開発者ガイド

iPlanet™ Application Server

Version 6.5

816-5275-01
2002 年 2 月

目 次

はじめに

『iPlanet Application Server 開発者ガイド』では、iPlanet Application Server 上での新しいオープン Java 標準モデル (Servlet、Enterprise JavaBeans (EJB)、JavaServer Pages (JSP)、および Java Database Connectivity (JDBC)) に準拠する Java™ 2 Platform Enterprise Edition (J2EE) アプリケーションの作成および実行方法について説明します。

このマニュアルは、WWW (World Wide Web) を使って幅広いユーザにクライアントサーバアプリケーションを広めようとする企業内の情報技術開発者を対象としています。このマニュアルでは、プログラミングの概念およびタスクを記述するほかに、サンプルコード、ヒント、および参照資料 (用語集など) を提供します。

この章には次のトピックがあります。

- マニュアルの使用法
- お読みになる前に
- このマニュアルの構成
- マニュアルの表記規則
- 関連情報

マニュアルの使用法

表 1 は、iPlanet Application Server の印刷版マニュアルおよびオンラインリリースノートに記述されているタスクや概念を示しています。特定のタスクを行う場合や特定の概念について調べる場合は、該当するマニュアルを参照してください。

印刷版マニュアルは、次のサイトから PDF または HTML 形式のオンラインファイルで入手することができます。

`http://docs.iplanet.com/docs/manuals/ias.html`

表 1　　iPlanet Application Server マニュアルの概要

情報の内容	参照するマニュアル	添付されている製品
ソフトウェアおよびマニュアルの最新情報	リリースノート	http://docs.iplanet.com から入手できます。
iPlanet Application Server およびそのコンポーネント (Web コネクタプラグイン、iPlanet Application Server Administrator) のインストールと、サンプルアプリケーションの設定	インストールガイド	iPlanet Application Server 6.5
次のタスクによる、オープン Java 標準モデル (Servlet、EJB、JSP、および JDBC) に準拠した iPlanet Application Server 6.5 アプリケーションの作成 • アプリケーションのプレゼンテーション層および実行層の作成 • EJB (Enterprise JavaBeans) コンポーネントへのビジネスロジックの個別部分およびエンティティの配置 • JDBC を使ったデータベースと通信 • 反復テスト、デバッグなどアプリケーションの調整機能を使用した、正確かつ高速に動作するアプリケーションの生成	開発者ガイド	iPlanet Application Server 6.5

表 1　iPlanet Application Server マニュアルの概要 (続き)

情報の内容	参照するマニュアル	添付されている製品
次のタスクを行うための、iPlanet Application Server Administrator Tool による 1 台または複数台のアプリケーションサーバの管理 • サーバの稼動状況の監視およびログ記録 • Netscape Application Server へのセキュリティの実装 • サーバリソースの高利用度の実現 • Web コネクタプラグインの設定 • データベース接続の管理 • トランザクションの管理 • 複数のサーバの設定 • 複数のサーバでのアプリケーションの管理 • サーバのロードバランス • 分散データ同期の管理 • 開発用 iPlanet Application Server の設定	管理者ガイド	iPlanet Application Server 6.5
iPlanet Application Server に付属するオンラインバンクアプリケーションの移行サンプルを含む、Netscape Application Server バージョン 2.1 から新しい iPlanet Application Server 6.5 プログラミングモデルへのアプリケーションの移行	移行ガイド	iPlanet Application Server 6.5
Java アプリケーションを作成する場合の Netscape Application Server クラスライブラリの共有クラスとインタフェース、およびそれらのメソッドの使用	Server Foundation Class Reference (Java)	iPlanet Application Server 6.5
C++ アプリケーションを作成する場合の Netscape Application Server クラスライブラリの共有クラスとインタフェース、およびそれらのメソッドの使用	Server Foundation Class Reference (C++)	別注文

お読みになる前に

このマニュアルでは、次の項目に精通していることを前提としています。

- J2EE 仕様
- HTML
- Java プログラミング
- EJB、JSP、および JDBC の仕様に定義されている Java API
- SQL などの構造化データベースクエリ言語
- リレーショナルデータベースの概念
- デバッグ、ソースコード制御を含むソフトウェア開発プロセス

このマニュアルの構成

第 1 部では、プログラム設計のための iPlanet Application Server 環境の概要について説明します。次のトピックがあります。

- 第 1 章「アプリケーションの開発」

第 2 部では、プレゼンテーションロジックおよびページ設計に関連するプログラミングタスクについて説明します。次のトピックがあります。

- 第 2 章「Servlet によるアプリケーションの制御」
- 第 3 章「JavaServer Pages によるアプリケーションページの表示」

第 3 部では、ビジネスロジックおよびデータアクセスに関連するプログラミングタスクについて説明します。次のトピックがあります。

- 第 4 章「Enterprise JavaBeans の紹介」
- 第 5 章「セッション EJB によるビジネスルール管理」
- 第 6 章「エンティティ EJB のビルド」
- 第 7 章「メッセージ駆動 Beans の使用」
- 第 8 章「EJB のトランザクション処理」
- 第 9 章「JDBC を使ったデータベースアクセス」
- 第 10 章「CORBA ベースクライアントの開発と配置」

第 4 部では、アプリケーションのすべての部分に影響を与える問題について説明します。次のトピックがあります。

- 第 11 章「配置のためのパッケージ化」
- 第 12 章「ユーザセッションの作成と管理」
- 第 13 章「安全なアプリケーションの作成」
- 第 14 章「iPlanet Application Server の機能の活用」

付録には、次の参照資料が含まれています。

- 付録 A「Java Message Service の使用法」
- 付録 B「実行時の注意事項」
- 付録 C「サンプル配置ファイル」

このマニュアルの最後には、用語集と索引があります。

マニュアルの表記規則

ファイルとディレクトリパスは Windows 形式で表示されます (ディレクトリ名を円記号で区切って表示)。Unix バージョンでは、ディレクトリパスについては Windows と同じですが、ディレクトリの区切り記号には円記号ではなくスラッシュが使われます。

このマニュアルでは次のように URL 形式を使います。

`http://`*`server`*`.`*`domain`*`/`*`path`*`/`*`file`*`.html`

これらの URL で、*server* はアプリケーションを実行するサーバ名で、*domain* はユーザのインターネットドメイン名、*path* はサーバ上のディレクトリの構造、および *file* は個別のファイル名を示します。URL の斜体文字の部分はプレースホルダです。

このマニュアルでは、フォントについて次の規則を採用しています。

- 固定ピッチフォントは、サンプルコード、コードの一覧表示、API および言語要素 (関数名、クラス名など)、ファイル名、パス名、ディレクトリ名、および HTML タグに使います。
- 斜体文字はコード変数に使います。
- 斜体文字は、マニュアル名、強調、変数およびプレースホルダ、およびリテラルに使われる文字にも使います。
- **太字**は、段落の先頭文字またはリテラルに使われる文字の強調に使います。

関連情報

公式の仕様書の URL ディレクトリには、*install_dir*/ias/docs/index.htm からアクセス可能です。また、次の書籍や Web サイトも参考にしてください。

Servlet および JSP を使ったプログラミング

『Java Servlet Programming』、Jason Hunter 著、O'Reilly 発行

『Java Threads, 2nd Edition』、Scott Oaks、Henry Wong 共著、O'Reilly 発行

Web サイト : http://www.servletcentral.com

EJB を使ったプログラミング

『Enterprise JavaBeans』、Richard Monson-Haefel 著、O'Reilly 発行

Web サイト : http://www.oreilly.com/catalog/entjbeans2/

JDBC を使ったプログラミング

『Database Programming with JDBC and Java』、George Reese 著、O'Reilly 発行

『JDBC Database Access With Java:A Tutorial and Annotated Reference (Java Series)』Graham Hamilton、Rick Cattell、Maydene Fisher 共著

第 1 章

アプリケーションの開発

この章では、アプリケーションの設計プロセスの概要を説明し、iPlanet Application Server の効果的な開発のためのガイドラインを提供します。

この章には次の節があります。

- アプリケーションの要件
- アプリケーションプログラミングモデルについて
- iPlanet アプリケーションの効果的なガイドライン

アプリケーションの要件

iPlanet Application Server アプリケーションの開発では、まず、アプリケーションの要件を明確にします。つまり、高速かつ安全であり、新規ユーザの追加条件について信頼性の高い処理が期待できる、広範囲に配置可能なアプリケーションとしての分散アプリケーションの開発を意味します。

iPlanet Application Server は、J2EE API だけでなく既存の高性能機能もサポートしており、これらのニーズを満たしています。たとえば、オンラインバンキングアプリケーションでは、次の条件を満たすことができます。

- パフォーマンスの向上
- スケーラビリティ
- 導入時間の短縮
- セキュリティ
- 特定機能の導入時間の短縮。口座振替、会計報告、オンライン取引、資格のあるお客様に対する特別サービスなど
- さまざまなタイプのエンドユーザの管理。個人、法人、社内ユーザ (銀行の従業員) など

- 社内レポート
- EIS (Enterprise Information System : 企業情報システム) の接続性。既存のデータベースに格納されている情報へのアクセスを提供

アプリケーションプログラミングモデルについて

分散アプリケーションモデルでは、さまざまなアプリケーション領域を個々の機能要素に集中させることができるため、パフォーマンスが向上します。たとえば、セキュリティ条件の設計がアプリケーションモデルの 1 つ以上のレイヤに影響を与えることがあります。

プレゼンテーションレイヤでは、ユーザの ID を確認する必要があります。これによって、アプリケーションは匿名ユーザ用のページと登録済みユーザ用のページを個別に表示できます。さらに、制限された機能にアクセスできない理由を説明し、メンバー登録を勧めるページを表示します。同様に、上得意の顧客は、一般の顧客がアクセスできない一部のページにアクセスできます。

ビジネスロジックレイヤでは、アプリケーションは登録されているユーザと照合してログインを認証し、特定のアプリケーション機能へのアクセス基準をユーザが満たしているかどうかを確認する必要があります。

データアクセスレイヤでは、アプリケーションはエンドユーザのカテゴリに基づいてデータベースへのアクセスを制限する必要があります。

プレゼンテーションレイヤ

プレゼンテーションレイヤでは、ユーザインタフェースがダイナミックに生成されます。アプリケーションには、次のアプリケーション要素が必要です。

- Servlet
- JSP
- HTML ページ
- クライアントサイド JavaScript 要素

Servlet

Servlet は、アプリケーションのプレゼンテーションロジックを処理します。Servlet は、ページ間移動ディスパッチャ、セッション管理、および簡単な入力確認を処理します。また、ビジネスロジックの要素を互いに関連付けます。

したがって、Servlet 開発者は、HTTP リクエスト、セキュリティ、国際化、および Web の状態のないこと (セッション、cookie、タイムアウトなど) に関するプログラミングの問題を理解する必要があります。iPlanet Application Server アプリケーションでは、Servlet を Java で記述する必要があります。Servlet は、JSP、EJB、および JDBC オブジェクトを呼び出します。したがって、Servlet 開発者はこれらのアプリケーション要素の開発者と緊密に協力して作業します。

JSP

JSP は、ほとんどのアプリケーション表示タスクを処理し、Servlet とともに動作することによって、アプリケーションの表示画面およびページ移動を定義します。JSP は、EJB および JDBC オブジェクトを呼び出します。EJB は通常、ビジネスロジックの機能をカプセル化して、計算や繰り返し要求されるほかのタスクを実行します。JDBC オブジェクトをは、データベースに接続し、クエリを実行し、クエリ結果を返します。

HTML ページ

適切に設計された HTML ページでは、次の効果が得られます。

- ブラウザが異なっていても外観が統一される
- 低速モデムによるコネクションでも HTML が効果的に読み込まれる
- Servlet または JSP がディスパッチされた、ダイナミックに生成されたページが表示される

クライアントサイド JavaScript

クライアントサイド JavaScript は、サーバにデータを送る前の簡単な入力確認を処理したり、ユーザインタフェースをより機能的なものにしたりします。クライアントサイド JavaScript 開発者は、Servlet 開発者および JSP 開発者と緊密に協力して作業します。

ビジネスロジックレイヤ

ビジネスロジックレイヤは通常、ビジネスルールおよびほかのビジネス機能をカプセル化した配置エンティティを含んでいます。

- セッション Beans
- エンティティ Beans
- メッセージ駆動 Beans

セッション Beans

セッション Beans で、ビジネスプロセスおよびルールロジックをカプセル化します。たとえば、セッション Beans で請求書の税金を計算できます。頻繁に変更が加えられる複雑なビジネスルールの場合 (新しい商慣習、政府規制など)、通常、アプリケーションはエンティティ Beans よりもセッション Beans を多く使うので、セッション Beans を絶えず改訂する必要があります。

セッション Beans は、ほかの EJB だけでなく、あらゆる JDBC インタフェースを呼び出します。セッション Beans に状態がない場合、アプリケーションは快適に動作します。たとえば、状態のあるセッション Beans で税金を計算すると仮定しましょう。アプリケーションは、その Bean のステート情報が保存されている特定のサーバにアクセスする必要があります。サーバがダウンすると、アプリケーション処理が遅れます。

詳細については、第 4 章「Enterprise JavaBeans の紹介」を参照してください。

エンティティ Beans

エンティティ Beans は、データベース行などのパーシスタントオブジェクトを表します。エンティティ Beans はあらゆる JDBC インタフェースを呼び出します。ただし、ほかの EJB を呼び出すことはありません。エンティティ Beans 開発者の役割は、組織のビジネスデータのオブジェクト指向ビューを設計することです。多くの場合、オブジェクト指向ビューを作成することは、エンティティ Beans へデータベーステーブルをマッピングすることです。たとえば、開発者は、Customer テーブル、Invoice テーブル、および Order テーブルを、対応する顧客オブジェクト、請求書オブジェクト、および注文書オブジェクトに変換できます。

エンティティ Beans 開発者は、セッション Beans 開発者および Servlet 開発者と協力して、アプリケーションでのパーシスタントビジネスデータへの高速でスケーラブルなアクセスを実現します。

詳細については、第 4 章「Enterprise JavaBeans の紹介」を参照してください。

メッセージ駆動 Beans

メッセージ駆動 Beans は、Enterprise JavaBeans によって提供されるフレームワークをサポートするという点で、セッション Beans とエンティティ Beans に似ています。ただし、メッセージ駆動 Beans は JMS (Java Messaging Service) のリスナでもあり、JMS メッセージの形式でクライアントから受信するリクエストに基づいてタスクを実行します。

セッション Beans やエンティティ Beans と異なり、メッセージ駆動 Beans はメッセージキューを非同期的に処理するので、サーバリソースを効率的に使用します。メッセージ駆動 Beans は多くのクライアントリクエストを同時に処理できるので、メッセージキューのボトルネックは生じません。

詳細については、第 7 章「メッセージ駆動 Beans の使用」を参照してください。

データアクセスレイヤ

データアクセスレイヤでは、カスタムコネクタは iPlanet Application Server の UIF (Unified Integration Framework) を使って、IBM の CICS などのレガシー EIS との通信を可能にします。

コネクタ開発者は C++ を使うことが多いため、JNI (Java Native Interfaces) などの Java への C++ の組み込みや UIF に関する事項も理解する必要があります。

UIF は、アプリケーションサーバから EIS データベースへの情報の送信を可能にする API フレームワークです。コネクタ開発者は次のシステムへのアクセスを統合します。

- CORBA アプリケーション
- メインフレームシステム
- サードパーティ製のセキュリティシステム

UIF の詳細については、『iPlanet Unified Integration Framework Developer's Guide』および次の URL に掲載されているリリースノートを参照してください。

`http://docs.iplanet.com/docs/manuals/ias.html#uifsp1`

iPlanet アプリケーションの効果的なガイドライン

この節では、iPlanet Application Server アプリケーションの設計および開発時に考慮する必要があるガイドラインについて簡単に説明します。詳細については、このマニュアルの後半の章を参照してください。

このガイドラインは次の目的に分類されます。

- Servlet および JSP を使ったデータ表示
- 再利用可能なアプリケーションコードの作成
- パフォーマンスの向上
- スケーラビリティの計画
- アプリケーションのモジュール化

Servlet および JSP を使ったデータ表示

Servlet はプレゼンテーションロジックによく使われ、ユーザ入力およびデータ表示のセントラルディスパッチャとして機能します。JSP は、プレゼンテーションレイアウトをダイナミックに生成します。Servlet および JSP を使うと、条件に応じてさまざまなページを生成できます。

ページのレイアウトが主な特徴であり、ページを作成するための処理がほとんどないか、またはまったくない場合は、対話に JSP のみを使った方が簡単な場合があります。

たとえば、オンラインブックストアアプリケーションでは、ユーザを認証すると、共通画面である「入口」というフロントページを表示します。このページでユーザは、本の検索や選択した本の購入などのタスクを選択できます。この「入口」自体で処理が行われることはほとんどないので、JSP として単独で実行できます。

JSP と Servlet を 1 枚のコインの表と裏と考えてください。JSP のタスクは Servlet でも実行でき、その逆の操作も可能です。しかし、JSP のタスクは JSP に最適化されており、Servlet のタスクは Servlet に最適化されています。Servlet の利点は処理能力と適合性にあり、また Java ファイルであるため、ファイルを作成するときに統合開発環境を利用できます。ただし、Java ファイルから HTML の出力を実行すると、扱いにくい `println` ステートメントが多量に発生します。JSP は HTML ファイルなので、計算や処理タスクの実行には不向きですが、HTML エディタで編集できるので、レイアウト作業に優れています。

JSP の詳細については、第 3 章「JavaServer Pages によるアプリケーションページの表示」を参照してください。

再利用可能なアプリケーションコードの作成

オブジェクト指向の最適な設計方針が使われている場合を除き、再利用性を最大限にするためには、アプリケーション開発時に次の点を考慮する必要があります。

- コードツリーを移動した場合でもリンクが無効にならないように、相対パスと相対 URL を使います。
- JSP 内での Java の使用を最小限に抑え、Servlet およびヘルパークラス内で Java を使います。JSP 設計者は、Java に関する高度な知識がなくても JSP を修正できます。
- データソース名、テーブル名、カラム名、JNDI オブジェクト名、ほかのアプリケーションプロパティ名などのハードコードされた文字列の保存には、プロパティファイルまたはグローバルクラスを使います。
- ドメイン固有のビジネスルールや入力確認などの頻繁に変更されるビジネスルールの保存には、Servlet や JSP でなく、セッション Beans を使います。

- パーシスタントオブジェクトにはエンティティ Beans を使います。エンティティ Beans を使うと、ユーザ 1 人あたり複数の Bean を管理できます。
- 柔軟性を最大限に高めるために、Java クラスでなく Java インタフェースを使います。
- レガシーデータにアクセスするには、UIF ベースのコネクタを使います。

パフォーマンスの向上

アプリケーションを iPlanet Application Server に配置した場合にパフォーマンスを向上させるいくつかのヒントがあります。

- 多くの場合、Servlet および JSP は iPlanet Web Server にではなく iPlanet Application Server に配置します。アプリケーションにトランザクションが多い場合、セッションデータを保持するためのフェールオーバーサポートが必要な場合、またはレガシーデータにアクセスする場合には、iPlanet Application Server が最適です。アプリケーションの大部分が状態なし、読み取り専用、およびトランザクションなしの場合は、iPlanet Web Server が適しています。
- エンティティ Beans と状態のないセッション Beans を使います。同じ場所に配置できるように設計して、時間のかかるリモートプロシージャコールを回避します。
- アプリケーションを配置したら、必要な EJB および JSP がレプリケートされ、Servlet を呼び出したプロセスと同じプロセスに読み込むことができることを確認します。
- 複数行にわたる情報を返す場合は、可能であれば JDBC の `RowSet` オブジェクトを使います。複雑なデータをデータベースにコミットする場合は、JDBC バッチ更新やダイレクト SQL オペレーションなどの効率的なデータベース機能を使います。
- Java のパフォーマンスを向上させるための一般的なプログラミングガイドラインに従います。

スケーラビリティの計画

顧客の要求の増加に合わせて簡単にスケーリングできるアプリケーションを作成するには、次の点に注意します。

- 情報分散用に設定した `HttpSession` オブジェクトにスケーリング情報や直列化情報を保存するようにアプリケーションを開発します。
- グローバル変数の使用を避けます。
- マルチマシンサーバファーム環境で動作するようにアプリケーションを設計します。

アプリケーションのモジュール化

J2EE アプリケーションをモジュール化するときは、次の主要な要素を考慮する必要があります。

- 機能の分離
- 再利用可能なコード
- パッケージ済みコンポーネント
- 一意の名前
- 共有フレームワーククラス
- セッションとセキュリティの問題

5 つのパッケージサンプル (A ～ E) は、ここで説明するパッケージ化の概念の例です。これらのサンプルの概要については、次のサイトを参照してください。

```
http://developer.iplanet.com/appserver/samples/pkging/docs/index.html
```

アプリケーションのパッケージ化の詳細については、第 11 章「配置のためのパッケージ化」を参照してください。

機能の分離

各モジュールには、特定の処理だけを割り当てる必要があります。たとえば給与システムでは、401k アカウントにアクセスする Enterprise JavaBeans と給与データベースにアクセスする Enterprise JavaBeans とは、分離する必要があります。タスクを機能別に分離することによって、ビジネスロジックは物理的に 2 つの Bean に分離されます。異なる開発チームがこれらの Bean を作成する場合、各チームは EJB JAR パッケージを個別に開発する必要があります。

シナリオ 1

UI 開発チームが、2 つの Bean 開発チームと協力して作業するとします。この場合、UI 開発チームは、Servlet、JSP、およびスタティックファイルを 1 つの WAR ファイルにパッケージ化する必要があります。次のようにします。

給与システム EAR ファイル = 給与 EJB jar
+ 401k ejb JAR
+ UI チームの 1 つの共通 war

EAR ファイル内でこのように機能を分離しても、各モジュール間で対話することができます。これらの Bean は、個別の EJB JAR ファイルから相互にビジネスメソッドを呼び出すことができます。このパッケージ化の例は、サンプル A にあります。

シナリオ 2

Bean の各開発チームに、UI 開発チームが個別に存在するとします。この場合、各 Web 開発チームは、Servlet、JSP、およびスタティックファイルを個別の WAR ファイルにパッケージ化する必要があります。次のようにします。

給与システム EAR ファイル = 給与 EJB jar
+ 401k ejb JAR
+ 1 つの給与 UI チームの war + 1 つの 401k UI チームの war

このように設定すると、各 WAR ファイルのコンポーネントからほかの WAR ファイルのコンポーネントにアクセスできます。このパッケージ化の例は、サンプル B にあります。

シナリオ 3

各モジュールが共有ライブラリから関数にアクセスするとします。いくつかのモジュールが共有ライブラリからメソッドにアクセスする場合は、共有ライブラリを EAR ファイルの特定のモジュールに追加する必要があります。この場合の例については、サンプル C を参照してください。

パッケージ化の式

モジュールとアプリケーションをパッケージ化するときの式は、通常、次のようになります。

表 1-1 パッケージ化の式

開発グループのタイプ	グループ内のチーム	モジュール化の式
小規模なワークグループ	1 つの Web 開発チーム + 1 つの ejb 開発チーム	1 つの EAR = 1 つの ejb + 1 つの war
企業のワークグループ	2 つの ejb 開発チーム + 1 つの Web 開発チーム + 1 つのコンポーネント	1 つの EAR = 2 つの ejb + 1 つの war + 1 つのスタンドアロンモジュール

再利用可能なコード

コンポーネントの再利用は、アプリケーションではなくモジュールをパッケージ化および配置するときに、特に有効です。ある開発者チームが開発したコードが、複数のアプリケーション (EAR ファイルが異なる場合) からアクセスされる再利用可能なコンポーネントである場合、次のコマンドを使ってそのコードをモジュールとしてパッケージ化および登録する必要があります。

```
iasdeploy deploymodule module_name
```

パッケージ済みコンポーネント

アプリケーションを最初から作成したくない場合は、パッケージ済みコンポーネントを利用できます。J2EE コンポーネントの主なベンダーは、サービスを全体的に管理するためのモジュールで構成される、さまざまなパッケージ済みコンポーネントを提供しています。パッケージ済みコンポーネントでは、アプリケーションに必要な標準コンポーネントの最大 60% を提供することを目標としています。iPlanet Application Server では、パッケージ済みの標準コンポーネントで構成したアプリケーションを簡単にパッケージ化できます。

一意の名前

各モジュール、アプリケーション、および EJB に対して一意の名前を割り当てることは、重要な要素です。いくつかの命名規則を作成すれば、2 つのエンティティに同じ名前を割り当てることがなくなります。たとえば、すべてのモジュールに確実に一意名を割り当てるには、アプリケーション名をモジュール名の接頭辞にします。この命名規則に従えば、アプリケーション `pkging.ear` 内の WAR モジュールの最適な名前は、`pkgingWar.war` になります。

