまたはアプリケーション・アセンブラは、Bean の全メソッドに **Required** 属性を設定することがよくあります。

Bean プロバイダは、Bean のデプロイメント・デスクリプタを作成する時点で、トランザクション属性を指定します。アプリケーション・アセンブラは、グラフィカル・アプリケーション・ビルダを使用して、これらのトランザクション属性を変更することができます。

コンテナ管理のトランザクション区分の利点

コンテナ管理のトランザクション区分を使用する場合、Bean のソースコードにトランザクション・ロジックを含める必要はほとんどありません。この方法では、トランザクション区分をコンテナが行うので、Bean のソースコードが簡略化され、よりロバストで保守を行いやすいコードを記述することができます。

トランザクション属性は宣言の形で指定されるので、アプリケーション・アセンブラはコンテナ管理のトランザクション区分を使用した Bean を自由に組み合わせることができます。アセンブラは特定アプリケーションのニーズに合わせてトランザクション属性を設定できるので、既存の Bean を再利用できる可能性が高くなります。

Bean 管理のトランザクション区分

Bean 管理のトランザクション区分を使用する場合、各トランザクションの開始、コミット、ロールバックを Bean のソースコードにより指定します。Bean 管理のトランザクション区分はセッション Bean でのみ使用でき、エンティティBean には使用できません。

Bean 管理のトランザクション区分を使用したセッション Bean は、JTA（Java Transaction API）トランザクションを作成できます。Bean プロバイダは Bean の開発時に `javax.transaction.UserTransaction` インターフェイスを使用して JTA トランザクションを作成します。

JTA トランザクションは、J2EE トランザクション・マネージャにより管理されます。JTA は、トランザクション・マネージャの実装と関係なく、Bean プロバイダがトランザクションを制御するために使用する標準インターフェイスを提供します。1 つの JTA トランザクションは複数ベンダが行う複数のデータベース更新を処理できます。

Bean 管理のトランザクション区分の利点

大半の EJB アプリケーションではコンテナ管理のトランザクション区分を使用しています。これは、Bean プロバイダがトランザクション区分のコードを記述する必要がないこと、またトランザクション・アプリケーション内での Bean の相互関係をアセンブラが構成できることなどが理由です。

これに対して Bean 管理のトランザクション区分では、Bean プロバイダが次のように細かいトランザクション制御を行えるのが最大の利点です。

- 1 つのメソッドで複数のトランザクションの開始と終了を行える。
- ステートフルなセッション Bean では、メソッドがデータベース・コネクションを解除する場合でも、1 つの JTA トランザクションで異なるメソッドへの複数の呼出しを処理できる。

このような細かいレベルの制御が必要なアプリケーションでは Bean 管理のトランザクション区分を使用します。

iPortal Application Server のトランザクション

概要

iPortal Application Server は、JTS（Java Transaction Service）を使用して分散トランザクションをサポートします。JTS はトランザクション・マネージャで、Bean プロバイダが JTA を使用できるようにします。JTS は低レベルでは Orbix オブジェクト・トランザクション・サービス（OTS）の Java マッピングを実装します。

図 9 に、トランザクション処理に関わる iPortal Application Server のコンポーネントを示します。

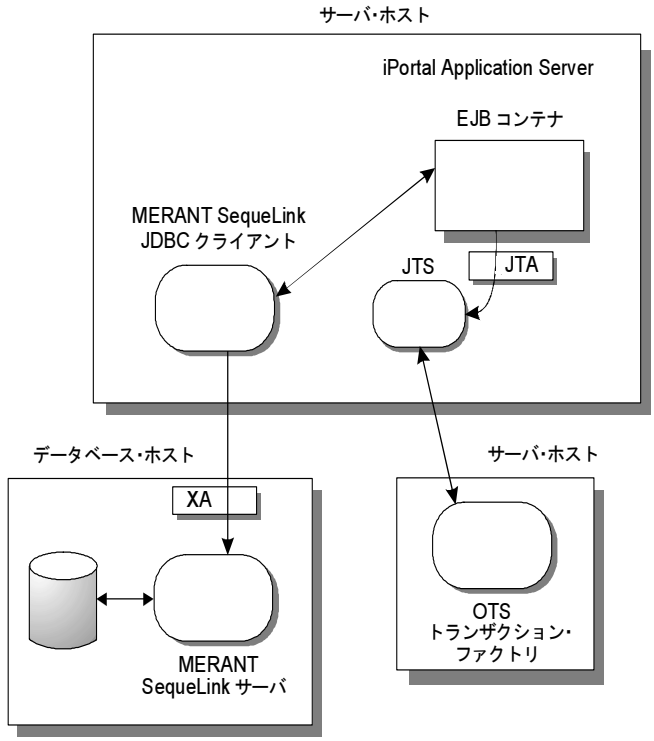


図 9: iPortal Application Server のトランザクション処理

EJB コンテナや Bean がトランザクションを開始すると、iPortal Application Server が Orbix OTS トランザクション・ファクトリに対して新規トランザクションの作成を要求します。トランザクション・ファクトリは、トランザクションの監視、コーディネート、およびコミットを処理します。

トランザクション中に EJB コンテナまたは Bean がデータソースにアクセスする場合、iPortal Application Server がトランザクション・ファクトリと通信してトランザクションでデータソースを使用できるようにします。その後、コンテナや Bean が JTA を使用してデータソースと通信します。130 ページの図 9 の例では、MERANT SequeLink データベース・ドライバにより、Oracle 8i へのアクセスが提供されています。

トランザクションを開始した EJB コンテナまたは Bean が、JTA を使用してトランザクションのコミットを試行します。iPortal Application Server は、トランザクションのコミットやロールバックのコーディネートを行うトランザクション・ファクトリとの通信を行います。

コンテナ管理のトランザクション区分

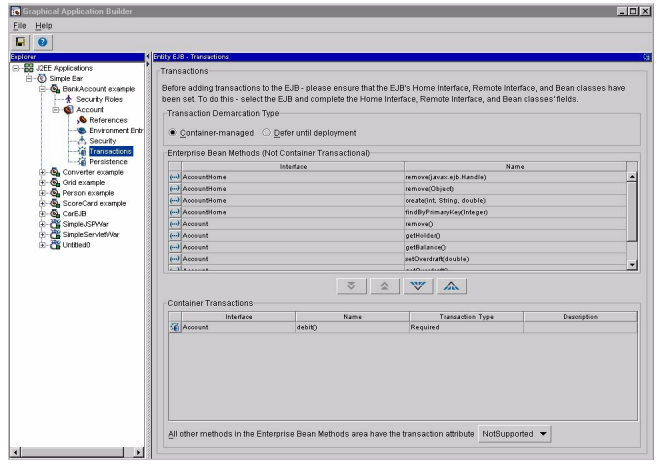
グラフィカル・アプリケーション・ビルダで、アプリケーションで使用するメソッドのトランザクション属性を指定します。iPortal Application Server で言う**コンテナ・トランザクション**とは、トランザクション属性 1 つとそれに関連付けられているメソッドにより構成されます。

コンテナ管理のトランザクションを新規作成するには、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.assembler アイコンをクリックして、グラフィカル・アプリケーション・ビルダを起動する。

2. メニューバーから File → Open Archive を選択し、Bean が入っている JAR ファイルを展開表示する。

3. Bean を選択し、Explorer パネルにあるリストから Transactions オプションを選択する。Bean のコンテナ・トランザクションの詳細が次のように表示される。



4. Transaction Demarcation Type エリアで Container-managed ボタンをオンにする。Enterprise Bean Methods エリアにコンテナ・トランザクションに関連付けられているメソッドが一覧表示される。

5. デプロイメントの時点でデプロイヤがコンテナ・トランザクションのメソッドを関連付けるためには、Enterprise Bean Methods エリアで Defer until deployment ボタンをオンにする。

6. Enterprise Bean Methods エリアでコンテナ・トランザクションに関連付けるメソッドをハイライトする。Add を選択して強調表示したメソッドを Container Transactions エリアに移動するか、全てのメソッドを関連付けるには、Add All を選択して Container Transactions エリアに全メソッドを移動する。

7. メニューバーから File → Save を選択すると、グラフィカル・アプリケーション・ビルダがアプリケーションの JAR ファイルを保存する。

Bean 管理のトランザクション区分

Bean 管理のトランザクション区分を使用する Bean を開発する場合、トランザクションの開始と終了を行うコードを記述する必要があります。トランザクションの一部をなすビジネス・ロジックはトランザクションの開始と終了の間に挿入します。

次の例は、トランザクションのコンテキスト内で 2 つのデータベースにアクセスするビジネス・メソッドを示してます。

```
//Java
public class DataAccessExampleEJB implements SessionBean {
  EJBContext ejbContext;

  public void accessDatasource (...) {
    javax.transaction.UserTransaction ut;
    javax.sql.DataSource ds1;
    javax.sql.DataSource ds2;
    java.sql.Connection con1;
    java.sql.Connection con2;
    java.sql.Statement stmt1;
    java.sql.Statement stmt2;

    // Start a transaction.
    ut = ejbContext.getUserTransaction();

1   ut.begin();

    InitialContext initCtx = new InitialContext();
```

```
    // Get connection to first data source.
2
    ds1 = (javax.sql.DataSource)
      initCtx.lookup ("java:comp/env/jdbcDatabase1");
    con1 = ds1.getConnection();
    stmt1 = con1.createStatement();

// Get connection to second data source.
    ds2 = (javax.sql.DataSource)
      initCtx.lookup ("java:comp/env/jdbcDatabase2");
    con2 = ds2.getConnection();
    stmt2 = con2.createStatement();

    // Use the two data sources.

3   stmt1.executeQuery(...);
    stmt1.executeUpdate(...);
    stmt2.executeQuery(...);
    stmt2.executeUpdate(...);
    stmt1.executeUpdate(...);
    stmt2.executeUpdate(...);

    // Close the connections.

4   stmt1.close();
    stmt2.close();
    con1.close();
    con2.close();

    // Commit the transaction.

5   ut.commit();
  }

  ...
}
```

このコードの動作は次のとおりです。

1. **accessDatasource()** メソッドがトランザクションを開始する。
2. このメソッドが 2 つの JDBC データソースへのアクセスを確立する。
3. 一連の SQL ステートメントがデータソースにアクセスし、更新とクエリーを実行する。

4. メソッドが 2 つのデータベース・コネクションを解除。

5. トランザクションをコミットする。

iPortal Application Server ではこれらの手順を正しい順序で実行することが重要です。Bean ではデータベース・コネクションを確立してからそのデータベースとの通信用にトランザクションを開始したり、その後このトランザクションをコミットしてからデータベース・コネクションを解除することはできません。Bean はデータベース・コネクションを確立する前に、必ずトランザクションを開始し、コネクションを解除した後でトランザクションをコミットする必要があります。

第 10 章

データソースの使用

コンポーネントへのリソース割当ては、アプリケーション・デプロイメントの基本的なタスクです。通常、Bean はデータソースをリソースとして使用するので、開発者は Bean がデータソースを使用する方法、およびこれらのデータソースへのアクセスを提供する方法を理解しておく必要があります。

エンタープライズ・アプリケーションは、分散システムの中層で動作してリモート・クライアントに必要なビジネス機能を提供し、これらのクライアントをバックエンド・データソースから隔離するように設計されています。したがって、データソースへのアクセスは通常 Bean が行います。

本章は、次のセクションで構成されます。

Bean によるデータソース・アクセス

概要

Bean がデータソースにアクセスする場合、メソッド呼出しを通じて直接アクセスする方法と、EJB コンテナが提供するサービスを通じて間接的にアクセスする方法の 2 つがあります。エンティティBean とセッション Bean は、状況に応じてこの両方のアクセス方法を使用します。

エンティティ Bean

エンティティBean は、アプリケーションまたは EJB サーバのライフタイム期間中に生存します。この期間中は Bean の状態が永続的である必要があるため、データベースなどの 2 次的なストレージによって Bean の永続状態が保持される必要があります。

エンティティBean は、コンテナ管理永続性（EJB コンテナが Bean の永続状態を 2 次ストレージを使用して保持）ならびに Bean 管理永続性（Bean が明示的に永続状態の読み書きを行う）の両方を使用します。

セッション Bean

セッション Bean には一貫性がありません。セッション Bean は 1 つのクライアントに関連付けられていて、そのクライアントがセッションを終了した時点で Bean も使用できなくなります。ただし、セッション Bean のソースコードが直接データソースに接続して、処理中に Bean が必要とするデータを取得することはあります。

セッション Bean には一貫性がありませんが、コンテナがステートフルなセッション Bean の状態をデータソースに書き込むことはあります。これは、例えば Bean の使用頻度が低いなどの理由でコンテナが一時的に Bean を非活性化した場合に起こります。これを Bean の**パッシブ化**（あるいは非活性化、パッシベーション）と呼びます。

デプロイメント・デスクリプタ

Bean のデプロイメント・デスクリプタは、その Bean に必要なリソースの条件を指定します。これは例えば、Bean のソースコードが特定タイプのデータソースに参照するかどうか、またエンティティBean がコンテナ管理永続性を必要とするかどうかなどの指定です。これらの条件を満たすには、デプロイメントの時点でコンテナ構成を使用します。

コンテナによるデータソース・アクセスの提供

概要

iPortal Application Server では、JDBC（Java Database Connectivity）2.0 を介したデータソースへのアクセスをサポートしています。JDBC は、データベース・コネクションの表現、SQL ステートメントの送信、およびデータベースから得た結果の処理を Java で行うための API を提供します。この API は使用するデータソースに依存しません。

データソースへ JDBC アクセスを提供する以外にも、iPortal Application Server では全てのデータ・アクセスをトランザクション・コンテキストで行うことができます。iPortal Application Server のトランザクション処理サポートは、CORBA オブジェクト・トランザクション・サービス（OTS）の IONA による実装である OrbixOTS の上に構築されます。

リソース・マネージャ

iPortal Application Server の EJB コンテナまたは Bean は、**リソース・マネージャ**を介してデータソースとの通信を行います。リソース・マネージャは、例えばデータベース・ドライバのようなシステムで、リカバリ可能なリソースの管理を担当します。EJB コンテナや Bean がリソース・マネージャに接続するには、リソース・マネージャ・コネクション・ファクトリというオブジェクトを使用します。

XA インターフェイス

iPortal Application Server システムでは全てのリソース・マネージャが X/Open 分散トランザクション処理（DTP：distributed transaction processing）モデルの XA インターフェイスを実装しています。XA インターフェイスは、DTP 環境におけるリソース・マネージャとトランザクション・マネージャ間のやり取りに関する規則を定義します。これにより、OrbixOTS がシステム内の共有リソースへのトランザクション・アクセスをコーディネートすることができます。

iPortal Application Server では、JDBC ドライバが関連付けられているデータベース、XA インターフェイスをサポートするデータベース、および IONA によりテスト済みのデータベースへのアクセスをサポートします。

図 10 に、iPortal Application Server でのデータソースとのやり取りを示します。

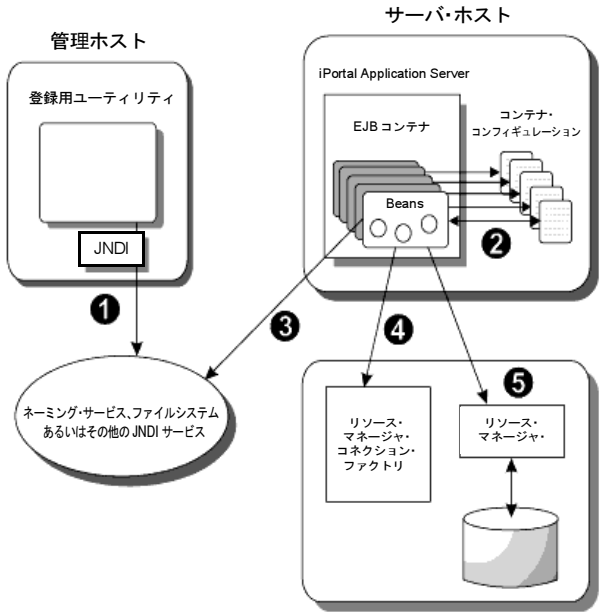


図 10: データソースへの接続

図 10 は、Bean のソースコードがデータソースへの呼出しを直接行う場合と、例えばコンテナ管理永続性を実装するためにコンテナがデータソースに接続する場合の両方に該当します。

手順 データベース・コネクションに必要な手順は次のとおりです。

1. システム管理者が JNDI（Java Naming and Directory Interface）を使用してデータソースを登録する。これにはシリアル化されたオブジェクトを格納できる JNDI SPI を使用する。

2. データソースへのアクセスが必要になると、コンテナがそのコンフィギュレーションを読み込み、JNDI がデータソース名を検索するために使用するプロパティを判断する。コンフィギュレーションは論理的なリソース・リファレンス名を実際の JNDI 名に関連付け、検索に必要な呼出しを提供する。

3. コンテナが JNDI 名を解決し、データソースのリソース・マネージャ・コネクション・ファクトリへのリファレンスを取得する。Bean のソースコードがデータソースに接続する場合、コンテナがこのリファレンスを Bean に渡す。

4. Bean またはコンテナがリソース・マネージャ・コネクション・ファクトリを使用して、データソースに接続する。コンテナは、コネクションを最適化するために、コネクション・プーリングを使用する。

5. Bean またはコンテナがコネクションを使用してデータソースとの間でデータをやり取りする。

コネクション・プーリング

iPortal Application Server ではアプリケーションのパフォーマンスとスケーラビリティを高めるために**コネクション・プーリング**を使用しています。コネクション・プーリングでは、特定のデータソースへアクセスする全アプリケーションをサポートするために必要なだけのコネクションを EJB コンテナが保管します。データソース・コネクションの確立はデータ・アクセス処理の中でも所要時間が一番長いので、コンテナはできるだけ既存のコネクションを再利用します。

コネクション・プーリングは、大半のアプリケーションではデータ・アクセスの頻度が低いことを前提としています。アクセスの頻度が低ければ、EJB コンテナは多数のアプリケーションに対して、使用可能な小数のデータソース・コネクションを動的に割り当てることができます。

データソースの登録

概要

iPortal Application Server には、データソース登録ユーティリティの例が付属しています。このユーティリティの実装は、登録するデータソースのタイプや JNDI への登録方法によって異なります。iPortal Application Server はラッパを使用して実際のデータソースをラッピングしてから、これを元のデータソースに委譲します。

データソース・ラッパのタイプ

iPortal Application Server には次の 2 タイプのデータソース・ラッパが用意されています。

- `PooledDataSource`（分散トランザクションはサポートしない）
- `DataSource`（分散トランザクションをサポート）

IONA のデータソース・ラッパは、トランザクション制御やその他のコンテナ固有の関数に加えてプーリングを実装しているので、データソースには必ずラッパを使用してください。

JNDI へのデータソースの登録

データソースを登録するには **iportal.datasource** ツールを使用します。また、iPortal Application Server インストール・ディレクトリの **util** ディレクトリに保存されている **RegisterDataSource.java** を使用すると、手動でデータソースを登録する方法を学習できます。

コネクション・プーリング

コネクション・プーリングは全てバックグラウンドで処理されるので、アプリケーション・コードに一切影響を与えません。ただし、コネクション・プーリングを使用するアプリケーションではコネクション取得の際に **DriverManager** クラスではなく **DataSource** オブジェクト（**DataSource** インターフェイスを実装するオブジェクト）を使用する必要があります。この **DataSource** オブジェクトは、JNDI ネーミング・サービスへの登録を行います。**DataSource** オブジェクトが登録されたら、アプリケーションは通常の方法で JNDI ネーミング・サービスからこのオブジェクトを取得します。

```
Context ctx = new InitialContext();
DataSource ds = (DataSource) ctx.lookup("DemoDataSource");
```

利点

DataSource オブジェクトがコネクション・プーリングを提供する場合、アプリケーションのコードを変更しなくても自動的にコネクションが再利用され、パフォーマンスが向上します。アプリケーション・ユーザにとっては実際に新しく確立されたコネクションと再利用されたコネクションはまったく同じです。アプリケーションは通常どおりにデータベースに接続して動作し、コネクションが不要になった場合はこれも通常どおりの方法でコネクションを明示的に解除します。プーリングされているコネクションを解除するイベントは、コネクションを後で再利用できるようにコネクション・プールに戻すことをプーリング・モジュールに指示します。

```
Connection con = ds.getConnection();
// Do some database activities using the connection...
con.close();
```

PooledDataSource と **DataSource** はどちらも **javax.sql.DataSource** インターフェイスを実装しているので、アプリケーションの開発者は使用されるデータソースの実際のタイプを気にする必要がなく、通常のデータソースとして使用することができます。

分散トランザクション

複数のデータソースを使用していてこれらのデータソースがグローバルなトランザクションで使用される場合には、**DataSource** がサポートする分散トランザクションを使用する必要があります。そうでない場合は **PooledDataSource** がサポートするローカル・トランザクションのみを使用します。

ローカル・トランザクションでサポートされるデータソース

ローカル・トランザクションでは、SequeLink ドライバ **OraclePooledDataSource** などの JDBC Core 2.0 API（**java.sql**）をサポートする JDBC 2.0 ドライバをデータソースとして使用できます。この API では、データソースとのコネクション確立、データソースへのクエリーと更新ステートメントの送信、および結果処理のためのクラス・オブジェクト作成を行うことができます。

注： **コネクション・プーリングの利点を活かすには、データソースが javax.sql.ConnectionPoolDataSource を実装している必要があります。データソースの実装の内容がわからない場合には、ドライバの提供元までお問い合わせください。**

分散トランザクションでサポートされるドライバ

分散トランザクションでは、SequeLink ドライバ **OracleXADataSource** などの、JDBC 2.0 の分散トランザクションを拡張する標準拡張インターフェイス（**javax.sql.XADataSource**、**javax.sql.XAConnection**、**javax.transaction.xa.XAResource**）をサポートする JDBC 2.0 ドライバを使用できます。

iPortal Application Server にはデータソースの登録と構成を行うための **iportal.datsource** というツールが用意されています。

iportal.datasource の使用方法

iportal.datasource を使用するには、次の手順を行います。

1. iportal.central ツールバーにある次の iportal.datasource アイコンをクリックし、register datasource ウィザードを起動する。

2. Next をクリックして JNDI のサービス・プロバイダを指定する。JNDI データソース情報を保管するための URL ロケーションを指定して、Next をクリックする。Next ボタンはクラスパスに有効なファクトリ・クラスを指定した場合のみ使用可能。

3. データソースの名前を指定して Next をクリックする。これは、通常では、直接使用しないデータソース。

4. ドロップダウン・リストから使用する JDBC データソースを選択する。必要なデータソースがリストにない場合、データソースのクラス名を入力する。このドライバがクラスパスにある場合のみ、Next ボタンが有効になる。データソースを分散トランザクション環境で使用する場合は XA? チェックボックスをオンにして Next をクリックする。

5. コネクション・プーリングを使用するには、このデータソースが `PooledDataSource` インターフェイスを実装している必要がある。

   実装しているかどうか確かでない場合にはデータソースの販売元にお問い合わせください。

6. 使用中の環境に適した JDBC コネクション・プロパティを指定し、Next をクリックする。

7. Oracle ドライバを使用する場合、ドライバのタイプを指定して Next をクリックする。

8. データソース・ラッパの JNDI 名を指定する。（コンテナ・コンフィギュレーション・ファイルの `cc.xml` で使用される名前。）Next をクリックする。

9. iPortal Application Server では、データソースを使用する場合にこれを独自のデータソースでラッピングする。ラッピング処理は `iportal.datasource` ツールを使用して行う。

10. コンフィギュレーション・ファイルの名前を指定する。この手順はオプションで、更新するコンテナ・コンフィギュレーション・ファイルの名前を指定する必要は無い。ファイル名を指定する場合、これにはすでにデータソース情報が保存されている必要がある。Finish をクリックしてウィザードを終了する。

コンテナ・コンフィギュレーション・ファイルの変更

次に、コンテナ・コンフィギュレーション・ファイルの例を示します。

```
<resources>
    <resource>
      <resource-name>JDBCAccounts</resource-name>
      <jndi-name>DemoDataSource</jndi-name>
      <property>
        <prop-name>java.naming.factory.initial</prop-name>
   <prop-value>com.sun.jndi.fscontext.RefFSContextFactory</prop-
   value>
      </property>
      <property>
        <prop-name>java.naming.provider.url</prop-name>
   <prop-value>file:/d:/iona/ipas1.3/ipas/1.3/util/jndi-bindings
   </prop-value>
      </property>
    </resource>
</resources>
<entity>
      <ejb-name>Account</ejb-name>
      <jndi-name>iona/ipas/examples/Account</jndi-name>
      <jndi-source-name>CosNaming</jndi-source-name>
      <resource-ref>
        <res-ref-name>jdbc/Accounts</res-ref-name>
        <res-ref-link>JDBCAccounts</res-ref-link>
      </resource-ref>
   </entity>
Looking up a Data Source in your Application
In your ejb-jar.xml file, declare the following (this example is
   for an entity bean):
<enterprise-beans>
  <entity>
    <resource-ref>
        <res-ref-name>jdbc/Accounts</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>
  </entity>
</enterprise-beans>
```

これで、次のようにこのデータソースを検索して通常どおりに使用できます。

```
InitialContext ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("java:comp/env/"+
   jdbc/Accounts);
Connection con =  ds.getConnection();
// Do some database activities using the connection...
con.close();
```

登録ユーティリティの実行

登録ユーティリティを実行するには、次の手順を行います。

1. 次のコマンドを実行する。

```
setenvs
```

2. クラスパスに **RegisterDataSource.class** ファイルが入っているディレクトリを追加する。次は Windows NT の例。

```
set ClassPath=%ClassPath%D:\iPortal\iPAS3\util;
```

3. **util** ディレクトリからユーティリティを実行する。

```
java RegisterDataSource
```

ユーティリティがホーム・ディレクトリの下の **jdbc** ディレクトリに **.bindings** ファイルを作成したかどうか確認します。このファイルには使用するデータソースの JNDI 名が指定されています。

アプリケーションへのデータソースの割当て

概要

アプリケーションをデプロイする場合、そのアプリケーションの各 Bean にデータソースを割り当てる必要があります。データソースを Bean に割り当てるには次の 2 つの手順を行います。

1. Bean のデータソース要件を確認する。
2. データソースへのリファレンスを解決する。

データソース要件の確認

Bean がデータソースに直接アクセスする必要がある場合、データソース要件は Bean のデプロイメント・デスクリプタに指定されています。一般にこれらの要件は、次の 2 つのカテゴリに分類できます。