EJB の JNDI 検索名も一意でなければなりません。この場合も、一貫した命名規則を作成すると有効です。たとえば、EJB 名にアプリケーション名とモジュール名を追加すると、確実に一意な名前になります。この場合、モジュール `pkgingEJB.jar` 内の EJB の JNDI 名は、アプリケーション `pkging.ear` にパッケージ化されているため、`mycompany.pkging.pkgingEJB.MyEJB` になります。

共有フレームワーククラス

LDAP SDK、Cocobase CMP ランタイムなどのアプリケーションでは、1 つのモジュール化ライブラリにアクセスする必要があります。この場合、次の 2 つの理由から、J2EE アプリケーションごとにライブラリを含めることはお勧めできません。

- **ライブラリのサイズ** : ほとんどのフレームワークライブラリはサイズが大きいため、アプリケーションに含めると、パッケージ化したときのアプリケーションサイズが大きくなります。
- **複数のインスタンス** : クラスローダが個別に各アプリケーションを読み込むため、実行時に複数のフレームワーククラスのコピーが生成されます。

このライブラリを iPlanet Application Server 実行時環境に含める方法の 1 つとして、このライブラリをシステムクラスパス (NT の場合は *install_dir*`/ias/env` ディレクトリにある `iasenv.ksh` スクリプト内と iPlanet Application Server レジストリ内) に追加する方法があります。この場合、フレームワークがシステムクラスローダによって読み込まれます。システムクラスローダについては、401 ページの「クラスローダの階層」を参照してください。

セッションとセキュリティの問題

セッションを共有する必要がある場合は、セッションにアクセスするすべてのコンポーネントを同じアプリケーションに含める必要があります。アプリケーションの境界にまたがったセッションの共有は、iPlanet Application Server ではサポートされていないうえ、J2EE 仕様書に違反しています。

HTTP セッションを EAR ファイル内の 2 つの WAR ファイル間で共有する場合は、そのセッションが分散することを配置記述子に記述する必要があります。この例はサンプル B にあります。

クラス、EJB などのリソースに対する認証されていない実行時アクセスは、禁止してください。モジュールは、そのモジュールに含まれるほかのリソースに対してアクセス権を持つクラスだけで構成してください。また、重要なタスクには、J2EE 標準の宣言セキュリティ (第 13 章「安全なアプリケーションの作成」を参照) を使用する必要があります。

第 2 章

Servlet によるアプリケーションの制御

この章では、iPlanet™ Application Server 上で行われるアプリケーション対話の制御に有効な Servlet (標準の Servlet を含む) の作成方法について説明します。また、標準を拡張する iPlanet Application Server の機能について説明します。

この章には次の節があります。

- Servlet について
- サーバエンジンについて
- Servlet の設計
- Servlet の作成
- Servlet の起動

Servlet について

Servlet はアプレットと同様に再利用可能な Java アプリケーションです。ただし、Servlet は Web ブラウザ上ではなく、アプリケーションサーバまたは Web サーバ上で動作します。

iPlanet Application Server でサポートされる Servlet は、Java Servlet 仕様書バージョン 2.2 に基づいています。この仕様書は *install_dir*`/ias/docs/index.htm` からアクセス可能です。*install_dir* は、iPlanet Application Server がインストールされている場所です。

Servlet はアプリケーションのプレゼンテーションロジックに使われます。Servlet はフォーム入力処理、EJB にカプセル化されているビジネスロジックのコンポーネント起動、JSP を使った Web ページ出力のフォーマットなど、アプリケーションのセントラルディスパッチャとして機能します。Servlet はユーザからのリクエストに応じてコンテンツを生成し、ユーザ対話から次のユーザ対話に続くアプリケーションフローを制御します。

Servlet の基本的な特徴は次のとおりです。

- Servlet は、iPlanet Application Server の Servlet エンジンによって実行時に作成され、管理される
- Servlet は、`request` オブジェクトにカプセル化されている入力データを処理する
- Servlet は、`response` オブジェクトにカプセル化されているクエリデータに応答する
- Servlet は EJB を呼び出してビジネスロジック機能を実行する
- Servlet は JSP を呼び出してページレイアウト機能を実行する
- Servlet は拡張可能であり、iPlanet Application Server とともに提供される API を使って機能を追加する
- Servlet は対話間のユーザセッション情報のパーシスタンスを提供する
- Servlet は、アプリケーションの一部にすることも、複数のアプリケーションに使えるように分離してアプリケーションサーバ上に置くこともできる
- Servlet はサーバの動作中にダイナミックに再読み込みされる
- Servlet は、URL でアドレスを指定できる。アプリケーションのページ上のボタンが Servlet を指している場合がある
- Servlet はほかの Servlet を呼び出すこともできる

iPlanet Application Server のいくつかの API 機能によって、アプリケーションは iPlanet の特定の機能をプログラムに基づいて利用できます。詳細については、54 ページの「iPlanet Application Server のオプション機能へのアクセス」を参照してください。

Servlet のデータフロー

ユーザが「Submit」ボタンをクリックすると、表示ページに入力された情報が Servlet に送信されます。Servlet は受信したデータを処理し、コンテンツを作成してレスポンスを構成します。このときビジネスロジックコンポーネント (EJB) を使う場合があります。コンテンツが作成されると、Servlet は通常、そのコンテンツを JSP に転送してレスポンスページを作成します。レスポンスはクライアントに返送され、次のユーザの対話について設定します。

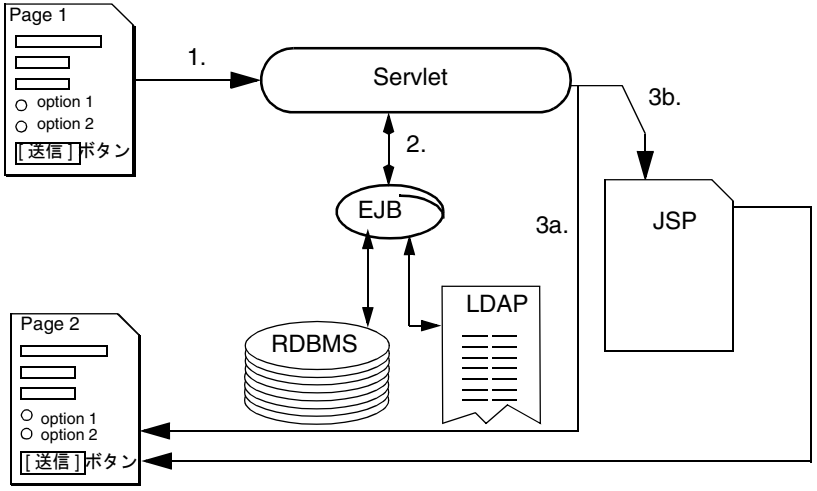
次の図は、Servlet との間でやり取りされる情報のフローを示しています。

1. Servlet がクライアントのリクエストを処理する
2. Servlet がコンテンツを作成する
3. Servlet がレスポンスを作成し、

a. それをクライアントに返送する

   または

b. そのタスクを JSP にディスパッチする



Servlet はほかのリクエストを処理できるようにメモリに残ります。

Servlet の種類

Servlet には主に次の 2 種類があります。

- 汎用 Servlet
  - `javax.servlet.GenericServlet` を拡張します。
  - プロトコルに依存しません。継承 HTTP のサポートやほかの転送プロトコルは含まれていません。
- HTTP Servlet
  - `javax.servlet.HttpServlet` を拡張します。
  - 組み込み HTTP プロトコルがサポートされているので、iPlanet Application Server 環境では便利。

これらの Servlet はそれぞれ、コンストラクタメソッドである `init()` とデストラクタメソッドである `destroy()` を実装し、リソースを初期化したり、リソースの割り当てを解除したりできます。

すべての Servlet が、Servlet に対するリクエストを処理する service() メソッドを実装する必要があります。汎用 Servlet の場合は、サービスメソッドをオーバーライドしてリクエスト処理用のルーチンを用意します。HTTP Servlet には、使用中の HTTP 転送メソッドに基づいて Servlet 内の別のメソッドに自動的にリクエストを転送するサービスメソッドがあります。したがって、HTTP Servlet の場合は doPost() をオーバーライドして POST リクエストを処理し、doGet() をオーバーライドして GET リクエストを処理します。

サーバエンジンについて

Servlet は iPlanet Application Server の Java サーバプロセスに存在し、Servlet エンジンによって管理されます。この Servlet エンジンは、Servlet のすべてのメタファンクションを処理する内部オブジェクトです。このメタファンクションには、インスタンス化、初期化、破棄、ほかのコンポーネントからのアクセス、設定管理などがあります。

Servlet のインスタンス化と削除

Servlet エンジンは、Servlet をインスタンス化したあと、その init() メソッドを実行して必要な初期化を行います。カウンタの初期化など、Servlet の生存期間の初期化関数を実行するには、このメソッドをオーバーライドします。

Servlet がサービスから削除されると、サーバエンジンは、最後のタスクを実行し、リソースの割り当てを解除できるように、Servlet 内の destroy() メソッドを呼び出します。ログメッセージを書き込んだり、ガベージコレクション時に検出されない残留コネクションを削除したりするには、このメソッドをオーバーライドします。

リクエスト処理

リクエストが行われると、iPlanet Application Server は受信データを Servlet エンジンに渡します。Servlet エンジンは、リクエストの入力データ (フォームデータ、cookie、セッション情報、URL の Name-value ペアなど) を HttpServletRequest リクエストオブジェクトタイプに処理します。

また、Servlet エンジンは、HttpServletResponse レスポンスオブジェクトタイプ内にカプセル化することによってクライアントのメタデータを取り込みます。その後、両方のオブジェクトをパラメータとして Servlet の service() メソッドに渡します。

HTTP Servlet 内のデフォルトの `service()` メソッドは、HTTP 転送メソッド (`POST` や `GET` など) に基づいてリクエストをほかのメソッドに転送します。たとえば、HTTP `POST` リクエストは `doPost()` メソッドに転送され、HTTP `GET` リクエストは `doGet()` メソッドに転送されます。これにより、Servlet は使用中の転送メソッドに応じてリクエストデータをさまざまな方法で処理できます。リクエストの転送はサービスメソッドによって行われるため、一般的に HTTP Servlet の `service()` をオーバーライドする必要はありません。その代わり、予想されるリクエストタイプに従って `doGet()` や `doPost()` などをオーバーライドします。

ヒント HTTP Servlet での自動転送を有効にするには、HTTP 転送メソッドを備えている `request.getMethod()` を呼び出します。iPlanet Application Server ですでにリクエストデータが Name-value リストに前処理されているため、機能を失うことなく簡単に HTTP Servlet の `service()` メソッドをオーバーライドできます。ただし、これによって前処理されたリクエストデータに依存することになるため、Servlet の移植性は低下します。

リクエストに応答するタスクを実行するには、`service()` メソッド (汎用 Servlet) または `doGet()` メソッドや `doPost()` メソッド (HTTP Servlet) をオーバーライドします。多くの場合、EJB にアクセスしてビジネストランザクションを実行し、リクエストオブジェクトまたは JDBC `ResultSet` オブジェクト内で情報を照合したあと、新たに作成されたコンテンツを JSP に渡してコンテンツのフォーマットとユーザへの配信を行います。

Servlet エンジンのリソースの割り当て

デフォルトでは、Servlet エンジンは新規リクエストごとにスレッドを作成します。この場合、リクエストごとにメモリ内で Servlet の新規コピーをインスタンス化するよりもリソースに与える影響は小さくなります。各スレッドが同じメモリ空間で動作し、変数が互いを上書きする場合があるので、スレッド問題の発生を避ける必要があります。

Servlet が特にシングルスレッドとして作成されている場合、Servlet エンジンは受信リクエストに使う Servlet のインスタンスを 10 個プールします。すべてのインスタンスが使用中であるときにリクエストを受信すると、インスタンスが使用可能になるまでリクエストはキューに入れられます。プールするインスタンスの数は、iPlanet Application Server 固有の XML ファイルである配置記述子 (DD) で設定できます。配置記述子の詳細については、第 11 章「配置のためのパッケージ化」を参照してください。

スレッド問題の詳細については、49 ページの「スレッドの処理」を参照してください。

実行時の Servlet のダイナミック再読み込み

コンフィグレーションファイルを変更する必要がない場合は、iPlanet Application Server での Servlet の再読み込みは、サーバを再起動しなくても、Servlet を配置し直すだけで行うことができます。同じディレクトリ内で Servlet クラスファイルを新しいクラスファイルに置き換えても、再読み込みを実行できます。

iPlanet Application Server は、新しいコンポーネントを検出し、次の Servlet 要求時にそれを再読み込みします。詳細については、付録 B「実行時の注意事項」を参照してください。

注	出荷時、この機能はデフォルトでオフになっています。必要に応じてオンにしてください。

配置用の Servlet の設定

Servlet を配置用に設定するときは、アプリケーションフレームワーク内での Servlet オブジェクトの作成および使用時にアプリケーションサーバが使うメタデータを指定します。Servlet の設定の詳細については、第 11 章「配置のためのパッケージ化」を参照してください。

Servlet ファイルの検索

Servlet ファイルとほかのアプリケーションファイルは、iPlanet Application Server によって `AppPath` として認識されるディレクトリ構造内にあります。この変数は、アプリケーションの論理ディレクトリツリーのトップを定義します。`AppPath` 変数は Web ブラウザのドキュメントパスに似ています。デフォルトでは、`AppPath` には値 *BasePath*`/APPS` が入っています。*BasePath* は iPlanet Application Server のベースディレクトリです。

`AppPath` と `BasePath` は、サーバおよびアプリケーションメタデータのリポジトリである、iPlanet Application Server のレジストリに保持されている変数です。詳細については、315 ページの「iPlanet Application Server レジストリ」と配置ツールのオンラインヘルプを参照してください。

`AppPath` と `BasePath` に加えて、レジストリには `ModulesDirName` という名前の 3 番目の変数があります。この変数は、J2EE アプリケーションの一部でない Web モジュールのホームである `AppPath` の下位ディレクトリに対応します。これらの Web モジュールはスタンドアロンモジュールとして登録されます。

表 2-1 に、重要なファイルおよび Servlet の場所を示します。

表 2-1　重要なファイルおよび Servlet の場所

場所変数	説明
`BasePath`	iPlanet Application Server ツリーのトップ。このディレクトリ内のすべてのファイルが iPlanet Application Server の構成要素である。レジストリ変数 `BasePath` によって定義される
`AppPath`	アプリケーションツリーのトップ。この場所のサブディレクトリにアプリケーションがある。レジストリ変数 `AppPath` によって定義される
`ModulesDirName`	スタンドアロンエンティティとして登録されたすべての J2EE Web モジュールおよび EJB モジュールが入っている特別なディレクトリ (`Default` アプリケーション内)。このディレクトリは *AppPath* の下位にある。レジストリ変数 `ModulesDirName` によって定義される。レジストリにあるこの変数のデフォルトの値は `modules` である
AppPath/*appName*/*	アプリケーション *appName* のサブツリーのトップ。*appName* ディレクトリには、アプリケーション内のさまざまなモジュールのサブディレクトリがある。詳細については、54 ページの「Servlet の起動」を参照してください。

Servlet の配置

Servlet は通常、iPlanet Application Server Deployment Tool を使って、アプリケーションの残りの部分と共に配置します。また、サーバの動作中に Servlet の検査または更新を行う場合は、Servlet を手動でも配置できます。詳細については、配置ツールのオンラインヘルプを参照してください。

Servlet の設計

この節では、アプリケーションの構成に役立つ Servlet の計画時に必要となる基本的な設計上の決定事項について説明します。

通常、Web アプリケーションはリクエスト / レスポンスパラダイムに従います。したがって、ユーザはフォームの入力および送信の手順に従って Web アプリケーションと対話できます。Servlet は各フォームのデータを処理し、ビジネスロジック関数を実行し、次の対話を設定します。

アプリケーション全体の設計方法が決まると、対話ごとに必要な入出力パラメータを定義することにより、各 Servlet の設計方法が決まります。

Servlet の種類の選択

`HttpServlet` を拡張する Servlet は HTTP 環境用に設計されているため、HTTP 環境で非常に役立ちます。この組み込み HTTP サポートを利用する場合は、すべての iPlanet Application Server Servlet を `GenericServlet` からではなく `HttpServlet` から拡張することをお勧めします。詳細については、37 ページの「Servlet の種類」を参照してください。

標準または非標準 Servlet の作成

アプリケーションの Servlet に関する重要な決定事項の 1 つに、公式仕様書に厳密に準拠した Servlet を作成して移植性を最大限に高めるか、または iPlanet Application Server の API で提供されている機能を利用するかがあります。この API を利用すると、iPlanet Application Server のフレームワークにおける Servlet の有用性が格段に向上します。

また、Servlet が iPlanet Application Server 環境で動作する場合は、iPlanet Application Server の機能だけを利用する移植性の高い Servlet も作成できます。

iPlanet Application Server 固有の API の詳細については、54 ページの「iPlanet Application Server のオプション機能へのアクセス」を参照してください。

Servlet の再利用計画

Servlet は本来、サーバ上で動作する、独立した再利用可能なアプリケーションです。Servlet を 1 つのアプリケーションに関連付ける必要はありません。`Default` という名前のアプリケーションに Servlet ライブラリを配置すると、複数のアプリケーションで共有できます。

ただし、特定のアプリケーションの一部でない Servlet を使う欠点もあります。特に、`Default` アプリケーションの Servlet は、アプリケーションの一部である Servlet とは別に設定します。

Servlet の作成

Servlet を作成するには、次のタスクを実行します。

- Servlet をアプリケーションに組み込みます。つまり、Servlet に汎用的な方法でアクセスされた場合、Servlet からアプリケーションデータにアクセスできないように設計します。
- `GenericServlet` または `HttpServlet` を拡張するクラスを作成し、リクエストを処理する適切なメソッドをオーバーライドします。
- iPlanet Application Server 管理ツール (iASAT) を使って、Servlet の Web アプリケーション配置記述子 (DD) を作成します。

iPlanet アプリケーションの Servlet ファイル

Servlet を構成するファイルは次のとおりです。

- Servlet のクラスファイル
- Servlet の配置記述子
- iPlanet Application Server のオプション機能へのアクセス

Servlet のクラスファイル

この節では、Servlet の作成方法、アプリケーションに関する決定事項、およびアプリケーション内の Servlet の配置場所について説明します。

クラス宣言の作成

Servlet を作成するには、基本的な I/O サポートと `javax.servlet` パッケージを持つ共有 Java クラスを記述します。このクラスは、`GenericServlet` または `HttpServlet` を拡張する必要があります。iPlanet Application Server Servlet は HTTP 環境に存在するので、HttpServlet の拡張をお勧めします。Servlet がパッケージに含まれる場合は、クラスローダがその Servlet を正しく配置できるようにパッケージ名も宣言する必要があります。

次のヘッダーの例は、`myServlet` という HTTP Servlet の宣言を示しています。

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class myServlet extends HttpServlet {
   ...servlet methods...
}
```

メソッドのオーバーライド

次に、1 つまたは複数のメソッドをオーバーライドして、意図したタスクを実行するための指示を Servlet に与えます。Servlet によるすべての処理が、リクエストごとに、サービスメソッド (汎用 Servlet の `service()` メソッドまたは HTTP Servlet の `do`*Operation*`()` メソッドのどれか) で行われます。このメソッドは受信リクエストを受け取り、与えられた指示に従ってリクエストを処理し、出力を適切に送信します。同じように、この Servlet のほかのメソッドも作成できます。

ビジネスロジックには、トランザクションを実行するためのデータベースアクセスや EJB へのリクエスト送信を含めることができます。

初期化メソッドのオーバーライド

カウンタなどの Servlet インスタンスの生存期間のリソースを初期化したり割り当てたりするには、クラスを初期化する `init()` をオーバーライドします。`init()` メソッドは、Servlet がインスタンス化されてからリクエストを受け入れるまでの間に実行されます。詳細については、Servlet の API 仕様書を参照してください。

注　範囲を設定するために、すべての `init()` メソッドで `super.init(ServletConfig)` を呼び出す必要があります。これにより、この Servlet の設定オブジェクトをほかの Servlet メソッドでも使うことができるようになります。この呼び出しを省略すると、Servlet の起動時に「`500 SC_INTERNAL_SERVER_ERROR`」がブラウザに表示されます。

次の `init()` メソッドの例では、`thisMany` という共有整数変数を作成することによってカウンタを初期化します。

```
public class myServlet extends HttpServlet {
   int thisMany;

   public void init (ServletConfig config) throws ServletException
   {
       super.init(config);
       thisMany = 0;
   }
}
```

これで、ほかの Servlet メソッドがこの変数にアクセスできるようになります。

破壊メソッドのオーバーライド

ログメッセージを書き込んだり、ガベージコレクション時に解放されないリソースを解放したりするには、クラスデストラクタ `destroy()` をオーバーライドします。`destroy()` メソッドは、Servlet 自体がメモリの割り当てから解放される直前に実行されます。詳細については、Servlet の API 仕様書を参照してください。

たとえば、上記の「初期化メソッドのオーバーライド」の例に基づいて、`destroy()` メソッドで次のようなログメッセージを書き込むことができます。

```
out.println("myServlet was accessed " + thisMany " times.¥n");
```

Service、Get、および Post メソッドのオーバーライド

リクエストが行われると、iPlanet Application Server は受信データを Servlet エンジンに渡してリクエストを処理します。リクエストにはフォームデータ、cookie、セッション情報、URL の Name-value ペアなどが含まれており、リクエストオブジェクトと呼ばれる `HttpServletRequest` タイプのオブジェクトによって処理されます。クライアントのメタデータは、レスポンスオブジェクトと呼ばれる `HttpServletResponse` タイプのオブジェクトとしてカプセル化されます。Servlet エンジンは、Servlet の `service()` メソッドのパラメータとして両方のオブジェクトを渡します。

HTTP Servlet 内のデフォルトの `service()` メソッドは、HTTP 転送メソッド (POST、GET など) に基づいてリクエストをほかのメソッドに転送します。たとえば、HTTP POST リクエストは `doPost()` メソッドに転送され、HTTP GET リクエストは `doGet()` メソッドに転送されます。これにより、Servlet は転送メソッドに応じてさまざまなリクエストデータ処理を実行できます。リクエストの転送は `service()` で行われるため、一般的に HTTP Servlet の `service()` をオーバーライドする必要はありません。その代わり、予想されるリクエストタイプに従って `doGet()` や `doPost()` をオーバーライドします。

HTTP Servlet での自動転送は、HTTP 転送メソッドを備えている `request.getMethod()` の呼び出しに基づいています。iPlanet Application Server では、Servlet がデータを参照する前にリクエストデータが Name-value リストに前処理されているため、HTTP Servlet の `service()` メソッドをオーバーライドしても機能は失われません。ただし、これによって前処理されたリクエストデータに依存することになるため、Servlet の移植性は低下します。

リクエストへの返答に必要なタスクを実行するには、service() メソッド (汎用 Servlet) または doGet() and/or メソッドや doPost() メソッド (HTTP Servlet) をオーバーライドします。多くの場合、EJB にアクセスしてビジネストランザクションを実行し、リクエストオブジェクトまたは JDBC ResultSet オブジェクト内で必要な情報を照合したあと、新たに作成されたコンテンツを JSP に渡してコンテンツのフォーマットとクライアントにへの配信を行います。

フォームを伴う多くのオペレーションで GET または POST オペレーションが使われるため、ほとんどの Servlet について doGet() または doPost() をオーバーライドします。次の例のように、両方のメソッドを実装して両方の入力タイプに備えたり、リクエストオブジェクトを中央処理メソッドに渡したりできます。

```
public void doGet (HttpServletRequest request,
                   HttpServletResponse response}
           throws ServletException, IOException {
   doPost(request, response);
}
```

HTTP Servlet におけるリクエストごとの負荷はすべて、適切な do*Operation*() メソッドで処理されます。このメソッドには、セッション管理、ユーザ認証、EJB や JSP のディスパッチ、および iPlanet Application Server の機能へのアクセスが含まれています。

Servlet が RequestDispatcher メソッドの include() または forward() を呼び出す場合は、リクエスト情報が HTTP の POST や GET として転送されなくなるので注意してください。つまり、Servlet が doPost() をオーバーライドする場合、呼び出し側の Servlet が HTTP GET 経由でデータを受け取ると、ほかの Servlet が doPost() を呼び出しても何も処理されません。このため、上記の例のように可能性のあるすべての入力タイプのルーチンを実装してください。RequestDispatcher メソッドは常に、service() を呼び出します。

詳細については、56 ページの「プログラムによる Servlet の呼び出し」を参照してください。

パラメータへのアクセスとデータの保存

受信データはリクエストオブジェクトにカプセル化されます。HTTP Servlet の場合、リクエストオブジェクトのタイプは HttpServletRequest です。汎用 Servlet の場合、リクエストオブジェクトのタイプは ServletRequest です。リクエストオブジェクトは、属性と呼ばれるユーザ独自のリクエスト値を含む、すべてのリクエストパラメータを持っています。

受信リクエストのすべてのパラメータにアクセスするには、getParameter() メソッドを使います。次のようにします。