- データソースへの直接アクセスする場合の要件
- コンテナ管理永続性を使用するエンティティ Bean 用の要件

データソースへの直接アクセス

セッション Bean やエンティティBean のコードがメソッド呼出しを使用してデータソースにアクセスする場合、その Bean のデプロイメント・デスクリプタには **resource-ref** エレメントが含まれています。

```
<!-- Deployment Descriptor for  Bean -->
<ejb-jar>
 <enterprise-beans>
  <session>
   <ejb-name>StockFeed</ejb-name>
     <resource-ref>
     <res-ref-name>StockPriceDB</res-ref-name>
     <res-type>javax.sql.DataSource</res-type>
     <res-auth>Container</res-auth>
  </session>
 </enterprise-beans>
</ejb-jar>
```

この例のデプロイメント・デスクリプタは、**StockFeed** というセッション Bean が **StockPriceDB** という JNDI 名を使用してリソース・マネージャ・コネクション・ファクトリへのリファレンスを取得するよう指定しています。**res-type** エレメントには、Bean がこのリファレンスを取得するときに受け入れるデータ型を示しています。**res-auth** エレメントの値は **Container** ですが、これはコンテナが Bean の代わりにリソース・マネージャにサインオンすることを示します。この値が **Application** の場合、Bean のコードが直接サインオンすることを示します。

コンテナ管理永続性を使用したエンティティ Bean

コンテナ管理永続性を使用したエンティティBean は、そのデプロイメント・デスクリプタで永続状態を成すフィールドを宣言する必要があります。コンテナはランタイムにこれらのフィールドの値をデータソースとの間でやり取りします。コンテナが管理するフィールドは、デプロイメント・デスクリプタの **cmp-field** エレメントにより宣言されます。次に例を示します。

```
<!-- Deployment Descriptor for Order Bean -->
<ejb-jar>
 <enterprise-beans>
```

```
        <entity>
         <ejb-name>Order</ejb-name>
         ...
         <persistence-type>Container</persistence-type>
         ...
         <cmp-field>
          <field-name>id</field-name>
         </cmp-field>
         <cmp-field>
          <field-name>code</field-name>
         </cmp-field>
         <cmp-field>
          <field-name>quantity</field-name>
         </cmp-field>
        </entity>
       </enterprise-beans>
      </ejb-jar>
```

ここでは**Order**Bean に **id**、**code**、および **quantity** という状態を成す、コンテナにより管理されるフィールドが 3 つあります。

データソース・リファレンスの解決

Bean のソースコードがリソースに直接アクセスする場合、Bean プロバイダが Bean のデプロイメント・デスクリプタでリソース・リファレンスを宣言します。アプリケーション開発者は、デプロイ時にこれらの各リソース・リファレンスを解決する必要があります。

リソース・リファレンスを解決するには、コンテナ・コンフィギュレーションの **resource-ref** エレメントを使用します。次に例を示します。

```
     <!-- Container Configuration -->
     <!DOCTYPE configuration SYSTEM "http://www.iona.com/
    dtds/container_configuration_3_0.dtd">
     <configuration>
      <enterprise-beans>
       <session>
        <ejb-name>StockFeed</ejb-name>
         <resource-ref>
          <res-ref-name>StockPriceDB</res-ref-name>
          <res-ref-link>AppDB1</res-ref-link>
          <jndi-name>databases/finance/stockprice</jndi-name>
          ...
         </resource-ref>
       </session>
      </enterprise-beans>
       <resources>
```

```
        <resource>
          <resource-name>AppDB1</resource-name>
          <jndi-name>Oracle8i</jndi-name>
           <property>
           ...
           </property>
        </resource>
      </resources>
    </configuration>
```

res-ref-name エレメントは Bean リソースを参照するために使用する名前を指定します。このエレメントには Bean のでプロイメント・デスクリプタ内の呼応する **res-ref-name** エレメントに含まれる値と同じ値が含まれている必要があります。

res-ref-link エレメントはロジカル・リソースの名前を指定します。**resource-name** エレメントは登録されたリソースとロジカル・リソースを関連付けます。

jndi-name エレメントは実際のリソースを登録する際に使用する JNDI 名を指定します。この名前は 139 ページの「データソースの登録」で解説されている **RegisterDataSource.java** と同じである必要があります。

また、リソースの JNDI 名に加え、名前の解決を行う際に使用する JNDI SPI に関する情報を EJB コンテナに提供する必要があります。この情報を提供するためには、**resource-ref** の宣言に属性のエレメントを追加します。次はその例です。

```
    <!-- Container Configuration -->
    <!DOCTYPE configuration SYSTEM
  "http://www.iona.com/dtds/container_configuration_3_0.dtd">
   <configuration>
    <enterprise-beans>
     <session>
      <ejb-name>StockFeed</ejb-name>
       <resource-ref>
        <res-ref-name>StockPriceDB</res-ref-name>
        <res-res-link>AppDB1</res-ref-link>
        <jndi-name>databases/finance/stockprice</jndi-name>
        ...
       </resources-ref>
     </session>
    </enterprise-beans>
     <resources>
      <resource>
       <resource-name>AppDB1</resource-name>
       <jndi-name>Oracle8i</jndi-name>
        <property>
```

```
          <prop-name>java.naming.factory.initial</prop-name>
          <prop-value>com.sun.jndi.fscontext.RefFSContextFac
          tory</prop-value>
         </property>
         <property>
          <prop-name>java.naming.provider.url</prop-name>
          <prop-value>file:///home/uid100/jdbc/</prop-value>
         </property>
       </resource>
     </resources>
   </configuration>
```

ここでは、**property** エレメントによって、EJB コンテナが JNDI 初期コンテキストを取得する際に必要な環境プロパティを指定しています。データベース名はファイルシステム SPI を使用して JNDI に登録されます。**resource-ref** エレメントには次の 2 つのプロパティが含まれています。

`java.naming.factory.initial`

ファイル・システム SPI の初期コンテキスト・ファクトリのクラス名

`java.naming.provider.url`

java.naming.factory.initial プロパティで指定されたサービス・プロバイダを構成するための URL 文字列

EJB コンテナはどの JNDI SPI を使用してもリソースの JNDI 名を検索できますが、リソース登録に使用したものと同じ SPI を使用する必要があります。またリソース・リファレンスの **property** エレメントには、コンテナが JNDI の初期コンテキストを検索するときに必要な環境値を指定します。

コンテナ管理永続性を使用したエンティティ Bean の構成

コンテナ管理永続性を使用したエンティティ Bean をデプロイする場合、コンテナが Bean の状態を保管するために使用するデータソースを指定する必要があります。これを指定するには、コンテナ・コンフィギュレーションの Bean の entity エレメントに **cmp-datasource** エレメントを追加します。次に例を示します。

```
    <!-- Container Configuration -->
    <!DOCTYPE configuration SYSTEM
  "http://www.iona.com/dtds/container_configuration_3_0.dtd">
   <configuration>
    <enterprise-beans>
     <entity>
      <ejb-name>Order</ejb-name>
       ...
```

```
        <cmp-datasource>
         <cmp-datasource-link>AppDB</cmp_datasource-link>
          <jndi-name>databases/retail</jndi-name>
           <property>
             <prop-name>java.naming.factory.initial
             </prop-name>
             <prop-value>com.sun.jndi.fscontext.RefFSContext
             Factory</prop-value>
           </property>
           <property>
            <prop-name>java.naming.provider.url</prop-name>
            <prop-value>file:///home/uid100/jdbc/
            </prop-value>
           </property>
        </cmp-datasource>
      </entity>
    </enterprise-beans>
     <resources>
     <resource>
      ...
     </resource>
    </resources>
  </configuration>
```

コンテナはこのコンフィギュレーションを使用して JNDI 名 **databases/retail** を検索し、データソースのリソース・マネージャ・コネクション・ファクトリへのリファレンスを取得します。コンテナは **Order** エンティティBean の状態を保管するためにこのデータソースを使用します。

cmp-datasource エレメントには、コンテナが JNDI 初期コンテキストを取得できるようにする **property** エレメントが含まれています。詳しくは「データソース・リファレンスの解決」を参照してください。

コンテナが管理するフィールドのデータベース・カラムへのマッピング

コンテナ管理永続性を使用したエンティティBean の場合、そのデプロイメント・デスクリプタにはコンテナが管理するフィールドの名前が指定されています。コンテナは JDBC を使用して、コンテナ・コンフィギュレーションの **cmp-datasource** エレメントで指定されたデータソースとの間でフィールド値をやり取りします。

コンテナ・コンフィギュレーションは、コンテナが Bean の状態を保管するデータソースを指定する以外に、コンテナが管理する各フィールドをデータベース・テーブルのカラム名にマッピングするためにも使用できます。次にこの例を示します。

```
<!-- Container Configuration for iPortal Application Server
   Copyright (c) 2000 IONA Technologies PLC. All Rights Reserved.
   -->

    <configuration>
        <enterprise-beans>
           <session>
              <ejb-name>SessionRef</ejb-name>
              <jndi-name>SessionRef</jndi-name>
              <jndi-source-name>CosNaming</jndi-source-name>
           </session>
           <entity>
              <ejb-name>Order</ejb-name>
                <jndi-name>Order</jndi-name>
              <jndi-source-name>CosNaming</jndi-source-name>
              <cmp-datasource>
                <res-ref-name>jdbc/EntityOrder</res-ref-name>
                <res-ref-link>JDBCEntityOrder</res-ref-link>
              </cmp-datasource>
              <cmp-config>
                <use-ipas-implementation>
                  <table-name>ORDERDEMO</table-name>
                  <cmp-field-mapping>
                    <field-name>id</field-name>
                    <column-name>order_number</column-name>
                  </cmp-field-mapping>
                  <cmp-field-mapping>
                    <field-name>code</field-name>
                    <column-name>product_code</column-name>
                  </cmp-field-mapping>
                  <cmp-field-mapping>
                    <field-name>quantity</field-name>
                    <column-name>order_quantity</column-name>
                  </cmp-field-mapping>
                </use-ipas-implementation>
              </cmp-config>
           </entity>
        </enterprise-beans>
        <resources>
        <resource>
          <resource-name>JDBCEntityOrder</resource-name>
          <jndi-name>iona/ipas/jdbc/OrderEntity</jndi-name>
          <property>
```

```
          <prop-name>java.naming.factory.initial</prop-name>
   <prop-value>com.sun.jndi.fscontext.RefFSContextFactory</
prop-value>
          </property>
          <property>
          <prop-name>java.naming.provider.url</prop-name>
   <prop-value>file:/vob/orbixhome/test/system_test/system/
bean2bean/jndi-bindings</prop-value>
          </property>
        </resource>
        </resources>
        <jndi-sources>
         <jndi-source>
           <jndi-source-name>CosNaming</jndi-source-name>
           <property>
             <prop-name>java.naming.factory.initial</prop-name>
             <prop-value>com.sun.jndi.cosnaming.CNCtxFactory</
prop-value>
           </property>
         </jndi-source>
       </jndi-sources>
     </configuration>
```

ここでは **Orders** エンティティBean に **id**、**code**、**quantity** というコンテナが管理するフィールドが 3 つ含まれていて、**cmp-config** エレメントがこれらのフィールドを **orders** データベース・テーブルの **order_number**、**product_code**、ならびに **order_quantity** という各カラムにマッピングしています。

第 11 章

インターオペラビリティと iPortal Application Server

OMG（Object Management Group）の CORBA（Common Object Request Broker Architecture）は、分散システムの開発を行うための標準アーキテクチャとして広く使用されています。エンタープライズ Bean では CORBA システムとのインターオペラビリティが重要な役割を果たします。CORBA はネットワーク上に分散したオブジェクトからアプリケーションを構築するための標準ソリューションです。

CORBA を使用すると、各コンポーネントの実装言語やそれが実行されるオペレーティング・システムやプラットフォームに関係無く、分散オブジェクトが相互に協力して処理を行えます。CORBA ソリューションは一般的な開発環境の全てに対応しているため、Visual Basic、C、C++、Java、COBOL の各言語で記述されたアプリケーションの統合に利用できます。

本章は、次のセクションで構成されます。

EJB-CORBA インターオペラビリティと CORBA-EJB インターオペラビリティ

概要

組込み式のシステム、Windows、UNIX、およびメインフレームで稼動する CORBA オブジェクトは、互いに直接通信することができます。これらのオブジェクトはさらに CORBA-COM ブリッジ技術を介して COM オブジェクトとインタラクトすることもできます。CORBA では分散オブジェクト間の相互通信を可能にするため、IIOP（Internet Inter-ORB Protocol）というプロトコルを使用します。

EJB-CORBA インターオペラビリティは、CORBA コンポーネントと EJB コンポーネント間の相互運用に必要です。このインターオペラビリティの基本となるのは Java RMI（Remote Method Invocation）と CORBA の組合せです。このためには次のような各仕様が必要です。

- IIOP（Internet Inter-ORB Protocol）
- Java から IDL へのマッピングと RMI-IIOP
- CORBA の値渡しの仕様
- ネーム・サービスのマッピング
- トランザクション・サービスのマッピング
- セキュリティ・サービスのマッピング

IIOP（Internet Inter-ORB Protocol）

分散システムで個々のコンポーネントが通信を行うには共通のプロトコルが必要です。IIOP は、こうした分散システム用の業界標準で、CORBA の重要な部分を成しています。iPortal Application Server では、Bean が CORBA サーバへ IIOP リクエストを送信することにより、CORBA クライアントとして動作することが可能です。

Java から IDL へのマッピングと RMI-IIOP

Java から IDL へのマッピングにより、標準化された方法で Java のデータ型を CORBA IDL にマッピングすることができます。また、RMI-IIOP は CORBA IIOP プロトコルを使用して EJB メソッド呼出しを実行します。これら 2 つの標準により、EJB が CORBA オブジェクトのように動作し、C++ や Visual Basic などの言語で記述された各 CORBA クライアントが EJB と通信できるようになります。

CORBA の値渡しの仕様

Java から IDL 言語へのマッピングでは、IDL valuetype の `construct` を広く使用します。これは CORBA の値渡しの仕様に含まれています。Java クラスを IDL で表現するには値渡しが必要です。

ネーミング・サービスのマッピング

このマッピングは、CORBA の COS（Common Object Services）ネーミング・サービスを使用して EJB ホーム・オブジェクトを見つける方法を定義します。COS ネーミング・サービスは CORBA アプリケーションのネーミング・サービスで、これにより、アプリケーションが CORBA オブジェクトへのリファレンスの保管とアクセスを行えるようになります。

iPortal Application Server は、JNDI（Java Naming and Directory Interface）を使用します。JNDI はネーミングおよびディレクトリ機能を提供するクライアント API で、CORBA ネーミング・サービスの上に実装されています。JNDI は Java で記述されていて、ネーミング・サービスやディレクトリ・サービス用クライアントでよく使用されるエレメントの抽象機構を提供します。

トランザクション・サービスのマッピング

これは、EJB のトランザクション機能から CORBA オブジェクト・トランザクション・サービス（OTS：Object Transaction Service）へのマッピングを定義します。iPortal Application Server EJB は、ランタイムに CORBA OTS の実装を使用してトランザクション制御を行います。

セキュリティ・サービスのマッピング

これは、EJB のセキュリティ機能から CORBA のセキュリティ機能へのマッピングを定義します。EJB のセキュリティ機能は主に EBJ へのアクセス制御を行うもので、クライアントのアイデンティティは EJB サーバが判断します。

エンタープライズ Bean からの CORBA オブジェクトの呼出し

概要

CORBA サーバには、新しく開発されたアプリケーション（例えば UNIX プラットフォームで C++ を使用してアプリケーション）と、OS/390 プラットフォーム上の IMS または CICS で稼動する旧式のシステムのラッパの両タイプがあります。エンタープライズ Bean という観点からは、新規アプリケーションとラッパは同様に扱

われます。EJB アプリケーション内で CORBA との通信を担当する Bean またはそのグループは全て、EJB アプリケーションへの Bean として CORBA を表すことになります。

IIOP を介した EJB から CORBA への通信では、Bean が CORBA サーバへの Java CORBA クライアントとして動作します。Bean でこの Java CORBA クライアントを実装するコードは、スタンドアロンの Java CORBA クライアントのコードと同じです。

例 この機能を使用した例については、iPortal Application Server インストール・ディレクトリの **`ipas3\demos\EjbCorba`** ディレクトリにある **`EjbCorba`** デモを参照してください。

CORBA クライアントからのエンタープライズ Bean の呼出し

概要 CORBA-EJB インターオペラビリティを実装する方法には次の 2 種類があります。

- Java から IDL へのマッピングを使用して直接アクセスする方法
- ラッパを使用して間接的にアクセスする方法

次のセクションではこの両方の方法について詳しく説明します。

本セクションの内容 本セクションは次のトピックで構成されています。

Java から IDL へのマッピングの使用

概要

iPortal Application Server は、標準の EJB-CORBA マッピングを完全にサポートしています。したがって EJB ホーム・オブジェクトおよび EJB リモート・オブジェクトは CORBA オブジェクトでもあります。EJB の CORBA ビューの IDL インターフェイスは、JDK に付属している RMI コンパイラ `rmic -idl -noValueMethods` を使用して Java インターフェイスから生成できます。EJB ホームのパブリッシュに使用される Java のネーミング・サービス（JNDI）が CORBA のネーミング・サービスにマッピングされるので、CORBA クライアントは EJB ホームを見つけ、追加の処理を行わずにそのまま CORBA オブジェクトとして使用することができます。

図 11 に、Java から IDL へのマッピング手順を示します。

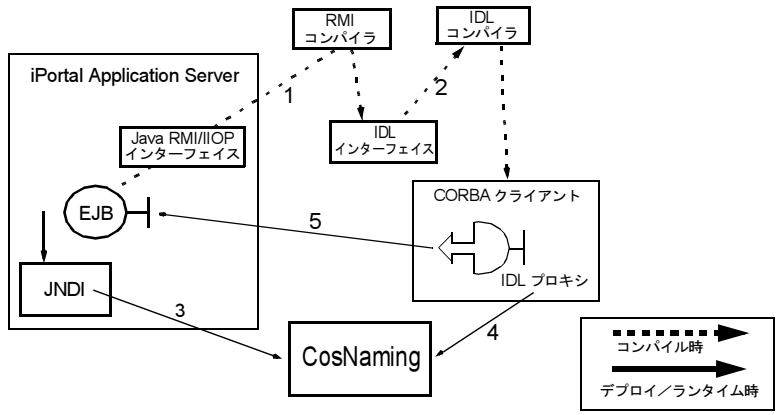


図 11:　Java から IDL へのマッピング

マッピングの手順

マッピングに必要な手順は次のとおりです。

1. RMI コンパイラ `rmic -idl` を使用して EJB のリモート Java インターフェイス用の IDL を生成する。
2. CORBA IDL コンパイラを使用してクライアントの実装言語によるスタブを生成し、これをクライアントにコンパイルする。
3. EJB サーバを構成し、JNDI プロバイダとして CORBA の `CosNaming` サービスを使用するよう設定する。

4. CORBA クライアントが CORBA ネーミング・サービスで EJB オブジェクトを検索する。

5. CORBA クライアントが EJB オブジェクトへの CORBA 呼出しを行う。

Java から IDL へのマッピングを使用した例については、iPortal Application Server インストール・ディレクトリの **ipas3\demos\CorbaEjbDirect** ディレクトリにある **CorbaEjbDirect** デモを参照してください。

Java から IDL へのマッピングに関連する問題

概要

自動生成された IDL にはいくつかの問題があり、その使用が困難となる場合もあります。IDL に問題が生じるのは、マッピングには互いに矛盾する次のような 2 つの目的があるためです。

- CORBA IIOP を使用して Java RMI を実行することで Java RMI/IIOP をサポートする。そのため IIOP を介した Java 間の通信を可能にするには Java から IDL への変換時に全ての情報を維持する必要がある。
- Java のリモート・オブジェクトを CORBA オブジェクトとして扱い、これを CORBA クライアントが使用できるようにすることで、CORBA クライアントをサポートする。

Java から IDL へのマッピングのサンプル・コード

次のサンプル・コードに、Java から IDL へのマッピングで生じる問題の例を示します。このコードは Sun Microsystems の Java 2 Enterprise Edition Developer's Guide バージョン 1.2.1 に記載されているごくシンプルな例からのものです。

```
/*
 *
 * Copyright 2000 Sun Microsystems, Inc. All Rights Reserved.
 *
 * This software is the proprietary information of Sun
   Microsystems, Inc.
 * Use is subject to license terms.
 *
 */
import java.util.*;
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
public interface Cart extends EJBObject
{
   public void addBook(String title) throws RemoteException;
```

```
    public void removeBook(String title) throws BookException,
    RemoteException;
    public Vector getContents() throws RemoteException;
}
```

このコードは、本の題名に基づいてショッピング・カートへの本の追加と削除を行い、題名を表す文字列のベクトルとしてカートの中身を返します。

手動マッピングによるサンプル IDL

次のコードは、上記の EJB の機能にアクセスする IDL インターフェイスを手動で記述した場合の例です。

```
//
// Custom IDL interface for Cart EJB
//
typedef wstring TitleType;
typedef sequence<TitleType> TitleSeq;
exception BookException { string message; };
interface Cart
{
    void addBook(in TitleType title);
    void removeBook(in TitleType title) raises BookException;
    readonly attribute TitleSeq contents;
};
```

この IDL インターフェイスはシンプルで、Java クライアントと EJB の両方で簡単に使用できるだけでなく、Java 以外の言語で記述されたクライアントでも使用できます。次のセクションにこの例を示します。

Java 以外のクライアントのサンプル・コード

これは Java 以外のクライアントのサンプル・コードです。

```
//
// C++ CORBA client
//
Cart_var my_cart = // find cart object somehow
my_cart->addBook("Moby Dick");
TitleSeq_var contents = my_cart->contents();
for (int i = 0; i < contents->length(); ++i)
{
    cout << "Cart contains: " << contents[i] << endl;
}
```

Java クライアントのサンプル・コード

次は Java クライアントのサンプル・コードです。

```
//
// Java CORBA client
//
Cart myCart = // find cart object somehow
myCart.addBook("Moby Dick");
String[] contents = myCart.contents();
for (int i = 0; i < contents.length; ++i)
{
    System.out.println("Cart contains: "+contents[i]);
}
```

自動マッピングによるサンプル IDL

上記のシンプルなインターフェイスを標準の方法でマッピングすると、次のようなコードが生成されます。

```
// Standard Java-to-IDL mapping of Cart
//
interface Cart: ::javax::ejb::EJBObject
{
   void addBook(in ::CORBA::WStringValue arg0);

   void removeBook(in ::CORBA::WStringValue arg0) raises
        (BookEx);
   readonly attribute ::java::util::Vector contents;
};
```

ここで問題となるのは **valuetype** の頻繁な使用です。このコードでは単純な数値以外の全てが **valuetype** にマッピングされています。

Valuetype の使用

CORBA 2.3 では、値渡しを可能にする **valuetype** という概念が導入されました。**valuetype** は C++ や Java のクラスと同様に、パブリックとプライベート両方のデータ・メンバ、オペレーション、初期化関数、および継承を持ちます。**valuetype** は IDL インターフェイスと異なり、リモート・オブジェクトを表すものではありません。**valuetype** のインスタンスは**値渡し**となるので、データ・メンバは転送された先でコピーされます。

オペレーションを **valuetype** にした場合、受取り側が実装を提供する必要があります。例えば C++ では、受取り側にこのオペレーションを実装するためのクラスと、実装のインスタンスを作成するファクトリ・クラスが必要です。

データ・メンバのみが **valuetype** の場合には、IDL コンパイラが実装を生成します。value-box は、パブリックなデータ・メンバを 1 つ持つ特殊なケースです。value-box は null 値を持つことができるので、Java から IDL へのマッピングでは value-box がよく使用されます。**valuetype** 以外の IDL データ型では null 値が許可されません。

コレクションの問題

自動マッピングで生成されたショッピング・カートのインターフェイスにはもう一つ問題があります。

```
// Standard Java-to-IDL mapping of Cart
//
interface Cart: ::javax::ejb::EJBObject
{
   void addBook(in ::CORBA::WStringValue arg0);

   void removeBook(in ::CORBA::WStringValue arg0) raises
        (BookEx);
   readonly attribute ::java::util::Vector contents;
};
```

ここでの問題は、**::java::util::Vector** というコードが Java 以外のクライアントにとって無意味である点です。Java から IDL へのマッピングでは、「データのみ」と「データとメソッド」の 2 つの方法から選択できます。

データのみのマッピング

データのみのマッピングを行うと、データ・クラスに含まれるデータ自体が IDL の valutype にマッピングされます。渡される全データを各メソッドのパラメータまたは戻り値として表現できるので、クライアントが EJB と通信するにはこの方法で十分です。

データのみのマッピングの場合、次のようなサンプル・コードになります。

```
//IDL
valuetype Vector: ::java::util::AbstractList, ::java::util::List
   supports ::java::lang::Cloneable {
        private long capacityIncrement;
        private long elementCount;
        private ::org::omg::boxedRMI::java::lang::seq1_Object
   elementData;
    };
```

このサンプル・コードでは、言語に依存性を持たない方式で文字列を表現するため、次のような問題が生じます。

- **型保証の欠如**
  typedef のリストの後にある **elementData** は IDL **any** 型のシーケンスなので、各エレメントが IDL の全ての型を取る可能性があります。

- **パフォーマンスの低下**
  ORB にとっては各エレメントがそれぞれ異なる型である場合があるので、型の記述情報が各エレメントについて別々に送られ、その結果パフォーマンスが低下します。

- **プライベート・メンバ**
  capacityIncrement と **elementCount** は実装詳細を処理する Java のベクトル・クラスのプライベート・メンバです。したがって Java 以外のクライアントに対して Java 固有の情報を大量に提供します。

- **継承**
  AbstractList、**List**、**Cloneable** はそれぞれ空のインターフェイスで、これらは Java の継承階層を維持するためだけに存在します。これは Java 以外のクライアントにとっては不要です。

さらにクライアントにとっては、これらのデータ・メンバがプライベートであるためにパブリック・アクセスができない点も問題となります。アクセスするためにはパブリック・メソッドが必要です。この問題を解決するには次の方法があります。

1. データのみのマッピングを手動で変更して各メンバをパブリックにする。

2. 言語に固有の手法を使用する。

   例えば C++ では **valuetype** インターフェイスの一部ではないパブリックな C++ メソッドを持つ **valuetype** の実装を記述し、動的キャスティングを使用してこれらのメソッドにアクセスします。

3. 後述のデータとメソッドのマッピング方法を使用する。

データとメソッドのマッピング

Java から IDL へのマッピングでは、データのみのマッピング、およびデータとメソッドのマッピングのどちらかを選択できます。データとメソッドのマッピングを使用すると、データ・クラスのメソッドも IDL の valuetype オペレーションにマッピングされます。この場合、各メソッドが持つパラメータもそれぞれマッピングする必要がありますが、これにより新しいデータ型が IDL に定義されます。するとそのデータ型のメソッド、そしてそのメソッドのパラメータもマッピングが必要となり、最終的に Java 言語の大部分が IDL にマッピングされてしまいます。マッピングされた IDL はその大半が無意味なものであり、CORBA クライアントの実装が非常に困難なコードが生成されます。

データとメソッドのマッピングでは同じ valuetype が定義されますが、これにはデータだけでなくマッピングの元になった Java クラスの全メソッドも含まれます。その結果、クライアントがこれらのメソッドを実装する必要があるという問題と、上に述べた再帰的なマッピングの問題が生じます。Java から IDL へのマッピングのドキュメンテーションでは、この後者の問題を**スパゲティ効果**と呼んでいます。

スパゲティ効果

サンプル・コードにはスパゲティ効果の例が見られます。このごくシンプルな例から、それぞれ 1 つの valuetype 宣言が含まれるだけの IDL ファイルが合計 27 個も生成されます。そしてクライアントは、数個の文字列を戻り値として取得することが目的でありながら、`Vector` 型により宣言されている 22 個のメソッドと、`AbstractList`、`List`、および `Cloneable` から継承される 11 個のメソッド全てを実装する必要があります。

これらのメソッドのほとんどは空ですが、実装は必要です。クライアントはスパゲティ効果により定義された `java.lang`、`java.io`、`javax.ejb` からの 10 個の非抽象 valuetype を実装する必要はありませんが、IDL コンパイラから全ての関連スタブ・コードをリンクする必要はあります。

その他の問題

このサンプル・コードでは見られないその他の問題としては次が挙げられます。

- **ファイル管理**
  マッピングにより入れ子のディレクトリ構造を持つファイルが多数作成されます。これは Java のビルド環境では一般的ですが、他の言語には適しません。
- **モジュールの分割**
  Java パッケージが複数のファイルで開かれる、ネストされた IDL モジュールにマッピングされます。これは例えばネームスペースがサポートされない C++ コンパイラなど、一部の言語で問題となります。
- **不正確なメソッド名**
  EJB メソッド名が IDL で正しい名前にマッピングされないことがあります。
- **例外継承の不一致**
  Java では例外の継承が許可されますが、IDL では許可されません。
- **標準クラスの紛失**
  スパゲティ効果の結果、`rmic` が `java.io` などの標準パッケージでよく使用されるクラスの IDL を生成することがあります。この IDL をコンパイルして Java コードに戻すと、元の標準クラスが作成されたクラスの中に紛れ込んでしまい、使用できなくなります。

CORBA に適した EJB

概要

標準のマッピングにより生じる各種の問題を回避するには、CORBA での使用に適した EJB を記述します。

IDL を使用した EJB データ型の定義

CORBA クライアントが使用しやすい EJB を作成するには、CORBA IDL を使用して EJB が使用するデータ型を定義することができます。これを行うには、IDL から Java へのマッピングを行います（IDL と Java 間では双方向のマッピングが可能です）。標準の CORBA-EJB インターオペラビリティを利用して Java から IDL へのマッピングを行う際の問題についてはすでに述べました。これに対して IDL から Java へのマッピングは Java ベースの CORBA システムを開発するための基盤として問題なく使用できます。IDL は Java のようなプログラミング言語へマッピングするために開発された言語なので、IDL から Java へのマッピングを行うのは簡単です。

したがって、IDL から Java へのマッピングを行う際は、IDL の全ての型に対応する型が Java にもあります。Java から IDL へのマッピングでは、派生型について特殊なケースを使用します。IDL で派生された Java クラスである EJB パラメータは、派生元の IDL 型を含んだ value-box にマッピングされます。つまり、IDL の型を元にして作成した Java コードは、マッピングを行った後で元の IDL コードに大変近いものに戻ります。次に例を示します。

```
// IDL: Data type definition
struct Customer {
  string name;
  string address;
}

// EJB: Interface using IDL-derived Customer class
interface Mailer extends EJBObject {
   void sendMail(Customer customer);
}

// IDL: Java to IDL mapping of EJB above
interface Mailer: ::javax::ejb::EJBObject {
   void sendMail(
     in ::org::omg::boxedIDL::Customer customer
   );
}
```

ここでは **`::org::omg::boxedIDL::Customer`** という型は、value-box になっています。これは元の IDL 型である **`Customer`** が含まれているラッパです。したがって ORB が valuetype をサポートしていれば、クライアント・コードでは元の IDL

型とほぼ同様にこれを使用できます。value-box は Java から IDL へのマッピングで null 値を処理するために導入されました。全ての Java クラスは IDL の value-box 型と同様に null 値を許可します。`struct` などこれ以外の IDL 型では null 値は許可されません。

CORBA に適した EJB を作成するには次のような方法もあります。

- IDL を使用して例外を定義する。

  IDL 定義による例外を EJB の throws 節で使用した場合、EJB が IDL にマッピングされるとその節は元の定義に戻ります。マッピングの結果 valuetype が例外に含まれてしまう問題を回避するには、これが唯一の方法です。

- コレクション・クラスではなく配列を使用する。

  Java の配列は、IDL でそれに対応する型のシーケンスを含んだ value-box にマッピングされます。コレクション・クラスを使用すると、Java の `valuetype` のプライベート・データ内に型定義が不確かな `sequence<any>` として保管してしまいます。

- パブリック・データを使用する。

  IDL にない Java クラスをデータ型に使用する場合、重要なメンバはパブリックに指定します。こうすればデータのみのマッピングが使用しやすくなります。

- `java.lang.Object` は、IDL の `any` 型にマッピングされるので使用可能。これは CORBA クライアントがランタイムに解釈できる自己記述式のデータです。

  ただしランタイムには、クライアントが解釈するオブジェクトのインスタンスを渡す必要があります。

- データのみのマッピング方法を使用してスパゲティ効果を回避する。

  データとメソッドのマッピングを行う必要があり、システムに含まれる全言語でオペレーションを実装する場合、余分な Java 機構が入り込まないよう EJB インターフェイスに渡すクラスのセットを定義する必要があります。ただし EJB のベースクラス自体が `java.io` や `java.lang` などのパッケージを多用しているので、これを行うのは非常に困難です。

上記の方法を使用することでマッピングから生じる問題に対処できますが、`valuetype` の問題だけは回避できません。Java の `other` クラスは、IDL のデータ型から派生されたものも含めて全て IDL の value-box にマッピングされます。

EJB に適した IDL からの EJB の生成

概要

CORBA に適した EJB インターフェイスを作成する場合、IDL でインターフェイスを定義し、それに基づいて EJB インターフェイスを作成するのが一番効率的です。この方法で作成した EJB インターフェイスは、IDL から Java へのマッピングで作成されるオペレーション・インターフェイスと非常に似ていますが、次の相違点があります。

- このインターフェイスは `javax.ejb.EJBObject` を拡張する。
- 各オペレーション・シグニチャに `throws java.rmi.remoteException` が追加される。

Orbix Code Generation Toolkit などを使用してコード生成スクリプトを作成すれば、IDL ファイルからの EJB リモート・インターフェイスの作成を自動化できます。そうすれば最初に IDL インターフェイスを定義して、EJB インターフェイスを自動生成できます。

逆マッピングで生じる問題

Java から IDL へのマッピングと ILD から Java へのマッピングは完全に一致しません。この方法で作成したインターフェイスの逆マッピングの結果は元の IDL インターフェイスとは異なり、次のような問題が生じます。

- 複雑なデータ型や文字列は全て value-box でラッピングされる。
- IDL 型のボックスには、`omg::org::boxedIDL::MyDataType` など何重にも入れ子されたパッケージ名が付けられる。
- 文字列の型は全て `wstring` を含んだ value-box になる。
- `EJBObject` の継承が IDL では公開される。
- Java では `in` を使用したパラメータ渡しのみがサポートされるので、IDL の `out` パラメータや `inout` パラメータが問題となる。

ラッパのビルドと生成

概要

ラッパは、Java で実装されている EJB に委譲する CORBA オブジェクトです。ラッパは標準の CORBA オブジェクトなので、Java から IDL へのマッピングによる、トラブルの無いクリーンな IDL インターフェイスを持っているのが利点です。

ラッパの使用

EJB に委譲するラッパのコードを記述する方法はさまざまですが、ラッパを使用して IDL インターフェイスから EJB インターフェイスを生成する際に重要な特殊ケースがあります。ラッパは元の IDL インターフェイスを実装し、派生された EJB に委譲します。CORBA クライアントは元のクリーンなインターフェイスを通信し、逆マッピングから生じる複雑な問題を回避します。

ラッパは EJB インターフェイスをほぼそのまま委譲するため、ラッパはパラメータや戻り値に操作を加える必要が無く、全てのパラメータを EJB に直接渡し、結果をクライアントに直接戻します。

ラッパが仲介処理を行うのは、次のいずれかの場合だけです。

- EJB が null 値を返した場合。これはエラーとみなされます。CORBA に適した EJB が null 値を返すことはありません。ラッパは **BAD_PARAM** などのシステム例外を送出することがあります。
- EJB が **RemoteException** 例外を送出した場合。ラッパがこの例外を受け取って、適切な CORBA システム例外に変換する必要があります。JDK で定義されている **RemoteExceptions** は、通常これに対応する CORBA 例外があり、**UNKNOWN** 例外は **RemoteException** への未知の拡張のフォールバックを提供します。

ラッパはシンプルでその動作も簡単なので、IDL からラッパを自動生成するコード生成スクリプトを使用することもできます。次にこの例を示します。

```
//
ReturnType some_op(ParameterType p1, ...) {
  try {
    ReturnType result = m_ejbTarget.some_op(p1, ...);
    if (result == null) throw new BAD_PARAM();
    return result;
  } catch (RemoteException ex) {
     throw_corba_system_exception_for(ex);
  }
}
```

EJB-CORBA インターオペラビリティに必要な手順

EJB-CORBA インターオペラビリティを確実にするには、次の手順を行います。

まず IDL でインターフェイスを定義します。この定義では **in** パラメータによる受渡しモードのみを使用し、**valuetype** の使用を避けます。

1. IDL から EJB インターフェイスを生成する。このインターフェイス用の EJB コードを記述する。

2. IDL からラッパを生成する。

3. 手順 2 の元の IDL 用クライアントのコードを記述する。

4. EJB とラッパをデプロイする。IIOP を使用してラッパと通信するクライアントをデプロイする。ラッパが EJB と通信するには RMI または RMI/IIOP を使用する。

ラッパを使用した例は、iPortal Application Server インストール・ディレクトリの **ipas3\demos\CorbaEjb** ディレクトリにある **CorbaEjb** デモを参照してください。

既存の EJB とインターオペラビリティ

概要

既存の EJB インターフェイスは IDL から生成されたものでない可能性が高いので、通常これらのインターフェイスでは先に述べたような使用しづらい CORBA Java 構造型のデータ型が一部使用されています。

この問題を解決するには、2 つの方法があります。

- IDL インターフェイスを実装し、使用しづらい EJB インターフェイスを隠す CORBA ラッパを記述する。
- CORBA に適した手順を使用して EJB をビルドし、使用しづらい EJB インターフェイスを隠す。

どちらの方法を使用するかは、開発者が EJB と CORBA どちらのプログラマであるかによって決まります。ただし、こうした問題を避けるには、まず最初に CORBA に適した EJB を作成し、この EJB によりクリーンな IDL インターフェイスの背後で CORBA に適さない EJB のサービスをまとめるのが理想的です。

CORBA に適した EJB の必要性

アプリケーション開発の際は、全ての EJB を CORBA に適したものにする必要があるかどうかを判断する必要があります。システムで既存の EJB インターフェイスを使用している場合には、これを再実装する必要があります。また、新しいシステムでは他の EJB に対してのみサービスを提供する EJB もあり、これらの EJB は CORBA クライアントが使用できません。したがって、全ての EJB を CORBA で使用しやすいものにするかは、次の点に留意して判断してください。

- システムを使用するサイトにおける CORBA および EJB 技術の浸透度
- 既存のソースコード
- 開発者の知識と経験レベル

まとめ

概要

EJB-CORBA インターオペラビリティの実装方法には次の 2 種類があります。

- Java から IDL へのマッピングを使用して直接アクセスする方法
- ラッパを使用して間接的にアクセスする方法

Java から IDL へのマッピング

この方法では中間処理を行う必要が無いので開発者の手間を省くことができます。ただし、マッピングされた IDL は使用しづらく、一部の ORB でサポートされない `wstring` や `valuetype` などの機能が含まれています。

ラッパ

ラッパを使用すると、クリーンで使用しやすい IDL を持つクライアントを作成でき、問題のある構成要素の使用を回避できます。ラッパを作成する手間はかかりますが、CORBA に適した手法を使用して自動生成することもできます。ラッパを使用する場合、デプロイメントや管理に際して追加のコンポーネントが必要となります。

パート 4

エンタープライズ・アプリケーションの設定

パート 4 は次の章で構成されます。

第 12 章

エンタープライズ・アプリケーションの構成と管理

iPortal Application Server では、エンタープライズ・アプリケーションの各コンポーネントに対してライフ・サイクル管理、セキュリティ、デプロイメント、ランタイム・サービスなどの各種サービスを提供する、複数の EJB コンテナをホストすることができます。本章では各コンテナをエンタープライズ・アプリケーション・コンポーネントのニーズに合わせて構成する方法について説明します。

本章は、次のセクションで構成されます。

コンテナ・コンフィギュレーション

概要

EJB アプリケーションは、通常のアプリケーションと異なり特定のオペレーティング環境に固有のコードを含んでいません。したがってアプリケーションのホストとなる EJB コンテナに対してオペレーティング環境を指定する必要があります。iPortal Application Server ではコンテナ・コンフィギュレーションを作成してこの指定を行います。

コンテナ・コンフィギュレーションは、コンテナでデプロイされている各コンポーネントの必須条件を、その環境での適切な機構にマッピングします。コンテナ・コンフィギュレーション・ファイルには、IONA のコンテナ・コンフィギュレーション DTD（Dcument Type Declaration）に準拠する XML ドキュメントが 1 つ含まれています。

コンテナ・コンフィギュレーションは XML ドキュメントです。コンフィギュレーションを作成するには、XML エレメントを使用してこのドキュメントを編集し、特定のオペレーティング環境用に Bean をカスタマイズします。本セクションでは、このドキュメントの上位構造を構成する基本的な XML エレメントについて説明します。

はじめに

各コンテナ・コンフィギュレーションには、コンフィギュレーション・エレメントが 1 つ含まれています。したがって、コンテナ・コンフィギュレーション・ファイルには次のタイプのエレメントを 1 つ定義します。

```
<configuration>
</configuration>
```

このコンフィギュレーション・エレメントには、コンフィギュレーションの構成要素となる他の全てのエレメントが含まれます。新しいコンテナ・コンフィギュレーションを作成する場合、まず始めに次のようなコンテナに関する一般的な情報を指定します。

```
<configuration>
 <description>Test Container</description>
 <display-name>My Test Container</display-name>
 <category>Test</category>
</configuration>
```

description、**display-name**、**category** の各エレメントを使用して、コンテナの識別とそのカテゴリに関する情報を人間が理解できる形で指定します。iPortal Application Server はこの情報の解釈を行いません。

この XML ドキュメントには、<!-- と --> を使用して次のようなコメントを挿入することもできます。

```
<!-- Test Container Configuration, v 1.3
     Author: John P. O'Sullivan
-->
<configuration>
<description>Test Container</description>
<display-name>My Test Container</display-name>
<category>Test</category>
</configuration>
```

セクションの内容

本セクションは次のトピックで構成されています。

Bean のコンフィギュレーション情報

概要

configuration エレメント内にある **enterprise-beans** エレメントを使用して、コンテナでデプロイされている各 Bean の詳細なコンフィギュレーションを指定します。次に例を示します。

```
<configuration>
<description>Test Container</description>
<display-name>My Test Container</display-name>
<category>Test</category>
<enterprise-beans>
...
</enterprise-beans>
</configuration>
```

コンテナでアプリケーションをデプロイする前に、コンポーネント Bean がターゲット環境で動作するために必要な全ての情報をそのコンテナの **enterprise-beans** エレメントで指定する必要があります。コンテナ・コンフィギュレーションで指定されていない情報がある場合、アプリケーションがデプロイ時にエラーを起こします。

enterprise-beans エレメント

enterprise-beans エレメントには、コンテナでデプロイする各セッション Bean につき 1 つの **session** エレメントと、同じくデプロイする各エンティティBean につき 1 つの **entity** エレメントを含めます。

```
<configuration>
<description>Test Container</description>
<display-name>My Test Container</display-name>
<category>Test</category>
<enterprise-beans>
<session>
<ejb-name>ShoppingCart</ejb-name>
...
</session>
<entity>
<ejb-name>Catalog</ejb-name>
...
</entity>
<entity>
<ejb-name>Orders</ejb-name>
...
</entity>
</enterprise-beans>
</configuration>
```

各 **session** エレメントと **entity** エレメントには、**ejb-name** エレメントを 1 つ含めます。このエレメントには、これに対応する Bean のデプロイメント・デスクリプタの **ejb-name** エレメントと同じ名前を指定します。

クライアントが Bean と通信する場合、まず Bean の EJB ホームへのリファレンスを取得してから、これを使用して Bean インスタンスを作成します。クライアントが EJB ホームへのリファレンスを取得するには JNDI 名を検索するので、Bean のホストとなるコンテナは、この Bean の EJB ホームを JNDI に登録する必要があります。

デフォルトでは、コンテナは EJB ホームの JNDI 名として Bean の **ejb-name** エレメントのコンテンツを使用しますが、次のように **jndi-name** エレメントを使用してこれ以外の JNDI 名を指定することもできます。

```
<configuration>
<description>Test Container</description>
<display-name>My Test Container</display-name>
<category>Test</category>
<enterprise-beans>
<session>
<ejb-name>ShoppingCart</ejb-name>
```

```
<jndi-name>MyCo/RetailApp/EJB/cart</jndi-name>
</session>
</enterprise-beans>
</configuration>
```

このコンフィギュレーションを使用すると、コンテナは **ShoppingCart** Bean の EJB ホームを **MyCo/RetailApp/EJB/cart** という JNDI 名で登録します。

session エレメントと **entity** エレメントには、これ以外にも次のエレメントを含められます。

- Bean の環境エントリを定義するエレメント
- Bean が使用するセキュリティ・ロールへのリファレンスを解決するエレメント
- Bean が使用するリソースへのリファレンスを解決するエレメント
- 他の Bean へのリファレンスを解決するエレメント

さらに **entity** エレメントでは、コンテナ管理永続性が必要なエンティティBean用にコンテナが使用するデータソースを指定し、コンテナ管理のフィールドをデータベース・テーブルの対応カラムにマッピングすることもできます。

enterprise-beans エレメントのコンテンツは新しいアプリケーションのインストール、アプリケーションの削除、オペレーティング環境の変更などに伴ない動的に変更されるので、コンテナ・コンフィギュレーション・ファイルは頻繁に更新されます。

Web コンポーネント用のコンフィギュレーション情報

概要

configuration エレメント内にある **web-app** エレメントを使用して、コンテナでデプロイされている各 Web コンポーネントの詳細なコンフィギュレーションを指定します。

```
<configuration>
<description>Test Container</description>
<display-name>My Test Container</display-name>
<category>Test</category>
<web-app>
...
</web-app>
</configuration>
```

コンテナでアプリケーションをデプロイする前に、Web コンポーネントがターゲット環境で動作するために必要な全ての情報をそのコンテナの **web-app** エレメントで指定する必要があります。コンテナ・コンフィギュレーションで指定されていない情報がある場合、アプリケーションがデプロイ時にエラーを起こします。

web-app エレメント

web-app エレメント内では、次のように **context-root** エレメントを使用してアプリケーションで使用する Web コンポーネントがあるディレクトリを指定します。

```
<configuration>
<description>Test Container</description>
<display-name>My Test Container</display-name>
<category>Test</category>
<web-app>
<context-root>TestDir</context-root>
</web-app>
</configuration>
```

Web コンポーネントのデプロイメント・デスクリプタでは **env-entry** エレメントを使用して Web コンポーネントに必要な環境エントリを宣言します。また **security-role-ref** エレメントにより Web コンポーネント用のセキュリティ・ロール・リファレンスを解決し、**resource-ref** エレメントにより Web コンポーネント内のリソース・リファレンスをオペレーティング・システム上にある実際のリソースにマッピングします。リソース・リファレンスの名前は Web コンポーネントのデプロイメント・デスクリプタで宣言されているものと一致する必要があります。

アプリケーションがリクエストに応じてもしくは開発過程の一部として jsp をビルドする場合や、アプリケーションを再デプロイせずに jsp を追加または変更する場合、コンテナ・コンフィギュレーション・ファイルに次のコードを挿入する必要があります。

```
<configuration>
<web-app>
     <jsp-earsco-dir>d:\applications\myapp</jsp-earsco-dir>
     <jsp-compile-mode>ONCE</jsp-compile-mode>
  </web-app>
</configuration>
```

jsp-earsco-dir エレメントには、EARSCO フレームワークの完全パスを指定します。**jsp-compile-mode** エレメントは次のどちらかのコンパイル・モードを指定します。

- **ONCE**：新しい jsp のみをコンパイルし、変更された jsp の再コンパイルは行いません。

- **ON-MODIFY**：新しい jsp と変更された jsp の両方をコンパイルします。

アプリケーション・アセンブリ情報の指定

概要

アプリケーションのデプロイメント・デスクリプタには、アプリケーションを構成する個々の Bean と Web コンポーネントのデプロイメント・デスクリプタが含まれているほか、アプリケーション全体における各コンポーネントの相互動作についての詳しい情報も指定されています。

これには次のような情報が含まれます。

- アプリケーションのセキュリティ・コントロール
- アプリケーションのセキュリティ・ロール
- セキュリティ・ロールに割り当てられたメソッド使用許可
- コンテナ管理によるトランザクション区分を使用する Bean のトランザクション属性

これらの情報は、通常アプリケーション・アセンブラがアプリケーションを組み立てる際に指定します。ただし、アセンブラがこの詳細情報を指定しないこともあり、その場合はデプロイメント時にコンテナ・コンフィギュレーションを使用して指定します。

assembly-descriptor エレメント

コンテナ・コンフィギュレーション内にある **assembly-descriptor** エレメントを使用して、次のようにアプリケーション・アセンブリ情報を指定します。

```
<configuration>
<description>Test Container</description>
<display-name>My Test Container</display-name>
<category>Test</category>
<enterprise-beans>
...
</enterprise-beans>
<web-app>
...
</web-app>
<assembly-descriptor>
...
</assembly-descriptor>
</configuration>
```

assembly-descriptor エレメントは、欠けているアプリケーション・アセンブリ情報を指定したり、アプリケーションのデプロイメント・デスクリプタでアセンブラが指定した設定を上書きするために使用できます。

アプリケーション・アセンブラがセキュリティ・ロールの作成など一部のタスクを完了しなかった場合、これらの情報を補足するにはアプリケーションの機能についての詳しい知識が必要です。例えば、アプリケーションで特定なビジネス・メソッドを呼び出すことのできるユーザを指定するには、各メソッドが行う処理について理解する必要があります。

コンテナ・コンフィギュレーションの検証

概要

コンテナ・コンフィギュレーションは IONA のコンテナ・コンフィギュレーション DTD に準拠している必要があります。iPortal Application Server にはコンテナ・コンフィギュレーションの検証を行うためのツールは用意されていませんが、標準の XML 検証ツールを使用してコンフィギュレーション・ファイルの形式を検査し、コンテナ・コンフィギュレーション DTD に準拠していることを確認できます。

DTD（ドキュメント・タイプ宣言）

コンテナ・コンフィギュレーション・ドキュメントを検証するには、検証するファイルに次の DTD を含める必要があります。

```
<!DOCTYPE configuration SYSTEM "http://www.iona.com/dtds/
container_configuration_3_0.dtd">
```

このようにドキュメント・タイプを宣言することで、検証ツールが IONA の Web サイトからコンテナ・コンフィギュレーション DTD を読み取り、ドキュメントの構造がこの DTD に準拠しているかを検査できます。DTD を宣言する場合、ドキュメントのコメントと標準の XML 処理指示（これは任意です）の直後に宣言を挿入します。

コンポーネントの環境エントリの構成

概要

Bean や Web コンポーネントの環境を設定することにより、これらのコンポーネントに対してランタイムにカスタムのコンフィギュレーション値を指定することができます。環境には一連のエントリが含まれ、各エントリにつき値が 1 つ指定されています。Bean や Web コンポーネントのソースコードは環境からこれらの値を読み込み、ビジネス・ロジックやプレゼンテーション・ロジックで使用します。

環境エントリが必要なコンポーネントの場合、コンポーネント・プロバイダがコンポーネントのデプロイメント・デスクリプタで必要なエントリを宣言します。アプリケーションを構成する全コンポーネントについて、デプロイ時に環境エントリの値を設定する必要があります。

Bean に必要な環境の確認

例えば、Bean のデプロイメント・デスクリプタでは、Bean プロバイダが env-entry エレメントを使用して必要な環境エントリを宣言します。セッション Bean の環境エントリは次のようになります。

```
<!-- Deployment Descriptor for Clock Bean -->

<ejb-jar>
<enterprise-beans>
<session>
<ejb-name>Clock</ejb-name>
<env-entry>
<env-entry-name>DisplayType</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
</env-entry>
<env-entry>
<env-entry-name>TimeZone</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
</env-entry>
 ...
 </session>
 </enterprise-beans>
 </ejb-jar>
```

Bean プロバイダやアプリケーション・アセンブラが次のように env-entry-value エレメントを使用して Bean の環境エントリに値を設定することもあります。

```
<!-- Deployment Descriptor for Clock Bean -->

<ejb-jar>
 <enterprise-beans>
 <session>
 <ejb-name>Clock</ejb-name>
 <env-entry>
 <env-entry-name>DisplayType</env-entry-name>
 <env-entry-type>java.lang.String</env-entry-type>
 <env-entry-value>digital</env-entry-value>
 </env-entry>
 <env-entry>
 <env-entry-name>TimeZone</env-entry-name>
 <env-entry-type>java.lang.String</env-entry-type>
 <env-entry-value>GMT</env-entry-value>
 </env-entry>
 ...
 </session>
 </enterprise-beans>
 </ejb-jar>
```