```
String username = request.getParameter("username");
```

リクエストオブジェクト内の値の設定および取得を行うには、それぞれ `setAttribute()` と `getAttribute()` を使います。次のようにします。

```
request.setAttribute("favoriteDwarf", "Dwalin");
```

上記の例は、データを JSP に転送する方法を示しています。これは、JSP に暗黙的 Bean としてのリクエストオブジェクトへのアクセス権があるためです。詳細については、84 ページの「JavaBeans の使用法」を参照してください。

セッションとセキュリティの処理

Web サーバまたはアプリケーションサーバから見ると、Web アプリケーションは関連のないサーバヒットの連続です。数秒前に対話したばかりでも、ユーザがそのサイトにアクセスしたことがあるかどうかの自動認識は行われません。セッションは、アプリケーションの状態を記憶することによってユーザとの複数の対話を関連付けます。クライアントは各対話時に cookie を使って自己を特定します。また、cookie のないブラウザの場合は、URL にセッション識別子を入れることによって自己を特定します。

セッションオブジェクトは、表形式データ、アプリケーションの現在のステートについての情報、現在のユーザについての情報などのオブジェクトを格納できます。セッションに関連付けられたオブジェクトは、同じセッションを使うほかのコンポーネントから使用できます。

詳細は、第 12 章「ユーザセッションの作成と管理」を参照してください。

ログインに成功したら、標準オブジェクト内でユーザの識別情報を確立するように Servlet に指示する必要があります。この標準オブジェクトはセッションオブジェクトと呼ばれ、ユーザのログイン名や保存の必要がある補足情報など、現在のセッションに関する情報を保持しています。アプリケーションコンポーネントはセッションオブジェクトに対してクエリを実行し、ユーザ認証を取得できます。

アプリケーションに対して安全なユーザセッションを行うには、第 13 章「安全なアプリケーションの作成」を参照してください。

ビジネスロジックコンポーネントへのアクセス

iPlanet Application Server のプログラミングモデルでは、データベーストランザクション、ディレクトリトランザクション、複雑な計算などのビジネスロジックを EJB に実装します。`request` オブジェクトの参照は、指定されたタスクを実行するための EJB パラメータとして渡すことができます。

データベーストランザクションからの結果を JDBC `ResultSet` オブジェクトに格納し、フォーマットとクライアントへの配信を行うためにコンポーネントにオブジェクト参照を渡します。また、`request.setAttribute()` メソッドを使ってリクエストオブジェクトの結果を格納したり、`session.putValue()` メソッドを使ってセッショ

ンに格納したりできます。リクエストオブジェクトに格納されるオブジェクトは、リクエストが有効であるかぎり有効です。つまり、個別の Servlet スレッドだけに有効です。セッションに格納されるオブジェクトは、セッションが持続している間は存続します。これによって、多くのユーザ対話にまたがることができます。

JDBC リザルトセットは直列化できないため、クラスタ内の複数のサーバに分散させることはできません。したがって、リザルトセットを分散セッションに保存しないでください。詳細は、第 12 章「ユーザセッションの作成と管理」を参照してください。

次の例は、ShoppingCart という EJB にアクセスする Servlet を示しています。Servlet は、カートのリモートインタフェースをインポートしてからユーザのセッション ID をカートに割り当てることによって、カートに対するハンドルを作成します。カートはユーザのセッション内に保存されます。

```
import cart.ShoppingCart;

   // ユーザのセッションおよびショッピングカートを取得します。
   HttpSession session = request.getSession(true);
   ShoppingCart cart =
      (ShoppingCart)session.getValue(session.getId());

   // ユーザがカートを持っていない場合は新規に作成します。
   if (cart == null) {
       String jndiNm = "java:comp/env/ejb/ShoppingCart";
       javax.naming.Context initCtx = null;
       Object home = null;
       try {
          initCtx = new javax.naming.InitialContext(env);
          java.util.Properties props = null;
          home = initCtx.lookup(jndiNm);
          cart = ((IShoppingCartHome) home).create();
       }
       catch (Exception ex) {
       .....
       .....
       }
   }
```

Java Naming Directory Interface (JNDI) を使って Servlet から EJB にアクセスし、EJB へのハンドルまたはプロクシを確立します。次に、正規オブジェクトとして EJB を参照します。このとき、オーバーヘッドは Bean のコンテナによって管理されます。

次の例は、ショッピングカートのプロクシを検索する JNDI を示しています。

```
String jndiNm = "java:comp/env/ejb/ShoppingCart";
javax.naming.Context initCtx;
Object home;
   try
```

```
    {
        initCtx = new javax.naming.InitialContext(env);
    }
    catch (Exception ex)
    {
        return null;
    }
    try
    {
        java.util.Properties props = null;
        home = initCtx.lookup(jndiNm);
    }
    catch(javax.naming.NameNotFoundException e)
    {
        return null;
    }
    catch(javax.naming.NamingException e)
    {
        return null;
    }
    try
    {
        IShoppingCart cart = ((IShoppingCartHome) home).create();

}
catch (...) {...}
```

EJB の詳細については、第 4 章「Enterprise JavaBeans の紹介」を参照してください。

スレッドの処理

デフォルトでは、Servlet はスレッドセーフになっていません。通常、1 つの Servlet インスタンス内のメソッドは、使用可能なメモリの範囲内で同時に何回も実行されます。メソッドの実行は、それぞれ別のスレッドで行われますが、Servlet エンジンには Servlet のコピーが 1 つしか存在しません。

これによってシステムリソースの使用率が向上しますが、Java でのメモリ管理方法に起因する危険性があります。パラメータ (オブジェクトおよび変数) が参照によって渡されるため、別のスレッドが同じメモリ空間を上書きしてしまうことがあります。Servlet (または Servlet 内のブロック) をスレッドセーフにするには、次のどちらかを行います。

- `public synchronized void` *method*`()` (メソッド全体) または `synchronized(this) {...}` (ブロックのみ) のように、すべてのインスタンス変数に対する書き込みアクセスの同期をとります。同期をとることによって応答時間が大幅に遅くなるため、ブロックだけの同期をとるか、または Servlet 内の同期が不要なブロックを記述してください。

たとえば、次の Servlet には doGet() 内にスレッドセーフのブロックがあり、さらに mySafeMethod() というスレッドセーフのメソッドがあります。

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class myServlet extends HttpServlet {

public void doGet (HttpServletRequest request,
                   HttpServletResponse response}
          throws ServletException, IOException {
    // 前処理
    synchronized (this) {
       // このブロック内のコードはスレッドセーフです。
    }
    // 前処理
    }

public synchronized int mySafeMethod (HttpServletRequest request)
    {
    // このメソッド内で行われる処理はすべてスレッドセーフです。
    }
}
```

- SingleThreadModel を使ってシングルスレッドの Servlet を作成します。この場合、現在のディレクトリを基準に .tld ファイルが検索されます。シングルスレッドの Servlet を iPlanet Application Server に登録すると、Servlet エンジンは受信リクエストに備えて Servlet インスタンスを 10 個プールします。つまり、同じ Servlet の 10 個のコピーをメモリ内に用意します。このプール内の Servlet インスタンスの数は、iPlanet Application Server 固有の Web アプリケーション配置記述子 (DD) の number-of-singles 要素を別の数値に設定することによって変更できます。iPlanet Application Server 固有の Web アプリケーション DD のこの数値を変更するには、iPlanet Application Server 配置ツールを使います。iPlanet Application Server の Web アプリケーション DD の詳細については、第 11 章「配置のためのパッケージ化」、IPlanet Application Server 配置ツールのオンラインヘルプ、および『管理者ガイド』を参照してください。シングルスレッドの Servlet は、新規リクエストを処理するためにインスタンスの空きを待つ必要があるため、その負荷によって動作が遅くなります。ただし、ロードバランスが有効な分散アプリケーションでは、比較的ビジーでない kjs プロセスに負荷が自動的にシフトするため、これが問題になることはありません。

  たとえば、次の Servlet は完全なシングルスレッドです。

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
public class myServlet extends HttpServlet
    implements SingleThreadModel {
    servlet methods...
}
```

クライアントへの結果の配信

ユーザとの対話の最終作業はクライアントにレスポンスページを配信することです。レスポンスページは次の 2 つの方法で配信できます。

- Servlet レスポンスページの作成
- JSP レスポンスページの作成

Servlet レスポンスページの作成

出力ストリームに書き込むことによって、Servlet 内で出力ページを生成します。出力のタイプによって推奨方法は異なります。

すべての出力を開始する前に、`setContentType()` を使って出力の MIME タイプを必ず指定します。次に例を示します。

```
response.setContentType("text/html");
```

単純な HTML などのテキスト出力の場合は、`PrintWriter` オブジェクトを作成してから `println` を使って書き込みます。次のようにします。

```
PrintWriter output = response.getWriter();
output.println("Hello, World¥n");
```

バイナリ出力の場合は、`ServletOutputStream` オブジェクトを作成してから `print()` を使って書き込むことによって、出力ストリームに直接書き込みます。次のようにします。

```
ServletOutputStream output = response.getOutputStream();
output.print(binary_data);
```

注 Servlet が、`PrintWriter` または `ServletOutputStream` オブジェクトから JSP を呼び出すことはできません。

注 iPlanet Application Server と iPlanet Web Server を併用する場合は、`setDateHeader()` を使って出力ストリーム内で日付ヘッダーを設定しないでください。設定すると、サーバがクライアントに返送するレスポンスページの HTTP ヘッダー内の日付フィールドが重複します。これは、iPlanet Web Server によってヘッダーフィールドが自動的に設定されるためです。これに対し、Microsoft の Internet Information Server (IIS) では日付ヘッダーが追加されないため、手動で設定する必要があります。

JSP レスポンスページの作成

Servlet は次の 2 つの方法で JSP を起動できます。

- `RequestDispatcher` インタフェースの `include()` メソッドは、JSP を呼び出し、JSP が返されてから対話処理を続行します。`include()` メソッドは、特定の Servlet 内で複数回呼び出すことができます。

  次の例は、`include()` を使った JSP を示しています。

```
RequestDispatcher dispatcher =
   getServletContext().getRequestDispatcher("JSP_URI");
dispatcher.include(request, response);
... // 処理の継続
```

- `RequestDispatcher` インタフェースの `forward()` メソッドは対話の制御を JSP に渡します。`forward()` を呼び出すと、Servlet は現在の対話の出力と無関係になります。つまり、特定の Servlet で `forward()` メソッドを呼び出すことができるのは 1 回だけです。

注 `PrintWriter` オブジェクトまたは `ServletOutputStream` オブジェクトをすでに定義している場合は、`forward()` メソッドを使うことができないので注意してください。

次の例は、`include()` を使った JSP を示しています。

```
RequestDispatcher dispatcher =
   getServletContext().getRequestDispatcher("JSP_URI");
dispatcher.forward(request, response);
```

注 URI (Universal Resource Identifier) を指定して、呼び出す JSP を指定してください。このパスは、`ServletContext` の範囲内にあるパスを記述する `String` です。また、絶対パスを示す `String` 引数を取るリクエストオブジェクトには、`getRequestDispatcher()` メソッドもあります。このメソッドの詳細については、Java Servlet 仕様書バージョン 2.2 の第 8 章を参照してください。

JSP の詳細については、第 3 章「JavaServer Pages によるアプリケーションページの表示」を参照してください。

Servlet の配置記述子

Servlet の配置記述子は、iPlanet Application Server 配置ツールによって作成されます。手動でも作成することができます。これらの記述子ファイルは、Web アプリケーション の aRchive (`.war`) ファイル内にパッケージされています。aRchive ファイルには、メタデータとともに、Servlet を識別しそのアプリケーションの役割を確立する情報が含まれています。

注 WAR モジュール名は、`.war` 拡張子ではなく、ファイル名の最初の部分によって識別されます。Application Server に配置する WAR モジュール名は、一意でなければなりません。これらのモジュールのファイル名には、Java パッケージ方式の命名規則を使ってください。Java パッケージ方式の命名規則を使えば、名前の衝突は発生しません。この命名規則は、iPlanet Application Server だけでなく、ほかの J2EE アプリケーションサーバでも使うことをお勧めします。

iPlanet Application Server のサンプルアプリケーションには、Servlet の配置記述子を作成する手順が含まれています。これらのサンプルアプリケーションは、*install_dir*`/ias/ias-samples` ディレクトリにあります。

要素

Servlet の配置記述子には、iPlanet Application Server 固有の要素だけでなく標準の J2EE 規定の要素があります。Servlet の配置記述子は、開発者、編成者、および配置者間で、Web アプリケーションの要素および設定の情報を伝えます。これらの要素の詳細については、第 11 章「配置のためのパッケージ化」を参照してください。

コンフィグレーションファイルの変更

配置記述子の設定を変更するには、配置ツールを使うか、またはエディタと Ant のようなコマンドラインユーティリティを組み合わせて使い、更新された配置記述子の情報を再構成し、配置します。

配置ツールの使用法

1. EAR、WAR、または EJB JAR ファイルを開きます。
2. 配置記述子を変更します。
3. EAR、WAR、または EJB JAR モジュールを再配置します。
4. アプリケーションサーバを再起動して変更した配置記述子の設定を取得します。

コマンドラインの使用法

Ant ベースの `build.xml` ファイルを使ってコマンドラインからアプリケーションおよびモジュールを再構成する例の詳細については、サンプルアプリケーション (*install_dir*`/ias/ias-samples` ディレクトリ内) を参照してください。

1. 適切な配置記述子ファイル `weblxml` または `ias-web.xml` を手動で編集します。
2. Ant ビルドコマンド (`build war` など) を実行して適切な EAR、WAR、または EJB JAR モジュールを再構成します。
3. `iasdeploy` コマンドを使って EAR または WAR のファイルまたはモジュールを配置します。
4. アプリケーションサーバを再起動して変更した配置記述子の設定を取得します。

iPlanet Application Server のオプション機能へのアクセス

数多くの iPlanet 機能の追加によって、iPlanet Application Server 環境での Servlet の使用が強化されます。これらの機能は公式の仕様書では規定されていませんが、一部の機能は Sun の標準に基づいており、将来の標準に準拠する予定です。

iPlanet Application Server の機能の詳細については、第 14 章「iPlanet Application Server の機能の活用」を参照してください。

iPlanet Application Server は、前バージョンモデルの iPlanet Application Server に基づいたより強力なセッションをサポートしています。このモデルは、Servlet 2.2 仕様に記述されているセッションモデルと同じ API を使っており、iPlanet Application Server はこの API もサポートしています。分散セッションの詳細については、第 12 章「ユーザセッションの作成と管理」を参照してください。

Servlet の起動

Servlet を起動するには、URL を使ってアプリケーションページから直接アドレス指定するか、またはすでに実行している Servlet からプログラムで呼び出します。Servlet パラメータを確認することもできます。次の節を参照してください。

- URL による Servlet の呼び出し
- プログラムによる Servlet の呼び出し
- Servlet パラメータの確認

URL による Servlet の呼び出し

多くの場合、アプリケーションのページにリンクとして埋め込まれた URL を使って Servlet を呼び出します。この節では、標準の URL を使って Servlet を起動する方法について説明します。

特定のアプリケーション Servlet の起動

リクエストに応答する Servlet への URL リクエストパスは、いくつかのセクションから構成されます。各セクションは適切な Servlet を特定する必要があります。リクエストオブジェクトは、リクエストの URI パスを取得すると次の要素を公開します。

- Context Path
- Servlet Path
- PathInfo

これらの要素の詳細については、Java Servlet 仕様書バージョン 2.2 の第 5.4 節を参照してください。

次のように、特定のアプリケーションの一部である Servlet をアドレス指定します。

`http://`*server*`:`*port*`/IASApp/`*moduleName*`/`*servletName*`?`*name*`=`*value*

表 2-2 は、URL の各セクションについて説明しています。

表 2-2 特定のアプリケーションに含まれる Servlet の URL フィールド

URL 要素	説明
server`:`*port*	リクエストを処理する Web サーバのアドレスおよびオプションのポート番号
`IASApp`	この URL が iPlanet Application Server アプリケーション用であることを Web サーバに示す。リクエストは、iPlanet Application Server の Executive Server に送信される
moduleName	Servlet のモジュール名 (サーバ全体で固有の名前)。*moduleName* はアプリケーションの一部として登録されたモジュールの *AppPath*`/`*applicationName* 下にあるディレクトリに対応する。これは Servlet と JSP が含まれている `.war` モジュール名を示し、その内容は `.war` モジュールの内容と同じ
servletName	XML ファイルに設定された Servlet の名前

次のようにします。

```
http://www.my-company.com/IASApp/OnlineBookings/directedLogin
```

汎用アプリケーション Servlet の起動

次のように、汎用の Default アプリケーションの一部である Servlet をアドレス指定します。

http://*server*:*port*/servlet/*servletName*?*name=value*

表 2-3 は、URL の各セクションについて説明しています。

表 2-3　汎用アプリケーションに含まれる Servlet の URL フィールド

URL 要素	説明
server:*port*	リクエストを処理する Web サーバのアドレスおよびオプションのポート番号
servlet	この URL が汎用 Servlet オブジェクトのものであることを Web サーバに示す
servletName	Web アプリケーションの XML ファイルの servlet-name 要素で指定された Servlet の名前
?*name=value*...	Servlet のオプションの Name-value パラメータ

次のようにします。

```
http://www.leMort.com/servlet/calcMortgage?rate=8.0&per=360&bal=180000
```

注　/servlet パスを使って配置するすべての Servlet が、Default というアプリケーション名で配置される必要があります。また、/servlet で始まるリクエストを iPlanet Application Server に渡すには、Web サーバインスタンスの Servlet エンジンを無効にする必要があります。

プログラムによる Servlet の呼び出し

まず、URI を指定して、呼び出す Servlet を指定します。通常、これは現在のアプリケーションに対応する相対パスになります。たとえば、Servlet が Office という名前のコンテキストルートを持つアプリケーションの一部である場合、ブラウザから ShowSupplies という名前の Servlet への URL は次のとおりです。

http://*server*:*port*/IASApp/Office/ShowSupplies?name=value

プログラムによってほかの Servlet からこの Servlet を呼び出す方法は次の 2 とおりあります。

- ほかの Servlet の出力を含めるには、RequestDispatcher インタフェースから include() メソッドを使います。このメソッドは、URI で Servlet を呼び出し、Servlet が返されてから対話処理を続行します。include() メソッドは、特定の Servlet 内で複数回呼び出すことができます。

  次のようにします。

```
RequestDispatcher dispatcher =
   getServletContext().getRequestDispatcher("/ShowSupplies");
dispatcher.include(request, response);
```

- ほかの Servlet に対話制御を渡すには、RequestDispatcher インタフェースの forward() メソッドに Servlet の URI をパラメータとして使います。

注 リクエストを転送した場合、forward() を呼び出すと、元の Servlet は現在の対話の出力とは無関係になるので注意してください。つまり、特定の Servlet で forward() を呼び出せるのは 1 回だけです。

次の例は、include() を使った JSP を示しています。

```
RequestDispatcher dispatcher =
   getServletContext().getRequestDispatcher("/ShowSupplies");
dispatcher.forward(request, response);
```

注 include() や forward() を使って Servlet をプログラムで起動するメカニズム、または Servlet を URL から起動するメカニズムでは、<servlet-name> エントリ内だけでなく、配置記述子の XML ファイルに指定されている Servlet の URL パターンを使うことができます。たとえば、web.xml ファイル内の Servlet の XML エントリが次のような場合を検討します。

```
<servlet-name>Fortune</ servlet-name>
<servlet-mapping>
<servlet-name>Fortune</ servlet-name>
<url-pattern>/Business</url-pattern>
</servlet-mapping>
```

この場合、次のどちらかの方法で Servlet にアクセスできます。

- http://*server*:*port*/IASApp/*context_root*/Fortune
- http://*server*:*port*/IASApp/*context_root*/Business

Servlet パラメータの確認

Servlet に渡されたパラメータを確認できます。この機能により、iPlanet Application Server の応答時間を増やし、開発時間を短縮できます。

iPlanet Application Server はパラメータの確認用に指定されたクラスを呼び出すことができます。確認した結果に基づき、サーバは Servlet メソッドを呼び出すか、または Servlet への呼び出しを中止して、ユーザをエラーページにリダイレクトできます。Servlet の配置時に、パラメータの確認クラスを用意し、配置ツールで指定する必要があります。どのパラメータが有効であることを指定できます。

パラメータの確認コードは、Servlet 内に存在する必要はありません。複数の Servlet が同じパラメータを受け入れる場合は、各 Servlet がそのパラメータが同じパラメータであるか確認する機能を使う必要があります。

配置ツールの Servlet 記述子の「IAS Params」タブでは、各パラメータに対して次の要素を指定できます。

- パラメータの名前
- 確認の必要性
- 確認のために呼び出すクラスおよびメソッド
- パラメータのフォーマット
- パラメータの範囲
- エラーの場合に表示するエラーページ

第 3 章

JavaServer Pages によるアプリケーションページの表示

この章では、JSP (JavaServer Pages) を iPlanet Appllication Server Web アプリケーションのページテンプレートとして使う方法について説明します。

この章には次の節があります。

- JSP の紹介
- JSP の機能
- JSP の設計
- JSP の作成
- 高度な JSP プログラミング
- JSP の配置
- JSP の起動
- JSP 1.1 タグの要約
- JSP 1.1 用カスタムタグの変更
- JSP のコンパイル : コマンドラインコンパイラ
- 付加価値機能

JSP の紹介

JSP は HTML または XML で書かれたブラウザページです。JSP には Java コードを含めることもできるため、複雑な処理を実行したり、出力に条件を付けたり、アプリケーション内のほかのオブジェクトと通信したりできます。iPlanet Application Server の JSP は JSP 1.1 仕様に準拠しています。この仕様書は *install_dir*`/ias/docs/index.htm` からアクセス可能です。*install_dir* は iPlanet Application Server がインストールされている場所です。

iPlanet Application Server アプリケーションでは、JSP はアプリケーションを構成する個々のページです。Servlet から JSP を呼び出してユーザ対話からの出力を処理できます。また、JSP はほかのアプリケーションコンポーネントと同じ方法でアプリケーション環境にアクセスするので、JSP を対話の相手として利用できます。

JSP の機能

JSP は、JSP 要素とテンプレートデータから構成されています。テンプレートデータとは、JSP 仕様書に定義されていないテキストや HTML タグなどのデータのことです。たとえば、最小の JSP は JSP エンジンによる処理が不要なスタティック HTML ページです。

iPlanet Application Server は、JSP が最初に呼び出されたときにその JSP を HTTP Servlet にコンパイルします。これにより、JSP を標準オブジェクトとしてアプリケーション環境で使えるようになり、URL を使ってクライアントから JSP を呼び出すことが可能になります。

JSP はサーバ上の Java プロセス内で実行されます。JSP エンジンと呼ばれるこのプロセスは、JSP 固有のタグを解釈し、そのタグが指定するアクションを実行することによってダイナミックコンテンツを生成します。このコンテンツは、それを囲んでいるテンプレートデータとともに出力ページにまとめられ、呼び出したユーザに返されます。

レスポンスオブジェクトには呼び出し側クライアントへの参照が含まれており、JSP は作成したページをここに表示します。`RequestDispatcher` インタフェースの `forward()` メソッドを使って Servlet から JSP を呼び出した場合は、`forward()` メソッドがレスポンスオブジェクトを JSP パラメータとして提供します。クライアントから JSP を直接起動した場合は、呼び出し側クライアントとの関係を管理するサーバがレスポンスオブジェクトを提供します。

どちらの場合も、ページは、レスポンスオブジェクトの参照によってクライアントに自動的に返されるので、それ以上のプログラミングは必要ありません。

特定のアプリケーションの一部ではない JSP を作成できます。このような JSP は汎用アプリケーションの一部であると見なされます。JSP は iPlanet Web Server やほかの Web サーバでも実行可能ですが、このような JSP はアプリケーションデータへのアクセス権を持たないため、使用法は制限されます。

JSP およびほかのアプリケーションコンポーネントはサーバを再起動しなくても実行時に更新できるので、サービスを中断しないでアプリケーションの外観や機能を簡単に変更することができます。詳細は、付録 B「実行時の注意事項」を参照してください。

JSP の設計

この節では、JSP を記述する際に考慮すべき決定事項について説明します。JSP は Servlet にコンパイルされるので、Servlet の設計上の決定事項は JSP にも関係します。Servlet の設計上の決定事項については、第 2 章「Servlet によるアプリケーションの制御」を参照してください。

ページの情報は、タグとページ構成情報から構成されるページレイアウト要素と、ユーザに送信される実際のページ情報から構成されるページコンテンツ要素に大別できます。

ページレイアウトは、ブラウザページの設計と同じように、必要な場所にコンテンツ要素をインターリーブして設計できます。たとえば、ページの一番上に「私たちのアプリケーションにようこそ!」のようなウェルカムメッセージで表示するとします。ユーザの認証後は、そのユーザの名前を使って「私たちのアプリケーションにようこそ、アインシュタインさん!」のようなメッセージを表示できます。

ページレイアウトは単純なタスクであるため、設計上必要な決定事項はむしろ JSP のアプリケーションとの対話方法や JSP の最適化の方法に関係しています。

この節には次の項があります。

- コンポーネントの選択
- メンテナンスの容易さを考慮した設計
- 移植性を考慮した設計
- 例外の処理

コンポーネントの選択

最初の作業は JSP と Servlet のどちらを使うかを判断することです。ページレイアウトを主眼とし、ページを作成するための処理がほとんどない場合は、対話には JSP だけを使います。

JSP と Servlet を 1 枚のコインの表と裏と考えてください。JSP のタスクは Servlet でも実行でき、その逆の操作も可能です。しかし、JSP のタスクは JSP に最適化されており、Servlet のタスクは Servlet に最適化されています。Servlet は処理能力と適応性に優れています。また、Servlet は Java ファイルなので Servlet を記述する際は統合開発環境を利用できます。ただし、Java ファイルから HTML の出力を実行すると、println ステートメントが多量に発生します。Printin ステートメントは、手動でコーディングする必要があるため処理が面倒です。それに対し、JSP は HTML ファイルなので、計算や処理タスクの実行には不向きですが、HTML エディタで編集できるので、レイアウト作業に優れています。作業に適したコンポーネントを選択してください。

たとえば、JSP と Servlet の簡単なコンポーネントを比較してみましょう。このコンポーネントは、複雑なコンテンツ生成作業がないため、JSP として最適に動作します。

JSP :