ここでは **DisplayType** という環境エントリの型が **java.lang.String**、値が **digital** に設定されていて、**TimeZone** 環境エントリの型は **java.lang.String**、値は GMT となっています。これらの設定値はアプリケーションのデプロイ時にコンテナ・コンフィギュレーション・ファイルを使用して上書きできます。

Bean の環境エントリの設定

コンテナ・コンフィギュレーションでは **env-entry** エレメントを使用して環境エントリの値を設定します。例えば **Clock** セッション Bean の環境エントリは次のように設定します。

```
<!-- Container Configuration -->

<!DOCTYPE configuration SYSTEM "http://www.iona.com/dtds/
container_configuration_3_0.dtd">

<configuration>
<enterprise-beans>
<session>
<ejb-name>Clock</ejb-name>
<env-entry>
<env-entry-name>DisplayType</env-entry-name>
<env-entry-value>analog</env-entry-value>
```

```
</env-entry>
<env-entry>
<env-entry-name>TimeZone</env-entry-name>
<env-entry-value>EST</env-entry-value>
</env-entry>
...
</session>
</enterprise-beans>
...
</configuration>
```

コンテナ・コンフィギュレーションでは、**env-entry-name** と **env-entry-value** のペアを使用して各環境エントリの値を割り当てます。Bean プロバイダやアプリケーション・アセンブラが設定した値をそのまま使用する場合は、コンテナ・コンフィギュレーションでこれに対応する環境エントリに対して **env-entry** エレメントを含めないようにします。

Bean は JNDI インターフェイスを介して環境にアクセスします。iPortal Application Server コンテナは、コンテナ・コンフィギュレーションがインストールされた時点で環境を JNDI に自動登録します。サーバは環境エントリを **java:comp/env** という JNDI コンテキストに登録します。

コンテナ・コンフィギュレーションやデプロイメント・デスクリプタの **env-entry-name** エレメントで指定する環境エントリ名は、**java:comp/env**JNDI コンテキストに対して相対的なものです。

パート 5

J2EE アプリケーションの管理

パート 5 は次の章で構成されます。

第 13 章

J2EE アプリケーション管理の概要

本章では Java アプリケーションの管理を行うためのツールについて簡単に説明します。ここでは IONA の iPortal Administrator 管理ツール、および Sun の JMX（Java Management Extensions）API について説明します。

本章は、次のセクションで構成されます。

iPortal Administrator の概要

概要

IONA の iPortal Administrator には、管理者が分散アプリケーションの構成、監視、制御をランタイムに行うための一連のツールが用意されています。iPortal Administrator では IONA スイートに含まれる全製品、およびこれらの製品を使用して開発した全アプリケーションのシームレスな管理を行えます。

Sun の JMX は、Java アプリケーションに管理機能を追加する標準機構を提供します。本章では JMX の主要なコンポーネントと概念について述べ、JMX と iPortal Administrator を組み合わせてアプリケーションに管理機能を追加する仕組みを説明します。

コンポーネント

iPortal Administrator には、次の主要なコンポーネントが含まれています。

- iPortal Administrator Console
- iPortal Administrator Web Console
- iPortal Administrator 管理サービス

iPortal Administrator Console

iPortal Administrator Console は、iPortal Administrator への Java GUI（Graphical User Interface）を提供します。このコンソールからアプリケーションの管理、コンフィギュレーションの設定、およびイベントのロギングを行えます。このコンソールは IIOP（Internet Inter-ORB Protocol）を介して管理サービスと通信します。

iPortal Administrator Web Console

iPortal Administrator Web Console は、HTML と JavaScript を使用して iPortal Administrator への Web ブラウザ・インターフェイスを提供します。Web Console を使用すると、ダウンロードやインストールを行わなくてもアプリケーションの管理とイベント・ロギングをどこからでも実行できます。

iPortal Administrator 管理サービス

iPortal Administrator 管理サービスは、各**ドメイン**の管理情報を一括処理するポイントです。ドメインとは、特定な物理ロケーションで管理されているサーバ・プロセスをまとめた抽象的なグループです。この管理サービスへは iPortal Administrator Console と iPortal Administrator Web Console の両方からアクセスできます。

図 12 に、iPortal Administrator の各ツールが提供する管理機能を示します。

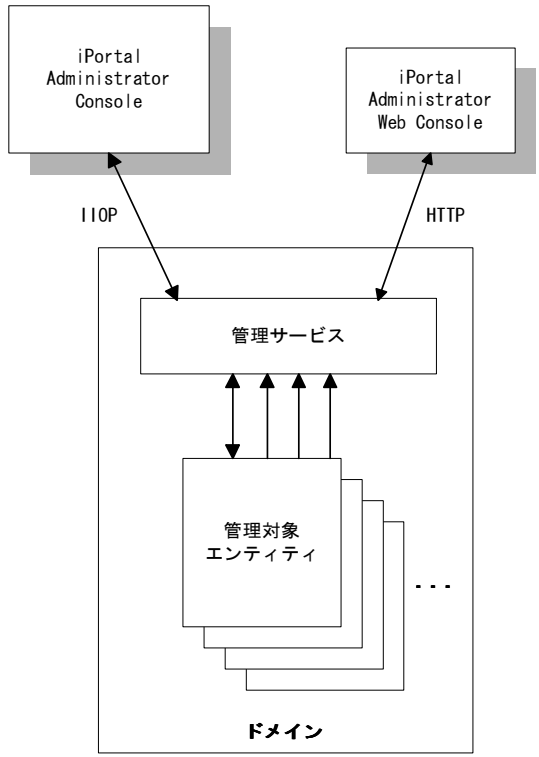


図 12:　iPortal Administrator のコンポーネント

iPortal Administrator についての詳細情報

iPortal Administrator に関するさらに詳しい情報は、『iPortal Administrator ユーザーズ・ガイド』を参照してください。

JMX（Java Management Extensions）の概要

概要

JMX（Java Management Extensions）はユーザ・アプリケーションに管理機能を追加するためのフレームワークを提供する Sun の標準 API です。ここでは JMX の主要な概念について述べ、JMX と iPortal Administrator を組み合わせて Java アプリケーションに管理機能を追加する仕組みを説明します。

MBean

JMX の中心を成すのは MBean（managed bean）という概念です。MBean とは、属性とオペレーションが関連付けられているオブジェクトです。例えば `Car` というオブジェクトの場合、`speed` という属性および、`start()` と `stop()` という各オペレーションを関連付けます。アプリケーション開発者は、ユーザによるアプリケーション管理を可能にする一連の MBean としてアプリケーションを表現します。

MBean サーバ

開発者が作成する全ての MBean は、MBean サーバにより管理および制御されます。作成した全ての MBean は、管理アプリケーション（例えば iPortal Administrator）からアクセスできるように MBean サーバに登録する必要があります。

図 13 に、これらの JMX コンポーネントと iPortal Administrator とのインタラクションにより管理機能が提供される仕組みを示します。

IONA の管理プラグイン

IONA の管理プラグイン（`it_mgmt`）をインストールすると、iPortal Administrator Console などの管理アプリケーションを使用して MBean を表示できます。これには管理プラグインに含まれている IIOP アダプタが必要です。`it_mgmt` 管理プラグインの役割は図 13 を参照してください。

JMX 情報の入手方法

JMX について詳しくは、Sun の『JMX Instrumentation and Agent Specification』および『Reference Implementation javadoc』を参照してください。これらのドキュメントは次の Web サイトから入手できます。

`http://java.sun.com/products/JavaManagement/`

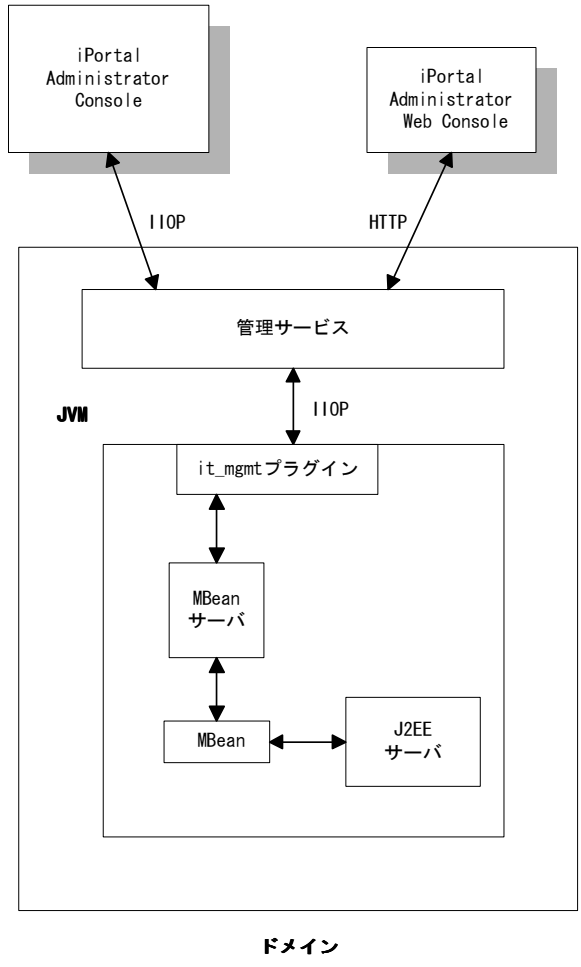


図 13:　JMX と iPortal Administrator のインタラクション

『JMX Reference Implementation』および IONA の **IT_IIOPAdaptorServer** javadoc は、『iPortal Application Server』インストール・ディレクトリにある次のディレクトリにも保存されています。

```
<install-dir>\ipas3\docs\ipaguide\ipa_javadocs
```

iBank のサンプルを使用したコード例

サンプルの EJB アプリケーション iBank に JMX 管理コードを追加する方法は、第 14 章を参照してください。第 14 章では次の操作方法について説明しています。

- サンプル MBean の定義と実装
- MBean サーバへの MBean の登録と削除
- iPortal Administrator Console を使用した MBean の表示

この操作によって、管理者が iPortal Administrator を使用して iBank アプリケーションの管理を行えるようになります。

JMX のノーティフィケーションを使用して iPortal Administrator にイベントを通知する方法については、第 15 章を参照してください。

第 14 章

アプリケーション管理機能の追加

本章では Java アプリケーションに管理機能を追加する方法について説明します。ここでは JMX（Java Management Extensions）API を使用して iPortal Administrator による管理を有効にする方法を解説します。アプリケーションに管理機能を追加するには、まずそのアプリケーションでユーザによる管理を可能にする機能を選択してから、アプリケーションに管理用のコードを追加する必要があります。これらの手順についてはサンプル・アプリケーションの iBank を例に挙げて説明を進めます。

本章は、次のセクションで構成されます。

管理を許可する機能の判断

概要

アプリケーションのどの機能を管理可能にするかの判断はアプリケーションによって異なりますが、主にアプリケーションの特性、ランタイム管理のタイプ、および必要な制御機能によって決定されます。

ここではサンプル・アプリケーションの iBank に管理機能を追加します。このアプリケーションは、一般的なバンキング・サービスをクライアントに提供します。

既存のアプリケーション機能

iBank アプリケーションを使用してユーザが行える操作は次のとおりです。

- ログインとログアウト
- 銀行口座の作成
- 資金の引出し
- パスワードの変更

図 14 に、上記の既存機能へのユーザ・インターフェイスを示します。

追加する管理機能

本章で iBank アプリケーションに追加する管理機能は次のとおりです。

- iBank アプリケーションの状態監視
- iBank アプリケーションの制御（銀行のオープンとクローズなど）
- アクティブな口座の表示
- 新規口座の作成と既存口座の変更
- 口座明細の表示
- 各口座の制御

これらの管理機能を iBank アプリケーションに追加する方法については、196 ページの「プログラミングの手順」を参照してください。追加した機能が iPortal Administrator Console でどう表示されるかについては、207 ページの「iPortal Administrator を使用した MBean の表示」を参照してください。

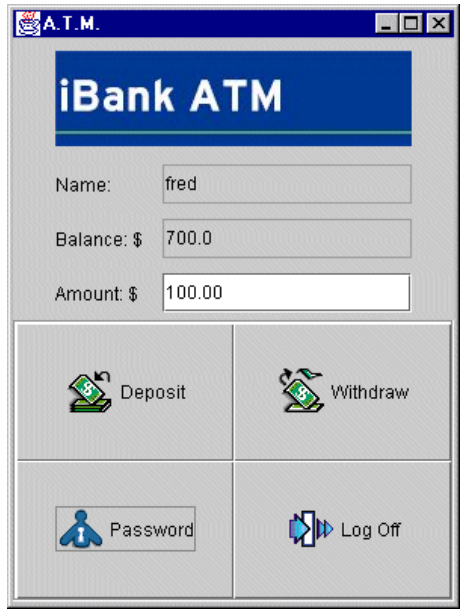


図 14:　iBank アプリケーションのユーザ・インターフェイス

プログラミングの手順

概要

本セクションでは、アプリケーションに管理機能を追加する手順について、iBank アプリケーションを例に挙げて詳しく説明します。

具体的には、管理の対象となる銀行や口座の MBean を定義して実装する方法、および実装した MBean を管理できるよう MBean サーバに登録する方法を説明します。

管理機能のインストルメンテーション

アプリケーションに管理コードを追加することを、**管理機能のインストルメンテーション**、もしくは**アプリケーションへのインストルメンテーション追加**と呼びます。

本セクションは次のトピックで構成されています。

ステップ 1：MBean の定義

概要

アプリケーションで管理の対象となる機能を決定したら、必要な管理機能を満たす MBean を定義できます。本セクションでは，

iBank アプリケーションのサンプル MBean インターフェイスを定義する方法を説明します。

MBean インターフェイスの定義規則

各 MBean オブジェクトが実装するインターフェイスは、例えば `ManagediBankMBean` のように MBean という文字列で終わっている必要があります。

この属性を公開するには MBean インターフェイスで一連の get オペレーションと set オペレーションを宣言します。get オペレーションのみが宣言されている MBean の属性は読取り専用となり、set オペレーションが宣言されている MBean の属性は書込み可能となります。

管理オペレーションを公開するには MBean インターフェイスで適切なメソッドを宣言し、これを MBean クラスで実装する必要があります。

サンプルの MBean

次に、iBank アプリケーションの機能を管理するために必要な MBean を示します。

MBean	機能
ManagediBankMBean	管理する銀行サービスを表します。例えば銀行のオープンとクローズなどに使用します。
ManagediBankAccountMBean	管理する銀行口座を表します。例えば既存口座の変更などに使用します。
ManagediBankAuthorizationMBean	管理するログイン認証サービスを表します。例えばユーザ・ログインの監視などに使用します。

ManagediBankMBean インターフェイス

次に **ManagediBankMBean** のインターフェイス定義を示します。

```
// MBean that represents the bank service to be managed.

public interface ManagediBankMBean  {

  // attributes
  public int            getNumberOfAccounts();
  public ObjectName[]   getAccounts();
  public ObjectName     getAuthorization();
  public boolean        getBankOpen();
  public String         getUptime();

  // operations
  public String         createAccount(String name, String
                                                  password);
  public String         removeAccount(String name);
  public String         closeBank();
```

```
  public String          openBank();
  public String          unregisterMBean();
};
```

ManagediBankAccount MBean インターフェイス

次に **ManagediBankAccountMBean** のインターフェイス定義を示します。

```
// MBean that represents the account to be managed.

public interface ManagediBankAccountMBean
{
  // attributes
  public String          getName();
  public double          getBalance();
  public void            setBalance(double newBalance);
  public boolean         getLocked();
  public ObjectName      getBankingServer();
  public int             getAccountID();

  // operations
  public String[]        changePassword(String oldPassword,
                                        String newPassword);
  public String[]        lockAccount();
  public String[]        unlockAccount();
};
```

ManagediBankBank AuthorizationMBean インターフェイス

次に ManagediBankAuthorizationMBean インターフェイスの定義を示します。

```
// MBean that represents the authorization service to be managed.

public interface ManagediBankAuthorizationMBean
{
  // attributes
  public Integer getSuccessfulLogons();
  public Integer getFailedLogons();
  public Integer getCurrentUsers();
  public ObjectName getBank();

  // operations
  public String createAccount(String name, String password);
  public String removeAccount(String name);
};
```

MBean の識別

MBean オブジェクトの名前は、MBean を識別するために使用されます。オブジェクト名は javax.management.ObjectName クラスで表され、このクラスは **java.lang.Object** クラスを拡張します。

サンプルのオブジェクト名

iBank アプリケーションでは ManagediBankAuthorizationMBean インターフェイスが **Bank** 属性について次の get メソッドを宣言しています。

```
public ObjectName getBank()
```

このメソッドはその銀行の MBean**ObjectName** を返します。

ManagediBankMBeanインターフェイスは、**Accounts**属性について次のgetメソッドを宣言しています。

```
public ObjectName[] getAccounts();
```

このメソッドは関連口座の MBean オブジェクト名の配列を返します。

詳細情報

MBean オブジェクト名の指定方法について詳しくは、202 ページの「ステップ 4 : MBean の登録」を参照してください。

ObjectName クラスについて詳しくは、Sun の『JMX Reference Implementation javadoc』を参照してください。このドキュメントは iPortal Application Server インストール・ディレクトリ内の次の場所に保存されています。

```
install-dir\ipas3\docs\ipaguide\ipa-javadocs
```

ステップ 2：MBean の実装

概要

MBean の定義が済んだら、MBean を実装します。通常 MBean の実装オブジェクトは管理対象のアプリケーションとインタラクトして監視と制御を可能にします。

MBean のサンプル実装

次のサンプル・コードは**ManagediBankMBean** の実装です。これには MBean が実行する一般的なタスクが含まれています。

```
public class ManagediBank implements ManagediBankMBean {

  private static IT_IIOPAdaptorServer adaptorServer = null;
  private static int sequenceNumber = 0;
  private iBankBean iBank;

  // Attributes
  public int getNumberOfAccounts() {
    return iBank.getNumberOfAccounts();
  }

  public ObjectName[] getAccounts() {
    return iBank.getAccounts();
  }

  public ObjectName getAuthorization() {
    return iBank.getAuthorizationObj();
  }

  public boolean getBankOpen() {
    return iBank.getBankOpen();
  }

  public String getUptime() {
    return iBank.getUptime();
  }

  //Methods
  public String createAccount(String name, String password) {
    return iBank.createAccount(name, password);
  }
  .
  .
  .
```

iBankBean オブジェクト

この例では **ManagediBank** を表す MBean が **iBankBean** というオブジェクトを使用して大半の操作を実行しています。

MBean の実装では、このように MBean が他のアプリケーション・オブジェクトの機能を使用して管理機能を提供するのが一般的な方法です。

このサンプル・コードで初期化されている IT_IIOPAdaptorServer オブジェクトについて詳しくは、201 ページの「ステップ 3 : MBean サーバへのアクセス取得」を参照してください。

ステップ 3 : MBean サーバへのアクセス取得

概要

MBean の定義と実装が済んだら、これを MBean サーバに登録します。登録した MBean は、iPortal Administrator Console と Web Console を使用して監視と制御を行えます。

MBean を登録する前に、まず MBean サーバへのアクセスを取得する必要があります。iPortal Application Server では管理機能がデフォルトで有効になっており、すでに MBean サーバが用意されています。

デフォルトの MBean サーバへのアクセス

次のサンプル・コードは、**ManagediBankAuthorization** クラスがデフォルトの MBean サーバにアクセスする方法を示しています。

```
IT_IIOPAdaptorServer adaptorServer =
      (IT_IIOPAdaptorServer)com.iona.management.
        jmx_iiop.IT_DynamicLoading.getDefaultIIOPAdaptorServer();

MBeanServer mBeanServer = adaptorServer.getMBeanServer();
```

IT_IIOPAdaptorServer オブジェクト

MBeanServer へのリファレンスを提供するには **IT_IIOPAdaptorServer** オブジェクトを使用します。デフォルトの **MBeanServer** にアクセスしたら、次に MBean を登録します。

注 : **IT_IIOPAdaptorServer** オブジェクトには、MBean サーバへのアクセスを提供する以外にも用途があります。詳しくは 208 ページの「ルート MBean」を参照してください。

詳細情報 **IT_IIOPAdaptorServer** についての詳しいリファレンス情報は、**IT_IIOPAdaptorServer** javadoc を参照してください。このドキュメントは iPortal Application Server のインストール・ディレクトリ内の次の場所に保存されています。

```
install-dir\ipas3\docs\ipaguide\ipa-javadocs
```

ステップ 4：MBean の登録

概要 MBean サーバへアクセスしたら、次に MBean を MBean サーバに登録します。MBean オブジェクトの登録は次のどちらかの方法で行います。

- **createMBean()** コンストラクタを使用して MBean オブジェクトを作成し MBean サーバに登録する。この場合 MBean は自動登録されます。
- MBean オブジェクトを手動で作成し、次に MBean サーバに登録する。この場合、**new()** コンストラクタと `registerMBean()` メソッドを使用します。

本セクションの例では 2 番目の方法を使用します。

例 次の例では **ManagediBankAuthorization** という MBean を手動で作成し、MBean サーバに登録します。

```
// Create the MBean.
authMBeanName authMBean = new ManagediBankAuthorization();
authMBean.setiBankAuthorization(this);

// Gain access to the default MBean server.
IT_IIOPAdaptorServer adaptorServer =
   (IT_IIOPAdaptorServer)com.iona.management.
    jmx_iiop.IT_DynamicLoading.getDefaultIIOPAdaptorServer();

MBeanServer mBeanServer = adaptorServer.getMBeanServer();

// Create the MBean object name.
authMBeanName = new
   ObjectName("iPAS:name=ManagediBankAuthorization");

if (mBeanServer.isRegistered(authMBeanName))
{
  try {
       mBeanServer.unregisterMBean(authMBeanName);
```

```
   } catch (Exception ex) {}
}

// Register the MBean with the MBean server.
mBeanServer.registerMBean(authMBean, authMBeanName);
```

MBean の作成

この例では、`new()` コンストラクタを使用して手動で **`ManagediBankAuthorization`** という MBean を作成します。この MBean は `authMBeanName` というオブジェクト名を使用します。

MBean の登録

この例では、MBean サーバの **`registerMBean()`** メソッドを使用して MBean を登録します。

`registerMBean()` メソッドには次の 2 つのパラメータを指定します。

- 登録するオブジェクトのクラス名。これはクラスパス上にある必要があります。この例では **`authMBean`** というクラス名を使用します。
- MBean を識別するための **`ObjectName`**。この例では **`authMBeanName`** というオブジェクト名を使用します。

MBean の識別

`registerMBean()` メソッドの **`ObjectName`** パラメータには MBean サーバで固有の名前を指定します。オブジェクト名にはドメイン名、コロン、名前と値のペアのリストをそれぞれ含めます。名前と値のペアには全ての型と値を使用できます。パラメータ指定のシンタックスは次のとおりです。

```
domain-name:name1=value1,name2=value2,...
```

iBank の例で使用するオブジェクト名 **`authMBeanName`** は、次のドメイン、および名前／値のペアを表します。

```
iPAS:name=ManagediBankAuthorization
```

詳細情報

`ObjectName` クラスについて詳しくは、Sun の『JMX Reference Implementation javadoc』を参照してください。このドキュメントは iPortal Application Server のインストール・ディレクトリ内の次のロケーションに保存されています。

```
install-dir\ipas3\docs\ipaguide\ipa-javadocs
```

MBean を登録するタイミング

概要

MBean を登録もしくは公開するタイミングは個々のアプリケーションによって異なりますが、次の状況下では、全てのアプリケーションが MBean の作成と登録を行います。

- アプリケーションの初期化中
- アプリケーションの実行中

アプリケーションの初期化中

アプリケーションの初期化シーケンスでは、アプリケーションの主要機能を提供する一連のオブジェクトが作成されます。これらのオブジェクトが作成された後で MBean を作成して登録し、アプリケーションの基本的な管理機能を有効にする必要があります。

アプリケーションの実行中

アプリケーションが正常に実行されている間にも、内部イベントや外部イベントの結果として新しいオブジェクトが作成されます。これには例えば内部タイマーや外部クライアントからのリクエストなどが該当します。新しいオブジェクトが作成された場合、これに対応する MBean を作成して登録し、新たにアプリケーションに追加されたエレメントの管理機能を有効にすることができます。

例えば iBank アプリケーションでは、新しい口座が作成されると `ManagedAccount` MBean が新規作成され、MBean サーバに登録されます。

ステップ 5 : サーバからの MBean の削除

概要

ユーザが行うシステム操作に応じてサーバから MBean を削除する場合があります。例えば、銀行から口座が削除された場合、その口座に対応する `ManagediBankAccountMBean` MBean をサーバから削除します。サーバから削除した MBean は管理するアプリケーションの一部として表示されなくなります。

MBean の削除

MBean を削除するには MBean サーバの `unregisterMBean()` メソッドを使用します。次の例では口座を削除し、その MBean を MBean サーバから削除しています。

```
// Delete an account from the bank.
boolean result = true;
   try {
```

```
        synchronized (accountListSyncObj) {
           iBankAccount account = null;
           int index = -1;
           for (int i = 0; i < accountList.size(); i++) {
                account = (iBankAccount)accountList.elementAt(i);
                if (account.getName().equals(name)) {
                    index = i;
                    break;
                }
                else {
                      account = null;
                }
           }
           if (account != null) {
              accountList.removeElementAt(index);
              account.close();
              account.remove();

// Gain access to the default MBean server.
             IT_IIOPAdaptorServer adaptorServer =
             (IT_IIOPAdaptorServer)com.iona.management.jmx_iiop.
             IT_DynamicLoading.getDefaultIIOPAdaptorServer();

             MBeanServer mBeanServer
                                =adaptorServer.getMBeanServer();


// Unregister the account MBean with the MBean Server.
              mBeanServer.unregisterMBean(account.getObjectName());
           }
           else {
               result = false;
           }
         }
      } catch (Exception ex) {
           result = false;
           System.out.println("Unexpected exception in
                                    removeAccoutn: " + ex);
           ex.printStackTrace();
      }
      if (result) {
           return "Account has been removed successfully for user:
           "+ name;
      }
      else {
```

```
            return "Failed to remove account for user: " + name;
        }
    }
```

unregisterMBean() メソッド

unregisterMBean() メソッドを使用して削除した MBean は、iPortal Administrator Console や iPortal Administrator Web Console に表示されなくなります。

unregisterMBean() メソッドは、パラメータとして MBean オブジェクト名を受け入れます。オブジェクト名について詳しくは、199 ページの「MBean の識別」を参照してください。

iPortal Administrator を使用した MBean の表示

概要

本セクションでは iPortal Administrator での MBean の表示内容について iPortal Administrator Console を例に挙げて説明します。MBean の表示は iPortal Administrator Web Console でも行うことができます。