```
<html><head><title>Feedback</title></head><body>
<h1>The name you typed is:<% request.getParameter("name"); %>.</h1>
</body></html>
```

Servlet :

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class myServlet extends HttpServlet {
   public void service (HttpServletRequest req,
                        HttpServletResponse res)
            throws ServletException, IOException
   {
   response.setContentType("text/html");
   PrintWriter output = response.getWriter();
   output.println("<html><head><title>Feedback</title></head>"
                + "<body>¥n"
                + "<h1>The name you typed is:"
                + req.getParameter("name") + ".</h1>"
                + "</body></html>";
   }
}
```

Servlet については、第 2 章「Servlet によるアプリケーションの制御」を参照してください。

メンテナンスの容易さを考慮した設計

各 JSP はほかの任意の JSP を呼び出したり取り込んだりできます。たとえば、汎用のコーポレートヘッダー、標準のナビゲーションバー、左側の目次カラムなどを作成できます。このカラムの各要素は、個別の JSP に入っており、作成されたページごとに取り込まれています。このページは、各サブフレームを読み込むページをダイナミックに決めるフレームセットとして機能する JSP で構成できます。JSP は、Servlet へのコンパイル時またはリクエストの到着時に取り込むこともできます。

移植性を考慮した設計

JSP は、異なるアプリケーションおよび異なるサーバ間で完全に移植できます。特定のアプリケーションデータの知識を保持しないという欠点がありますが、そのようなデータが不要な場合は問題ありません。

汎用 JSP を使う例としては、ナビゲーションバーやコーポレートヘッダーおよびフッターのような移植性のあるページ要素があります。これはほかの JSP に取り込まれることを想定しています。再利用可能な汎用ページ要素のライブラリを作成してアプリケーション全体で使ったり、ほかの複数のアプリケーションで使ったりできます。

たとえば、もっとも簡単な汎用 JSP は JSP 固有のタグを持たないスタティック HTML ページです。これよりやや複雑な JSP は、日時の印刷などの一般データを操作したり、リクエストオブジェクトの標準の値セットに基づいたページ構造に変更したりする Java コードを持っています。

例外の処理

検出されていない例外が JSP ファイルで発生すると、iPlanet Application Server では、通常、`404` または `500` エラーの例外を生成します。この問題を避けるには、`<%@ page%>` タグの `errorPage` 属性を設定します。

JSP の作成

JSP は基本的に、スタティック HTML ファイルと同様に作成します。HTML エディタを使ってページを作成したりレイアウトを編集したりできます。ページを JSP にするには生のソースコードの適切な位置に JSP 固有のタグを挿入します。

次の節では、HTML ファイルで JSP 固有のタグを使って、JSP 要素、ディレクティブ要素、スクリプト要素、アクション要素を含む、JSP を作成する方法について説明します。

この節には次の項があります。

- 一般シンタックス
- ディレクティブ
- スクリプト要素
- アクション
- 暗黙的オブジェクト

一般シンタックス

JSP 1.1 仕様に準拠する JSP は、ほとんどの部分が、HTML と似ていてより厳密な XML シンタックスに従っています。つまり、タグは < と > で囲まれています。構造体には開始タグ (<tag>) および終了タグ (</tag>) があります。タグでは、大文字と小文字が区別されます。たとえば、<tag> は <Tag> や <TAG> とは異なります。

JSP タグは通常、標準の HTML タグと同様に、ファイル内の適切な場所に挿入します。たとえば、ユーザ名を格納するパラメータ name がリクエストに含まれている場合、ウェルカムメッセージは次のようになります。

```
<p>Hello, <%= request.getParameter("name"); %>.</p>
```

JSP タグ

JSP タグは XML の形式をとる <jsp:*tag*> の形式を使います。特にスクリプトタグなどの一部のタグには、HTML ファイルで使うためのショートカットがあり、一般に <% で始まり、%> で終わります。

注　これらのショートカットは XML ファイルでは有効ではありません。

空の要素、つまり開始タグと終了タグの間に何もないタグ構成は、/> で終わる 1 つのタグに省略できます。次に例を示します。

本体のない Include タグ :

```
<jsp:include page="/corporate/banner.jsp"></jsp:include>
```

本体のない Include タグの省略 :

```
<jsp:include page="/corporate/banner.jsp" />
```

通常、空白は重要ではありませんが、開始タグと任意の属性との間には空白文字を入れる必要があります。たとえば、`<%= myExpression %>` は有効ですが、`<%=myExpression %>` は無効です。

エスケープ文字

属性で、入れ子の単一引用符および二重引用符の扱いが困難な場合は、次のようなエスケープ文字を使います。

- `'` は `¥'` で囲む
- `"` は `¥"` で囲む
- `%>` は `%¥>` で囲む
- `<%` は `<¥%` で囲む

コメント

JSP には次の 2 種類のコメントがあります。

- ページの動作を文書化する、JSP ページへのコメント
- クライアントに送信される、ドキュメントに生成されたコメント

JSP コメント

JSP コメントは `<%--` と `--%>` で囲み、`--%>` を除く任意のテキストを含めることができます。したがって、次の例は誤りです。

```
<%-- 閉じる以外の操作 --%> ... --%>
```

また、Java コメントを使って JSP にコメントを配置する方法もあります。次のようにします。

```
<% /** これはコメントです ... **/ %>
```

クライアントへの出力でのコメントの生成

レスポンス出力ストリーム内でリクエストを出したクライアントへのコメントを生成するには、次の例のような HTML および XML コメントシンタックスを使います。

```
<!-- コメント ... -->
```

JSP エンジンは、解釈されていないテンプレートテキストとしてコメントを扱います。生成されたコメントがダイナミックデータを持つ場合、このダイナミックデータは、次の例のような expression シンタックスを使って取得します。

```
<!-- コメント <%= expression %> さらにコメント ... -->
```

ディレクティブ

ディレクティブを使って JSP にプリファレンスを設定します。各ディレクティブには、JSP の動作や状態に影響を与える多くの属性があります。

```
<%@ directive { attr="value" }* %>
```

有効なディレクティブは次のとおりです。

- `<%@ page%>`
- `<%@ include%>`
- `<%@ taglib... %>`

<%@ page%>

page ディレクティブは JSP のページレベルのプリファレンスを設定します。

シンタックス

```
<%@ page language="java"
          extends="className"
          import="className{,+}"
          session="true|false"
          buffer="none|sizeInKB"
          autoFlush="true|false"
          isThreadSafe="true|false"
          info="text"
          errorPage="jspUrl"
          isErrorPage="true|false"
          contentType="mimeType{;charset=charset}"
%>
```

属性

表 3-1 に有効な属性を示します。

表 3-1　JSP の page ディレクティブ

属性	有効な値	説明
language	java	デフォルトは java。この JSP を記述する言語。現在、iPlanet Application Server がサポートしているのは java のみ
extends	有効な Java クラス名	この JSP に特定のスーパークラスを定義する。これはいろいろな意味で JSP エンジンを制限するので、できるかぎり避ける必要がある
import	カンマで区切られている有効な Java クラス名のリスト	この JSP のほかのメソッドで使用可能なタイプおよびクラス。これは、Java クラスの import ステートメントと同じ import ディレクティブは、JSP ファイルに複数個指定できる唯一のディレクティブである
session	true または false	デフォルトは true。ページが HTTP セッションを共有する必要があることを示す。language=java および session=true の場合、このオプションは javax.servlet.http.HttpSession タイプのセッションを指したり、作成したりする session と呼ばれる暗黙的変数を作成する
buffer	none またはバッファサイズ (単位は KB)	出力バッファを定義する。none に設定すると、すべての出力が出力ストリーム (PrintWriter オブジェクト) に直接書き込まれる。サイズが指定されている場合にバッファが出力で満杯になると、その内容が出力ストリームに書き出されるか例外が発生する。動作は autoFlush 属性によって決まる
autoFlush	true または false	出力バッファが満杯になったときの動作を決める。true の場合、バッファが満杯になると出力は出力ストリームに書き出される。false の場合、バッファが満杯になると例外が発生する
isThreadSafe	true または false	デフォルトは false。このページ内のスレッドの安全性のレベルを示す。この値は JSP エンジンの動作を決定する。true の場合は、JSP インスタンスに対して複数のリクエストが同時に作成されるが、それ以外の場合は、複数のリクエストは順番に処理される。この設定はセッションやコンテキストなどの共有オブジェクトに影響を及ぼさないので、多くの場合は、この設定に関係なく JSP をスレッドセーフにする
info	テキスト	翻訳されたページ内の文字列。そのページ内の Servlet.getServletInfo() メソッドから取得

表 3-1　JSP の page ディレクティブ (続き)

属性	有効な値	説明
errorPage	JSP エラーページの有効な URL	この JSP のエラーページ。JSP でなければならない。元のページによってスローされるが検出されない Throwable オブジェクトはエラーページに転送される。エラーページには、検出されなかった例外への参照を持つ exception と呼ばれる暗黙的変数がある。autoFlush=true の場合に、最初の JspWriter のコンテンツが ServletResponse 出力ストリームに書き出されていると (ページの一部がすでにクライアントに送信されている場合など)、あとからエラーページを起動しても失敗することがある
isErrorPage	true または false	デフォルトは false。現在の JSP ページが、別の JSP ページの errorPage のターゲットになる可能性があるかどうかを示す。true の場合は、暗黙的変数 exception が定義され、その値は、エラーが発生したソース JSP ページからの問題がある Throwable への参照である
contentType	コンテンツタイプ、オプションで charset を持つ	デフォルトは text/html;charset=ISO-8859-1 。レスポンスの MIME タイプおよび文字エンコードを定義する。値は TYPE または TYPE;charset=CHARSET のどちらかの形式

例

```
<%@ page errorpage="errorpg.htm" %>
<%@ page import="java.io.*,javax.naming.*" %>
```

<%@ include%>

include ディレクティブを使うと、JSP を Servlet にコンパイルするときに、ほかの JSP (またはスタティックページ) を取り込むことができます。このリソースは JSP の一部として扱われます。

ほかのリソースを取り込むためには、要求時にリソースを取り込む <jsp:include> アクションを使う方法もあります。ファイルを取り込む方法については、82 ページの「ほかのリソースの取り込み」を参照してください。

シンタックス

```
<%@ include file="file" %>
```

属性

表 3-2 に有効な属性を示します。

表 3-2　JSP の include ディレクティブ

属性	有効な値	説明
file	有効な URL (絶対パス) または URI (相対パス)	取り込むファイル

file 属性は、現在の JSP の相対パス、またはアプリケーションのコンテキストルートの絶対パスになります。相対 file 属性の場合は、ファイル名をスラッシュ ('/') で始めないでください。絶対 file 属性の場合は、ファイル名をスラッシュ ('/') で始める必要があります。

例

who.jsp がアプリケーション MyApp 内にある場合 (通常は *install_dir*/ias/APPS/MyApp に置かれている)、who.jsp には次のタグが含まれています。

```
<%@include file="/add/baz.jsp"%>
```

次に、システムは *install_dir*/ias/APPS/MyApp/add/baz.jsp から baz.jsp ファイルを取り込みます。

baz.jsp が次のタグを含んでいる場合、

```
<%@include file="who.jsp"%>
```

システムはファイル *install_dir*/ias/APPS/MyApps/add/who.jsp を取り込みます。

<%@ taglib... %>

tag library ディレクティブを使うと、カスタムタグを作成できます。カスタムタグの作成方法については、95 ページの「付加価値機能」を参照してください。

シンタックス

```
<%@ taglib uri="uriToTagLibrary" prefix="prefixString" %>
```

属性

表 3-3 に有効な属性を示します。

表 3-3　JSP の <taglib> ディレクティブ

属性	有効な値	説明
uri	有効な URI (相対パス)	URI は、アプリケーションのコンテキストルートからの絶対参照、またはタグリブを記述する .tld XML ファイルの相対参照。URI は、Web アプリケーションの JSP 記述子内の <taglib> エントリによってエイリアス指定が解除されるエイリアスにできる。詳細については、JSP v1.1 仕様書の第 5.2 節を参照
prefix	文字列	カスタムタグのプレフィックス

例

アプリケーション MyApp 内の次のような JSP ファイル who.jsp と、それに対応する次のような Web アプリケーションセクションを持つ XML 配置記述子ファイルがあるとします。

```
<taglib>
   <taglib-uri> http://www.mytaglib.com/spTags </taglib-uri>
   <taglib-location> /who/add/baz.tld</taglib-location>
<taglib>
```

JSP ファイルには次のセクションがあります。

```
<%@ taglib uri="http://www.mytaglib.com/spTags" prefix="mytags" %>
<mytags:specialTag attribute="value"> ...</mytag:specialTag>
```

JSP エンジンは web app 記述子の内部で、http://www.mytaglib.com/spTags と一致するタグリブの場所を検出します。エンジンは /who/add/baz.tld を探します。つまり、XML ファイル *install_dir*/ias/APPS/MyApp/who/add/baz.tld を探します。これは、このファイルで使われているタグを記述するタグリブ記述子ファイルです。

URI、つまりタグリブの場所 (URI がエイリアス指定されている場合) を相対パスにすることもできます。この場合、現在のディレクトリを基準に .tld ファイルが検索されます。詳細については、JSP 仕様書 バージョン 1.1 の第 5.2 節を参照してください。

スクリプト要素

スクリプト要素は次のタグから構成されています。

- 宣言 <%! ... %>
- 式 <%= ... %>
- スクリプトレット <%...%>

スクリプトには、リクエスト / レスポンスオブジェクトを含む複数の暗黙的オブジェクトを使用できます。暗黙的オブジェクトについては、80 ページの「暗黙的オブジェクト」を参照してください。

宣言 <%! ... %>

宣言の要素は、JSP 全体で使われる有効な変数を定義します。宣言が完全であるかぎり、メソッドなどの、正しい Java のスクリプトはすべて宣言できます。宣言の結果として、出力ストリームには何も表示されません。

シンタックス

```
<%! declaration %>
```

例

```
<%! int i=0; %>
<%! String scriptname="myScript"; %>
<%! private void myMethod () { ... } %>
```

式 <%= ... %>

式の要素は変数を評価します。式の値は式が発生する場所に代入されます。結果は出力ストリームに現れます。

式の結果は、文字列、または結果を文字列にキャストできる式でなければなりません。

シンタックス

```
<%= expression %>
```

例

```
<p>My favorite color is <%= userBean.favColor %>.</p>
```

スクリプトレット <%...%>

スクリプトレットの要素は実行するコードブロックを定義します。正しいコードはすべてここに表示されます。

シンタックス

```
<% script %>
```

例

```
<% int balance = request.getAttribute("balance");
   if (balance < LIMIT) {
       println (UNDERLIMIT_ALERT);
   }
   String balString = formatAsMoney(balance);
%>
Your current balance is <%= balance %>.
```

アクション

アクションは、ほかの JSP の取り込み、必要なプラグインの指定、JavaBeans の作成や読み込み、Bean プロパティの設定や取得などのアクティビティを実行します。

要求時の式をパラメータとして許可するアクションでは、これらの属性の値をダイナミックにリクエストに設定できます。式をパラメータとして許可する属性には、<jsp:setProperty> の value および name 属性と <jsp:include> および <jsp:forward> の page 属性があります。

次に、標準のアクションについて説明します。

- <jsp:useBean> は JavaBeans を作成したり、アクセスしたりする
- <jsp:setProperty> は Bean のプロパティを設定する
- <jsp:getProperty> は Bean のプロパティを取得する
- <jsp:include> は要求時にほかの JSP または HTML ページを取り込む
- <jsp:forward> は別の JSP に実行の制御権を渡す
- <jsp:plugin> は特別なデータタイプのブラウザプラグインをダイナミックに読み込む

<jsp:useBean>

<jsp:useBean> アクションは指定された名前 (id) と scope を持つ JavaBeans を検出します。Bean が存在する場合はこのアクションを使用できますが、存在しない場合は、入力された名前、scope、およびタイプ / クラスの情報を使って作成されます。アクションが成功したらそのオブジェクトにアクセスできるように、属性 id="*name*" で指定する *name* と呼ばれる変数を JSP で使用できます。

<jsp:useBean> は <jsp:useBean .../> のように空のタグにできます。または、ほかのアクションを含めたり、終了タグ </jsp:useBean> で終了することもできます。通常ここに表示されるアクションは、主に新規作成された Bean のプロパティを設定する <jsp:setProperty> アクションです。テンプレートテキスト、ほかのスクリプト、宣言などは正常に処理されます。<jsp:useBean> タグボディは、Bean 作成時に一度だけ実行されます。

<jsp:useBean> アクションには必ず固有の id="*name*" 属性を指定します。アクションがオブジェクトの作成やアクセスに成功した場合、この名前によって、JSP 細部のスクリプトタグでもこのオブジェクトを使用できます。

シンタックス

```
<jsp:useBean id="name" scope="scope"
             class="className" |
             class="className" type="typeName" |
             beanName="beanName" type="typeName" |
             type="typeName">
// optional body
</jsp:useBean>
```

属性

表 3-4 に有効な属性を示します。

表 3-4 <jsp:useBean> の属性

属性	説明
id	オブジェクト固有の識別名
scope	オブジェクトのライフサイクルは次のどれかになる • page: リクエストが 2 ページ以上あってもオブジェクトはこのページだけで有効。このオブジェクトはほかのページに転送されない • request: オブジェクトはリクエストオブジェクトにバインドされる (このオブジェクトは getAttribute(name) を使って取得。ここで、name はオブジェクトの id)。したがって、このリクエストが有効であるかぎり使用可能 • session: オブジェクトはセッションオブジェクトにバインドされる (このオブジェクトは getValue(name) を使って取得。ここで、name はオブジェクトの id)。したがって、このセッションが有効であるかぎり使用可能。この scope を使うには、この JSP をアクティブにする必要がある • application: オブジェクトは ServletContext にバインドされる (このオブジェクトは、getAttribute(name) を使って取得。name はオブジェクトの id)。したがって、特に破棄されなければ、このアプリケーションが存在するかぎり使用可能

表 3-4　<jsp:useBean> の属性 (続き)

属性	説明
class	有効な Bean クラス名。存在しない場合は、Bean をインスタンス化するときに使う。type が指定されている場合は、class を type に割り当てる必要がある。beanName と class の両方を同じ Bean に指定できない
beanName	a.b.c (クラス名) または a/b/c (リソース名) 形式の Bean の有効な名前。beanName と class の両方を同じ Bean に指定できない。beanName 属性は式として要求時に評価される
type	Bean 変数 type を定義する。この属性を使うと、変数 type を、指定された実装クラスの変数タイプと区別できる。タイプは、クラス自体、クラスのスーパークラス、または指定されたクラスに実装されたインタフェース。指定されていない場合、値は class 属性の値と同じ

例

次の例で、com.iplanet.myApp.User タイプで currentUser という名前の Bean の作成、または既存の Bean へのアクセスを示します。

```
<jsp:useBean id="currentUser" class="com.iplanet.myApp.User" />
```

この例では、オブジェクトがこのセッションに存在します。その場合、オブジェクトには WombatType を持つローカル名 wombat が指定されます。このオブジェクトのクラスが正しくない場合は、ClassCastException が発生する可能性があり、オブジェクトが定義されていない場合は、InstantiationException が発生する可能性があります。

```
<jsp:useBean id="currentUser"
          type="com.iplanet.myApp.User"
          scope="session" />
```

詳細については、75 ページの「例」を参照してください。

<jsp:setProperty>

<jsp:setProperty> アクションは Bean プロパティの値を設定します。<jsp:userBean> タグボディの内側または外側の両方で、Bean プロパティを設定できます。プロパティの値は、式を使って決めたり、リクエストオブジェクトから直接決めたりできます。

シンタックス

```
<jsp:setProperty name="beanName"
                 property="propertyName"
                 param="requestParameter" | value="value"
</jsp:setProperty>
```

属性

表 3-5 に有効な属性を示します。

表 3-5　<jsp:setProperty> の属性

属性	説明
name	プロパティを設定する Bean の名前。名前は、<jsp:useBean> を使って、ファイル内であらかじめ定義しておく必要がある
property	値を設定する Bean プロパティの名前。プロパティは、有効な Bean プロパティでなければならない。property="*" の場合、タグは、リクエストオブジェクトのパラメータを繰り返して、パラメータ名および値のタイプを Bean のプロパティ名およびセッターメソッドタイプと比較し、比較したパラメータの値に対して一致した各プロパティを設定する。パラメータの値が空の場合、対応するプロパティは変更されない。パラメータの以前の値はすべて保持される
param	Bean プロパティに与える値を持つリクエストオブジェクトパラメータの名前。param を削除すると、リクエストのパラメータ名は Bean のプロパティ名と同じであると想定される。param がリクエストオブジェクト内に設定されていない場合や空の値を持つ場合は、<jsp:setProperty> アクションによる効果はない。<jsp:setProperty> アクションは、param と value のどちらの属性も持たないことがある
value	指定されたプロパティに割り当てる値。この属性は式を値として受け入れる。この式は要求時に評価される。<jsp:setProperty> アクションは、param と value のどちらの属性も持たないことがある

例

この例では、name および permissions プロパティが次のように設定されています。

```
<jsp:useBean id="currentUser" class="com.iplanet.myApp.User" >
   <jsp:setProperty name="currentUser"
                    property="name"
                    param="name">
   <jsp:setProperty name="currentUser"
                    property="permissions"
                    param="permissions">
</jsp:useBean>
```

この例では、プロパティ name の値を、対応するリクエストパラメータに設定します。これらのプロパティは次の name でも呼ばれます。

```
<jsp:setProperty name="myBean" property="name" param="name" />

<jsp:setProperty name="myBean" property="name"
              value="<%= request.getParameter(¥"name¥" %>)" />
```

<jsp:getProperty>

<jsp:getProperty> アクションは、文字列に変換された Bean プロパティの値を出力ストリームに配置します。

シンタックス

```
<jsp:getProperty name="beanName"
                 property="propertyName">
```

属性

表 3-6 に有効な属性を示します。

表 3-6　<jsp:getProperty> の属性

属性	説明
name	プロパティを取得する Bean の name。name は、<jsp:useBean> を使って、ファイル内であらかじめ定義しておく必要がある
property	値を取得する Bean property の名前。property は、有効な Bean プロパティでなければならない。

例

```
<jsp:getProperty name="currentUser" property="name" />
```

<jsp:include>

<jsp:include> アクションは、現在のページのコンテキストを保持しながら、要求時に、指定されたページを現在のページに取り込みます。この方法を使うと、取り込まれたページは出力ストリームに書き込まれます。