ここではまず**ルート MBean** という概念について述べた後、ルートへの MBean の追加、ナビゲーション・ツリーでの MBean の表示、およびアイコンのカスタマイズを行う方法を説明します。

iPortal Administrator Console

図 15 に iPortal Administrator Console の Server View を示します。この例は `ManagediBank` アプリケーションで管理される属性とオペレーションを示しています。ここに表示されている属性とオペレーションは、196 ページの「ステップ 1：MBean の定義」で宣言されているものに対応します。

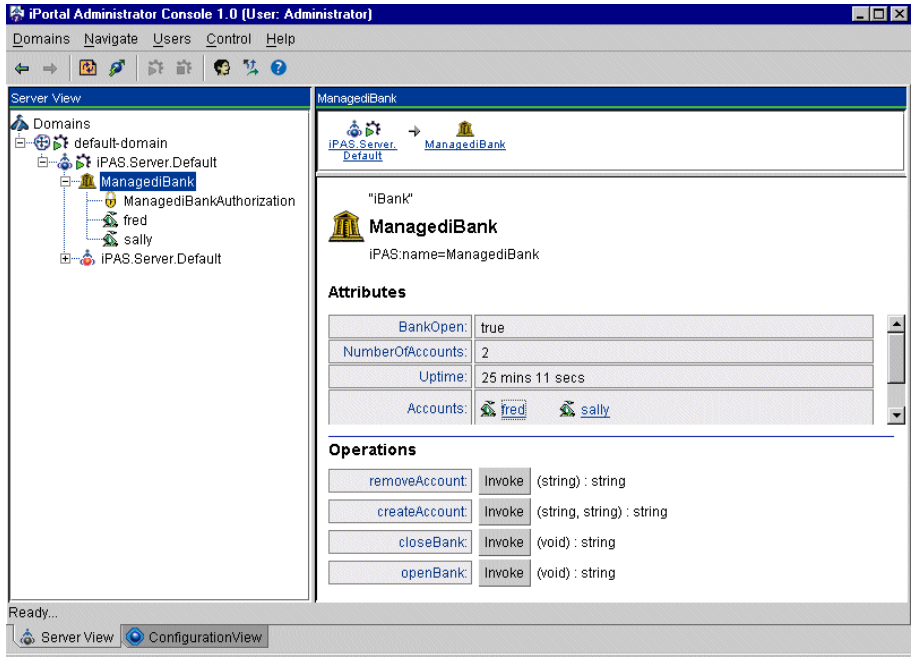


図 15:　iPortal Administrator Console

ルート MBean

201 ページの「ステップ 3 : MBean サーバへのアクセス取得」には、`IT_IIOPAdaptorServer` オブジェクトを使用してデフォルトの MBean サーバにアクセスする方法が示されています。`IT_IIOPAdaptorServer` オブジェクトは iPortal Application Server の起動時に自動作成されます。

`IT_IIOPAdaptorServer` のインスタンスが作成されると、iPortal Administrator Console がその Server View のナビゲーション・ツリーにエントリを作成します。このエントリが**ルート MBean** で、これは公開される最上位の MBean を表します。ルート MBean は iPortal Administrator Console を使用してサーバをナビゲートする際の出発地点となります。

ルート MBean の例

図 16 のナビゲーション・ツリーでは `iPAS.Server.Default` というルート MBean が選択されていて、詳細パネルにはこのルート MBean エントリに関連付けられているプロセス・タイプ、アップタイム、ホストなどのデフォルト属性が表示されています。

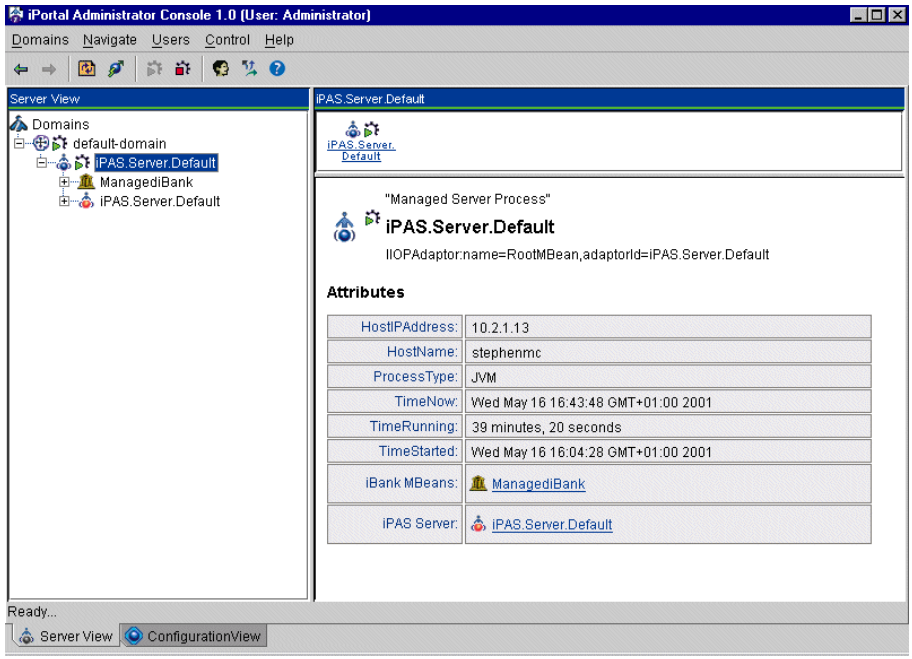


図 16:　ルート MBean

セクションの内容 本セクションは次のトピックで構成されています。

アプリケーション MBean のルートへの追加

概要 iPortal Administrator Console でアプリケーションの MBean を表示するには、まずアプリケーションの MBean をルート MBean に追加する必要があります。MBean を追加するには **IT_IIOPAdaptorServer** インスタンスの **addToRootMBean()** メソッドを使用します。

ルート MBean に追加した MBean は、ナビゲーション・ツリーではルート MBean の子として、また詳細パネルではその属性として表示されます。図 16 のナビゲーション・ツリーでは **ManagediBank** MBean が子として表示され、詳細パネルでは属性として表示されています。

サンプル・コード 次のサンプル・コードは **ManagediBankBean** を作成して登録し、ルート MBean に追加します。

```
if (managediBank == null) {
   try {
     managediBank = new ManagediBank();
     managediBank.setiBank(this);

     // Gain access to the default MBean server.
     IT_IIOPAdaptorServer adaptorServer =
        (IT_IIOPAdaptorServer)com.iona.management.
       jmx_iiop.IT_DynamicLoading.getDefaultIIOPAdaptorServer();

     // Create the MBean.
     ObjectName managediBankObjName =
              new ObjectName("iPAS:name=ManagediBank");

     // Get the default MBean server.
     MBeanServer mBeanServer = adaptorServer.getMBeanServer();
      // Check if MBean is already registered.
      if (mBeanServer.isRegistered(managediBankObjName)) {
```

```
            try {
             adaptorServer.removeFromRootMBean(managediBankObjName);
                mBeanServer.unregisterMBean(managediBankObjName);
              }
            catch (Exception ex) {}
          }

       // Register MBean with the MBean server.
       mBeanServer.registerMBean(managediBank,managediBankObjName);


      // Add MBean to root MBean.
      adaptorServer.addToRootMBean(
                     managediBankObjName, // ObjectName
                     "iBank MBeans");     // Attribute name
      }
      catch (Exception ex) {
         System.out.println("Unexpected exception while registering
                               iBankMBean: " + ex);
      }
}
```

addToRootMBean() メソッド

`addToRootMBean()` メソッドは、MBean がルート MBean の属性として表示される場合に使用される属性名と、オブジェクト名の 2 つのパラメータを受け入れます。

注： ルート MBean に追加した MBean を表示するには、MBean の登録が済んでいる必要があります。MBean の登録方法について詳しくは、202 ページの「ステップ 4：MBean の登録」を参照してください。

ナビゲーション・ツリーでの MBean の表示

概要

MBean をナビゲーション・ツリーで表示するには、アプリケーション MBean をルート MBean に追加するだけでなく、`ipa_view.xml` ファイルを更新する必要があります。このファイルは、次のディレクトリに保存されています。

```
install-dir/ipas3/etc/resources
```

本セクションではこのファイルの更新方法について例を挙げて説明します。

サンプル・ファイル

次に、**ipa_view.xml** ファイルを使用してデフォルトのナビゲーション・ツリー階層を指定する方法を示します。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE root_element SYSTEM "ipa_view.dtd">
<root_element>
   <ipa_view>
      <name>iPASHierarchy</name>
      <server>iPAS*</server>
      <mbean>
         <name>iPAS</name>
         <objectname_tagelement>type</objectname_tagelement>
         <objectname_tagvalue>Server</objectname_tagvalue>
         <child_mbean>
            <attribute_name>DeployedApplications</attribute_name>
         </child_mbean>
         <child_mbean>
           <attribute_name>DataSources</attribute_name>
         </child_mbean>
         <child_mbean>
            <attribute_name>Namespaces</attribute_name>
         </child_mbean>
         <child_mbean>

   <attribute_name>JMSConnectionFactories</attribute_name>
         </child_mbean>
      </mbean>

      <mbean>
         <name>iPASContainer</name>
         <objectname_tagelement>type</objectname_tagelement>
         <objectname_tagvalue>Container</objectname_tagvalue>
         <child_mbean>
            <attribute_name>Modules</attribute_name>
         </child_mbean>
      </mbean>
     .
     .
     .
   </ipa_view>
```

MBean の識別方法

MBean の型を識別するには次の XML エレメントを使用します。

```
<objectname_tagelement>
<objectname_tagvalue>
```

例えば、**ManagediBank** 型の MBean の場合は次のようになります。

```
<objectname_tagelement>name</objectname_tagelement>
<objectname_tagvalue>ManagediBank</objectname_tagvalue>
```

ManagediBank 型の MBean は、次を含む **ObjectName** を持つことになります。

```
SomeDomain:...,name=ManagediBank,...
```

ツリーでの階層指定

階層の指定には **<child_bean>** という XML エレメントを使用します。**<child_bean>** および **</child_bean>** タグは **<mbean>** と **</mbean>** タグの間に挿入します。このタグは、コンソールが **<child_bean>** エレメントで指定された属性を表示する際にツリー形式を使用するように指定します。

ツリー階層の例

次のコードは **ipa_view.xml** ファイルでのツリー階層の例を示しています。

```
  <ipa_view>
      <name>SampleHierarchy</name>
      <server>*Sample*Server*</server>
      <mbean>
         <name>SampleMBean1</name>

   <objectname_tagelement>SampleTag1</objectname_tagelement>
         <objectname_tagvalue>SampleValue1</objectname_tagvalue>
         <child_mbean>
            <attribute_name>SampleAttribute1.1</attribute_name>
         </child_mbean>
         <child_mbean>
            <attribute_name>SampleAttribute1.2</attribute_name>
         </child_mbean>
      </mbean>

      <mbean>
         <name>SampleMBean2</name>

   <objectname_tagelement>SampleTag2</objectname_tagelement>
         <objectname_tagvalue>SampleValue2</objectname_tagvalue>
```

```
            <child_mbean>
               <attribute_name>SampleAttribute2.1</attribute_name>
            </child_mbean>
            <child_mbean>
               <attribute_name>SampleAttribute2.2</attribute_name>
            </child_mbean>
         </mbean>
      </ipa_view>
</root_element>
```

注： ツリー構造の表示を指定するのは、アプリケーションの instrumentation が階層式である場合のみです。循環的なリファレンスがある場合にツリー表示を指定すると、例えば MBean A が MBean B に参照し、この MBean B が逆に MBean A に参照するというように、管理者にとってわかりづらい内容になります。

アプリケーション MBean のアイコンのカスタマイズ

概要

新しく追加された MBean は、デフォルトでは青い MBean アイコンとして表示されます。iPortal Administrator ではカスタムのアイコンを使用してアプリケーション MBean を表示するようにも指定できます。例えば iBank アプリケーションは **ManagediBank** MBean を銀行アイコンで表し、**ManagediBankAccountMBean** MBean をキャッシュのアイコンで表します。207 ページの図 15 を参照してください。

本セクションではイメージ・マッピング・ファイルにカスタム・アイコンを追加する方法および、追加したアイコンをクラスパスからアクセスできるようにする方法を説明します。

イメージ・マッピング・ファイルの更新

カスタム・アイコンを使用するには、**image_mapping.properties** ファイルを更新する必要があります。このファイルは次のディレクトリに保存されています。

```
install-dir/ipas3/etc/domains/resources
```

例えばこのファイルにリストされている iBank MBean の場合、各 MBean につき 3 つのエントリが指定されています。ManagediBank MBean のエントリは次のとおりです。

```
examples.ejb.management.ibank.ManagediBank.small =
                                        resources/images/bank16.gif
examples.ejb.management.ibank.ManagediBank.large =
                                        resources/images/bank32.gif
examples.ejb.management.ibank.ManagediBank.text = "iBank"
```

これらのエントリは、**ManagediBank** の小さなアイコン（16x16）、大きなアイコン（32x32）、およびアイコン用の表示テキストをそれぞれ指定します。

各プロパティ名の最初の部分（**examples.ejb.management.ibank.ManagediBank** など）は MBean のクラス名です。

クラスパスからアイコンへのアクセス

指定したアイコンを使用するには、クラスパスからこのアイコンへのアクセスを可能にする必要があります。例えば image_mapping.properties ファイルにある

```
com.acme.myapp.ManagedThing.small =
                                    resources/images/mgdthng16.gif
```

というエントリの場合、**mgdthng16.gif** を **my-dir/etc/resources/images** ディレクトリに保存します。iPortal Administrator Cosole または iPortal Administrator 管理サービスの起動時に **my-dir/etc** がクラスパス上にあることを確認します。

この方法の代わりにリソースをアプリケーションの jar ファイルに追加して、アプリケーション JAR ファイルをパスに含めることもできます。

MBean の接続

概要

iPortal Administrator Console では、ルート MBean がアプリケーションをブラウズする際の出発地点となります。このルート MBean はアプリケーションにより自動登録されます。

iPortal Administrator Console ではアプリケーションが相互に接続されている一連の MBean として表されます。本セクションでは、MBean を相互接続する方法について説明します。

2 つの MBean の接続

2 つの MBean を接続する方法は簡単です。MBean を接続するには get メソッドを使用します。例えば **ManagediBank** MBean は、その全ての `ManagediBankAccount` MBean に接続する必要があります。

ManagediBank MBean をその全ての `ManagediBankAccount` MBean に接続するには、**ManagediBank** MBean が持つ次の get メソッドを使用します。

```
//  public ObjectName[] getAccounts();
```

このメソッドは、管理される口座のオブジェクト名を返します。返されたオブジェクト名がすでに MBean 名として登録されている場合、これらの MBean は **ManagediBank** MBean の **Accounts** 属性として表示されます。

詳細情報

オブジェクト名について詳しくは、199 ページの「MBean の識別」を参照してください。

標準 MBean と動的 MBean

概要 本セクションでは標準の MBean と動的 MBean という概念を説明し、これら 2 タイプの MBean の特長について述べます。

標準 MBean 本章でこれまでに述べた MBean は全て**標準 MBean** です。標準 MBean は管理されるデータの構造が事前定義されていて更新が頻繁でない、比較的シンプルな管理を行う場合に適しています。

動的 MBean JMX の仕様には、**動的 MBean** についての概要も述べられています。これらの MBeans は、MBean の公開データがアプリケーションの使用中に更新されるような場合に適しています。

制限事項 動的 MBean は実装が複雑で、標準 MBean を使用する場合と比べて時間がかかります。既存の管理プロトコルやプラットフォームを新しいものと統合する必要がある管理機能で動的 MBean を使用すると、これらの統合が困難になることがあります。

詳細情報 動的 MBean について詳しくは、JMX の仕様を参照してください。このドキュメントは次のサイトから入手可能です。

`http://java.sun.com/products/JavaManagement/`

第 15 章

JMX ノーティフィケーションを使用したイベント送信

本章では JMX ノーティフィケーションを使用して Orbix などの管理アプリケーションに対してイベントを送信する方法について説明します。操作手順では iBank アプリケーションを例にとって説明を進めます。

JMX ノーティフィケーションを使用して、関連パーティーにイベントを通知できます。MBean は、ブロードキャスト元の MBean をリスンしている他の全ての MBean に対して、ノーティフィケーション・イベントを送信（つまりブロードキャスト）することができます。

例えば iBank アプリケーションでは、不正なパスワードが原因で特定口座へのユーザ・ログインが失敗した場合にノーティフィケーションを送ることができます。

本章は、次のセクションで構成されます。

イベント・ノーティフィケーションの送信

概要

オブジェクトが JMX イベントのノーティフィケーションを送信するには、このオブジェクトがノーティフィケーション・ブロードキャスト・インターフェイスを実装しているか、これを実装するクラス（例えば **NotificationBroadcasterSupport** クラス）を拡張する必要があります。

ここではユーザ・ログインの失敗時にイベント・ノーティフィケーションを送信する例について説明します。

サンプル・ノーティフィケーション

iBank アプリケーションでは **ManagediBankAuthorization** MBean がノーティフィケーションの送信元（ブロードキャスト元）です。

イベント・ノーティフィケーションを実装するには **NotificationBroadcasterSupport** クラスを拡張するのが一番簡単な方法です。次のコード例を参照してください。

```
public class ManagediBankAuthorization
   extends NotificationBroadcasterSupport
   implements ManagediBankAuthorizationMBean {

  private static int sequenceNumber = 0;
  .
  .
  .

   public void sendLoginFailedNotification(String username) {
   try {
     if (adaptorServer == null) {
        adaptorServer =(IT_IIOPAdaptorServer)com.iona.management.
        jmx_iiop.IT_DynamicLoading.getDefaultIIOPAdaptorServer();
      }

      String msg = "Attempt to log into account FAILED for
                        username: " + username;

      // Send a Notification showing that the login failed.
      adaptorServer.sendNotification(
        new Notification("ManagediBankAuthorization.loginFailed",
        this, sequenceNumber++, new java.util.Date().getTime(),
        msg));
```

```
    } catch (Exception ex) {}
  }
}
```

ログイン失敗のノーティフィケーション送信

銀行がログイン失敗を検知した場合、**ManagediBankAuthorization** MBean の **loginFailed()** メソッドを呼び出して、sendNotification() メソッドを使用してこれに対応する sendLoginFailedNotification を送信できます。

詳細情報

iPortal Administrator Console でのノーティフィケーションの表示について詳しくは、219 ページの「iPortal Administrator での ノーティフィケーション表示」を参照してください。

iPortal Administrator での ノーティフィケーション表示

概要

iPortal Administrator により、イベント・ノーティフィケーションを iPortal Administrator 管理サービスに送信することができます。管理サービスはノーティフィケーションを iPortal Administrator Console および Web Console で表示できるように、これらを一括して保管します。

iPortal Administrator の Events Console

図 17 に、iPortal Administrator の Events Console に表示された iBank からのイベント・ノーティフィケーションの例を示します。Event Name フィールドには JMX.mbean.registered と ManagediBankAuthorisation.loginFailed イベントの例が表示されています。

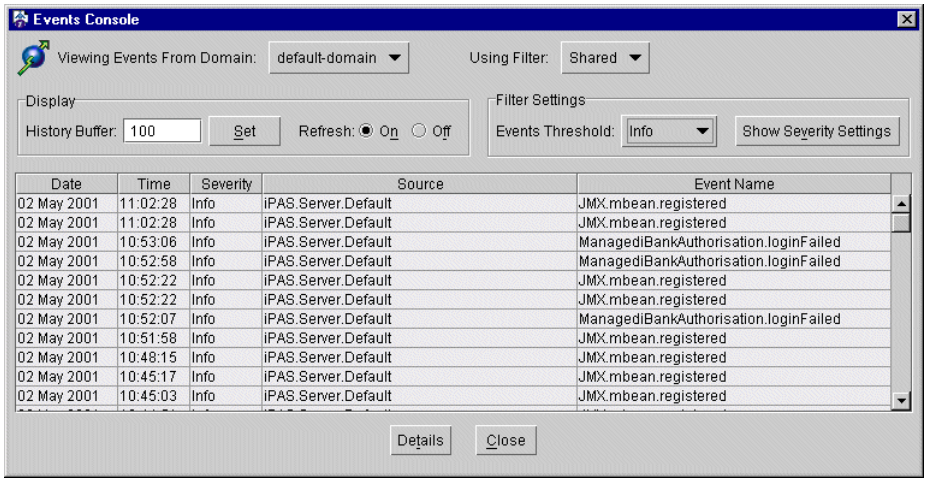


図 17: iPortal Administrator Console の Events Console

登録と削除のノーティフィケーション

デフォルトでは MBean のサーバ登録と削除に関するノーティフィケーションは全て管理サービスに送られ、iPortal Administrator Console および Web Console に表示されます。図 17 に JMX.mbean.registered イベントの例を示します。

その他のノーティフィケーションの表示

iPortal Administrator Console や Web Console でその他の全イベント・ノーティフィケーションを表示するには、IT_IIOPAdaptorServer sendNotification() メソッドを使用してイベント・ノーティフィケーションをブロードキャストします。

IT_IIOPAdaptorServer sendNotification() メソッド使用して、javax.managment.Notification またはこれから派生したノーティフィケーションを送信できます。本章の例では ManagediBankAuthorisation.loginFailed オブジェクトを送信しています。

ノーティフィケーションの表示例

図 17 に、iPortal Administrator Console の Events Console に表示されるイベント・ノーティフィケーションのリストの例を示します。図 18 は、Event Details ダイアログに表示される個々のイベントに関する詳細情報（ここでは `ManagediBankAuthorisation.loginFailed` イベントに関する詳細）の例です。

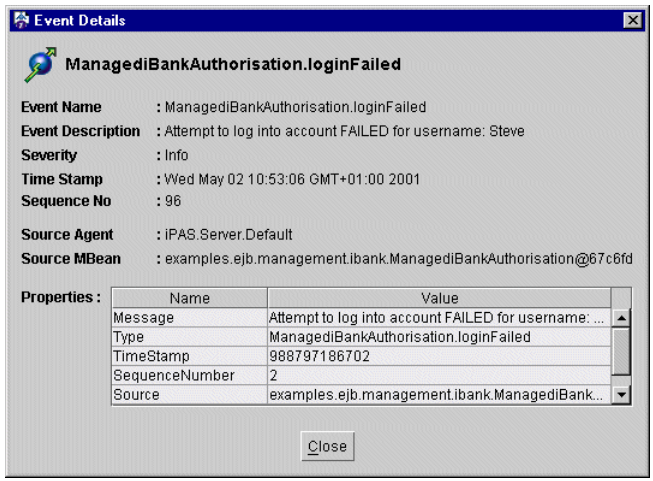


図 18:　Event Details ダイアログ

詳細情報

本章では JMX ノーティフィケーションの基本概念について述べました。さらに詳しい情報については、次のサイトから入手可能な JMX の仕様を参照してください。

`http://java.sun.com/products/JavaManagement/`

第 16 章

iPortal Application Server の高度な機能

本章では iPortal Application Server の本リリースでサポートされる高度な機能について説明します。これらのトピックでは XML（eXtensible Markup Language：拡張マークアップ言語）、分散 HTTP サービス、および Web サービスについて触れます。XML は HTML の問題点を改善するために World Wide Web Consortium が開発した新しいマークアップ言語です。また、本章には iPortal Application Server で分散 HTTP セッションを使用する方法や、IONA の Web サイトに関する情報も記載されています。

本章は、次のセクションで構成されます。

XML の概要

概要

本セクションでは XML の概要について説明します。XML は HTML の問題点を改善するために W3C（World Wide Web Consortium）が開発した新しいマークアップ言語で、HTML の使用を補充および拡張します。XML の仕様により、抽象ドキュメントのシンタックスを定義できます。

XML の特長

XML は、SGML（Standard Generalized Markup Language）を Web で使用するために開発されました。XML は事前定義された特定のマークアップ言語ではなく、他の言語を記述するための言語（これをメタ言語と呼びます）です。XML には事前定義されたタグやエレメントがないので、独自のタグを作成してデータを表すことができます。したがって、HTML にはないタグが必要な場合に言語を拡張することができる一方、必要な数だけのタグを使用することも可能です。XML はクロス・プラットフォーム言語で、これを使用すればソフトウェアやハードウェアに依存せずに情報を転送できます。

セクションの内容

本セクションは、次のトピックで構成されています。

XML の利点

概要

XML は HTML 風のシンタックスを使用するため、その使用は企業間で急速に普及しています。柔軟性に富み、全プラットフォームで使用できるのも XML の利点です。

XML は、コンテンツとデータを構成するメカニズムを提供する標準ツールです。XML を使用すれば、ページのコンテンツだけでなく、その構造に関する情報もアプリケーションが送信できます。また、XML は Web 上で SGML を使用しやすくするために設計されているので、ドキュメント・タイプの定義、SGML で記述されたドキュメントの作成と管理、および Web を介したデータの転送と共有が容易に行えます。

XML は、SGML のごくシンプルなダイアレクトを定義します。この定義は XML の仕様に含まれています。XML の目的は、汎用の SGML を現在 HTML で行えるような方法により Web 上で配信、受信、処理できるようにすることです。このため XML は実装の容易さと SGML および HTML とのインターオペラビリティを重視して開発されました。

XML はもともと出版業界で植字の記述にレイアウト指示を追加するために開発された言語です。ワープロ・ドキュメントなどの電子ドキュメントでは、電子処理に必要な情報を定義するマークアップがドキュメント内にコードとして埋め込まれています。

XML の用途

XML は次のような分野での使用に適しています。

- パブリッシング
  - HTML を使用した大規模な Web サイトの保守。XML により HTML ドキュメントの作成を容易にします。
  - WAP および HTML によるメディア作成。複数の Web サイトやメディアでコンテンツを使用できるようにします。
  - 科学用アプリケーション。数式や化学式を表現できます。
  - 電子ブック。表現に大量のマークアップを含めることができるので研修マニュアルなど短いドキュメントの記述に適しています。
- 情報交換
  - 企業間の通信。複数システムや複数企業間でやり取りするドキュメント（またはその一部）の記述に適しています。
  - 電子商取引アプリケーションの相互運営。複数企業が協力して顧客にサービスを提供できます。
  - データベースのオフロード／リロード。データベース・マイグレーションを容易にします。

XML アプリケーションの種類

XML アプリケーションは次の 2 種類に分類されます。

- ドキュメント・アプリケーション
- データ・アプリケーション

ドキュメント・アプリケーション

ドキュメント・アプリケーションは、人間が読んで理解するための情報を操作するアプリケーションです。XML は配信メディアに依存しないので、XML ドキュメントをさまざまなメディアで自動的にパブリッシュし、例えば HTML をブラウザに含めたり、WAP 電話に WML や HTML を含めたり、またプリンタにポストスクリプトや PDF 形式として送ることができます。

データ・アプリケーション

データ・アプリケーションは、ソフトウェアのみが使用する情報を操作するアプリケーションです。データ・アプリケーションでは、ドキュメント・アプリケーションでデータベースなどのデータを管理するために使用されるのと同じツールやアプリケーションを使用できます。すると、データベースのデータを XML で表現することにより、データベースをドキュメントと同じ方法で処理できます。自動ソリューションを使用してデータベースにある提携企業の Web サイト上の変更情報を XML により更新することで、リアルタイムの最新情報を顧客に提供できます。

XML は構造化された情報を保存するのに適しています。これはファイル全体に関してだけでなく、ファイルの一部を処理する場合にも便利です。特に XML パーサーはイベント指向のシステムで動作するので、ドキュメントの残り部分をパースする前にまず最初の部分を処理することができます。

XML ドキュメントに含まれているデータの構造は、ドキュメント・タイプ定義（DTD：Document Type Definition）に指定されています。ドキュメントを DTD に照合して妥当性をチェックし、無効なデータは自動的に拒否することもできます。

ドキュメント・タイプ定義（DTD）ファイル

DTD ファイルにはドキュメントのクラスの構文を提供するマークアップ宣言が含まれています。このファイルは特定タイプのドキュメントの正式な記述を定義する、XML の宣言シンタックスで記述されています。このファイルにはエレメント・タイプに使用できる名前、その使用個所や相互の関係などが定義されています。例えば、**<Item>** を含む **<List>** を記述できるドキュメント・タイプを宣言する場合、DTD には次のようなコードを含めます。

このコードは、リストを 1 つ以上のアイテムを含むエレメント・タイプとして定

```
<!ELEMENT List (Item)+>

<!ELEMENT Item (#PCDATA)>
```

義されています。（プラス記号はアイテムが 1 つ以上含まれることを示します。）また、アイテムは、テキストのみを含み、マークアップが全て除外されたエレメント・タイプとして定義しています（#PCDATA は Parsed Character DATA を示します）。XML は正式な記述言語で、プロセッサは XML を使用してまず DTD を

パースし、次にドキュメントを読み込み各エレメント・タイプの場所とその相互関係を判断します。これによりスタイルシート、ナビゲータ、ブラウザ、サーチ・エンジン、データベース、プリント・ルーチン、その他のアプリケーションを使用できるようになります。上記の DTD の場合、次のような形式のリストを作成できます。リストはそのままの形式で保存されます。

```
<List>
      <Item>Chocolate</Item>
       <Item>Music<Item>
       <Item>Surfing</Item>
</List>
```

DTD により、アプリケーションは特定のドキュメント・タイプで使用される名前と構造を事前に判断できます。ファイルの編集時に DTD を使用すると、特定タイプの全ドキュメントの構造と名前を一貫させることができます。

XML の利点

XML には次のように多くの利点があります。

- XML は既存の国際標準に基づいている。
- XML ドキュメントは拡張可能でタグの制限がないので容易に作成できる。
- XML は Unicode に基づくので国際化に対応している。
- XML は妥当性のチェックと編集時のコントロールをサポートしていて、あらゆる種類のデータをモデリングできる。
- XML ではリンクやナビゲーション支援機能を自動生成できる。
- XML はユーザによるデータ・アクセスを高速化する。
- XML は同じソースから印刷用とオンラインの両バージョンを作成できる。
- XML は Web サイトのシステム管理を容易にし、次世代のハイパーテキスト機能を備えている。

XML の問題点

XML の問題点は次の 2 点だけです。

- XML は大きなドキュメントの保存用に設計されていない。通常の XML ベースの処理プログラムはドキュメントの冒頭から読み込みを開始してドキュメントの終わりまで読み込みを続行する。
- XML ドキュメントの検索はデータベース・プログラムに比べて時間がかかる。

詳細情報

XML に関する詳細情報は、OASIS（Organization for the Advancement of Structured Information Standards）が運営する XML Cover Pages サイトを参照してください。

http://www.oasis-open.org/cover/sgml-xml.html

分散 HTTP セッションの使用

概要

分散 HTTP セッションはデータベースでの永続性を持つセッションで、これにより HTTP セッションのフェールオーバとリカバリが可能になります。分散 HTTP セッションはデフォルトのメモリ常駐セッションよりも処理に時間がかかります。

HTTP セッション・タイプの指定方法

使用する HTTP セッションのタイプ（分散タイプまたは非分散タイプ）は **cc.xml** ファイルで指定します。分散 HTTP セッションは各リクエストの後でディスクに保存されるので、Web サーバがクラッシュした場合に他のサーバがリクエストの処理を続行し、元のサーバが再起動した時点でその処理を完了できます。

分散セッションを有効にするには、**cc.xml** ファイルに次を追加します。

```
<configuration>
<web-app>
<session-config>
   <distributed>true</distributed>
   <res-ref-link>DavidDistSessions</res-ref-link>
   <tx-isolation-level>request</tx-isolation-level>
   <scrub-timeout>100000</scrub-timeout>
</session-config>
</web-app>

<resources>
<resource>
   <resource-name>DavidDistSessions</resource-name>
   <jndi-name>iona:cloudscape/demos</jndi-name>
</resource>
</resources>
</configuration>
```

<distributed> タグは分散セッションを有効または無効にします。デフォルト値は **false** です。

<res-ref-link> は、分散セッションに使用するデータベースを指定します。この値はリソース・タグを参照します。このタグを指定しない場合、システムはデフォルト設定で Cloudscape デモ・データベースを使用します。

データベースに必要なテーブルが無い場合には、プラグインがこれを自動的に作成します。テーブルの作成には次の SQL ステートメントを使用します。

```
CREATE TABLE HttpSession ( ID CHAR(12) PRIMARY KEY, CreationTime
   NUMERIC(20,0), LastAccessTime NUMERIC(20,0), MaxInactiveInt
   NUMERIC(11,0))

CREATE TABLE HttpSessionAttribute ( ID CHAR(12), ValueCounter
   NUMERIC(5,0), ValuePart CHAR(40), PRIMARY KEY (ID,
   ValueCounter))
```

<tx-isolation-level> の値には、**request** または **invocation** を指定します。**request** を指定すると、このリモート・セッションで使用するトランザクションはリクエストが完了するまで永続します。この場合、各 HTTP リクエストにつき一度データベースにセッションが保存されます。通常のアプリケーションではこれで十分ですが、セッションへの各呼出しの後で情報を保存することもできます。呼出しごとに保存するにはタグの値を **invocation** に設定します。デフォルト値は **request** です。

Web サーバがクラッシュした後で、一部の分散セッション・オブジェクトがデータベースに残ってしまうことがあります。**<scrub-timeout>** タグは、分散 HTTP セッション・システムが期限切れとなったオブジェクトをデータベースから自動削除するように指定します。オブジェクトの削除は分散セッションの起動時に一度行われます。サーバがクラッシュした後で処理が続行されなかったセッションを調べたい場合もあるので、このタイムアウト期間は通常のセッション・タイムアウトとは異なります。この値は分単位で指定され、デフォルトでは 31 日に設定されています。デフォルト設定を使用した場合、プラグインの起動時に、31 日以上経過している全セッションがデータベースから削除されます。

Web サービス

概要

iPortal Application Server には J2EE 用の Web サービスが統合されています。Web サービス・テクノロジーは e ビジネス・アプリケーションの配信チャネルを増やすことができる、Web ベースの B2B 環境にとって不可欠なコンポーネントです。iPortal Application Server の Web サービス・サポートにより、開発者は XML やマーシャル SOAP を開発しなくても、既存の J2EE アプリケーション用 Web サービスのブラウズ、実装、デプロイ、テスト、およびアクセスを簡単に行えます。

このようにリスクを減らしながら最先端のテクノロジーを提供することができるのは、iPortal Application Server だけの利点です。弊社では、Web サーバ・テクノロジーを iPortal Application Server 3.0 製品への追加機能として提供しています。Web サーバ・コンポーネントは、次のサイトからダウンロード可能です。

http://www.xmlbus.com/ipas3/

www.xmlbus.com サイトでは、Web サービス・テクノロジーに関するホワイトペーパー、データシート、チュートリアル、FAQ、用語集、プレス・リリースなどの役立つ情報が提供されているほか、Web サービス・テクノロジーを実際に使用してみたいというユーザのために、アップデート機能も用意されています。

パート 6

ユーザーズ・リファレンス

パート 6 は次のユーザーズ・リファレンスで構成されます。

第 17 章

コマンド・クイック・リファレンス

ここではコマンドラインで使用できる各ユーティリティのオプションについて説明します。

コマンド	オプション
java iportal.appadmin	java iportal.appadmin [-u -usage] [-s -silent] [-d, -dir **"project start directory"**]
java iportal.assembler	java iportal.assembler
java iportal.build	java iportal.build [-u,-usage] [-s,-silent] [-clean] [-d,-eardir **"eardir"**] [-compilerargs "**java compiler arguments**"] [-noear] [-novalidation] [-compress] [-j,-jsp **"jsp to be compiled"**] [-i,-index **"Jsp index used in development mode"**]
java iportal.central	java iportal.central
java iportal.client	java iportal.client [-u,-usage] [-s,-silent] [-m,-module **"appClient module"**] [-e, -ear **"ear archive"**] [-c,-config **"configuration file"**] [-o,-ORBname **"configuration scope selector"**] client-arguments... [The option -module must be specified]
java iportal.datasource	java iportal.datasource
java iportal.deploy	java iportal.deploy [-u,-usage] [-s,-silent] [-f,-propsfile "properties file"] [-d,-dir "directory containing properties file"] [-o,-ORBname "identifies the configuration scope"] [-n,-server "target server"] [-edit]
java iportal.earsco	java iportal.earsco [-u -usage] [-s -silent] [-d -dir **project-start-directory**]
java iportal.earscoify	java iportal.earscoify [-u -usage] [-s -silent] [-e,-ear **"ear-name"**] [-srcpath **"source-path"**]

java iportal.jms.startbr	java iportal.jms.startbr [-u -usage] [-s -silent] [-ini **"broker initialization file"**]
java iportal.jms.explorer	java iportal.jms.explorer
java iportal.server	java iportal.server java com.iona.j2ee.server.Main [-h] [-s] [-n "**server-name**"] [-d "**domain-name**"]
java iportal.undeploy	java iportal.undeploy [-u,-usage] [-s,-silent] [-f,-url **"the URL of the deployed application"**] [-a,-application **"application name"**] [-o,-ORBname **"identifies the configuration scope"**] [-n,-server **"target server"**] [The option **-ORBname** must be followed by an argument that identifies the configuration scope]

第 18 章

EARSCO リファレンス

EARSCO

EARSCO（Enterprise ARchive Source Code Organization）は、よりクリーンで管理しやすい J2EE アプリケーションの開発を目的としたディレクトリ構造に関する規則です。

iPortal Application Server にはエンタープライズ・アプリケーションの開発を支援するツールが多数用意されていますが、これらのツールは EARSCO 形式を認識します。

次に EARSCO ディレクトリ構造の概要を示します。

```
<project-root-directory>/
   |- <EAR1-directory>/
   |- <EAR2-directory>/
1          |- src/
2         |- etc/
             |- application.xml
3         |- <warfilename>.war/

             |- etc/

                |- web.xml

                    |- src/
                     |- lib/

                |- <libfile1>.jar
                |- <libfile2>.jar

                |- <userlib1>.jar/

                                |- src/

              |- <userlib2>.jar/
                                |- src/

                      |- web/
```

```
                |- <warfilename2>.war/
4           |- <ejbjarfilename1>.jar/
                    |- etc/
                |- ejb-jar.xml
5           |- src/
            |- <ejbjarfilename2>.jar/
6           |- <appclientjarfile1>.jar/
                    |- etc/
                        |- application-client.xml
                |- MANIFEST.MF
7                   |- src/
                |- <appclientjarfile2>.jar/
8           |- tmp/
```

1. 最上位の **src/** ディレクトリには、EAR ファイルの全ソース・ファイルを保存する。このディレクトリには、事前にビルドされた **ejb-jar**、**war**、または **application-client** アーカイブ（別のビルダ・ツールで生成されたアーカイブ）も保存される。

2. **etc/** ディレクトリには、エンタープライズ・アプリケーションの **application.xml** を保存する。

3. **<warfilename>.war/** ディレクトリには、Web アプリケーションの全ソース・ファイルを保存する。

   etc/ ディレクトリには、**web.xml** および、**WEB-INF/** ディレクトリ用の全てのファイルを保存します。**src/** ディレクトリには、全サーブレット・コードとそれが依存する Java コードを保存します。**lib/** ディレクトリには、サードパーティー提供によるユーザが開発した JAR（ROF、XML、ユーザ定義の JSP タグ・ライブラリなど）を保存します。ユーザ開発 JAR の Java ソースコードは **<userlib1>.jar/src/** ディレクトリに保存します。**web/** ディレクトリには、JSP を含む全ての静的なコンテンツを保存します。

4. **<ejbjarfilename1>.jar/** ディレクトリには、EJB の全ソース・ファイルを保存する。

5. **src/** ディレクトリには、EJB とこれが依存するオブジェクトの全ソースコードを保存する。

6. **<appclientjarfile1>.jar/** ディレクトリには、アプリケーション・クライアントの全ソース・ファイルを保存する。

7. **src/** ディレクトリには、アプリケーション・クライアントの全 Java ソースを保存する。

8. `tmp/` ディレクトリには、EAR ファイル用の全派生オブジェクト（例えばクラスなど）を保存する。

   派生オブジェクト・ディレクトリの構造仕様は、次の URL から入手可能です。

http://java.sun.com/j2se/1.3/docs/guide/jar

第 19 章

コマンド・リファレンス

概要 iPortal Application Server には、アプリケーションのビルドとデプロイおよびサーバの実行を行うためのコマンドライン・ユーティリティが用意されています。本章では各ユーティリティで使用できるコマンドライン・オプションについて説明します。

java iportal.appadmin

```
java iportal.appadmin [ -u -usage ] [ -s -silent]
[ -d, -dir "project start directory" ]
```

java iportal.appadmin コマンドにより、iPortal Application Server でエンタープライズ・アプリケーションの管理を行います。

オプション： 次のオプションを使用できます。

-u -usage	**iportal.build** コマンドで使用できるオプションを一覧表示する。
-d -dir **project start directory**	新しい EARSCO プロジェクトを作成するディレクトリの名前を指定する。
-s -silent	**iportal.appadmin** をサイレント・モードで実行する。

java iportal.assembler

```
java iportal.assembler
```

java iportal.assembler コマンドにより、iPortal Application Server に付属のグラフィカル・アプリケーション・ビルダを使用して JAR および WAR ファイルのデプロイメント・デスクリプタを更新します。

java iportal.build

```
java iportal.build [ -u,-usage ] [ -s,-silent ] [ -clean ]
   [-d,-eardir "eardir" ]  [ -compilerargs "java compiler
   arguments" ] [ -noear ] [ -novalidation ] [ -compress ]
[ -j,-jsp "jsp to be compiled" ]
[ -i,-index "Jsp index used in development mode" ]
```

java iportal.build コマンドを使用して、iPortal Application Server でデプロイできる EAR ファイルをビルドします。

オプション： 次のオプションを使用できます。

-u usage	**iportal.build** コマンドで使用できるオプションを一覧表示する。
-d **eardirectory**	使用する EAR ファイル・ディレクトリの名前を指定する。
-clean	以前に作成されたアーカイブを全てディレクトリから削除する。
-s -silent	**iportal.build** をサイレント・モードで実行する。

-compilerargs **javac_args**	指定した *javac_args* はコンパイル時に Java コンパイラに渡される。引数は二重引用符（"）で囲み、各オプションの前にハイフン（-）を付けずに指定する。例えば **javac** に **-debug** と **-verbose** を渡すには、次のように入力する。 **java iportal.build -compilerargs "debug verbose"** 事前設定された Java コンパイラ引数のリストを使用するには、**compilerargs.properties** ファイルを EARSCO ディレクトリのルートに作成し、このファイルに **compiler_args** 変数を含めその値として Java コンパイラで使用するプロパティを設定する。例えばアプリケーションを **-debug** と **-verbose** の各オプションを使用してコンパイルするには、**compilerargs.properties** ファイルに次のエントリを含める。 **compiler_args=debug verbose**
-noear	EARSCO アプリケーションをコンパイルするだけで、EAR ファイルは作成しない。
-novalidation	EJB の妥当性をチェックしない。
-compress	ビルダからの出力を圧縮します。大きなアーカイブ・ファイルを圧縮すると、iPortal Application Server がアプリケーションを実行する所要時間が長くなる。デフォルトではビルダは EAR を圧縮せずに作成。
-jsp **jsp_file**	*jsp_file* で指定された JSP ファイルのみをコンパイルする。
-index **jsp_index**	**-jsp** フラグを指定した場合のみ有効。渡された *jsp_index* 文字列を生成されたサーブレット・クラス名の前に追加する。例えば JSP 名が **main.jsp** で *jsp_index* の値が 001 の場合、**_001_ipas_generated_main.java** というサーブレット Java ファイルが生成される。

java iportal.central

```
java iportal.central
```

java iportal.central コマンドを使用して、開発者用ツールキットを起動します。iPortal Application Server に付属している全ツールへのショートカットとしても使用できます。

java iportal.client

```
java iportal.client [ -u,-usage ] [ -s,-silent ] [ -m,-module
   "appClient module" ] [ -e, -ear "ear archive" ] [ -c,-config
   "configuration file" ] [ -o,-ORBname "configuration scope
   selector" ] client-arguments...
[The option -m, -module must be specified]
```

java iportal.client コマンドを使用して、iPortal Application Server でアプリケーション・クライアントを実行します。

オプション： 次のオプションを使用できます。

-u usage	**iportal.client** コマンドで使用できるオプションを一覧表示する。
-e **ear archive**	使用する EAR ファイルの名前を指定する。
-c -config **configuration file**	使用するコンテナ・コンフィギュレーション・ファイルの名前を指定する。
-m -moduile [**appClient module**]	使用するモジュールの名前を指定。このオプションの指定は必須。
-s silent	**iportal.client** をサイレント・モードで実行する。
-o -OrRBname **configuration scope selector**	使用する Orb の名前を指定する。

java iportal.datasource

```
java iportal.datasource
```

java iportal.datasource コマンドを使用して、iPortal Application Server のデータソースを構成します。

java iportal.deploy

```
java iportal.deploy [ -u,-usage ] [ -s,-silent ] [ -f,-propsfile
   "properties file" ] [ -d,-dir "directory containing properties
   file" ] [ -o,-ORBname "identifies the configuration scope" ] [
   -n,-server "target server" ] [ -edit ]
```

java iportal.deploy コマンドを使用して、iPortal Application Server で EAR ファイルをデプロイします。

オプション： 次のオプションを使用できます。

-u usage	**iportal.deploy** コマンドで使用できるオプションを一覧表示する。
-f -propsfile **properties-file**	使用するプロパティ・ファイルの名前を指定。
-d -dir **property-file-dir**	使用するプロパティ・ファイル・ディレクトリの名前を指定する。
-o -ORBname **Orbname-config-scope**	使用する Orb の名前を指定する。
-s silent	**iportal.deploy** をサイレント・モードで実行。
-n **target-server**	ターゲット・サーバの名前を指定する。
-edit	deploy tool ウィザードが必ず起動されるようにする。

java iportal.earsco

```
java iportal.earsco [ -u -usage ] [ -s -silent ] [ -d -dir
   project-start-directory]
```

java iportal.earsco コマンドを使用して、エンタープライズ・アプリケーション用の EARSCO フレームワークを作成します。

オプション： 次のオプションを使用できます。

-u usage	**iportal.earsco** コマンドで使用できるオプションを一覧表示する。
-d -dir **project-start-directory**	プロジェクト・ディレクトリを指定する。
-s silent	**iportal.earsco** をサイレント・モードで実行。

java iportal.earscoify

```
java iportal.earscoify [ -u -usage ] [ -s -silent ]
[ -e,-ear "ear-name" ] [ -srcpath "source-path" ]
```

java iportal.earscoify コマンドを使用して、EARSCO フレームワークにファイルをインポートします。

オプション： 次のオプションを使用できます。

-u usage	**iportal.earscoify** コマンドで使用できるオプションを一覧表示する。
-e -ear **ear-name**	EARSCO フレームワークに変換する EAR の名前を指定する。
-srcpath **source-path**	Java ソース・ファイルを検索するパスを指定する。
-s silent	**iportal.earscoify** をサイレント・モードで実行する。

java iportal.jms.startbr

```
java iportal.jms.startbr [ -u -usage ] [ -s -silent ]
[ -ini "broker initialization file" ]
```

java iportal.jms.startbr コマンドを使用して、JMSProducer デモで使用する SonicMQ ブローカを起動します。

オプション： 次のオプションを使用できます。

-u usage	**iportal.earscoify** コマンドで使用できるオプションを一覧表示する。
-ini **broker init file**	起動時にブローカの初期化に使用するファイルを指定する。
-s silent	**iportal.earscoify** をサイレント・モードで実行する。

java iportal.jms.explorer

```
java iportal.jms.explorer
```

java iportal.jms.explorer コマンドを使用して、SonicMQ エクスプローラ・ウィンドウを起動します。

java iportal.server

```
java iportal.server [ -h ] [ -s ] [ -n "server-name" ] [ -d
   "domain-name" ]
```

java iportal.server コマンドは、iPortal Application Server のインスタンスを実行します。1 つのホストで複数のサーバを実行できますが、各サーバには固有のサーバ名が必要です。サーバ名について詳しくは、『Orbix 2000 管理者ガイド』を参照してください。このドキュメントについての情報は、**http://www.iona.co.jp** から入手できます。

オプション： 次のオプションを使用できます。

-d **domain-name**	サーバがコンフィギュレーションの詳細を取得するために使用する iPortal Application Server のコンフィギュレーション・ドメインを指定する。デプロイヤが値を指定しない場合、サーバはデフォルトのドメインを使用。 コンフィギュレーション・ドメインについて詳しくは『Orbix 2000 管理者ガイド』を参照してください。このドキュメントについての情報は、**http://www.iona.co.jp** から入手できます。
-h	コマンドの使用方法を表示する。
-n **server-name**	**iPAS.Server.RetailServer** などのサーバ名を指定する。サーバ名を指定しない場合、デフォルトのサーバ名 **iPAS.Server.Default** が使用される。デフォルト以外のサーバ名を使用するには、そのサーバが正しく構成されている必要がある。 サーバの構成について詳しくは『Orbix 2000 管理者ガイド』を参照してください。このドキュメントについての情報は、**http://www.iona.co.jp** から入手できます。
-s	**iportal.server** をサイレント・モードで実行する。

java iportal.undeploy

```
java iportal.undeploy  [ -u,-usage ] [ -s,-silent ] [ -f,-url
   "the URL of the deployed application" ] [ -a, -application
   "application name"] [ -o,-ORBname "identifies the
   configuration scope" ] [ -n,-server "target server" ]
[The option -ORBname must be followed by an argument that
   identifies the configuration scope]
```

java iportal.undeploy コマンドを使用して、iPortal Application Server からエンタープライズ・アプリケーションをアンデプロイします。

オプション： 次のオプションを使用できます。

-u usage	**iportal.undeploy** コマンドで使用できるオプションを一覧表示する。
-a **app_name**	アンデプロイするアプリケーションの表示名前を指定する。
-o **Orbname-cfg-scp**	使用する Orb の名前を指定する。

-f -url **url-name**	アンデプロイするアプリケーションの完全パスを指定する。
-n -server **target-server**	ターゲット・サーバの名前を指定する。
-s silent	**iportal.server** をサイレント・モードで実行する。

パート 7

付録

パート 7 は次の付録で構成されます。

付録 A

コンテナ・コンフィギュレーション DTD

各 EJB コンテナは、コンテナ・コンフィギュレーションと呼ばれる XML ドキュメントを使用して構成されています。特定のオペレーティング環境に合わせたコンテナの構成は、アプリケーション・デプロイヤが行います。

コンテナ・コンフィギュレーション DTD

次に、コンテナ・コンフィギュレーション・ドキュメントの DTD（ドキュメント・タイプ宣言）を示します。

```
<!-- Container Configuration V1.1 DTD
     Copyright (c) IONA Technologies, 1999, 2000, 2001
 -->
<!-- -->
<!ELEMENT access-control (#PCDATA)>
<!-- -->
<!ELEMENT application-client (application-client-name*,
   env-entry*, ejb-ref*, security-role-ref*, resource-ref*)>
<!-- -->
<!ELEMENT assembly-descriptor (security-control?, security-role*,
   method-permission*, container-transaction*)>
<!-- -->
```