コンパイル時にリソースを取り込む <%@ include%> ディレクティブを使って、ほかのリソースを取り込むこともできます。ファイルを取り込む方法については、82 ページの「ほかのリソースの取り込み」を参照してください。

シンタックス

```
<jsp:include page="URI" flush="true|false" />
```

属性

表 3-7 に有効な属性を示します。

表 3-7　<jsp:include> の属性

属性	説明
ページ	ページへの絶対参照または相対参照を取り込む。絶対参照の場合、このフィールドはスラッシュ (/) で始まり、アプリケーションのコンテキストルートがそのルートになる。相対参照の場合、このフィールドは取り込みを実行している JSP ファイルに対応しており、要求時に評価される式を持つ
flush	取り込んだページを出力ストリームに書き出すかどうかを判断する

例

```
<jsp:include page="/templates/copyright.html" flush="true" />
```

<jsp:forward>

<jsp:forward> アクションを使うと、現在のページの実行を終了して、現在のページと同じコンテキスト内にあるスタティックリソース、JSP ページ、または Java Servlet への現在のリクエストを実行時にディスパッチできます。このアクションは RequestDispatcher インタフェースの forward() メソッドと同じです。

シンタックス

```
<jsp:forward page="URL" />
```

属性

表 3-8 に有効な属性を示します。

表 3-8　<jsp:forward> の属性

属性	説明
ページ	取り込むページを指す有効な URL。この属性は要求時に評価される式を含んでいる場合がある。式は有効な URL でなければならない

注	ページ出力が <% page buffer="none" %> を使ってバッファから書き出され、データが出力ストリームにすでに書き込まれている場合、このタグはランタイムエラーになります。

例

```
<jsp:forward page="/who/handleAlternativeInput.jsp" />
```

次の要素は、ダイナミックな状態に基づいてスタティックページを転送する方法を示しています。

```
<% String whereTo = "/templates/"+someValue; %>
<jsp:forward page="<%= whereTo %>" />
```

<jsp:plugin>

<jsp:plugin> アクションを使うと、JSP の作成者は、適切なクライアントブラウザの従属構造体 (object または embed) を含む HTML を作成して、必要に応じて適切な Java プラグインをダウンロードし、アプレットまたは JavaBeans コンポーネントを実行するようにブラウザに指示できます。<jsp:plugin> タグの属性は、要素を表現するための設定データを提供します。

<jsp:plugin> タグは、リクエストを出したユーザエージェントの適切な <object> または <embed> タグのどちらかに置換され、レスポンスの出力ストリームに送信されます。

関連する次の 2 つのアクションは <jsp:plugin> アクション内だけで有効です。

- <jsp:params> は、アプレットまたは JavaBeans コンポーネントにパラメータブロックを送信します。個々のパラメータは次のように設定されます。

  ```
  <jsp:param name="name" value="value">
  ```

  このセクションは、</jsp:params> で終わります。名前および値はコンポーネントに依存します。

- object または embed がサポートされていないなどの問題でプラグインが起動できない場合、<jsp:fallback> はブラウザのコンテンツを示します。<jsp:plugin> で囲まれた部分で障害が発生すると、このタグの本体がブラウザに表示されます。次のようにします。

  ```
  <jsp:plugin ...>
      <jsp:fallback><b>Plugin could not be
      started!</b></jsp:fallback>
  </jsp:plugin>
  ```

プラグインが起動しても、アプレットまたは JavaBeans コンポーネントが見つからないか、または起動できない場合は、プラグイン固有のメッセージがユーザに送信されます。このメッセージは、多くの場合、ClassNotFoundException を報告するポップアップウィンドウとして表示されます。

シンタックス

```
<jsp:plugin type="bean|applet"
          code="objectCode"
          codebase="objectCodebase"
          { align="alignment" }
          { archive="archiveList" }
```

```
            { height="height" }
            { hspace="hspace" }
            { jreversion="jreversion" }
            { name="componentName" }
            { vspace="vspace" }
            { width="width" }
            { nspluginurl="URL" }
            { iepluginurl="URL" } >
            { <jsp:params
                    <jsp:param name="paramName" value="paramValue" />
                </jsp:params> }
            { <jsp:fallback> fallbackText </jsp:fallback> }
</jsp:plugin>
```

属性

<jsp:plugin> タグは、その属性の多くを HTML の <applet> および <object> タグから取得します。<applet> は HTML 3.2 で定義されて廃止され、<object> は HTML 4.01 で定義されています。これらのタグについては、HTML 4.01 の公式仕様書を参照してください。

http://www.w3.org/TR/REC-html40/

表 3-9 に有効な属性を示します。

表 3-9 <jsp:plugin> の属性

属性	説明
type	コンポーネントのタイプを識別する (bean または applet)
code	HTML の仕様に準拠
codebase	HTML の仕様に準拠
align	HTML の仕様に準拠
archive	HTML の仕様に準拠
height	HTML の仕様に準拠
hspace	HTML の仕様に準拠
jreversion	コンポーネントが動作するために必要な JRE の仕様書のバージョン番号を指定する。デフォルトは 1.1
name	HTML の仕様に準拠
vspace	HTML の仕様に準拠
title	HTML の仕様に準拠
width	HTML の仕様に準拠

表 3-9　<jsp:plugin> の属性 (続き)

属性	説明
nspluginurl	Netscape Navigator の JRE プラグインをダウンロードできる URL。デフォルトは定義済みの実装
iepluginurl	Microsoft Internet Explorer の JRE プラグインをダウンロードできる URL。デフォルトは定義済みの実装

例

```
<jsp:plugin type="applet"
          code="Tetris.class"
          codebase="/html" >
   <jsp:params>
      <jsp:param name="mode" value="extraHard"/>
   </jsp:params>

   <jsp:fallback>
      <p> unable to load Plugin </p>
   </jsp:fallback>
</jsp:plugin>
```

暗黙的オブジェクト

JSP 1.1 仕様書では、すべての JSP に暗黙的に使用できる複数のオブジェクトを定義しています。これらのオブジェクトは、<jsp:useBean> などであらかじめ定義しておかなくても JSP の任意の場所で参照できます。

表 3-10 に、すべての JSP に暗黙的に使用できるオブジェクトを示します。

表 3-10　すべての JSP に暗黙的に使用できるオブジェクト

オブジェクト	説明	「Scope」	Java のタイプ
request	この JSP の実行をトリガしたリクエスト	request	プロトコルに従属する javax.servlet.ServletRequest のサブタイプ。例 : javax.servlet.HttpServletRequest
response	リクエストに対するレスポンス (例 : 呼び出し側に返されたページとそのパス)	ページ	プロトコルに従属する javax.servlet.ServletResponse のサブタイプ。例 : javax.servlet.HttpServletResponse
pageContext	JSP のページコンテキスト	ページ	javax.servlet.jsp.PageContext

表 3-10　すべての JSP に暗黙的に使用できるオブジェクト (続き)

オブジェクト	説明	「Scope」	Java のタイプ
session	呼び出し側にセッションオブジェクトが作成されると関連付けられる	session	javax.servlet.http.HttpSession
アプリケーション	getServletConfig()、getContext() を介した Servlet の設定オブジェクトからの、この JSP の Servlet コンテキスト	アプリケーション	javax.servlet.ServletContext
out	出力ストリームに書き込みを行うオブジェクト	ページ	javax.servlet.jsp.JspWriter
config	この JSP の Servlet 設定オブジェクト (ServletConfig)	ページ	javax.servlet.ServletConfig
ページ	現在のリクエストを処理しているこのページのクラスのインスタンス	ページ	java.lang.Object
exception	エラーページ専用の、エラーページを起動する検出されていない Throwable 例外	ページ	java.lang.Throwable

たとえば、<%= request.getParameter("*param*"); %> のように、リクエストパラメータの一つを使ってリクエストオブジェクトを参照できます。

高度な JSP プログラミング

この節では、高度なプログラミングテクニックを使うための手順について説明します。この節には次の項があります。

- ほかのリソースの取り込み
- JavaBeans の使用法
- ビジネスオブジェクトへのアクセス

ほかのリソースの取り込み

JSP の重要な特徴は、ほかのページが生成するリソースやその結果を実行時にダイナミックに取り込む機能があることです。スタティック HTML ページのコンテンツを取り込んだり、別の JSP を処理してその結果を出力ページに取り込んだりできます。

たとえば、コーポレートヘッダーおよびフッターは、取り込まれた要素だけを持つページスタブを作成することによって、各ページに取り込むことができます。条件付きの基準に従ってページ全体を取り込めるので、単層型ナビゲーションバーやコーポレートヘッダーを単に挿入するよりもはるかに柔軟性が向上します。

JSP にリソースを取り込むには、次の 2 つの方法があります。

- `<%@ include%>` ディレクティブの場合

  `<%@ include file="`*filename*`" %>`

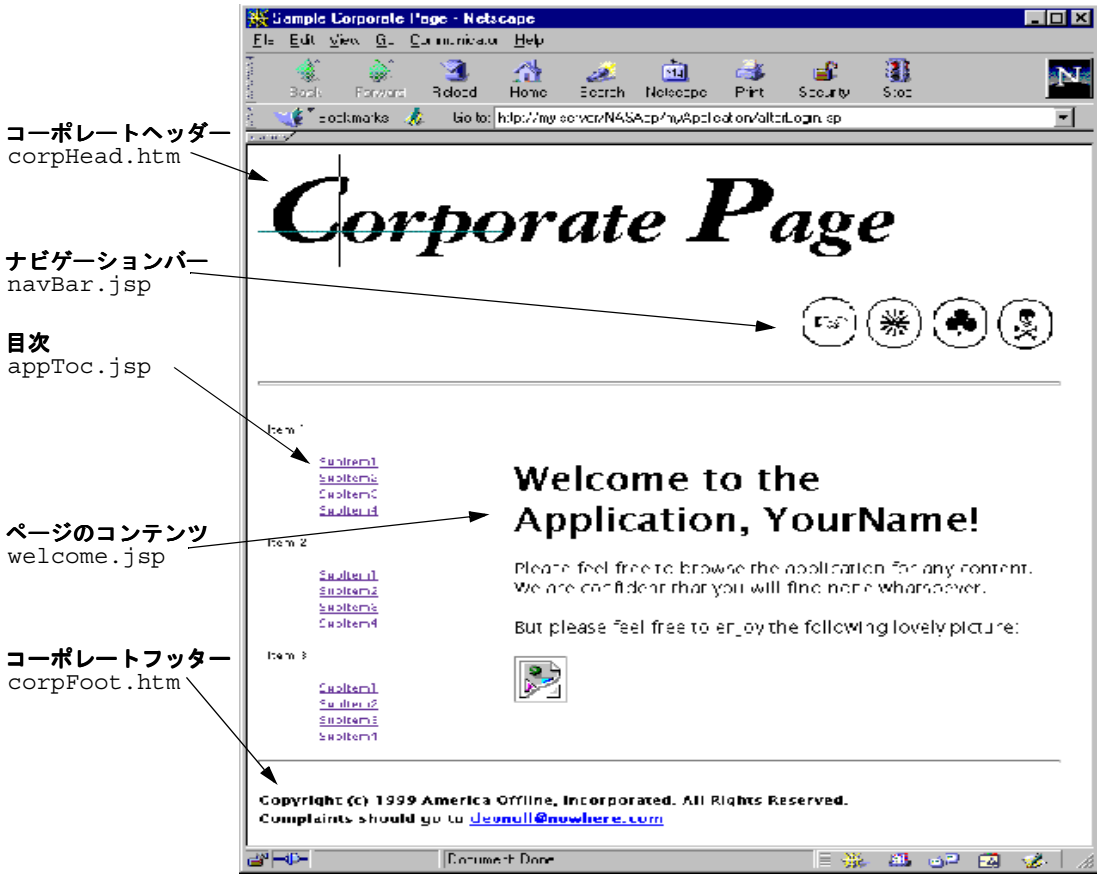
- `<jsp:include>` アクションの場合

  `<jsp:include page="`*URI*`" flush="true|false" />`

`<%@ include%>` ディレクティブを使ってリソースを取り込む場合、そのリソースは、JSP が Servlet にコンパイルされるときに取り込まれます。したがって、このリソースは元の JSP の一部として処理されます。取り込んだリソースも JSP である場合、そのコンテンツは親 JSP とともに処理されます。詳細については、66 ページの「ディレクティブ」を参照してください。

`<jsp:include>` アクションを使ってリソースを取り込む場合、リソースは、JSP が呼び出されたときに取り込まれます。詳細については、72 ページの「アクション」を参照してください。

次の例は、ある JSP からアクセスした場合に、ページの各部分が個々のリソースからどのように提供されるかを示します。このページのソースコードは、リソースを取り込むための方法を示します。スタティックリソースは `<jsp:include>` アクションを使って取り込まれ、ダイナミックリソースは `<%@ include%>` ディレクティブを使って取り込まれます。



afterLogin.jsp

```
<html><head><title>Sample Corporate Page</title></head><body>

<p align="left"><jsp:include page="corpHead.htm" flush="true" /></p>
<%@ include file="navBar.jsp" %>
<hr size="3">

<table border=0><tr>
<td width="25%"><%@ include file="appToc.jsp" %></td>
<td width="75%"><%@ include file="appToc.jsp" %></td>
</tr></table>

<hr>
<p align="left"><jsp:include page="corpFoot.htm" flush="true" /></p>
</body></html>
```

JavaBeans の使用法

JSP は、JavaBeans をインスタンス化したりアクセスしたりするためのタグをサポートしています。Bean は計算を行ってリザルトセットを取得します。この結果は Bean のプロパティとして格納されます。JSP では、Bean の作成とプロパティの調査が自動的にサポートされます。

Bean 自体は、JavaBeans の仕様に従って作成された個々のクラスです。JavaBeans については、以下のサイトを参照してください。

`http://java.sun.com/beans`

一般に、Bean ではゲッターメソッドとセッターメソッドを使って、Bean プロパティを取得したり、設定したりします。ゲッターメソッドには `get`*Xxx*`()` という名前が付けられます。ここで、*Xxx* は *xxx* と呼ばれるプロパティです (メソッド名の先頭文字は大文字)。これに対応する `set`*Xxx*`()` という名前のセッターがある場合、このセッターは、ゲッター戻り値とパラメータタイプが同じである必要があります。

この機能では標準の JavaBeans をサポートしていますが、EJB は対象外です。JSP から EJB にアクセスする方法については、84 ページの「ビジネスオブジェクトへのアクセス」を参照してください。JSP 0.92 仕様では、リクエストおよびレスポンスオブジェクトは「暗黙的 Bean」からアクセスしました。JSP 1.1 仕様では、このサポート内容が変更され、リクエストおよびレスポンスオブジェクトを含む複数のオブジェクトを広範囲に、かつ暗黙的に使用できます。詳細については、80 ページの「暗黙的オブジェクト」を参照してください。

ビジネスオブジェクトへのアクセス

JSP は、実行時に Servlet にコンパイルされるため、すべてのサーバプロセス (EJB を含む) にアクセスできます。Java コードがエスケープタグに埋め込まれている場合に限り、Servlet からのアクセスと同じ方法で、Bean や Servlet にアクセスできます。

ここで記述する EJB にアクセスするためのメソッドは、Servlet からのアクセスに使うメソッドと同じです。EJB へのアクセスについては、47 ページの「ビジネスロジックコンポーネントへのアクセス」を参照してください。

次の例では、カートのリモートインタフェースをインポートし、ユーザのセッション ID を使ってカートへのハンドルを作成して、`ShoppingCart` という名前の EJB にアクセスする JSP を示しています。

```
<%@ import cart.ShoppingCart %>;
...
<% // ユーザのセッションおよびショッピングカートを取得します。
   ShoppingCart cart =
(ShoppingCart)session.getValue(session.getId());
```

```
   // ユーザがカートを持っていない場合は新規に作成します。
   if (cart == null) {
       cart = new ShoppingCart();
       session.putValue(session.getId(), cart);
   } %>
...
<%= cart.getDataAsHTML() %>
```

次の例は、カートのプロクシまたはハンドルを検索する JNDI (Java Naming Directory Interface) の使用法を示しています。

```
<% String jndiNm = "java:/comp/ejb/ShoppingCart";
   javax.naming.Context initCtx;
   Object home;
       try {
          initCtx = new javax.naming.InitialContext;
       } catch (Exception ex) {
          return null;
       }
       try {
          java.util.Properties props = null;
          home = initCtx.lookup(jndiNm);
       }
       catch(javax.naming.NameNotFoundException e)
       {
          return null;
       }
       catch(javax.naming.NamingException e)
       {
          return null;
       }
       try {
          IShoppingCart cart = ((IShoppingCartHome) home).create();
             ...
       } catch (...) {...}
%>
...
<%= cart.getDataAsHTML() %>
```

注 上記の `getDataAsHTML()` のように、ページが受け入れ可能なフォーマットに生データを変換するには、EJB メソッドを指定する必要があります。

JSP の配置

iPlanet Application Server が JSP を配置する方法は 2 つあります。未登録 JSP か 登録 JSP のいずれかで配置します。

未登録 JSP

未登録 JSP は、`AppPath` 内の対応するディレクトリ構造体 (`applicationName/moduleName`) にコピーすることによって配置します。これらの JSP は、次のような URL アクセスを使って起動します。

`http://`*server*`:`*port*`/`*AppPrefix*`/`*ModuleName*`/`*JSPFileName*

詳細については、87 ページの「JSP の起動」を参照してください。

登録 JSP

iPlanet Application Server では、XML を使って JSP を `GUID` に登録できます。これによって、JSP は、ロードバランスのような iPlanet Application Server の付加価値機能を使用できます。この機能は、Servlet 2.2 仕様に記載されているように、`<jsp-file>` エントリとともに XML ファイルを使うことによって実行されます。

次の XML ファイルは、登録 JSP の配置記述子の例です。これは `web.xml` ファイルです。

```
<?xml version="1.0" ?>
<!DOCTYPE web-app>
<web-app>
   <display-name> An Example Registered JSP File </display-name>
   <Servlet>
       <servlet-name>JSPExample</servlet-name>
       <jsp-file>JSPExample.jsp</jsp-file>
   </servlet>
   <servlet-mapping>
       <servlet-name>JSPExample</servlet-name>
       <url-pattern>/jspexample</url-pattern>
   </servlet-mapping>
</web-app>
```

これは `ias-web.xml` ファイルです。

```
<?xml version="1.0" ?>
<ias-web-app>
   <Servlet>
       <servlet-name>JSPExample</servlet-name>
       <guid>{aaaabbbb-A456-161A-8be4-0800203942f2}</guid>
   </servlet>
</ias-web-app>
```

この例では、ias-MyApp.xml ファイルに指定された GUID に JSP を登録します。この例では Servlet 名を JSPExample としていますが、.jsp 拡張子は不要です。JSPExample.jsp という Servlet 名にすることも可能です。

この JSP は、次の例のどちらかの URL からアクセスされます。

- http://*server*:*port*/*AppPrefix*/*ModuleName*/JSPExample
- http://*server*:*port*/*AppPrefix*/*ModuleName*/JSPExample.jsp (XML ファイルの servlet-name エントリが JSPExample.jsp の場合に使用)

JSP の起動

JSP は、プログラムで Servlet から起動するか、URL を使ってクライアントから直接アドレス指定することによって起動します。また、JSP を取り込むこともできます。詳細については、82 ページの「ほかのリソースの取り込み」を参照してください。

URL による JSP の呼び出し

JSP は、アプリケーションページにリンクとして埋め込まれている URL を使って呼び出すことができます。この節では、標準の URL を使って JSP を起動する方法について説明します。

特定のアプリケーションでの JSP の起動

特定のアプリケーションの一部である JSP は次のようにアドレス指定します。

http://*server*:*port*/*AppPrefix*/*ModuleName*/*jspName*?*name=value*

表 3-11 で、URL の各セクションを説明します。

表 3-11 URL の各セクション

URL 要素	説明
server:*port*	リクエストを処理する Web サーバのアドレスおよびオプションのポート番号

表 3-11　URL の各セクション (続き)

URL 要素	説明
AppPrefix	iPlanet Application Server アプリケーションの URL であることを Web サーバに示す。リクエストは iPlanet Application Server の Executive Server に転送される。この要素は、レジストリのエントリ `SSPL_APP_PREFIX` を使って設定する
moduleName	Web モジュールの名前 (この名前はサーバー上で一意であること)
jspName	JSP のファイル名。`.jsp` という拡張子を持つ
`?`*name=value*`...`	JSP に対するオプションの *name=value* パラメータ。`request` オブジェクトからアクセス可能

次のようにします。

```
http://www.mycompany.com/BookApp/OnlineBookings/directedLogin.jsp
```

JSP の汎用アプリケーションの使用には、Servlet の汎用アプリケーションの使用と同様の要件と制限があります。XML ファイルが登録されている `Default` と呼ばれるアプリケーションが必要です。`/servlet/` エントリを使って Servlet または JSP にアクセスする URL リクエストは、汎用アプリケーション `Default` に送信されます。この要件については、56 ページの「汎用アプリケーション Servlet の起動」を参照してください。

汎用アプリケーションでの JSP の起動

特定のアプリケーションの一部でない JSP は次のようにアドレス指定されます。

`http://`*server*`:`*port*`/servlet/`*jspName*`?`*name=value*

表 3-12 で、URL の各セクションを説明します。

表 3-12　URL の各セクション

URL 要素	説明
server`:`*port*	リクエストを処理する Web サーバのアドレスおよびオプションのポート番号
`servlet`	汎用 Servlet オブジェクトの URL であることを Web サーバに示す
jspName	JSP の名前。`.jsp` という拡張子を持つ
`?`*name=value*`...`	JSP に対するオプションの *name=value* パラメータ。`request` オブジェクトからアクセス可能

次のようにします。

```
http://www.Who.com/servlet/calcMort.jsp?rate=8.0&per=360&bal=180000
```

Servlet からの JSP の起動

Servlet は、次のどちらかの方法で JSP を呼び出すことができます。

- `RequestDispatcher` インタフェースの `include()` メソッドが、JSP を呼び出し、JSP が返されてから処理を続行する。
- `RequestDispatcher` インタフェースの `forward()` メソッドが、対話の制御を JSP に渡す。

これらのメソッドについては、51 ページの「クライアントへの結果の配信」を参照してください。

次のようにします。

```
public class ForwardToJSP extends HttpServlet
{
   public void service (HttpServletRequest req,
                        HttpServletResponse res)
   throws ServletException, IOException
   {
       RequestDispatcher rd = req.getRequestDispatcher("/test.jsp");
       rd.forward(req, res);
   }
}
```

JSP 1.1 タグの要約

次の節では、JSP 1.1 タグの概要について説明します。

ディレクティブ

```
<%@ page|include|taglib { attr="value" }* %>

   attr: page language="java"
                 extends="className"
                 import="className{,+}"
                 session="true|false"
                 buffer="none|sizeInKB"
                 autoFlush="true|false"
```

```
                isThreadSafe="true|false"
                info="text"
                errorPage="jspUrl"
                isErrorPage="true|false"
                contentType="mimeType{;charset=charset}"

          include file="filename"

       taglib uri="uriToTagLibrary"
          prefix="prefixString"
```

詳細については、66 ページの「ディレクティブ」を参照してください。

式

```
<%= expression %>
```

詳細については、70 ページの「スクリプト要素」を参照してください。

スクリプトレット

```
<% scriptlet %>
```

詳細については、70 ページの「スクリプト要素」を参照してください。

コメント

```
<%-- コメント --%>        クライアントに渡されない JSP コメント
<!-- コメント -->         クライアントに渡される標準 HTML コメント
<% /** コメント **/ %>    スクリプトレットにカプセル化されてクライアントに渡される Java コメント
```

詳細については、65 ページの「コメント」を参照してください。

Bean 関連アクション

```
<jsp:useBean id="name" scope="scope"
             class="className" |
             class="className" type="typeName" |
             beanName="beanName" type="typeName" |
             type="typeName">
// optional body
</jsp:useBean>
```

```
<jsp:setProperty name="beanName"
                 property="propertyName"
                 param="requestParameter" | value="value"
</jsp:setProperty>

<jsp:getProperty name="beanName"
            property="propertyName">
```

詳細については、72 ページの「アクション」を参照してください。

その他のアクション

```
<jsp:include page="relativeUrl"
             flush="true|false" />

<jsp:forward page="URL" />

<jsp:plugin type="bean|applet"
             code="objectCode"
             codebase="objectCodebase"
             { align="alignment" }
             { archive="archiveList" }
             { height="height" }
             { hspace="hspace" }
             { jreversion="jreversion" }
             { name="componentName" }
             { vspace="vspace" }
             { width="width" }
             { nspluginurl="URL" }
             { iepluginurl="URL" } >
             { <jsp:params
                 <jsp:param name="paramName" value="paramValue" />
                 </jsp:params> }
          { <jsp:fallback> fallbackText </jsp:fallback> }
</jsp:plugin>
```