```
<!ELEMENT application-client-name (#PCDATA)>
<!-- -->
<!ELEMENT authenticator-factory-classname (#PCDATA)>
<!-- -->
<!ELEMENT bean-map (bean-map-class, bean-map-constructor?,
   bean-map-argument?, bean-map-arg-type?, bean-map-arg-value)>
<!-- -->
<!ELEMENT bean-map-arg-type (#PCDATA)>
<!-- -->
<!ELEMENT bean-map-arg-value (#PCDATA)>
<!-- -->
<!ELEMENT bean-map-argument (bean-map-arg-type?,
   bean-map-arg-value)>
<!-- -->
<!ELEMENT bean-map-class (#PCDATA)>
<!-- -->
<!ELEMENT bean-map-constructor (bean-map-argument*)>
<!-- -->
<!ELEMENT bean-pool (bean-pool-low-watermark?,
   bean-pool-high-watermark?, bean-pool-preferred-ready?,
   bean-pool-restricted?, ready-map-max-size?)>
<!-- -->
<!ELEMENT bean-pool-high-watermark (#PCDATA)>
<!-- -->
<!ELEMENT bean-pool-low-watermark (#PCDATA)>
<!-- -->
<!ELEMENT bean-pool-preferred-ready (#PCDATA)>
<!-- -->
<!ELEMENT bean-pool-restricted (#PCDATA)>
<!--  -->
<!ELEMENT category (#PCDATA)>
<!-- -->
<!ELEMENT cmp-config (use-ipas-implementation |
   cmp-implementation-link)>
<!-- -->
<!ELEMENT cmp-datasource (res-ref-name, res-ref-link)>
<!-- -->
<!ELEMENT cmp-field-mapping ((field-name, column-name)+)>
<!-- -->
<!ELEMENT cmp-implementation (cmp-implementation-name,
   cmp-implementation-factory-classname, property)>
<!-- -->
<!ELEMENT cmp-implementation-factory-classname (#PCDATA)>
<!-- -->
```

```
<!ELEMENT cmp-implementation-link (#PCDATA)>
<!-- -->
<!ELEMENT cmp-implementation-name (#PCDATA)>
<!-- -->
<!ELEMENT cmp-implementations (cmp-implementation)>
<!-- -->
<!ELEMENT column-name (#PCDATA)>
<!-- -->
<!ELEMENT configuration (description?, display-name?, category?,
   enterprise-beans?, web-app?, application-client*,
   assembly-descriptor?, resources?, jndi-sources?, passivators?,
   cmp-implementations?)>
<!-- -->
<!ELEMENT container-transaction (description, method,
   trans-attribute)>
<!-- -->
<!ELEMENT context-root (#PCDATA)>
<!-- -->
<!ELEMENT default-timeouts (transaction-timeout?,
   passivation-timeout?, session-timeout?)>
<!-- -->
<!ELEMENT description (#PCDATA)>
<!-- -->
<!ELEMENT display-name (#PCDATA)>
<!-- -->
<!ELEMENT distributed (#PCDATA)>
<!-- -->
<!ELEMENT ejb (ejb-name, jndi-name, jndi-source-name, env-entry*,
   ejb-ref, security-role-ref*, resource-ref*, replica)>
<!-- -->
<!ELEMENT ejb-link (#PCDATA)>
<!-- -->
<!ELEMENT ejb-name (#PCDATA)>
<!-- -->
<!ELEMENT ejb-ref (description, ejb-ref-name, ejb-ref-type,
   home?, remote?, ejb-link?)>
<!-- -->
<!ELEMENT ejb-ref-name (#PCDATA)>
<!-- -->
<!ELEMENT ejb-ref-type (#PCDATA)>
<!-- -->
<!ELEMENT ejbStore-control (#PCDATA)>
<!-- -->
```

```
<!ELEMENT enterprise-beans ((session | entity)+,
   default-timeouts?)>
<!-- -->
<!ELEMENT entity (ejb-name, jndi-name, jndi-source-name,
   env-entry*, ejb-ref*, security-role-ref*, resource-ref*,
   cmp-datasource?, cmp-config?, ejbStore-control?,
   transaction-timeout?, bean-map?, bean-pool?)>
<!-- -->
<!ELEMENT env-entry (env-entry-name, env-entry-value)>
<!-- -->
<!ELEMENT env-entry-name (#PCDATA)>
<!-- -->
<!ELEMENT env-entry-value (#PCDATA)>
<!-- -->
<!ELEMENT field-name (#PCDATA)>
<!-- -->
<!ELEMENT home (#PCDATA)>
<!-- -->
<!ELEMENT jndi-name (#PCDATA)>
<!-- -->
<!ELEMENT jndi-source (jndi-source-name, property)>
<!-- -->
<!ELEMENT jndi-source-name (#PCDATA)>
<!-- -->
<!ELEMENT jndi-sources (jndi-source)>
<!-- -->
<!ELEMENT login-authenticator (authenticator-factory-classname,
   property)>
<!-- -->
<!ELEMENT mail-authenticator
   (mail-authenticator-factory-classname, property)>
<!-- -->
<!ELEMENT mail-authenticator-factory-classname (#PCDATA)>
<!-- -->
<!ELEMENT mail-resource (resource-name, property,
   mail-authenticator)>
<!-- -->
<!ELEMENT method (description?, ejb-name, method-intf,
   method-name, method-params?)>
<!-- -->
<!ELEMENT method-intf (#PCDATA)>
<!-- -->
<!ELEMENT method-name (#PCDATA)>
<!-- -->
```

```
<!ELEMENT method-param (#PCDATA)>
<!-- -->
<!ELEMENT method-params (method-param)>
<!-- -->
<!ELEMENT method-permission ((role-name | principal |
   principal-group)+, method+)>
<!-- -->
<!ELEMENT passivation-timeout (#PCDATA)>
<!-- -->
<!ELEMENT passivator (passivator-name, passivator-class,
   property)>
<!-- -->
<!ELEMENT passivator-class (#PCDATA)>
<!-- -->
<!ELEMENT passivator-name (#PCDATA)>
<!-- -->
<!ELEMENT passivators (passivator)>
<!-- -->
<!ELEMENT plugin-class (#PCDATA)>
<!-- -->
<!ELEMENT principal (#PCDATA)>
<!-- -->
<!ELEMENT principal-group (#PCDATA)>
<!-- -->
<!ELEMENT prop-name (#PCDATA)>
<!-- -->
<!ELEMENT prop-value (#PCDATA)>
<!-- -->
<!ELEMENT property (prop-name, prop-value)>
<!-- -->
<!ELEMENT ready-map-max-size (#PCDATA)>
<!-- -->
<!ELEMENT remote (#PCDATA)>
<!-- -->
<!ELEMENT replica (#PCDATA)>
<!-- -->
<!ELEMENT res-ref-link (#PCDATA)>
<!-- -->
<!ELEMENT res-ref-name (#PCDATA)>
<!-- -->
<!ELEMENT resource (resource-name, jndi-name, property*)>
<!-- -->
<!ELEMENT resource-name (#PCDATA)>
```

```
<!-- -->
<!ELEMENT resource-ref (res-ref-name, res-ref-link)>
<!-- -->
<!ELEMENT resources (resource*, mail-resource*)>
<!-- -->
<!ELEMENT role-link (#PCDATA)>
<!-- -->
<!ELEMENT role-mapping (#PCDATA)>
<!-- -->
<!ELEMENT role-name (#PCDATA)>
<!-- -->
<!ELEMENT scrub-timeout (#PCDATA)>
<!-- -->
<!ELEMENT security-control (access-control, role-mapping,
   security-view-from)>
<!-- -->
<!ELEMENT security-role (description?, role-name, (principal |
   principal-group)*)>
<!-- -->
<!ELEMENT security-role-ref (description?, role-name, role-link)>
<!-- -->
<!ELEMENT security-view-from (#PCDATA)>
<!-- -->
<!ELEMENT servlet (servlet-name)>
<!-- -->
<!ELEMENT servlet-name (#PCDATA)>
<!-- -->
<!ELEMENT session (ejb-name, jndi-name, jndi-source-name,
   env-entry*, ejb-ref?, security-role-ref*, resource-ref*,
   transaction-timeout?, passivation-timeout?, session-timeout?,
   bean-map?, bean-pool?, passivator-name?)>
<!-- -->
<!ELEMENT session-config ((distributed | plugin-class),
   res-ref-link?, tx-isolation-level?, scrub-timeout?)>
<!-- -->
<!ELEMENT session-timeout (#PCDATA)>
<!-- -->
<!ELEMENT table-name (#PCDATA)>
<!-- -->
<!ELEMENT trans-attribute (#PCDATA)>
<!-- -->
<!ELEMENT transaction-timeout (#PCDATA)>
<!-- -->
<!ELEMENT tx-isolation-level (#PCDATA)>
```

```
<!-- -->
<!ELEMENT use-ipas-implementation (table-name,
   cmp-field-mapping*)>
<!-- -->
<!ELEMENT web-app (display-name, context-root, jsp-source-dir?,
   jsp-output-dir?, jsp-earsco-dir?, jsp-compile-mode?,
   jsp-builder?, env-entry?, servlet*, session-config,
   security-role-ref?, resource-ref?, security-role?,
   login-authenticator?)>
<!-- -->
<!ELEMENT jsp-source-dir (#PCDATA)>
<!-- -->
<!ELEMENT jsp-output-dir (#PCDATA)>
<!-- -->
<!ELEMENT jsp-earsco-dir (#PCDATA)>
<!-- -->
<!ELEMENT jsp-compile-mode (#PCDATA)>
<!-- -->
<!ELEMENT jsp-builder (#PCDATA)>
```

付録 B

トラブルシューティング

本章では iPortal Application Server の既知の問題点とその解決方法について述べ、一般的なエラー・コードの詳細とその解決方法を説明します。

本章は、次のセクションで構成されます。

外部でデプロイされた EJB へのアクセス

概要 アプリケーションやクライアントが、コンテナ内でデプロイされた EJB に、必要に応じてアプリケーション・クライアント・コンテナを使わずにアクセスすることがあります。その場合、アプリケーションは **cc.xml** ファイルで定義されている JNDI 名を使用してネーム・サービスで必要な Bean のホーム・インターフェイスを検索する必要があります。

例 サンプルの EAR ファイルから **Account** Bean にアクセスする必要がある場合、次のコードを使用します。

```
import javax.naming.*;
..............
   System.setProperty("javax.rmi.CORBA.PortableRemoteObjectClass
   ","com.iona.corba.rmi.PortableRemoteObjectDelegateImpl");

   System.setProperty("org.omg.CORBA.ORBClass","com.iona.corba.a
   rt.artimpl.ORBImpl");

   System.setProperty("org.omg.CORBA.ORBSingletonClass","com.ion
   a.corba.art.artimpl.ORBSingleton");
    System.setProperty("ORBname", "iPAS");
    System.setProperty("ORBid", "iPAS");
    java.util.Hashtable environment=new java.util.Hashtable();

   environment.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
   "com.sun.jndi.cosnaming.CNCtxFactory");
            Context ctx = new InitialContext(environment);
   AccountHome home

   =(AccountHome)PortableRemoteObject.narrow(ctx.lookup("iona/ip
   as/examples/Account"),AccountHome.class);
```

この方法を使用するには、**CLASSPATH** に Account のホーム・クラスとリモート・クラスを含める必要があります。

Account のサンプルでは次の操作を行います。

1. デプロイされたサンプル EAR から **bankaccount.jar** ファイルを抽出する。

```
jar xvf examples.ear bankaccount.jar
```

2. **CLASSPATH** 環境変数に **bankaccount.jar** を含める。

```
set CLASSPATH=%CLASSPATH%;./bankaccount.jar
```

iPortal Application Server を使用した EJB での CORBA オブジェクト・リファレンスの取得

概要 CORBA ネーミング・サービスとの通信には CosNaming SPI が使用されます。必要なオブジェクト・リファレンスがネーミング・サービスの場合、EJB Bean に次のコードを追加してオブジェクトに参照します。

```
Properties orbProperties = new Properties();
                orbProperties.put("org.omg.CORBA.ORBClass",
                com.iona.corba.art.artimpl.ORBImpl");

   orbProperties.put("org.omg.CORBA.ORBSingletonClass",
                "com.iona.corba.art.artimpl.ORBSingleton");
   String[] orbArgs = new String[]
                {"-ORBname","iPAS"}; orb =
   org.omg.CORBA.ORB.init(orbArgs, orbProperties));
                try { objRef =
   orb.resolve_initial_references("NameService"); context =
   NamingContextExtHelper.narrow(objRef);
   org.omg.CosNaming.NameComponent[]
                cname = context.to_name("RequiredObject");
```

iPortal Application Server への接続試行時の ClassCastException 例外の送出

概要 iPortal Application Server に接続するクライアント・アプリケーションを実行すると、次のエラーが発生します。

```
Exception in thread "main" java.lang.ClassCastException at
com.sun.corba.se.internal.javax.rmi.PortableRemoteObject.
narrow(PortableRemoteObject.java:296)
```

原因 このエラーは、**PortableRemoteObject** クラスの JDK 1.3 実装が iPortal Application Server で提供されるバージョンと異なるために起こります。

解決方法 クライアントが IONA によるクラス実装を使用するには、クライアント・コードに次のコードを追加します。

```
System.setProperty("javax.rmi.CORBA.PortableRemoteObjectClass",
          "com.iona.corba.rmi.PortableRemoteObjectDelegateImpl");
```

あるいは、クライアント・アプリケーションの起動時に、次のコマンドライン・スイッチを使用することもできます。

```
java -Djavax.rmi.CORBA.PortableRemoteObjectClass
=com.iona.corba.rmi.PortableRemoteObjectDelegateImpl Client
```

クライアント側での UserTransaction 取得

概要 iPortal Application Server でクライアントが **UserTransaction** を取得するには、次のコードが必要です。

```
// obtain InitialContext the usual way...
Context ctx=new InitialContext(...);
UserTranscation utx=(UserTransaction)
   ctx.lookup("java:comp/UserTransaction");
```

クライアントが完全なスタンドアロン Java アプリケーションの場合は、J2EE 環境にアクセスしないので、**InitialContext** に次の環境プロパティを渡す必要があります。

```
Hashtable env2 = new Hashtable();
   env2.put("java.naming.factory.initial","com.iona.j2ee.ejb.jnd
   i.references.ReferencesContextFactory");
Context myCtx = new InitialContext(env2);
UserTransaction
   utx=(UserTransaction)myCtx.lookup("UserTransaction");
```

ClassNotFoundException と ClassCastException

問題 EJB が正しくデプロイされていても **ClassNotFoundException** と **ClassCastException** 例外が送出される。

解決方法 EJB アプリケーション・コードの入っているディレクトリが iPortal Application Server の CLASSPATH に含まれていないことを確認してください。

iPortal Application Server アプリケーションのデバッグ

概要

アプリケーションの開発ではデバッグを行う必要があります。iPortal Application Server アプリケーションなどの分散 Java アプリケーションでは、Sun が専用に開発した JPDA（Java Platform Debugger Architecture）を使用します。

デバッグの手順

デバッグには次の手順を行います。

1. アプリケーション・コードを適切な方法でコンパイルしてデバッグ情報を取得し、これらのクラスファイルを使用して EJB JAR をパッケージ化する。

2. 次のコマンドを使用して iPortal Application Server をデバッグ・モードで起動する。

```
java -Xdebug -Xnoagent -Djava.compiler=NONE
   -Xrunjdwp:transport=dt_socket,server=y,address=7777,
    suspend=n iportal.server
```

-Xdebug オプションにより、JVM による iPortal Application Server と EJB のデバッグ機能をオンにします。JVM の古いバージョンではこのパラメータを指定すると JVM がパスワードを提供し、このパスワードを使用して JVM インスタンスに接続してデバッグを行う必要がありました。新しいバージョンのデバッグ・アプリケーションではこのパスワード機能がサポートされません。この機能をオフにするには **-Xnoagent** オプションを指定します。

通常の場合、JVM はジャスト・イン・タイム（JIT）コンパイラを使用して Java のバイトコードをマシンに依存するネイティブ・コードに翻訳します。デバッグを行う場合は **-Djava.compiler=NONE** を指定してこの機能をオフにしてください。

JPDA の使用

JPDA は JVM の通常のデバッグ機能に接続し、リモート・デバッグを行うための JDWP（Java Debug Wire Protocol）という通信プロトコルを提供します。このプロトコルを構成するには **-Xrunjdwp** オプションとそのサブオプションを使用します。

上記のコマンドは、デバッグ対象のアプリケーションとデバッガ 間の通信に TCP/IP を使用し（**transport=dt_socket**）ソケット 7777 を使用して（**address=7777**）、アプリケーションの JVM がサーバ・ポートをオープンして（**server=y**）デバッガがこのポートにクライアントとして接続することを指定します。

アドレスを指定しない場合は JVM によりサーバ・ポートが割り当てられ、これがサーバの起動時に標準出力にプリントされます。

通常は JVM がデバッグ・モードで起動されると JVM が使用するリソースを初期化してアプリケーションの起動に必要なクラスをロードし、**main()** の呼出し直前で実行をサスペンドしてデバッガが実行コマンドを呼び出すのを待ちます。**suspend=n** サブオプションを指定すると、実行がサスペンドされません。JVM は通常どおりに起動されますが、サーバ・ポートはデバッグ用にオープンされます。

3. EJB をデプロイする。

4. デバッガ・アプリケーションを起動する。

Forte for Java、JBuilder、Visual Cafe などほとんどの統合型 Java 開発環境は JPDA をサポートしています。デバッグ・アプリケーションを使用する場合、デバッグ対象アプリケーションのソース・ファイルが入っているプロジェクトを定義する必要があります。プロジェクトを定義したら、次にデバッガを iPortal Application Server が稼動する JVM のインスタンスに接続します。これにはデバッグ・メニューの Connect... オプションを使用するのが普通です。

JVM のインスタンスに接続したら、breakpoint や watch の設定など通常のデバッグ作業を開始できます。breakpoint を設定した個所でクライアントがオペレーションを呼び出すと、デバッガ・アプリケーションがアクティブになりサーバ処理がサスペンドされます。

JSSE 1.0.2 による HTTPS の有効化

概要

iPortal Application Server の本リリースには、Sun Micro Systems による JSSE（Java Secure Socket Extension）パッケージの非商用リファレンス実装（RI）が付属しています。

開発者は Sun の JSSE RI を使用して、HTTP、Telnet、NNTP、FTP などのアプリケーション・プロトコルを実行しているサーバとクライアントの間で、TCP/IP を介してセキュアなデータ転送を行う純粋な Java ソリューションを提供することができます。

HTTPS の有効化

iPortal Application Server 1.3 で Sun の JSSE 1.0.2 を使用してデフォルトの HTTPS ポリシーを有効にするには、次の手順を行います。

1. iPortal Application Server に同梱されている JSSE の zip ファイル、**jsse102.zip** を見つけ、展開する。（このファイルは iPortal Application Server インストーラとは別の CD に収録。）**\jsse1.0.2\lib** ディレクトリに **jcert.jar**、**jnet.jar**、**jsse.jar** の各ファイルが抽象される。

2. これらのファイルを Java Runtime Environment の **NT\Program Files\JavaSoft\JRE\1.3\lib\ext** というディレクトリにコピーする。

3. NT の JRE（**\Program Files\JavaSoft\JRE\1.3\lib\security**）にある **java.security** ファイルを開き、次のセキュリティ・プロバイダが定義されていることを確認する。

```
#
# List of providers and their preference orders:
#
security.provider.1=sun.security.provider.Sun
security.provider.2=com.sun.net.ssl.internal.ssl.Provider
security.provider.3=com.sun.rsajca.Provider
```

4. 次に、iPortal Application Server がセキュアなクライアントに対して ID（身分証明）を提供し、セキュア・ソケットを開くことができるよう、証明書を作成する。Sun が提供する **keytool** というツールを使用して自己署名証明書を

生成できる。この証明書は後日、認証局（CA:Certificate Authority）が発行する証明書に変更することもできる。JDK1.3 に付属の **keytool** で自己署名証明書を生成するには、**太字で記述された**部分をコマンドラインで入力する。

```
1  keytool -genkey -keyalg RSA -alias server
2  Enter keystore password: changeit
3  What is your first and last name? [unknown]: Joe Bloggs
4  What is the name of your organizational unit? [unknown]:
     Engineering
5  What is the name of your organization? [unknown]: IONA
     Technologies
6  What is the name of your City or Locality? [unknown]: Dublin
7  What is the name of your State or Province? [unknown]: Ireland
8  What is the two letter country code for this unit? [unknown]: ie
9  Is <CN=Joe Bloggs, OU=Engineering, O=IONA Technologies, L=Dublin,
     ST=Ireland, C=ie> correct ? [no]: yes
10 Enter Key password for server (Return if same as keystore
     password): Return
```

証明書が生成され、NT の **\WINNT\Profiles\YourUserName\.keystore** ディレクトリに保存される。

5. iPortal Application Server のコンフィギュレーション・ファイル **\Program Files\IONA\ipas\1.3\etc\ipas-defaults.cfg** を編集し、**ipas::sever::default** スコープに次の行を追加する。

```
iPAS {
   Server {
     Default {
                ipas:httpsd:listen="true";
                ipas:httpsd:port="443";

# Port 443 is the default port for HTTPS.
# once you don't clash with a port already in use.
             }
          }
     }
```

6. 次のコマンドを入力して iPortal Application Server を起動する。

```
java
   -Djavax.net.ssl.keyStore="C:\WINNT\Profiles\YourUserName\.key
   store"
-Djavax.net.ssl.keyStorePassword=changeit
   com.iona.j2ee.server.Main
```

7. iPortal Application Server の例を通常どおりにビルドおよびデプロイしてから、ブラウザに次の URL を入力する。

https://localhost:443/SimpleServletRoot/SimpleServlet

ブラウザが iPortal Application Server からの証明書を受け入れるかどうかを尋ねるので、accept または yes を選択する。SSL コネクションを介してページが配信される。ブラウザ・ウィンドウの右下に表示されるロックのアイコンにより SSL コネクションが使用されていることがわかる。

上記の手順では、サーバ認証に際して iPortal Application Server のみが証明書を必要とすることを前提としています。

トラブルシューティング例 1

iPortal Application Server が起動すると次のメッセージが表示されることがあります。

```
    Setting servant and servant manager...done
    Starting HTTP listener on port 9000...done
    Starting HTTPS listener on port 443...done
    done
    Ready...
    javax.net.ssl.SSLException: No available certificate
  corresponds to the
    SSL cipher suites which are enabled at
  com.sun.net.ssl.internal.ssl.SSLServerSocketImpl.a([DashoPro-
  V1.2-120198])  at
  com.sun.net.ssl.internal.ssl.SSLServerSocketImpl.accept([Dash
  oPro-V1.2-120198]) at
  com.iona.j2ee.httpd.jws.SSLListener.run(SSLListener.java:47)
  javax.net.ssl.SSLException: No available certificate
  corresponds to the SSL cipher suites which are enabled at
  com.sun.net.ssl.internal.ssl.SSLServerSocketImpl.a([DashoPro-
  V1.2-120198]) at
  com.sun.net.ssl.internal.ssl.SSLServerSocketImpl.accept([Dash
  oPro-V1.2-120198])
```

これは、iPortal Application Server の起動時に必要な追加のパラメータがコマンドラインで指定されなかったか、指定された追加のパラメータが正しくないことを示します。さらにキー・ストア・ファイル **.keystore** が存在していてこれが正しく設定されていることも確認してください。

トラブルシューティング例 2

iPortal Application Server の起動中に例外が送出され、SSL サービスが使用不可である旨が報告されることがあります。例えば次のような例外が発生します。

```
Exception in thread "main" java.net.SocketException: no SSL
   Server
Sockets Exception in thread "main": SSL implementation not
   available
```

上記のセキュリティ・プロバイダ情報を使用して **java.security** ファイルで正しい変更を行ってください。

iPortal Application Server のエラー

概要

本セクションでは iPortal Application Server でよく発生するエラーについて、次の項目に分けて説明します。

- エラーの概要：問題の概要とナレッジ・ベースの関連記事
- 発生時点：エラーが発生する時点（ビルド時、デプロイ時、ランタイム）
- エラー・メッセージ：エラー発生時に画面に表示されるメッセージ
- 解決方法：エラーの解決方法

ここでは次のエラーについて解説しています。

1. Java コンパイラ・プロセスが開始しない
2. iPortal Application Server の起動時に組込み式サービスが失敗する
3. アンインストールで削除されないディレクトリやファイルがある
4. Bean クラスのビジネス・メソッドに一致するホームとリモートのペアが存在しない
5. **CORBA.UNKNOWN** 例外

6 `org.omg.CORBA.INITIALIZE` : ORB コンフィギュレーション・ファイルを開けることができない

7 iportal.server が再起動されない

8 クライアント実行時のエラー：例 1

9 クライアント実行時のエラー：例 2

10 クライアント実行時のエラー：例 3

11 クライアント実行時のエラー：例 4

12 アプリケーション・クライアント実行時のエラー：例 1

13 アプリケーション・クライアント実行時のエラー：例 2

14 iportal.server 起動エラー

エラーの概要 ビルド使用時に Java compiler process could not be started というエラー・メッセージが表示される。

発生時点 ビルド時

エラー・メッセージ

```
Error occurred when building module , <module_name>
   [IT_Builder:1701]
  The java compiler process could not be started
   [IT_Builder:1001]
  java.io.IOException: CreateProcess: javac -nowarn -sourcepath
<paths>
at java.lang.Win32Process.create(Native Method)
at java.lang.Win32Process.<init>(Unknown Source)
at java.lang.Runtime.execInternal(Native Method)
at java.lang.Runtime.exec(Unknown Source)
at java.lang.Runtime.exec(Unknown Source)
at com.iona.common.process.Spawn.startProcess(Spawn.java:64)
at com.iona.common.process.Spawn.startIfNecessary(Spawn.java:72)
at
   com.iona.common.process.Spawn.waitForCompletion(Spawn.java:10
   4)
at com.iona.j2ee.builder.JavaBuilder.build(JavaBuilder.java:131)
at com.iona.j2ee.builder.WarBuilder.build(WarBuilder.java:100)
```

```
at com.iona.j2ee.builder.EarBuilder.build(EarBuilder.java:152)
at com.iona.j2ee.builder.EarBuilder.main(EarBuilder.java:303)
at iportal.build.main(build.java:14)
```

解決方法 システム変数 **PATH** に **jdk1.3\bin** を設定する必要があります。

エラーの概要 iPortal Application Server の起動時に組込み式サービスが失敗する。

発生時点 iPortal Application Server の起動時

エラー・メッセージ

```
IONA iPortal Application Server - Version 3.0 Iteration 0910
   [20010514-1534]
Copyright (c) IONA Technologies plc., 1999-2001. All Rights
   Reserved.
Java 1.3.0_01 on Windows NT 4.0 (x86)
Starting Embedded Service: Locator...done
Could not start embedded service.  Using external service:
   Locator...Starting HTTP Listener on port 9000 ... done
org.omg.CORBA.COMM_FAILURE: Problem trying to create ATLI TCP
   Listener -  java.net.BindException: Address in use: JVM_Bind
   minor code: 1230246912  completed: No
   at
   com.iona.corba.atli.tcp.TCPListenerImpl.<init>(TCPListenerImp
   l.java:92)
   at
   com.iona.corba.atli.tcp.TCPTransportImpl.create_listener(TCPT
   ransportImpl.java:214)
   at
   com.iona.corba.atli_iop.tcp.TCPTransportAdapter.create_listen
   er(TCPTransportAdapter.java:100)
   at
   com.iona.corba.atli_iop.tcp.TCPServiceEndpointManager.create_
   atli_tcp_listener(TCPServiceEndpointManager.java:201)
   at
   com.iona.corba.atli_iop.tcp.TCPServiceEndpointManager.create_
   listener(TCPServiceEndpointManager.java:108)
   at
   com.iona.j2ee.web.dispatch.WebDispatchPerORBState.init_comple
   te(WebDispatchPerORBState.java:131)
```