詳細については、72 ページの「アクション」を参照してください。

JSP 1.1 用カスタムタグの変更

iPlanet Application Server のカスタムタグは、JSP 1.1 用に変更が必要な場合があります。理由は次のとおりです。

- `.tld` ファイルは次の場所にある DTD に適合しない

  `http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd`

  たとえば、`prefix` 属性へのすべての参照を `shortname` に変更する必要がある

- 次の `DOCTYPE` 要素がない

```
<!DOCTYPE taglib
PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
"http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
```

JSP コマンドラインコンパイラを使う場合にこれらの変更が必要です。このコンパイラについては、92 ページの「JSP のコンパイル : コマンドラインコンパイラ」を参照してください。

JSP のコンパイル : コマンドラインコンパイラ

iPlanet Application Server Service Pack 3 では、Apache Tomcat 3.2 から Jasper JSP コンパイラを使って JSP 1.1 互換のソースファイルを Servlet にコンパイルします。このバージョンの Jasper で使用可能なすべての機能が、iPlanet Application Server 環境で利用できます。

注	Jasper は iPlanet Application Server の必要条件を満たすように変更されているため、iPlanet Application Server とともに提供される Jasper バージョンを使ってください。ほかのバージョンは iPlanet Application Server で動作しません。

開発者は、配置前に JSP コンパイラを使って JSP ファイルのシンタックスチェックを実行できます。また、WAR ファイルをアプリケーションサーバに配置する前に、JSP ファイルを WAR ファイルにプリコンパイルすることもできます。

`jspc` コマンドラインツールは、*install_dir*`/ias/bin` の下にあります。このディレクトリがパスに含まれていることを確認してください。`jspc` コマンドのフォーマットは次のとおりです。

`jspc [`*options*`] `*jsp_files*

jsp_files は次のどちらかになります。

files	コンパイルされる 1 つまたは複数の JSP ファイル
`-webapp` *dir*	Web アプリケーションがあるディレクトリ。指定したディレクトリとそのサブディレクトリ内のすべての JSP がコンパイルされる。WAR、JAR、または ZIP ファイルを指定できない。最初に、`iasdeploy` を使ってオープンディレクトリ構造にそれらのファイルを配置する必要がある

`jspc` コマンドの基本的な *options* は次のとおりです。

`-q`	消音モードを有効にする (`-v0` と同じ)。重大なエラーメッセージだけ表示する
`-d` *dir*	コンパイル済み JSP の出力ディレクトリを指定する。コンパイルされていない JSP が含まれているディレクトリに基づいてパッケージディレクトリが自動的に生成される。デフォルトのトップレベルディレクトリは `jspc` が起動されるディレクトリである
`-p` *name*	すべての指定済み JSP にターゲットパッケージの名前を指定し、`-d` オプションによって実行されるデフォルトのパッケージ生成をオーバーライドする
`-c` *name*	最初にコンパイルされる JSP のターゲットクラス名を指定する。後続の JSP は影響を受けない
`-uribase` *dir*	コンパイルに相対的な URI ディレクトリを指定する。コマンドに一覧表示された JSP ファイルだけに適用され、`-webapp` で指定された JSP ファイルには適用されない `uriroot` に相対的な各 JSP ファイルの場所。指定されない場合、デフォルトは `/`
`-uriroot` *dir*	URI ファイルを解決するルートディレクトリを指定する。コマンドに一覧表示された JSP ファイルだけに適用され、`-webapp` で指定された JSP ファイルには適用されない このオプションを指定しない場合は、`WEB-INF` サブディレクトリにある最初の JSP ページのすべての親ディレクトリを検索する。WEB-INF サブディレクトリを持つ JSP ページにもっとも近いディレクトリが使われる どの JSP の親ディレクトリにも `WEB-INF` サブディレクトリがない場合は、`jspc` が起動されるディレクトリが使われる
`-webinc` *file*	`-webapp` オプションの部分的な Servlet マッピングを作成する。これを `-webapp` ファイルに貼り付け可能

`-webxml` *file*	`-webapp` オプションの `web.xml` ファイル全体を作成する
`-ieplugin` *class_id*	Internet Explorer の Java プラグイン COM クラス ID を指定する。`<jsp:plugin>` タグによって使われる

`jspc` コマンドの高度な *options* は次のとおりです。

`-v` [*level*]	詳細モードを有効にする。*level* はオプション。デフォルトは 2。可能な *level* 値は次のとおり • `0` - 重大なエラーメッセージのみ • `1` - エラーメッセージのみ • `2` - エラーおよび警告メッセージのみ • `3` - エラー、警告、および情報メッセージ • `4` - エラー、警告、情報、およびデバッグメッセージ
`-dd` *dir*	コンパイルされる JSP のリテラル出力ディレクトリを指定する。パッケージディレクトリは作成されない。デフォルトは、`jspc` が起動されるディレクトリ
`-mapped`	各 HTML 行の `write` 呼び出しと、JSP ファイルの各行の場所を記述するコメントを生成する。デフォルトでは、すべての隣接した `write` 呼び出しが結合され、場所のコメントは生成されない
`-die` [*code*]	重大なエラーが発生した場合に、JVM を終了し、エラー戻り *code* を生成する。*code* がない場合や解析できない場合は、デフォルトで `1` に設定される
`-webinc` *file*	`-webapp` オプションの部分的な Servlet マッピングを作成する。これを `-webapp` ファイルに貼り付け可能
`-webxml` *file*	`-webapp` オプションの `web.xml` ファイル全体を作成する
`-ieplugin` *class_id*	Internet Explorer の Java プラグイン COM クラス ID を指定する。`<jsp:plugin>` タグによって使われる

JSP がコンパイルされると、それに対してパッケージが作成されます。パッケージは、*install_dir*`/ias/APPS/`*appName*`/`*moduleName*`/WEB-INF/compiled_jsp/` にあります。コードを個々のモジュールとして配置する場合、*moduleName* は `modules` です。パッケージ名は、iPlanet Application Server のデフォルトのパッケージプレフィックス名である `jsp.APPS` で始まる必要があります。

iPlanet Application Server 用に JSP をコンパイルするときは、jspc の基本オプションを使います。iPlanet Application Server では標準の Jasper 命名規則を使わないので、生成されるファイル名、クラス名 (-c)、パッケージ (-p)、およびディレクトリ (-d) を指定する必要があります。

たとえば、fortune.jsp を fotune.java にプリコンパイルするには、次のコマンドを使います。

```
cd install_dir/ias/APPS/fortune/fortune
jspc -d WEB-INF/compiled_jsp -p jsp.APPS.fortune -c fortune fortune.jsp
```

fortune.java ファイルとそれぞれのクラスは、次のディレクトリに生成されます。

```
install_dir/ias/APPS/fortune/fortune/WEB-INF/compiled_jsp/jsp/APPS/fortune
```

fortune が JSP のパッケージ名であり、iPlanet Application Server では jsp.APPS をプレフィックスとして使うので、fortune.class ファイルのパッケージ名は、jsp.APPS.fortune になります。

注 iPlanet Application Server 6.5 では、Forte For Java Internet Edition 3.0 によって、コンパイルされた Servlet コードをデバッグできます。ただし、コンパイルされていない JSP ページはデバッグできません。

JSP コンパイラのその他のマニュアルは次の Jakarta サイトにあります。

```
http://jakarta.apache.org/tomcat-4.0/jakarta-tomcat-4.0/jasper/doc/jspc.html
```

付加価値機能

ここでは、iPlanet の付加価値機能の概要について説明します。

- カスタムタグエクステンション
- JSP ロードバランス
- JSP ページキャッシュ

カスタムタグエクステンション

JSP 1.1 仕様は、ユーザ定義のカスタムタグプロトコルをサポートしています。この仕様にタグについての規定はありませんが、iPlanet Application Server では、付加価値機能として JSP 1.1 定義のタグエクステンションプロトコルに従うカスタムタグがサポートされています。詳細については、JSP 1.1 仕様書の第 5 章を参照してください。

一部のタグは LDAP およびデータベースクエリをサポートしますが、この仕様では基本的にサポートされないため、それ以外のタグは JSP 内で条件付きでサポートしています。

JSP のページキャッシュをサポートするために、iPlanet Application Server は Cache タグリブとともに配布されます。詳細については、113 ページの「JSP ページキャッシュ」を参照してください。

iPlanet Application Server で使われるほかのタグは、GX タグの変換をサポートするために内部使用に限定されています。これらのタグは、GX タグがサポートされている JSP 0.92 ページから JSP 1.1 ページを生成する際に使います。外部使用はできません。

iPlanet Application Server には、次のタグリブが用意されています。

- Query
- LDAP
- Conditional
- Attribute

カスタムタグエクステンションの例については、*install_dir*/ias/ias-samples/iastags/ ディレクトリのサンプルを参照してください。

データベースクエリタグリブ

クエリタグリブは、リザルトセットをループするループやカラム値を表示する表示タグとともに、JSP ページでの行セットの宣言をサポートします。次の節ではクエリタグリブについて説明します。

useQuery タグ

useQuery タグは使用するリザルトセットを宣言します。useQuery タグは、どのクエリが行われているのか、またどのフィールドが使用可能かを定義します。useQuery が保存するリザルトセットが scope 内にすでに存在する場合、このタグの本体はスキップされます。また、行セットが作成されても無視されます。

リザルトセットが存在しない場合、作成された行セットは、指定された scope で useQuery タグの id 属性を使って、デフォルトであれば request にエクスポートされます。指定された command が使われたり、queryFile に配置されたクエリが読み込まれたりすることがあります。読み込まれたクエリのファイル名は、queryName 属性に配置されている名前です。名前が指定されていない場合は、タグの id 属性の値になります。行セットが初期化されると、execute タグが指定済みであれば実行されます。ループの外で field タグを使うには、クエリを実行する必要があります。

command 属性に指定されていないファイルからクエリが読み込まれると、そのファイルは QueryLoader クラスによって読み込まれ、キャッシュされます。2 つの属性、queryFile と queryName は連動します。queryFile はクエリファイルを検索します。属性の値が相対パスの場合、この値は RDBMS.path.query パスで検索されます。この変数が設定されていない場合は、iPlanet Application Server 固有の GX.path.query プロパティが使われます。このファイルが JSP に対して相対的に配置されていない場合、クエリファイルは次のようになります。

```
query name1 [using (ODBC, cdx, iplanet)] is select *
from Who, add where :whereClause
/* :whereClause は、指定されたパラメータ bindOnLoad の例です */

query name2 is select * from Who, add where Who.x = add.y and
Who.name = :name
/* :name は、指定されたパラメータの例です */
```

query ... is 構造体を使って指定するクエリは、スペースやタブなどがない空白行で区切ります。

シンタックス

```
<rdbm:useQuery id="export_name"
scope="[page|request|session|application]" command="select *
from..."="Who.gxq" queryName="firstQuery"
execute="[true|false]" dataSourceName="jdbc/..."
url="odbc:...">...</rdbm:useQuery>
```

param タグ

param タグは行セットにパラメータを設定します。パラメータの名前は、インデックスか、またはディクショナリに保存されている実際のパラメータ名のどちらかです。bindOnLoad パラメータは、bindOnLoad 以外のパラメータを指定する前に、useQuery タグの本体に存在している必要があります。パラメータの値は、値属性に格納されている値か、または param タグ本体のコンテンツのどちらかです。JSP 1.1 のタグは、一般にネストしないので (<%= ...%> は顕著な例外)、別のクエリからの値にパラメータをバインドする唯一の方法は、param タグの本体に field タグを配置して、param タグがその本体を値として使うようにします。これによって、タグの本体に値を配置できます。

param タグが useQuery タグ内に存在する場合は、その親クエリに直接パラメータを設定します。それ以外の場合は、loop タグが行セットを再実行する前に、useQuery タグがエクスポートした行セットにパラメータを設定します。

シンタックス

```
<rdbm:param query="query-declaration-export-name"
name="name-of-parameter" value="value" bindOnLoad="[true|false]"
type="[String|Int|Double|Float|BigDecimal|Date|Boolean|Time|Timesta
mp"
format="java-format-string for dates">value</rdbm:param>
```

loop タグ

loop タグはリザルトセットのコンテンツをループします。query 属性は、リザルトセットまたは囲んでいる useQuery タグ内を検索するときに使います。start 属性はループの開始位置の指定に使います。start はパラメータまたは属性を参照するか、定数の整数値を参照します。属性の検索には PageContext.findAttribute() を使います。この値は、開始するレコード番号または last のレコード番号を指定します。これにより、行セットは最後までスクロールされたあと、max 行に戻ります。max 属性は表示する最大レコード数の指定に使います。execute が指定されると、行セットはループ開始前に実行されます。

シンタックス

```
<rdbm:loop id="export_name"
scope="[page|request|session|application]"
query="query-declaration-export-name"
start="[request-parameter-name|request-attribute-name|last|constant
]"
max="integer-maximum-number-of-rows"
execute='{true|false]">...</rdbm:loop>
```

field タグ

field タグは、リザルトセットの特定のカラムを表示します。query 属性は、それを囲んでいる useQuery タグまたは以前に useQuery タグによってエクスポートされたリザルトセットを検索します。name 属性は表示するカラムの名前を識別します。format 属性は文字列、数値、または日付を適切なタイプにフォーマットできるようにします。urlEncode 属性は、文字列のエンコードに使用できます。カラムが null の場合、field タグの本体が出力されます。

シンタックス

```
<rdbm:field query="query-declaration-export-name" name="field name"
format="format for doubles" urlEncode="{false/true}">default
value</rdbm:field>
```

close タグ

close タグはシステムリソースを解放します。resource 属性は、エクスポートされたクエリリソース (リザルトセット) を検索し、それに対して close() を呼び出します。

シンタックス

```
<rdbm:close resource="query-declaration-export-name"/>
```

execute タグ

execute タグは識別されたクエリを実行します。

シンタックス

```
<rdbm:execute query="query-declaration-export-name"/>
```

goRecord タグ

goRecord タグは指定されたクエリを実行し、start 属性によって指定されたレコードにそのリザルトセットを移動します。start は、パラメータ、属性、または定数を参照します。start 属性に last が指定された場合は、リザルトセットを最後のレコードに移動します。

シンタックス

```
<rdbm:goRecord query="query-declaration-export-name"
execute="{false/true}"
start="[request-parameter-name|request-attribute-name|last|constant
]">
default start</rdbm:goRecord>
```

例

次のタグは、最後に出力表示を生成します。

```
<HTML>
<BODY>
<%@ taglib prefix="rdbm" uri="rdbmstags6_0.tld" %>
<h2>Now let us see</h2>
<rdbm:useQuery id="a" queryFile="dbms/queries.gxq"
   dataSourceName="jdbc/cdx">
</rdbm:useQuery>
<rdbm:useQuery id="b" queryFile="dbms/queries.gxq"
   dataSourceName="jdbc/cdx">
</rdbm:useQuery>

<table border=1 cellPadding=3>
<tr><th>name</th><th>phone</th><th>Titles Owned</th></tr>
<rdbm:loop id="loop1" query="a" max="5" execute="true">
   <tr>
   <td><rdbm:field query="a" name="name"/></td>
   <td><rdbm:field query="a" name="phone"/></td>
   <td>
       <rdbm:param query="b" id="owner" type="Int">
      rdbm:field query="a" name="id"/></rdbm:param>
          <table border=1 cellPadding=3 width="100%">
```

```
            <tr><th>title</th><th>price</th><th>artist</th></tr>
            <rdbm:loop id="loop2" query="b" max="5"execute="true">
            <tr>
            <td><rdbm:field query="b" name="title"/></td>
            <td><rdbm:field query="b" format="$#,###.00"name="price"/>
            </td>
            <td><rdbm:field query="b" name="artist"/></td>
            </tr>
            </rdbm:loop>
            </table>
    </td>
    </tr>
    </rdbm:loop>
    </table>
    </td>
    </tr>
</rdbm:loop>
</table>
<rdbm:close resource="a"/>
<rdbm:close resource="b"/>
</BODY>
</HTML>
```

結果は次のとおりです。

name	phone	Titles Owned
John Seller	555-1234	title / price / artist Bye Bye Birdie / $03.99 / Flop House
vijay	4335	title / price / artist foo / $12.00 / Bar flop House / $15.00 / spam

LDAP タグリブ

LDAP コネクションで問題となる点は、リクエスト固有のコネクションになる可能性が高いことです。つまり、現在のユーザが、そのユーザのデータの LDAP 属性を読み取れるように認証されている唯一のユーザであるということです。この理由から、LDAP 検索の実行に使うコネクションと現在のユーザ間のマッピングをプログラミン

グ可能にする LDAPAuthenticate/Authorize タグを追加する必要があります。LDAP サーバがリモートで、一般認証可能なログインを利用できない場合は、LDAPAuthenticate タグを使います。次の節では LDAP タグリブについて説明します。

authenticate タグ (コネクションとも呼ばれる)

authenticate タグは LDAPTagSearch のコンテキストで動作します。LDAPTagSearch は、findAttribute および query 属性の名前を使って PageContext から取得されるか、または親 useQuery タグを検索し LDAPTagSearch を得ることによって取得されます。url および password 属性は、LDAPTagSearch が保持する LDAPConnection の認証に使われます。url 属性がパラメータを持つ場合、つまり、この属性が標準の ldap://server:portNumber/ LDAP URL セクションのあとに : Who 値を持つ場合は、authenticate タグの本体は各パラメータに対する param タグを含んでいる必要があります。password 属性が指定されていない場合、authenticate タグの本体は password タグも含んでいる必要があります。このタグは、タグの終わりで LDAPTagSearch を認証します。

シンタックス

```
<ldap:[authenticate|connection] query="name of ldap exported query"
url="ldap://..." password="..."> </ldap:[authenticate|connection]>
```

authorize タグ

authorize タグは LDAPTagSearch のコンテキストで動作します。LDAPTagSearch は、findAttribute および query 属性の名前を使って PageContext から取得されるか、または親 useQuery タグを検索し LDAPTagSearch を得ることによって取得されます。dn 属性は、LDAPTagSearch が保持する LDAPConnection を認証するために使います。dn 属性がパラメータを持つ場合、つまり、属性が内部に : Who 値を持つ場合、authorize タグの本体は各パラメータに対する param タグを含んでいる必要があります。このタグは、タグの終わりで LDAPTagSearch を認証します。

シンタックス

```
<ldap:authorize query="name of ldap exported query"
dn="distinguished name for the user to authorize against">
</ldap:authorize>
```

param タグ

param タグは LDAP URL にパラメータを設定します。LDAP URL は、authorize タグの url 属性および dn 属性と、field タグおよび useQuery タグの url 属性で指定します。

URL の param は、.gxq ファイルのクエリパラメータと同様に、「:」が先頭に付いた Java レベルの識別子です。次のようにします。

```
ldap://iplanet.com:389/uid=:user,ou=People,dc=iplanet,dc=com
```

すべてのパラメータが、field、authenticate、authorize、または useQuery タグの終わりで解消される必要があります。389 が LDAP URL の DN セクションの前にあり、Java レベルの識別子ではないので、389 はタグではありません。

param タグ自体に値が指定されていないものとして、param タグの本体は、name 属性によって指定されたパラメータ値になります。

シンタックス

```
<ldap:param name="parameter name in authenticate userDN or query
url" query="name of ldap exported query" value="...">default
value</ldap:param>
```

password タグ

password タグは、authenticate タグのパスワードを設定します。password タグの属性として値が指定されていないものとして、password タグの本体は、param タグと同様にパスワードの値になります。password タグは、authenticate タグ内だけで有効です。

シンタックス

```
<ldap:password query="name of ldap exported query"
value="...">default value</ldap:password>
```

useQuery タグ

useQuery タグは、LDAP リポジトリの検索に使われる URL を記述します。タグの本体の終わりに、LDAPTagSearch は、id によって指定された名前を使って scope に指定されたレベルのコンテキスト内に配置されます。url プロパティには query の URL が含まれています。この URL は通常、loop タグがループするか、または field タグが表示します。これは、loop タグが本体以外ではパラメータのマッピングを指定できないからです。loop を使って結果の有無を調べると時間がかかります。field タグは URL を指定できるので、クエリを参照する必要はありませんが、参照することもできます。

URL はクエリファイルから読み込むこともできます。2 つの属性、queryFile と queryName は連動します。queryFile はクエリファイルを検索します。属性の値をファイルへの相対パスで指定すると、このファイルは LDAP.path.query で検索されます。この変数が設定されていない場合は、iPlanet Application Server 固有の GX.path.query プロパティが使われます。このファイルは、JSP に対して相対的に検索できません。クエリファイルは次のようになります。

query *name1* is
ldap://directory:389/dc=com?blah

```
query name2 is
ldap://directory:389/dc=org?blah
```

query ... is 構造体を使って指定するクエリは、スペースやタブなどがない空白行で区切ります。

シンタックス

```
<ldap:useQuery id="exported LDAPTagSearch"
scope="[page|request|session|application]" url="ldap://...
queryFile="filename for ldap query" queryName="name of the query in
the ldap query file" connection="classname of an LDAPPoolManager"
authorize="distinguished name for the user to authorize
against">...</ldap:useQuery>
```

loopEntry タグ

loopEntry タグは、複数のエントリを返す検索から生じる一連の LDAPEntries をループします。query 属性は、エクスポートされた LDAPTagSearch を指します。詳細については、102 ページの「useQuery タグ」を参照してください。start および end タグは、クエリの loop タグに指定されたとおりに動作します。useVL 属性が true の場合は、VirtualListResponse の contentCount に対応する {id}_contentCount 値がエクスポートされます。loop を経由する各パス上では、現在の LDAPEntry は、id を使って指定された scope でエクスポートされます。pre および jump 属性は、VirtualListControl 構造体内の beforeCount および jumpTo パラメータに対応します。loop が VirtualListControl を使っており、useVL 属性が設定されている場合、返されたエントリのウィンドウは VirtualListControl を使って配置されます。VirtualList の実際の公開ドラフト URL はこの場所です。

シンタックス

```
<ldap:loop[Entry] id="name of attribute to export loop'd value"
scope="[page|request|session|application]" query="name of ldap
exported query" start="request variable name" max="number"
pre="number of records before jump" jump="value of sort to jump to"
useVL="true/false"> </ldap:loop[Entry]>
```

loopValue タグ

loopValue タグは、LDAPEntry 複数値属性または LDAPSearchResults の最初の LDAPEntry をループします。query 属性は、エクスポートされた LDAPTagSearch を指します。詳細については、102 ページの「useQuery タグ」を参照してください。このタグが指定されていない場合、entry 属性は、含まれている loop タグによって指定されたとおりにエクスポートされたエントリを指します。どちらか一方を指定する必

要があります。両方を指定することはできません。attribute タグは複数値属性を指定します。start および end タグは、クエリの loop タグに指定されたとおりに動作します。ループを経由する各パス上では、現在の LDAPAttribute 値は、id を使って指定された scope でエクスポートされます。

シンタックス

```
<ldap:loopValue id="name of attribute to export loop'd value"
scope="[page|request|session|application]" query="name of ldap
exported query" entry="name of ldap exported entry from loopEntry"
attribute="name of attribute to loop through" start="..."
max="..."></ldap:loopValue>
```

field タグ

field タグは、query 属性、url 属性、または entry 属性、および attribute 属性に指定された単一値属性の値を出力します。値が存在しない場合は、field タグの本体が渡されます。field タグの本体が評価されるのは、url がパラメータを持っていて、本体に評価し設定する必要があるパラメータがバインドがある場合、またはマッピングされた値が null の場合にだけです。属性の名前が DN の場合、entry の識別名は field の値として返されます。

シンタックス

```
<ldap:field query="name of query to use" entry="name of ldap
exported entry from loopEntry" url="ldap://..." attribute="name of
attribute to display"> </ldap:field>
```

sort タグ

sort タグは、useQuery タグと連動して、囲んでいるクエリのソート順を設定します。query 属性は、囲んでいる useQuery タグ (sort タグが useQuery タグの本体の外にある場合は、エクスポートされた LDAPTagSearch) を識別します。order 属性は、LDAPSortKey 構造体の keyDescription パラメータで記述されたとおりにソート順を指定します。useQuery タグは、複数のソートをサポートします。ソートは指定された順番で優先順位が付けられます。

シンタックス

```
<ldap:sort query="name of ldap exported query" order="..."/>
```

close タグ

close タグはリソースをシステムに返します。resource 属性は、エクスポートされたクエリリソース (LDAPTagSearch) を検索し、それに対して close() を呼び出します。この呼び出しによって、実行中のすべての SearchResults が中断し、コネクションがコネクションプールに戻されます。コネクションがコネクションプールからのものではなく authenticate タグから生じた場合は、そのコネクションで disconnect() を呼び出します。

シンタックス

```
<ldap:close resource="name of ldap exported query"/>
```

例

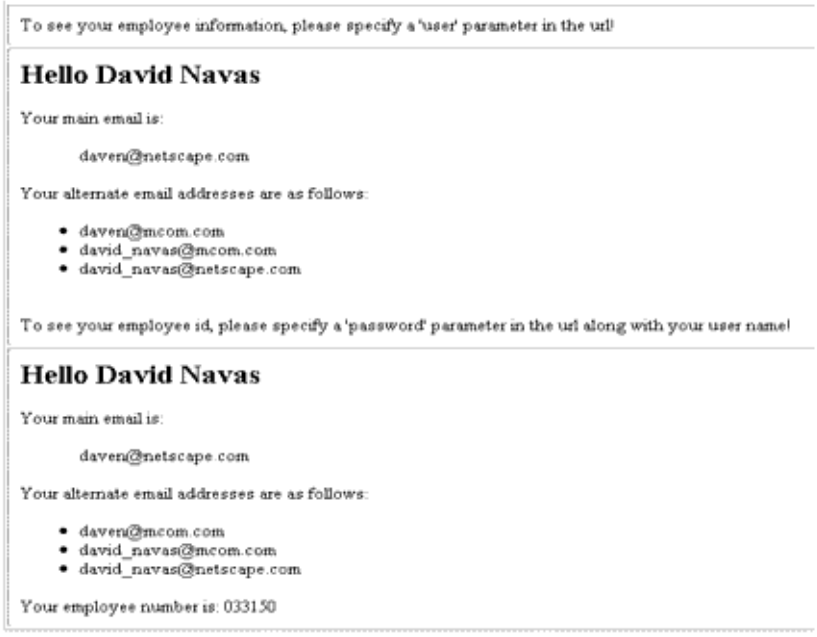
次の例では、LDAP タグと switch タグの両方を使います。switch タグは、多くの場合自明であると想定します。

```
<HTML>
<BODY>
<%@ taglib prefix="cond" uri="condtags6_0.tld" %>
<%@ taglib prefix="ldap" uri="ldaptags6_0.tld" %>
<%@ taglib prefix="attr" uri="attribtags6_0.tld" %>

<cond:parameter name="user">
   <cond:exists>
       <ldap:query id="c" url="ldap://localhost:389/uid=:user,

ou=People,dc=iplanet,dc=com?cn,mailalternateaddress,mail">
          <cond:parameter name="password">
             <cond:exists>
                <ldap:authenticate query="c"
                    url="ldap://localhost:389/dc=
                        com??sub?(uid=:user)">
                    <ldap:param name="user">
                       <attr:getParameter name="user" />
                    </ldap:param>
                    <ldap:password>
                       <attr:getParameter name="password" />
                    </ldap:password>
                </ldap:authenticate>
             </cond:exists>
          </cond:parameter>
       <ldap:param name="user"><attr:getParameter name="user" />
       </ldap:param>
   </ldap:query>
   <h2>Hello
   <ldap:field query="c" attribute="cn">
```

```
No Contact Name for <attr:getParameter name="user" /> in LDAP!

   </ldap:field></h2>
   <p>

Your main email is:
   <blockquote>
   <ldap:field query="c" attribute="mail"/>
   </blockquote>

Your alternate email addresses are as follows:

<ul>
<ldap:loopValue id="Who" scope="request" query="c"
attribute="mailalternateaddress">
<li><attr:get name="foo" scope="request"/>
</ldap:loopValue>
</ul>
<cond:ldap name="c">
<cond:authenticated>
<p>

Your employee number is:
<ldap:field attribute="employeenumber" query="c">
They removed the employee numbers from ldap -- not good!

</ldap:field>
</cond:authenticated>
<cond:else>
<cond:parameter name="password">
<cond:exists>Your specified password is incorrect.Please
retry!</cond:exists>
<cond:else>To see your employee id, please specify a 'password'
parameter in the url along with your user name!<p></cond:else>

</cond:parameter>
</cond:else>
</cond:ldap>
<p>
<ldap:close resource="c"/>
</cond:exists>
<cond:else>

To see your employee information, please specify a 'user' parameter
in the url!

<p>
```

```
</cond:else>
</cond:parameter>
</body></html>
```

次のいずれかの結果が作成されます。



条件タグリブ

cond タグのファミリは、switch タグや case タグをサポートします。これによって、行セットが最後にある場合、ユーザが管理情報だけを必要とする場合、または高度な処理能力を必要とするコンテンツをユーザが要求する場合に対応できます。

ただし、使いやすさと読みやすさを維持するために、次の等価タグを使用できます。

1. `<cond:role> ...</cond:role>`
2. `<cond:rowset name="rowset name"> ...</cond:rowset>`
3. `<cond:ldap name="ldap connection name"> ...</cond:ldap>`
4. `<cond:attribute name="attribute name"> ...</cond:attribute>`
5. `<cond:parameter name="parameter name | $REMOTE_USER$"> ...</cond:parameter>`
6. `<cond:else> ...</cond:else>`

7. `<cond:equals value="..."> ... </cond:equals>`
8. `<cond:equalsIgnoreCase value="..."> ... </cond:equalsIgnoreCase>`
9. `<cond:exists> ...</cond:exists>`
10. `<cond:notEmpty> ...</cond:notEmpty>`
11. `<cond:executeNotEmpty> ...</cond:executeNotEmpty>`
12. `<cond:isLast> ...</cond:isLast>`
13. `<cond:Connected> ...</cond:connected>`
14. `<cond:authenticated> ...</cond:authenticated>`

必要以上に表現力豊かなコードもあります。次のようにします。

```
<cond:parameter name="Who"> ...</cond:parameter>
```

これは、次のコードとまったく同じです。

```
<cond:switch><cond:value><%= request.getParameter("Who")
%></cond:value> ...</cond:switch>
```

別のコードを示します。

```
<cond:rowset value="rowset name">
<cond:exists> ...</cond:exists></cond:rowset>
```

次のコードと同じです。

```
<cond:rowset value="rowset name">
<cond:case operation="="> ...</cond:case></cond:rowset>
```

優れた表現力がユーザの混乱を招きかねないことを考慮してください。

次に、ルートタグについて説明します。

switch タグ

`switch` タグは、デフォルトでは単なる値の比較ですが、一般に、`DBRowSet` が指定するいくつかのコールバックを置き換える `RowSet` タイプの switch タグとして使われます。`switch` タグは、特定の case ステートメントが switch ステートメントの条件を満たしているかどうかを調べて、その本体だけをコンテンツページにエクスポートします。

シンタックス

```
<cond:switch type="[value|role|rowset|ldap|attribute|parameter]"
value="constant value, role name, rowset name, etc."> ...
</cond:switch>
```

case タグ

case タグは、オペレーションと、場合によって 2 番目のオペランドを含んでいます。オペランドは、switch タグの条件を満たすかどうかを調べるときに使います。値が必要であっても指定されていない場合に case と switch の組み合わせが使われると、値は囲んでいる cond:dynamicValue タグから取得されます。これにより、case タグはタグインタフェースだけを実装し、JSP をより効率よく構築します。case タグの本体は、case ステートメントが switch ステートメントの条件を満たさないかぎり評価されません。

オペレーションが指定されていない場合、オペレーションは else と見なされます。つまり、switch を無条件に満たします。オペレーションが指定されず、switch のタイプが role の場合、オペレーションは equals と見なされます。

switch タイプが特定の場合にだけ意味を持つ case オペレーションもあります。次のようにします。

- isLast および notEmpty タグは、switch タイプが ldap (query または entry) および RowSet の場合に有効です。
- executeNotEmpty オペレーションは、switch タイプが RowSet の場合にだけ意味を持ちます。
- connected および authenticated オペレーションは、switch タイプが ldap の場合にだけ意味を持ちます。
- 「=、<、>」などのオペレーションは、数値を比較する場合にだけ意味を持ちます。switch および case の値は、Double (必要なら) に変換され、比較されます。
- equals は switch 値の equals メソッドに対して呼び出されますが、equals および equalsIgnoreCase オペレーションは文字列を比較する場合にだけ意味を持ちます。この equals メソッドは、それ自体を文字列 (常に case 値) と比較するオブジェクトによって実装される場合があります。また、notEmpty は、パラメータに 0 以外の長さが指定された場合に、確認として文字列の意味を持ちます。

シンタックス

```
<cond:case
operation="[=|<|>|<=|>=|!=|<>|><|=>|=<|~=|equals|equalsIgnoreCase|e
lse|exists|notEmpty|executeNotEmpty|isLast|connected|authenticated|
{method-name}]"
value="..."></cond:case>
```

value タグ

value タグの本体は評価され、value タグの親に渡されます。親は、switch と dynamicValue の両方が実行する IValueContainingTag を実装します。この値は、value タグの属性にも指定できます。ただし、値を指定する場合は、switch または case に値を直接指定することをお勧めします。

シンタックス

```
<cond:value value="blah">default value</cond:value>
```

dynamicValue タグ

dynamicValue タグの本体には少なくとも 2 つの要素があります。1 つは、対象となるダイナミック値を構築する value タグです。もう 1 つは、囲んでいる dynamicValue インスタンスから抽出された value 属性を持つ case タグです。dynamicValue タグは次のような value タグを持ちます。

```
<cond:attribute name="Who">
      <cond:dynamicValue value="10">
            <cond:case operation="<">less than ten</cond:case>
      <cond:case operation="=">equal to ten</cond:case>
            <cond:case operation=">">greater than ten</cond:case>
      </cond:dynamicValue>
</cond:attribute>
```

ステータスレジスタには比較ビットに相当するマシン値がありません。そのため、このオペレーションは 3 回実行されます。

シンタックス

```
<cond:dynamicValue value="blah"> ... <cond:value/> ...
<cond:*case*/> ...</cond:dynamicValue>
```

例

次の例は、switch の使用法を示しています。最後の 3 つのリンクは、異なる 3 つのタイプの出力を生成します。

```
<%@ taglib prefix="cond" uri="condtags6_0.tld" %>

<cond:parameter name="showHeader">
       <cond:equalsIgnoreCase value="true">
             h2>Now let us see</h2>
       </cond:equalsIgnoreCase>
       <cond:dynamicValue>
             <cond:value value="false"/>
             <cond:equalsIgnoreCase>
                   I'm not showing a header.Nope, not me!
             </cond:equalsIgnoreCase>
       </cond:dynamicValue>
       <cond:else>
             showHeader not specified or illegal value
       </cond:else>
</cond:parameter>
```

次のように出力されます。

http://localhost/servlet/Query4.jsp	showHeader not specified or illegal value
http://localhost/servlet/Query4.jsp?showHeader=true	Now let us see
http://localhost/servlet/Query4.jsp?showHeader=false	I'm not showing a header. Nope, not me!

Attribute タグリブ

次に、attribute タグリブについて説明します。

getAttribute タグ

`getAttribute` タグは、指定された `scope` から抽出された、特定の属性の値を出力します。scope が指定されていない場合は、`findAttribute()` を使って属性を検索します。値が見つからない場合は、タグの本体が出力されます。`format` は `query:field` タグと同様に使います。

シンタックス

```
<attr:getAttribute name="attributeName"
scope="[|page|request|session|application]" format="...">default
value</attr:getAttribute>
```

setAttribute タグ

`setAttribute` タグは、指定された `scope` に特定の属性の値を設定します。`scope` が指定されていない場合は、`page` と見なされます。この値は、value 属性に指定した値です。値を指定しない場合はタグの本体です。

シンタックス

```
<attr:setAttribute name="attributeName" value="..."
scope="[page|request|session|application]">value</attr:setAttribute
>
```

getParameter タグ

`getParameter` タグは特定のパラメータの値を出力します。パラメータ値が存在しない場合は、代わりにタグの本体が出力されます。format 属性は `query:field` タグと同様に使います。

シンタックス

```
<attr:getParameter name="parameterName" format="urlEncode">default
value</attr:getParameter>
```

getRemoteUser タグ

`getRemoteUser` タグは Servlet のリモートユーザ名を出力します。

シンタックス

```
<attr:getRemoteUser>default value</attr:getRemoteUser>
```

例

詳細については、100 ページの「LDAP タグリブ」および 107 ページの「条件タグリブ」を参照してください。

JSP ロードバランス

Servlet では、各 Servlet に割り当てられた GUID があるので、ロードバランスを実行できます。すべての iPlanet Application Server インスタンスにわたって Servlet を分散するだけです。ただし、JSP は実行時に iPlanet Application Server によって Servlet に変換され、最初は個々の GUID が割り当てられていません。このため、JSP をブラウザから直接呼び出すときに、Servlet から呼び出す場合と違って、JSP のロードバランスおよびフェールオーバーを実行できません。

iPlanet Application Server 6.5 では JSP のロードバランスを個別にサポートしています。ブラウザから直接呼び出される JSP のロードバランスおよびフェールオーバー機能を使うには、次の手順を実行します。

1. XML 記述子で、ロードバランスを行う各 JSP に GUID を割り当てます。GUID の JSP への割り当てについては、86 ページの「登録 JSP」を参照してください。
2. iPlanet Application Server で、システム Servlet `JSPRunner` および `JSPRunnerSticky` を使って JSP を実行します。これらの Servlet はインストール時に登録されます。管理ツールを使って、ロードバランスに含めるサーバ全体にこれらのシステム Servlet を分散させます。
3. `System_JSPRunner` および `System_JSPRunnerSticky` の Servlet コンポーネントプロパティを確認します。JSP のロードバランスが行われるすべてのサーバが正しく一覧表示されていることを確認します。
4. 管理ツールによって Servlet のロードバランスを行うように JSP のロードバランスを実行します。ロードバランスに含めるサーバ全体に JSP を分散します。
5. Web サーバを再起動します。

アプリケーションコンポーネントの分散およびコンポーネントプロパティの変更については、『管理者ガイド』を参照してください。

JSP ページキャッシュ

JSP キャッシュと呼ばれる新しい機能は、JSP を合成するときに役立ちます。これによって、Java エンジンの内部で JSP をキャッシュする機能が提供されます。この機能によって、マスタ JSP に複数の JSP (ポータルページなど) を取り込むことができます。各 JSP に異なるキャッシュ基準を使ってキャッシュできます。たとえば株式相場を表示するウィンドウや気象通報を表示するウィンドウなどを含んでいるポータルページでは、株式相場のウィンドウを 10 分間キャッシュし、気象情報のウィンドウを 30 分間キャッシュするというように設定できます。

JSP キャッシュはリザルトキャッシュへの追加機能です。JSP は、それぞれに異なるキャッシュ基準を持つ複数の JSP から構成できます。構成された JSP は、`GUID` とともにリザルトキャッシュを使って KXS 内にキャッシュできます。詳細については、86 ページの「登録 JSP」を参照してください。

注 JSP 内で `HttpServletRequest` から拡張した独自のリクエストオブジェクトを使用するときに、Jasper JSP コンパイラを使用する場合、`CacheLib.tld` から提供される JSP キャッシュは、その JSP では利用できません。

JSP キャッシュには JSP 1.1 で提供されているカスタムタグリブのサポートを使います。キャッシュ可能な一般的な JSP ページは次のとおりです。

```
<%@ taglib prefix="ias" uri="CacheLib.tld"%>
<ias:cache>
<ias:criteria timeout="30">
<ias:check class="com.iplanet.server.servlet.test.Checker"/>
<ias:param name="y" value="*" scope="request"/>
</ias:criteria>
</ias:cache>
<%! int i=0; %>
<html>
<body>
<h2>Hello there</h2>
I should be cached.
No? <b><%= i++ %></b>
</body>
</html>
```

`<ias:cache>` および `</ias:cache>` タグはキャッシュの各制約条件を区切ります。`<ias:criteria>` タグはタイムアウト値を指定し、別のキャッシュ基準を囲みます。キャッシュ基準は `<ias:check>` と `<ias:param>` の両方のタグを使って表現できます。タグのシンタックスは次のとおりです。

- `<ias:criteria timeout="val" >` は、キャッシュされた要素のタイムアウトを秒単位で指定します。キャッシュ基準は、このタグと終了タグ `</ias:criteria>` の間に指定します。
- `<ias:check class="classname" />` は、キャッシュ基準を指定するメカニズムの一つです。`classname` は、`check` と呼ばれるメソッドを持つクラスを参照します。これは次のシグネチャを持ちます

  ```
  public Boolean check(ServletRequest, Servlet)
  ```

  これは、要素がキャッシュされたかどうかを示すブール値を返します。
- `<ias:param name="paramName" value="paramValue" scope="request" />` は、キャッシュ基準を指定する別のメカニズムです。

`paramName` は属性の名前です。この名前は `setAttribute` を使ってリクエストオブジェクトまたは URI に渡されます。このパラメータはキャッシュの基準として使われます。

表 3-13 に、`paramValue` パラメータの値を示します。これは、キャッシュを実行するかどうかを決める値です。

表 3-13 `paramValue` パラメータの値

制約	説明
`x = ""`	`x` はパラメータまたは属性として存在している必要がある
`x = "v1|...|vk"`、`vi` は `*` でもかまわない	`x` は任意の文字列 (パラメータ / 属性) にマッピングされる。`x=*` の場合は、`x` のリクエストパラメータが、キャッシュされたバッファを格納するときに使われた値と同じ値を持つとき、現在のリクエストの制約は true になる
`x = "1-u"`、`1` および `u` は整数	`x` は、`[1,u]` の範囲内の値にマッピングされる

この範囲は確認する属性のソースを指定します。範囲に指定できるのは、`page`、`request` (デフォルト)、`session`、または `application` です。

例

次の例はキャッシュされた JSP ページを示しています。

```
<%@ taglib prefix="ias" uri="CacheLib.tld"%>
<ias:cache>
<ias:criteria timeout="30">
<ias:check class="com.iplanet.server.servlet.test.Checker"/>
<ias:param name="y" value="*" scope="request"/>
</ias:criteria>
```

```
</ias:cache>
<%! int i=0; %>
<html>
<body>
<h2>Hello there</h2>

I should be cached.

No? <b><%= i++ %></b>
</body>
</html>
```

Checker は次のように定義されます。

```
package com.iplanet.server.servlet.test;

import com.iplanet.server.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Checker {
   String chk = "42";
   public Checker()
   {

   }
   public Boolean check(ServletRequest _req, Servlet _serv)
   {

       HttpServletRequest req = (HttpServletRequest)_req;
       String par = req.getParameter("x");
       return new Boolean(par == null ? false :par.equals(chk));
   }
}
```

この例でキャッシュされた要素は、リクエストのパラメータが x=42 で、y がその要素の保存に使われた値と等しい場合に有効です。<ias:criteria> ブロック内に <ias:param> および <ias:check> の複数の組み合わせを持たせることができます。また、JSP 内に複数の <ias:criteria> ブロックを持たせることもできます。

第 4 章

Enterprise JavaBeans の紹介

この章では、iPlanet Application Server アプリケーションプログラミングモデルにおける Enterprise JavaBeans (EJB) の動作について説明します。この章ではまず、EJB の役割と配信メカニズムを定義します。次に、エンティティ Beans、セッション Beans、およびメッセージ駆動 Beans の 3 つの EJB タイプについて説明し、どのような場合に使うかについて詳しく説明します。さらに、ビジネスロジックのカプセル化に EJB を使う、オブジェクト指向の iPlanet Application Server アプリケーション設計の概要についても説明します。

この章には次の節があります。

- Enterprise JavaBeans の役割
- Enterprise JavaBeans とは
- Enterprise JavaBeans について
- iPlanet Application Server アプリケーションにおける EJB の役割
- オブジェクト指向アプリケーションの設計
- ejbc コンパイラの使用法
- JNDI による EJB の参照

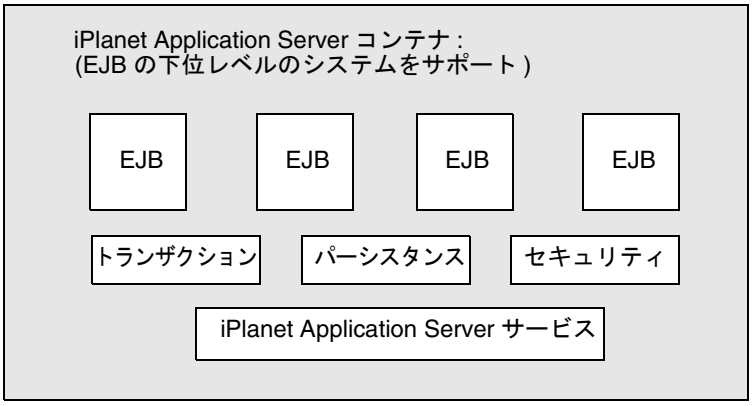
注

- EJB を理解し、すでに iPlanet Application Server で EJB を使っている場合は、iPlanet Application Server で使う EJB を開発するための手順とガイドラインに進んでください。第 6 章「エンティティ EJB のビルド」および第 12 章「ユーザセッションの作成と管理」を参照してください。
- メッセージ駆動 Beans のサポートは EJB 2.0 の仕様に基づいており、このリリースの iPlanet Application Server の新機能です。この機能は開発者だけが使用し、運用環境では使用されません。
- セッション Beans とエンティティ Beans は EJB 1.1 の仕様に基づいており、運用環境での使用が認められています。この章では、EJB 1.1 仕様に準拠したセッション Beans とエンティティ Beans に関連するプロパティおよびプロシージャについて説明します。

Enterprise JavaBeans の役割

iPlanet Application Server では、EJB はアプリケーションにとって重要な役割を果たします。Servlet はアプリケーションのセントラルディスパッチャとして動作し、プレゼンテーションロジックを処理します。EJB はアプリケーションの実際のデータとルールを大量に処理しますが、ユーザインタフェースサービスを表示しません。EJB を使うと、ビジネスロジック、ルール、およびオブジェクトをスケーラブルな個別のモジュールに分割できます。各 EJB は、データ構造体やオペレーションメソッドなどの 1 つまたは複数のアプリケーションタスク、またはアプリケーションオブジェクトをカプセル化します。EJB はパラメータを受け取って戻り値を返します。

EJB は常にコンテナのコンテキスト内で動作します。コンテナは、EJB とそれらを管理するサーバ間のリンクとして機能します。iPlanet Application Server ソフトウェア環境では EJB コンテナがサポートされています。このコンテナは、Sun の EJB 仕様書に規定されているすべての標準コンテナサービスに加え、iPlanet Application Server に限定されたサービスも提供します。



コンテナはリモートアクセス、セキュリティ、同時性、トランザクションコントロール、およびデータベースアクセスを処理します。実際の実装方法の詳細はコンテナに含まれており、コンテナと EJB の間には規定の標準インタフェースが存在するので、Bean 開発者がプラットフォーム固有の実装に関する詳細を理解したり、処理したりする必要はありません。Bean 開発者は、EJB 標準をサポートするベンダーの製品とともに使用できる、一般的なタスク限定の EJB を作成できます。

Enterprise JavaBeans とは

EJB のアーキテクチャは、オブジェクト指向の分散型エンタープライズアプリケーションの開発および配置を行うコンポーネントベースのモデルです。EJB はアプリケーション内の一つのコンポーネントです。EJB を使って作成されたアプリケーションはスケーラブルであり、トランザクションをカプセル化し、安全なマルチユーザアクセスを許可します。これらのアプリケーションを一度作成すると、EJB をサポートする任意のサーバに配置できます。

EJB の基本的な特性は次のとおりです。

- Bean の作成および管理は、iPlanet Application Server で提供されるコンテナによって実行時に処理されます。
- クライアントアクセスの仲介は Bean が配置されたコンテナおよびサーバによって処理されるので、Bean 開発者がそれを処理する必要はありません。

- EJB 仕様書で定義されている標準コンテナサービスを使うように Bean を制限すると、Bean は移植可能になり、EJB 互換コンテナに配置できます。
- ほかの個別の Bean 要素で構成されるアプリケーション (「複合」アプリケーション) に Bean を入れたり、Bean を追加したりしても、ソースコードの変更や Bean の再コンパイルを行う必要はありません。
- クライアントの Bean 定義ビューは、Bean 開発者によって完全に制御されます。ビューは、Bean が動作するコンテナや Bean が配置されたサーバの影響を受けません。
- EJB は iPlanet Application Server の動作中にダイナミックに再読み込みされます。

さらに EJB 仕様書には、Enterprise JavaBeans によってクライアント、コンポーネント、および JAR ファイルの 3 つの規約が確立されることが記載されています。

クライアント規約を理解する

クライアント規約によって、クライアントと EJB コンテナ間の通信ルールが決定し、EJB を使う統一されたアプリケーション開発モデルが確立され、Bean の優れた再利用性が保証されます。クライアント規約には、EJB オブジェクトの識別方法、そのメソッドの起動方法、EJB オブジェクトの作成および破棄方法が明記されています。