```
    at
    com.iona.corba.art.artimpl.ORBDelegate.set_parameters(ORBDele
    gate.java:2077)
    at
    com.iona.corba.art.artimpl.ORBImpl.set_parameters(ORBImpl.jav
    a:125)
    at org.omg.CORBA.ORB.init(Unknown Source)
    at
    com.iona.j2ee.server.OrbServices.getORB(OrbServices.java:87)
    at
    com.iona.j2ee.server.IonaContextServerExtensions.populateCont
    ext(IonaContextServerExtensions.java:31)
    at
    com.iona.j2ee.server.IonaContextServerExtensions.bootContext(
    IonaContextServerExtensions.java:26)
    at com.iona.j2ee.server.Main.invoke(Main.java:200)
    at
    com.iona.j2ee.tools.CommandLineParser.service(CommandLinePars
    er.java:268)
    at
    com.iona.j2ee.tools.CommandLineParser.service(CommandLinePars
    er.java:198)
    at
    com.iona.j2ee.tools.CommandLineTool.main(CommandLineTool.java
    :23)
    at com.iona.j2ee.server.Main.main(Main.java:339)
    at iportal.server.main(server.java:12)
Error: org.omg.CORBA.COMM_FAILURE: Problem trying to create ATLI
    TCP Listener -  java.net.BindException: Address in use:
 JVM_Bind  minor code: 1230246912  completed: No
```

解決方法 このエラーは、最初のサーバと同じコンフィギュレーションを使用して別のサーバを起動した場合に発生します。iPortal Application Server の複数インスタンスを実行するには、各インスタンスで別々のポート番号を使用する必要があります。

エラーの概要 アンインストールで削除されないディレクトリやファイルがある。

発生時点 iPortal Application Server のアンインストール時

エラー・メッセージ

```
Unable to remove directory: d:\iona\ipas3\ipas3\lib
Unable to remove directory: d:\iona\ipas3\ipas3\etc\persistent
Unable to remove directory: d:\iona\ipas3\ipas3\etc\logs
Unable to remove directory: d:\iona\ipas3\ipas3\etc\domains
Unable to remove directory: d:\iona\ipas3\ipas3\etc
Unable to remove directory: d:\iona\ipas3\ipas3
Unable to remove directory: d:\iona\ipas3\ipas3
Unable to remove directory: d:\iona\ipas3\ipas3
```

解決方法 これはエラーではありません。アンインストールはインストール以降に変更されていないアイテムのみを削除します。インストール時に作成されたアイテムや、インストール以降に作成や変更されたアイテムは削除されません。

エラーの概要 Bean クラスのビジネス・メソッドに一致するホームとリモートのペアが存在しない。

エラー・メッセージ

```
No home/remote pairs match the bean class business methods.
   Exception in thread "main" java.lang.NullPointerException"
```

解決方法 このエラーは、EJB のソース・ファイルをビルドする際に Bean の実装クラスと Bean のリモート・インターフェイス・メソッド間に不一致がある場合に発生します。リモート・インターフェイスで定義されている各メソッドにつき、Bean の実装クラスにもこれに一致するメソッドが必要です。このメソッドのペアは、メソッド名、引数の数と型、および戻り値の型が一致している必要があります。

エラーの概要 `CORBA.UNKNOWN` 例外

発生時点 クライアントの起動時

エラー・メッセージ

```
Exception in thread "main"
   java.lang.reflect.InvocationTargetException:
   org.omg.CORBA.UNKNOWN: minor code: 1230242048 completed: No
at java.lang.Class.newInstance0(Native Method) at
   java.lang.Class.newInstance(Class.java:237) at
   com.iona.corba.art.streamables.SystemExceptionStreamable._rea
   d(SystemExceptionStreamable.java:96) at
   com.iona.corba.giop.GIOP_1_2_ReplyProcessorImpl.process_reply
   (GIOP_1_2_ReplyProcessorImpl.java:302) at
   com.iona.corba.giop.GIOPClientStreamRequestInterceptorImpl.in
   voke(GIOPClientStreamRequestInterceptorImpl.java:300)
at
   com.iona.j2ee.ejb.container.interceptor.TransactionClientRequ
   estInterceptor.invoke(TransactionClientRequestInterceptor.jav
   a:144)
at
   com.iona.j2ee.ejb.container.interceptor.SecurityClientRequest
   Interceptor.invoke(SecurityClientRequestInterceptor.java:85)
at
   com.iona.corba.art.binding.IORRequest.invoke(IORRequest.java:
   126) at
   org.omg.CosNaming._NamingContextStub.resolve(_NamingContextSt
   ub.java:156) at
   com.sun.jndi.cosnaming.CNCtx.callResolve(CNCtx.java:324) at
   com.sun.jndi.cosnaming.CNCtx.lookup(CNCtx.java:373) at
   com.sun.jndi.cosnaming.CNCtx.lookup(CNCtx.java:351) at
   javax.naming.InitialContext.lookup(InitialContext.java:350)
   at com.titan.Clients.Client_1.main(Client_1.java:24) at
   java.lang.reflect.Method.invoke(Native Method) at
   com.iona.j2ee.client.ApplicationClientRunner.run(ApplicationC
   lientRunner.java:149) at
   com.iona.j2ee.client.ApplicationClientRunner.main(Application
   ClientRunner.java:210) at iportal.client.main(client.java:11)
```

解決方法 `jndiContext.lookup()` に不正な名前が渡されたために起こります。

エラーの概要 `org.omg.CORBA.INITIALIZE` : ORB コンフィギュレーション・ファイルを開けることができない

発生時点 iPortal Application Server の起動時

エラー・メッセージ

```
Starting Embedded Service: Locator...18:55:24 4/12/2001
   [windsurfer/10.2.5.74] ( IT_CORE:5) E - exception while
   initializing orb: org.omg.CORBA.INITIALIZE: unable to open ORB
   configuration file
   "D:\ProgramFiles\IONA\etc\domains\iPAS.cfg" for
reading, check file existence (on CLASSPATH) and read permissions
   minor code:
1230242048 completed: No done Could not start embedded service.
   Using external service: Locator...18:55:25 4/12/2001
   [windsurfer/10.2.5.74] (IT_CORE:5) E - exception while
   initializing orb: org.omg.CORBA.INITIALIZE: unable to open ORB
   configuration file
   "D:\ProgramFiles\IONA\etc\domains\iPAS.cfg" for reading,
   check file existence on CLASSPATH) and read permissions minor
   code: 1230242048 completed: No org.omg.CORBA.INITIALIZE:
   unable to open ORB configuration file
   "D:\ProgramFiles\IONA\etc\domains\iPAS.cfg" for reading,
   check file existence (on CLASSPATH) and read permissions minor
   code: 1230242048 completed: No
at
   com.iona.corba.art.configuration.art_config_file.ConfigFile.r
   eadConfigSource(ConfigFile.java:183)
at
   com.iona.corba.art.configuration.art_config_file.ConfigFile.r
   eadConfigSource(ConfigFile.java:78)
at
   com.iona.corba.art.configuration.art_config_common.ConfigSour
   ce.nodifyStatement(ConfigSource.java:101)
at
   com.iona.corba.art.configuration.art_config_file.Configuratio
   nParser.Stmt(ConfigurationParser.java:158)
at
   com.iona.corba.art.configuration.art_config_file.Configuratio
   nParser.StmtList(ConfigurationParser.java:112)
at
   com.iona.corba.art.configuration.art_config_file.Configuratio
   nParser.ConfigurationFile(ConfigurationParser.java:96)
at
   com.iona.corba.art.configuration.art_config_file.ConfigFile.c
   reateParser(ConfigFile.java:213)
at
   com.iona.corba.art.configuration.art_config_file.ConfigFile.r
   eadConfigSource(ConfigFile.java:193)
```

```
at
   com.iona.corba.art.configuration.art_config_file.ConfigFile.r
   eadConfigSource(ConfigFile.java:78)
at
   com.iona.corba.art.configuration.art_config_file.ArtConfigFil
   ePlugInImpl.get_repository(ArtConfigFilePlugInImpl.java:160)
at
   com.iona.corba.art.artimpl.ORBDelegate.set_parameters(ORBDele
   gate.java:1550)
at
   com.iona.corba.art.artimpl.ORBImpl.set_parameters(ORBImpl.jav
   a:123) at org.omg.CORBA.ORB.init(Unknown Source)
 at com.iona.j2ee.server.OrbServices.getORB(OrbServices.java:83)
at
   com.iona.j2ee.server.IonaContextServerExtensions.populateCont
   ext(IonaContextServerExtensions.java:31)
at
   com.iona.j2ee.server.IonaContextServerExtensions.bootContext(
   IonaContextServerExtensions.java:26)
 at com.iona.j2ee.server.Main.invoke(Main.java:175)
at
   com.iona.j2ee.tools.CommandLineParser.service(CommandLinePars
   er.java:268)
at
   com.iona.j2ee.tools.CommandLineParser.service(CommandLinePars
   er.java:198)
at
   com.iona.j2ee.tools.CommandLineTool.main(CommandLineTool.java
   :23) at com.iona.j2ee.server.Main.main(Main.java:307)
at iportal.server.main(server.java:12)
Error: org.omg.CORBA.INITIALIZE: unable to open ORB configuration
   file "D:\ProgramFiles\IONA\etc\domains\iPAS.cfg" for reading,
   check file existence (on CLASSPATH) and read permissions minor
   code: 1230242048 completed: No
```

解決方法

このエラーは、ドライブのルート・ディレクトリに古い **default-domain.cfg** コンフィギュレーション・ファイルが保存されている場合に発生します。サーバは、**iPAS\etc\domains** ディレクトリにあるファイルではなく、このコンフィギュレーション・ファイルを使用します。この問題は、コンフィギュレーション・ファイルの検索アルゴリズムが原因で起こります。

これは、ユーザが iPortal Application Server の最新バージョンにアップグレードする前にドライブのルートに古いコンフィギュレーション・ファイルを保存した場合に発生します。

この問題を回避するには **default-domain.cfg** の名前を変更するか、ドライブのルート・ディレクトリから移動または削除します。

エラーの概要 iportal.server が再起動されない。

発生時点 ランタイム

エラー・メッセージ

```
Loading: SomeApp from repository...done
   java.lang.ClassCastException:
   com.sun.jndi.fscontext.FSContext$FSFile
at
   com.iona.j2ee.repository.Repository.hasConfiguration(Reposito
   ry.java:79) at
   com.iona.j2ee.server.Main.invoke(Main.java:285) at
   com.iona.j2ee.tools.CommandLineParser.service(CommandLinePars
   er.java:268) at
   com.iona.j2ee.tools.CommandLineParser.service(CommandLinePars
   er.java:198) at
   com.iona.j2ee.tools.CommandLineTool.main(CommandLineTool.java
   :23) at com.iona.j2ee.server.Main.main(Main.java:307) at
   iportal.server.main(server.java:12)
Error: java.lang.ClassCastException:
   com.sun.jndi.fscontext.FSContext$FSFile
```

解決方法 デプロイ済みのアプリケーション **SomeApp** にデプロイメントの問題があります。**[installDir]/ipas3/etc/repository/SomeApp** ディレクトリ全体を削除してから再試行してください。

エラーの概要 クライアント実行時のエラー：例 1

発生時点 ランタイム

エラー・メッセージ

```
Exception in thread "main"
   javax.naming.NoInitialContextException: Need to specify class
   name in environment or system property, or as an applet
   parameter, or i n an application resource file:
   java.naming.factory.initial
at javax.naming.spi.NamingManager.getInitialContext(Unknown
   Source) at
   javax.naming.InitialContext.getDefaultInitCtx(Unknown Source)
   at javax.naming.InitialContext.getURLOrDefaultInitCtx(Unknown
   Source) at javax.naming.InitialContext.lookup(Unknown Source)
   at
   examples.ejb.entity.cmp.person.PersonClient.<init>(PersonClie
   nt.java:16)
at
   examples.ejb.entity.cmp.person.PersonClient.main(PersonClient
   .java:38)
```

解決方法 このエラーは **iportal.client** を使わずにスタンドアロン・クライアントを実行した場合や、**jndi.naming.factory.initial** プロパティの設定が正しくない状態でクライアントを実行した場合に発生します。**java.naming.factory.initial** システム・プロパティを **com.sun.jndi.cosnaming.CNCtxFactory** か、これ以外の JNDI ソースを使用する場合は別のプロパティに設定します。

エラーの概要 クライアント実行時のエラー：例 2

発生時点 ランタイム

エラー・メッセージ

```
D:\ipash\ipas3\code_samples\simple\tmp\personclient.jar\classes>
   java
   -Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFac
   tory examples.ejb.entity.cmp. person.PersonClient
Exception in thread "main" javax.naming.CommunicationException:
   Cannot connect to ORB. Root exception is
   org.omg.CORBA.COMM_FAILURE: minor code: 1398079490 completed:
   No
at
   com.sun.corba.se.internal.iiop.IIOPConnection.writeLock(Unkno
   wn Source)
at com.sun.corba.se.internal.iiop.IIOPConnection.send(Unknown
   Source) at
   com.sun.corba.se.internal.iiop.IIOPOutputStream.invoke(Unknow
   n Source)
at
   com.sun.corba.se.internal.iiop.ClientRequestImpl.invoke(Unkno
   wn Source)
at com.sun.corba.se.internal.corba.ClientDelegate.invoke(Unknown
   Source)
at
   com.sun.corba.se.internal.corba.InitialNamingClient.resolve(U
   nknown Source)
at
   com.sun.corba.se.internal.corba.InitialNamingClient.cachedIni
   tialReferences(Unknown Source)
at
   com.sun.corba.se.internal.corba.InitialNamingClient.resolve_i
   nitial_references(Unknown Source)
at
   com.sun.corba.se.internal.corba.ORB.resolve_initial_reference
   s(Unknown Source)
```

```
at com.sun.jndi.cosnaming.CNCtx.setOrbAndRootContext(Unknown
   Source) at
   com.sun.jndi.cosnaming.CNCtx.initOrbAndRootContext(Unknown
   Source) at com.sun.jndi.cosnaming.CNCtx.<init>(Unknown
   Source) at
   com.sun.jndi.cosnaming.CNCtxFactory.getInitialContext(Unknown
   Source)
at javax.naming.spi.NamingManager.getInitialContext(Unknown
   Source) at
   javax.naming.InitialContext.getDefaultInitCtx(Unknown Source)
   at javax.naming.InitialContext.init(Unknown Source) at
   javax.naming.InitialContext.<init>(Unknown Source) at
   examples.ejb.entity.cmp.person.PersonClient.<init>(PersonClie
   nt.java:15)
at
   examples.ejb.entity.cmp.person.PersonClient.main(PersonClient
   .java:38)
```

解決方法

naming プロパティを設定してからクライアントを実行した後でこのスタック・トレースが出力された場合、IONA の ORB ではなく SUN の ORB が使用されています。その場合は追加のプロパティを設定する必要があります。

エラーの概要

クライアント実行時のエラー：例 3

発生時点

ランタイム

エラー・メッセージ

```
Exception in thread "main" java.lang.ClassCastException
at
   com.sun.corba.se.internal.javax.rmi.PortableRemoteObject.narr
   ow(PortableRemoteObject.java:296)
at
   javax.rmi.PortableRemoteObject.narrow(PortableRemoteObject.ja
   va:137) at
   examples.ejb.entity.cmp.person.PersonClient.<init>(PersonClie
   nt.java:17)
at
   examples.ejb.entity.cmp.person.PersonClient.main(PersonClient
   .java:39)
```

解決方法 **-Djavax.rmi.CORBA.PortableRemoteObjectClass=com.iona.corba.rmi.PortableRemoteObjectDelegateImpl** をコマンドラインのプロンプトで設定するか、コード中のシステム・プロパティとして設定します。

エラーの概要 クライアント実行時のエラー：例 4

発生時点 ランタイム

エラー・メッセージ

```
Exception in thread "main"
   java.lang.reflect.InvocationTargetException: javax.na
   ming.NameNotFoundException: GridBean
at
   com.iona.j2ee.ejb.environment.EnvironmentContext.lookup(Envir
   onmentContext.java:71)
at
   com.iona.j2ee.jndi.url.java.JavaContext.lookup(JavaContext.ja
   va:35) at
   com.iona.j2ee.jndi.url.java.JavaContext.lookup(JavaContext.ja
   va:51) at
   javax.naming.InitialContext.lookup(InitialContext.java:350)
   at
   examples.ejb.session.stateful.grid.GridClient.<init>(GridClie
   nt.java:28)
at
   examples.ejb.session.stateful.grid.GridClient.main(GridClient
   .java:44)
at java.lang.reflect.Method.invoke(Native Method) at
   com.iona.j2ee.client.ApplicationClientRunner.run(ApplicationC
   lientRunner.java:149)
at
   com.iona.j2ee.client.ApplicationClientRunner.main(Application
   ClientRunner.java:210)
at iportal.client.main(client.java:11)
```

解決方法 **lookup()** に使用した名前が間違っています。**application-client.xml** で **ejb-ref-name** をチェックし、さらに **cc.xml** で Bean の JNDI 名を確認してください。

エラーの概要 アプリケーション・クライアント実行時のエラー：例 1

発生時点 ランタイム

エラー・メッセージ

```
IONA iPortal Application Server - Version 3.0 Iteration 0910
   [20010419-1043]
Copyright (c) IONA Technologies plc., 1999-2001. All Rights
   Reserved.
Java 1.3.0_01 on Windows NT 4.0 (x86)

13:28:24 4/26/2001 [Silent/10.2.2.44] (IT_IIOP:8) W - failed to
   create socket for client (while attempting to connect()
to host 'localhost/127.0.0.1' on port 3074) - exception:
   com.iona.corba.IT_ATLI.IOError:
   IDL:iona.com/IT_ATLI/IOError:1.0
13:28:25 4/26/2001 [Silent/10.2.2.44] (IT_IIOP:8) W - failed to
   create socket for client (while attempting to connect()
to host 'localhost/127.0.0.1' on port 3074) - exception:
   com.iona.corba.IT_ATLI.IOError:
   IDL:iona.com/IT_ATLI/IOError:1.0
13:28:27 4/26/2001 [Silent/10.2.2.44] (IT_IIOP:8) W - failed to
   create socket for client (while attempting to connect()
to host 'localhost/127.0.0.1' on port 3074) - exception:
   com.iona.corba.IT_ATLI.IOError:
   IDL:iona.com/IT_ATLI/IOError:1.0
13:28:28 4/26/2001 [Silent/10.2.2.44] (IT_IIOP:8) W - failed to
   create socket for client (while attempting to connect()
to host 'localhost/127.0.0.1' on port 3074) - exception:
   com.iona.corba.IT_ATLI.IOError:
   IDL:iona.com/IT_ATLI/IOError:1.0
13:28:30 4/26/2001 [Silent/10.2.2.44] (IT_IIOP:8) W - failed to
   create socket for client (while attempting to connect()
to host 'localhost/127.0.0.1' on port 3074) - exception:
   com.iona.corba.IT_ATLI.IOError:
   IDL:iona.com/IT_ATLI/IOError:1.0
13:28:31 4/26/2001 [Silent/10.2.2.44] (IT_IIOP:8) W - failed to
   create socket for client (while attempting to connect()
to host 'localhost/127.0.0.1' on port 3074) - exception:
   com.iona.corba.IT_ATLI.IOError:
   IDL:iona.com/IT_ATLI/IOError:1.0
```

```
13:28:33 4/26/2001 [Silent/10.2.2.44] (IT_IIOP:8) W - failed to
   create socket for client (while attempting to connect()
to host 'localhost/127.0.0.1' on port 3074) - exception:
   com.iona.corba.IT_ATLI.IOError:
   IDL:iona.com/IT_ATLI/IOError:1.0
13:28:35 4/26/2001 [Silent/10.2.2.44] (IT_IIOP:8) W - failed to
   create socket for client (while attempting to connect()
to host 'localhost/127.0.0.1' on port 3074) - exception:
   com.iona.corba.IT_ATLI.IOError:
   IDL:iona.com/IT_ATLI/IOError:1.0
13:28:36 4/26/2001 [Silent/10.2.2.44] (IT_IIOP:8) W - failed to
   create socket for client (while attempting to connect()
to host 'localhost/127.0.0.1' on port 3074) - exception:
   com.iona.corba.IT_ATLI.IOError:
   IDL:iona.com/IT_ATLI/IOError:1.0
13:28:38 4/26/2001 [Silent/10.2.2.44] (IT_IIOP:8) W - failed to
   create socket for client (while attempting to connect()
to host 'localhost/127.0.0.1' on port 3074) - exception:
   com.iona.corba.IT_ATLI.IOError:
   IDL:iona.com/IT_ATLI/IOError:1.0
Exception in thread "main" org.omg.CORBA.TRANSIENT: cannot
   establish binding  minor code: 1230242562  completed: No
        at
   com.iona.corba.art.binding.ProxyBinding.get_binding(ProxyBind
   ing.java:488)
        at
   com.iona.corba.art.binding.IORRequest.invoke(IORRequest.java:
   102)
        at
   com.iona.corba.art.binding.IORProxy.is_a(IORProxy.java:285)
        at org.omg.CORBA.portable.ObjectImpl._is_a(Unknown
   Source)
        at org.omg.CosNaming.NamingContextHelper.narrow(Unknown
   Source)
        at
   com.sun.jndi.cosnaming.CNCtx.setOrbAndRootContext(Unknown
   Source)
        at
   com.sun.jndi.cosnaming.CNCtx.initOrbAndRootContext(Unknown
   Source)
        at com.sun.jndi.cosnaming.CNCtx.<init>(Unknown Source)
        at
   com.sun.jndi.cosnaming.CNCtxFactory.getInitialContext(Unknown
   Source)
        at
   javax.naming.spi.NamingManager.getInitialContext(Unknown
   Source)
```

```
        at javax.naming.InitialContext.getDefaultInitCtx(Unknown
Source)
        at javax.naming.InitialContext.init(Unknown Source)
        at javax.naming.InitialContext.<init>(Unknown Source)
        at
com.iona.j2ee.environment.EnvironmentBuilder.resolveEJBReferen
ces(EnvironmentBuilder.java:318)
        at
com.iona.j2ee.environment.EnvironmentBuilder.checkConfigurati
on(EnvironmentBuilder.java:79)
        at
com.iona.j2ee.environment.EnvironmentFactory.checkConfigurati
on(EnvironmentFactory.java:103)
        at
com.iona.j2ee.client.ApplicationClientRunner.create(Applicati
onClientRunner.java:117)
        at
com.iona.j2ee.client.ApplicationClientRunner.create(Applicati
onClientRunner.java:106)
        at
com.iona.j2ee.client.ApplicationClientRunner.create(Applicati
onClientRunner.java:91)
        at
com.iona.j2ee.client.ApplicationClientRunner.<init>(Applicati
onClientRunner.java:69)
        at
com.iona.j2ee.client.ApplicationClientRunner.main(Application
ClientRunner.java:204)
        at iportal.client.main(client.java:11)
```

解決方法 サーバが稼動していない状態で、アプリケーション・クライアントを実行しようとしています。サーバを起動し、アプリケーション・クライアント JAR が含まれているアプリケーションをデプロイしてから再試行してください。

エラーの概要 アプリケーション・クライアント実行時のエラー：例 2

発生時点 ランタイム

エラー・メッセージ

```
IONA iPortal Application Server - Version 3.0 Iteration 0910
   [20010514-1534]
Copyright (c) IONA Technologies plc., 1999-2001. All Rights
   Reserved.
Java 1.3.0_01 on Windows NT 4.0 (x86)

Exception in thread "main" The application-client jar does not
   appear to contain a manifest file. Without a Main-Class
   manifest entry it isn't possible to infer which class contains
   the main method to run [IT_Client:1002]

        at
   com.iona.j2ee.client.ApplicationClientRunner.create(Applicati
   onClientRunner.java:124)
        at
   com.iona.j2ee.client.ApplicationClientRunner.create(Applicati
   onClientRunner.java:108)
        at
   com.iona.j2ee.client.ApplicationClientRunner.create(Applicati
   onClientRunner.java:93)
        at
   com.iona.j2ee.client.ApplicationClientRunner.<init>(Applicati
   onClientRunner.java:71)
        at
   com.iona.j2ee.client.ApplicationClientRunner.main(Application
   ClientRunner.java:207)
        at iportal.client.main(client.java:11)
```

解決方法 **app-client.jar\etc** ディレクトリに **MANIFEST.MF** ファイルが存在していて、このファイル名が全て大文字であることを確認します。

エラーの概要 iportal.server 起動エラー

発生時点 ランタイム

エラー・メッセージ

```
Exception in thread "main" java.lang.NullPointerException
at sun.misc.Launcher$AppClassLoader.loadClass
   (Launcher.java:279)#
at java.lang.ClassLoader.loadClass(ClassLoader.java:290)
at java.lang.ClassLoader.loadClass(ClassLoader.java:253)
at
   com.iona.j2ee.client.ApplicationClientRunner.create(Applicati
   onClientRunner.java:99)
at
   com.iona.j2ee.client.ApplicationClientRunner.create(Applicati
   onClientRunner.java:74)
at
   com.iona.j2ee.client.ApplicationClientRunner.create(Applicati
   onClientRunner.java:54)
at
   com.iona.j2ee.client.ApplicationClientRunner.main(Application
   ClientRunner.java:176)
at iportal.client.main(client.java:11)
```

解決方法 この問題は、main のクライアント・クラスをポイントする MANIFEST.MF の最後にキャリッジ・リターンがないために発生します。JAR の仕様により、manifest ファイルの最後にはキャリッジ・リターンを挿入することが必須とされています。詳しくは
http://java.sun.com/j2se/1.3/docs/guide/jar/jar.html#Name-Value pairs and Sections
を参照してください。

索引

J

L

M

N

O

R

さ

し

す

せ

そ

て

と

の

ひ

ふ

ま

ら

り

る