EJB コンテナによって、ユーザ独自のコンポーネントやほかの供給元に提供されたコンポーネントを使った分散アプリケーションを構築できます。iPlanet Application Server には高レベルのトランザクション、ステート管理、マルチスレッド、およびリソースプールラッパーが備わっているので、ユーザが低レベル API の詳細を理解する必要はありません。

EJB インスタンスの作成および管理はコンテナクラスによって実行時に行われますが、EJB 自体は環境プロパティを編集することによって配置時にカスタマイズされます。トランザクションモードやセキュリティ属性などのメタデータは Bean 自体から切り離され、設計および配置時にコンテナのツールによって制御されます。実行時のクライアントによる Bean へのアクセスは、EJB が配置されたサーバによってコンテナ制御されます。

さらに、EJB コンテナは、EJB を定義する特定のビジネスメソッドをクライアントが確実に起動できるようにする役割を果たします。Bean 開発者は、Bean にメソッドを実装するときに、クライアントが Bean メソッドを呼び出す方法をコンテナに指示する「リモートインタフェース」をコンテナに提供する必要があります。

さらに、EJB によって「ホームインタフェース」がコンテナに提供されます。ホームインタフェースは、EJB 仕様で定義されている `javax.ejb.EJBHome` インタフェースを拡張します。これによって、クライアントが EJB を作成したり破棄したりするメカニズムが提供されます。ホームインタフェースは、そのもっとも基本的な機能として、

Bean を作成する各方法に関連する 0 または 1 つ以上の `create(...)` メソッドを定義します。さらに、エンティティ *Beans* と呼ばれるタイプの EJB では、Bean または Bean のコレクションの検索に使う各方法についても検索メソッドを定義する必要があります。

コンポーネント規約を理解する

コンポーネント約定は EJB とそのコンテナ間の関係を確立しており、クライアントに対して完全にトランスペアレントです。特定の Bean のコンポーネント規約は、次の部分から構成されています。

- **ライフサイクル**: EJB セッション Beans の場合は、`javax.ejb.SessionBean` および `javax.ejb.SessionSynchronization` インタフェースが実装されています。EJB エンティティ Beans の場合は、`javax.ejb.EntityBean` インタフェースが実装されています。
- **セッションコンテキスト**: コンテナは `javax.ejb.SessionContext` インタフェースを実装して、Bean インスタンス作成時にセッション Beans インスタンスにサービスおよび情報を渡します。
- **エンティティコンテキスト**: コンテナは `javax.ejb.EntityContext` インタフェースを実装して、Bean インスタンス作成時にエンティティ Beans にサービスおよび情報を渡します。
- **環境**: コンテナは `java.util.Properties` を実装し、これらのプロパティをそのコンテナの EJB で利用できるようにします。
- **サービス情報**: コンテナは、そのコンテナのすべての EJB でそのサービスを利用できるようにします。

さらに、コンポーネント規約を拡張して、アプリケーション固有のサービスを追加できます。

JAR ファイル規約を理解する

Enterprise JavaBeans をパッケージに入れるときに使われる標準フォーマットは EJB-JAR ファイルです。このフォーマットは、Bean プロバイダとアプリケーション編成者間、およびアプリケーションの編成者と配置者間の規約です。iPlanet Application Server では、iPlanet Application Server 配置ツールを使って、EJB を含んだ `.jar` ファイルを作成できます。詳細については、配置ツールのオンラインヘルプを参照してください。

注 EJB JAR 名は、.jar 拡張子ではなく、ファイル名の最初の部分によって識別されます。Application Server に配置する EJB JAR 名は、一意でなければなりません。ejb-jar.xml ファイルの <ejb-name> 部分に指定する EAR ファイル名および EJB 名には、Java パッケージ方式の命名規則を使ってください。Java パッケージ方式の命名規則を使えば、名前の衝突は発生しません。この命名規則は、iPlanet Application Server だけでなく、ほかの J2EE アプリケーションサーバでも使うことをお勧めします。

EJB-JAR ファイルは、配置記述子 (DD) だけでなく、次のクラスおよびインタフェースが使うすべてのクラスファイルを含んでいる必要があります。

- Enterprise JavaBeans クラス
- Enterprise JavaBeans のホームおよびリモートインタフェース
- エンティティ Beans のプライマリキークラス

さらに、EJB-JAR ファイルは、Enterprise JavaBeans クラスと、リモートおよびホームインタフェースが使うすべてのクラスおよびインタフェースのクラスファイルを含んでいる必要があります。EJB-JAR ファイルの内容の詳細については、第 11 章「配置のためのパッケージ化」を参照してください。

Enterprise JavaBeans について

EJB は、次のいずれかを表すオブジェクトです。

- 状態のないサービス
- 特定のクライアントとのセッションであり、クライアントによって起動された複数のメソッドの状態を自動的に管理する
- 複数のクライアントによって共有される可能性のあるパーシスタントエンティティオブジェクト

EJB には、エンティティ、セッション、メッセージ駆動の 3 種類があります。Bean のタイプによってサーバアプリケーションでの使用法が異なります。次の節では、3 種類の Bean について説明します。

- セッション Beans について
- エンティティ Beans について
- メッセージ駆動 Beans について

セッション Beans について

セッション EJB には次の特性があります。

- 1 つのクライアントに対して実行されます。
- オプションで、プロパティの設定値に従ってトランザクション管理を処理します。
- オプションで、基礎となるデータベース内の共有データを更新します。
- 比較的短命です。
- 状態のあるセッション Beans の場合、iPlanet Application Server のフェールオーバーサポートを使わないかぎりサーバクラッシュに耐える保証はありません。

セッション Beans はビジネスルールまたはビジネスロジックを実装しています。リモートアクセス、セキュリティ、同時性、およびトランザクションのすべての機能が EJB コンテナによって提供されます。セッション EJB は、そのセッション EJB を作成するクライアントだけが使うプライベートリソースです。たとえば、EJB を作成して電子ショッピングカートをシミュレートするとします。ユーザがアプリケーションにログインするたびに、アプリケーションはセッション Beans を作成してそのユーザが購入したアイテムを保持します。ユーザがログアウトしたりショッピングを終了したりすると、セッション Beans は削除されます。

エンティティ Beans について

エンティティ EJB には次の特性があります。

- エンタープライズ情報システム (EIS) のリソース (通常はデータベース) のデータ表示
- Bean によって管理されるトランザクション区分
- コンテナによって管理されるトランザクション区分
- すべてのユーザによる共有アクセス
- データベースにデータが存在するかぎり存続する
- EJB のサーバクラッシュを透過的に回避する

EJB および EJB コンテナを管理するサーバは、同時にアクティブになっているエンティティ EJB にスケーラブルな実行時環境を提供します。エンティティ EJB は、パーシスタントデータを表します。

メッセージ駆動 Beans について

iPlanet Application Server, Enterprise Edition 6.5 では、メッセージ駆動 Beans の配置がサポートされます。この実装は EJB 2.0 の仕様に基づいており、開発者専用です。このリリースのメッセージ駆動 Beans は、運用環境ではテストされていません。

メッセージ駆動 Beans は、Enterprise JavaBeans によって提供されるフレームワークをサポートするという点で、セッション Beans とエンティティ Beans に似ています。ただし、メッセージ駆動 Beans は JMS (Java Messaging Service) のリスナでもあり、JMS メッセージの形式でクライアントから受信するリクエストに基づいてタスクを実行します。

JMS については、付録 A「Java Message Service の使用法」を参照してください。

セッション Beans やエンティティ Beans と異なり、メッセージ駆動 Beans はメッセージキューを非同期的に処理するので、サーバのリソースをより効率的に使用します。メッセージ駆動 Beans は多くのクライアントリクエストを同時に処理できるので、メッセージキューのボトルネックは生じません。

メッセージ駆動 Beans とセッションおよびエンティティ Beans のもっとも大きな目に見える違いは、クライアントがインタフェースを通じてメッセージ駆動 Beans にアクセスしないことです。セッションまたはエンティティ Beans と異なり、メッセージ駆動 Beans には 2 つの標準インタフェースを実装する 1 つの Bean クラスしかありません。

MDB プロパティ

メッセージ駆動 Beans には次の属性があります。

- メッセージ駆動 Beans のインスタンスは、特定のクライアントのデータまたは会話型ステートを保持しません。
- メッセージ駆動 Beans のすべてのインスタンスは同等であり、EJB コンテナは、どのメッセージ駆動 Beans インスタンスにもメッセージを割り当てることができます。コンテナはこれらのインスタンスをプールして、メッセージのストリームが同時に処理されるようにします。
- 1 つのメッセージ駆動 Beans が複数のクライアントからのメッセージを処理できます。

メッセージ駆動 Beans インスタンスのインスタンス変数には、JMS API コネクション、オープンデータベースコネクション、Enterprise JavaBeans オブジェクトへのオブジェクト参照などの、クライアントメッセージの処理に関連する状態のいくつかを含めることができます。

iPlanet Application Server は、JMS 仕様を実装するメッセージングミドルウェアのアプリケーションとして、iPlanet Message Queue for Java, 2.0 SP1 を使います。iPlanet Application Server でメッセージ駆動 Beans を使う前に、iMQ for Java, 2.0 SP1 をインストールしておく必要があります。

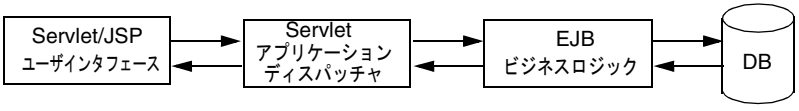
メッセージ駆動 Beans の配置については、第 7 章「メッセージ駆動 Beans の使用」を参照してください。

iMQ for Java, 2.0 SP1 は、iPlanet Application Server インストール CD に収録されています。このインストール CD がない場合は、`http://www.iplanet.com/products/iplanet_message_queue/home_message_queue.html` から開発者用を無償でダウンロードできます。

iPlanet Application Server アプリケーションにおける EJB の役割

EJB は、iPlanet Application Server アプリケーションのビジネスロジックおよびデータ処理の大部分を実行します。EJB は、認識されることなく背後でアプリケーションの実行を支えます。EJB は iPlanet Application Server アプリケーションの中核にありますが、ユーザがその存在を意識することはほとんどなく、EJB と直接対話することもありません。

ユーザがブラウザから iPlanet Application Server のアプリケーション Servlet を起動すると、Servlet は 1 つまたは複数の EJB を起動してアプリケーションのビジネスロジックおよびデータ処理の大部分を行います。たとえば、Servlet はユーザのブラウザに JavaServer Page (JSP) を読み込んでユーザ名およびパスワードを要求し、さらにその内容をセッション Beans に渡して確認します。



有効なユーザ名およびパスワードの組み合わせを受け取ると、Servlet は 1 つまたは複数のエンティティ Beans およびセッション Beans をインスタンス化してアプリケーションのビジネスロジックを実行し、その後、終了します。Bean 自体もほかのエンティティ Beans またはセッション Beans をインスタンス化して、ビジネスロジックおよびデータ処理をさらに行います。

たとえば、カスタマサービス代理店による部品データベースへのアクセスを可能にするエンティティ Beans を、Servlet が起動するとします。部品データベースへのアクセスとは、データベースを検索し、顧客が購入するアイテムをキューに入れ、顧客の注文を出してデータベースの部品在庫数を減らし、顧客に請求書を発送するまでの一連のプロセスです。また、在庫が不足した場合に部品を追加注文する処理も含まれます。

顧客注文プロセスの一環として、Servlet は「ショッピングカート」を表すセッション Beans を作成し、顧客から注文を受けた製品を一時的に記録します。注文が終了すると、ショッピングカートのデータが注文データベースに転送され、在庫データベースの各アイテムの数量が減り、ショッピングカートのセッション Beans が解放されます。

この簡単な例が示すように、EJB は Servlet によって起動され、アプリケーションのビジネスロジックおよびデータ処理の大半を処理します。エンティティ Beans は基本的に、Java Database Connectivity (JDBC) API を使ったデータアクセスの処理に使われます。セッション Beans は一時的なアプリケーションオブジェクトを提供し、個々のビジネスタスクを実行します。

EJB を使うアプリケーションを作成する際の課題は、アプリケーションを Servlet、JSP、セッション Beans、エンティティ Beans にどのように分割するかを決めることです。

オブジェクト指向アプリケーションの設計

開発者として最大の仕事は、iPlanet Application Server アプリケーションのビジネスロジックおよびデータ処理をもっとも効率的な EJB セットに分割することです。EJB を使ってオブジェクト指向の設計を行うには、エンティティ Beans のインスタンスは生存期間の長い、パーシスタントな、複数のクライアント間で共有されるものにします。一方、セッション Beans のインスタンスは生存期間の短い、1 つのクライアントだけに使われるものにします。それ以外に明確なルールはありません。したがって、次の節の説明は、アプリケーションを高速化したり、EJB をモジュール化したり、共有可能にしたり、簡単にメンテナンスしたりできるようにするための、iPlanet Application Server に限定される高度な情報です。

すべてのオブジェクト指向の開発と同様に、まず、ビジネスロジックおよびデータ処理に必要な細分レベルを決める必要があります。細分レベルにより、アプリケーションをどの程度の数に分割するかが決まります。細分レベルが高い場合 (アプリケーションを多数の狭義な EJB に分割する場合) は、サイトのさまざまなアプリケーション間で EJB の共有および再利用を促進するアプリケーションが作成されます。細分レベルが低い場合は、実行速度の速いモノリシックなアプリケーションが作成されます。

注　アプリケーションを中程度から多数の EJB に分割すると、アプリケーションパフォーマンスが著しく劣化し、オーバーヘッドが増加します。JavaBeans などの EJB は単なる Java オブジェクトではありません。EJB は Java オブジェクトよりも高レベルです。これらの EJB は、リモート呼び出しインタフェースセマンティック、セキュリティセマンティック、トランザクションセマンティック、およびプロパティから構成されるコンポーネントです。

ガイドラインの計画

一般に、複数のアプリケーションおよびクライアント間で EJB を共有して複数のサーバ間に配置するニーズと実行速度のニーズのバランスをとるように iPlanet Application Server アプリケーションを作成します。

- EJB をプレゼンテーションロジック (Servlet および JSP) とともに同じサーバ上に配置するようにサーバ管理者に依頼して、アプリケーション実行時のリモートプロシージャコール (RPC) の数を減らします。
- 状態のあるセッション Beans ではなく、できるだけ状態のないセッション Beans を作成します。状態のあるセッション Beans を作成する必要がある場合は、パフォーマンスを向上させるために「スティッキーロードバランス」をオンにするようにサーバ管理者に依頼します。
- タスク限定の小さな一般セッション EJB を作成します。これらの EJB は、多数のアプリケーションで使われる動作をカプセル化すると理想的です。

これらの一般的な注意事項に加えて、アプリケーションのどの部分がエンティティ Beans およびセッション Beans となるか決めます。

セッション Beans の使用法

セッション Beans は、1 つのデータベースレコードの更新、編集するドキュメントのコピーの更新、またはショッピングカートのように個々のクライアントに固有のビジネスオブジェクトの更新など、一時的なオブジェクトおよび処理を表します。これらのオブジェクトの利用は 1 つのクライアントに限られます。このため、セッション Beans は「会話型ステート」と呼ばれるクライアント固有のセッション情報を維持できます。会話型ステートを維持するセッション Beans は状態のあるセッション Beans と呼ばれ、会話型ステートを維持しないセッション Beans は状態のないセッション Beans と呼ばれます。

クライアントによるセッションオブジェクトの利用が終了すると、そのオブジェクトは解放されます。アプリケーションを設計するとき、一時的な 1 つのクライアントオブジェクトを潜在的なセッション Beans として指定します。たとえば、オンラインショッピングのアプリケーションでは、各ショッピングカートは一時オブジェクトです。カートの生存期間は、顧客が購入するアイテムを選択している間に限られます。顧客の買い物が終了し注文が処理されると、カートオブジェクトは不要になるため解放されます。

エンティティ Beans と同様に、セッション Beans は JDBC 呼び出しを介してデータベースにアクセスします。また、セッション Beans はトランザクション設定も提供できます。これらのトランザクション設定および JDBC 呼び出しは、セッション Beans のコンテナによって参照されます。このコンテナは透過的です。iPlanet Application Server によって提供されるコンテナは、JDBC 呼び出しおよびリザルトセットを処理します。

iPlanet Application Server アプリケーションで、セッション Beans を使って単一のクライアントアクセス用に一時オブジェクトおよびルールを定義する方法の詳細については、第 5 章「セッション EJB によるビジネスルール管理」を参照してください。

エンティティ Beans の使用法

エンティティ Beans は通常、パーシスタントデータを表します。このデータは、データベースで直接管理されます。すなわち、オブジェクトとしての EIS アプリケーションからアクセスされます。エンティティ Beans の簡単な例としては、データベーステーブルの 1 行を表すように定義され、各 Bean インスタンスが特定の行を表すものがあります。より複雑な例としては、データベース内の複雑に組み合わされたテーブルを表すように設計され、各 Bean インスタンスが 1 つのショッピングカートの内容を表すエンティティ Beans があります。

セッション Beans とは異なり、エンティティ Beans のインスタンスには、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、使用するトランザクションごとにインスタンスのステートの同期をとる役割を果たします。この役割がコンテナに委譲された場合、Bean 開発者は複数のトランザクションからメソッドへの同時アクセスを考慮せずにすみます。

エンティティ Beans のパーシスタンスは、Bean またはコンテナによって管理できます。パーシスタンスがエンティティ Beans によって管理されると、「Bean 管理パーシスタンス」と呼ばれます。Bean がこの機能をコンテナに委譲した場合は、「コンテナ管理パーシスタンス」(CMP) と呼ばれます。

- **Bean 管理パーシスタンス** : Bean 開発者が EJB クラスメソッドにパーシスタンスコード (JDBC 呼び出しなど) を直接実装します。専用インタフェースを使うと、ダウンサイドでは移植性が失われる可能性があり、Bean が特定のデータベースに関連付けられるというリスクがあります。

- **コンテナ管理パーシスタンス**：コンテナプロバイダ配置ツールを使ってコンテナパーシスタンスを実装します。コンテナは透過的にパーシスタンスステートを管理します。したがって、ユーザが Bean メソッドのデータアクセスコードを実装する必要はありません。この方法によって、簡単に実装できるだけでなく、特定のデータベースに関連付けずに完全に Bean を移植できます。

エンティティ Beans を使って、iPlanet Application Server アプリケーションでパーシスタントオブジェクトおよびビジネスロジックを定義する方法の詳細については、第 6 章「エンティティ EJB のビルド」を参照してください。

フェールオーバーリカバリ計画

フェールオーバーリカバリは、サーバのクラッシュ後に Bean がそれ自体をインスタンス化し直すプロセスです。フェールオーバーリカバリは、状態のないセッション Beans と状態のあるセッション Beans の両方でサポートされます。セッション Beans のフェールオーバープロパティの設定には配置ツールを使います。これらの設定値の詳細については、配置ツールのオンラインヘルプを参照してください。セッション Beans のフェールオーバーリカバリの詳細については、第 5 章「セッション EJB によるビジネスルール管理」を参照してください。

エンティティ Beans は、サーバクラッシュ後に Bean 参照が失われたという警告を表示するフェールオーバーリカバリをサポートしています。エンティティ Beans を復元するには、ファインダを使って Bean の新規参照を作成する必要があります。詳細については、157 ページの「ファインダーメソッドの使用法」を参照してください。

データベース操作

iPlanet Application Server では、トランザクション属性とともに JDBC API を使ってデータベース操作を行うことをお勧めします。データベースコネクションを確立するには、JNDI (Java Naming and Directory Interface) を使います。JNDI は、アプリケーションが JDBC ドライバから独立してデータベースサービスを検索して、アクセスする標準的な手段を提供します。

エンティティ Beans を使って iPlanet Application Server アプリケーションでパーシスタントオブジェクトおよびビジネスロジックを定義する方法の詳細については、第 9 章「JDBC を使ったデータベースアクセス」を参照してください。

セッションおよびエンティティ Beans を介して利用できるトランザクションコントロールの詳細については、第 8 章「EJB のトランザクション処理」を参照してください。

EJB の配置

Deployment Tool を使って、アプリケーションの残りの部分と共に EJB を配置します。詳細については、配置ツールのオンライン ヘルプを参照してください。配置ツールによって設定されるプロパティ設定値や、それらのアプリケーションへの影響の詳細については、第 11 章「配置のためのパッケージ化」を参照してください。

EJB のダイナミックな再読み込み

iPlanet Application Server での EJB の再読み込みは、サーバを再起動しなくても、EJB を配置し直すだけで行うことができます。同じディレクトリ内で新しい EJB 実装のクラスファイルに置き換えても行うことができます。

iPlanet Application Server は、新しいコンポーネントを検出し、次の EJB 呼び出しの作成時に再読み込みします。

注

- EJB の再読み込みは、EJB 実装クラスにだけ適用されます。
- 出荷時、ダイナミックな再読み込み機能はデフォルトでオフになっています。必要に応じてオンにしてください。

詳細については、付録 B「実行時の注意事項」を参照してください。

ejbc コンパイラの使用法

iPlanet Application Server には、次の機能を持つ `ejbc` ユーティリティがあります。

- すべての EJB クラスおよびインタフェースを調べて、EJB 仕様書に準拠しているかどうかを確認する
- スタブおよびスケルトンを生成する

スタブとスケルトンは EJB コンテナに必要で、アプリケーションファイルとともに配置する必要があります。これらのスタブとスケルトンによってリモート通信が可能になり、コンテナはすべての Bean リクエストを捕捉できます。

`ejbc` ユーティリティは、次のファイルを生成します。

表 4-1　ejbc ユーティリティによって生成されるファイル

ファイル	説明	必要な ejbc オプション
_*Home*_Stub.class	OMG JavaIDL 07-59-99 仕様書準拠のホームスタブクラス	-iiop
_*Remote*_Stub.class	OMG JavaIDL 07-59-99 仕様書準拠のリモートスタブクラス	-iiop
_ejb_RmiCorbaBridge_*Home*_Tie.class	ホームインタフェースタイクラス	-iiop
_ejb_RmiCorbaBridge_*Remote*_Tie.class	OMG JavaIDL 07-59-99 仕様書準拠のタイクラス	-iiop
ejb_RmiCorbaBridge_*Remote*.class	リモートインタフェースブリッジ	-iiop
ejb_RmiCorbaBridge_*Home*.class	ホームブリッジ	-iiop
ejb_fac_*Implementation*.class	ホームファクトリ	
ejb_home_*Implementation*.class	ホームスケルトン	
ejb_kcp_skel_*Remote*.class	KCP リモートスケルトン	
ejb_kcp_skel_*Home*.class	KCP ホームスケルトン	
ejb_kcp_stub_*Remote*.class	KCP リモートスタブ	
ejb_kcp_stub_*Home*.class	KCP ホームスタブ	
ejb_skel_*Implementation*.class	リモートスケルトン	
ejb_stub_*Remote*.class	リモートスタブ	
ejb_stub_*Home*.class	ホームスタブ	

通常使う ejbc のシンタックスは次のとおりです。

```
ejbc options Home Remote Implementation
```

RMIC (Remote Method Invocation Compiler) モードでは、IIOP スタブとスケルトンクラスだけが生成され、EJB 仕様書に対する準拠の確認は省略されます。RMIC モード用の ejbc のシンタックスは次のとおりです。

```
ejbc options -rmic Remote
```

オプションは次のとおりです。-sl、-sf、または -cmp が指定されていない場合、Bean は BMP エンティティ Beans としてコンパイルされます。

表 4-2　ejbc のオプション

オプション	説明
-sl	Bean を状態のないセッション Beans としてコンパイルする
-sf	Bean を状態のあるセッション Beans としてコンパイルする
-fo	可用性の高い状態のあるセッション Beans をコンパイルする
-cmp	Bean を CMP エンティティ Beans としてコンパイルする
-iiop	追加の CORBA クラスを生成する
-gs	Java ソースファイルを生成する
-d *dir*	出力ディレクトリを指定する
-help	シンタックスの要約を表示する
-rmic	RMIC コードを生成する
-classpath *classpath*	クラスパスを設定する
-cp	廃止。代わりに -classpath を使用
-javaccp *classpath*	javac クラスパスに接頭辞を追加する
-debug	ejbc ユーティリティをデバッグモードで実行し、デバッグ情報を出力する

JNDI による EJB の参照

ここでは、EJB を検索するための JNDI の命名方式の例 (サーバに添付の HelloWorld サンプルの型に準拠) を示します。Servlet のソースファイル GreeterServlet.java は、TheGreeter Bean のホームを検索します。

注　ここで説明する原則は、EJB 間の EJB 検索にも適用できます。

GreeterServlet.java ファイルの JNDI 検索は次のようになります。

```
initContext = new javax.naming.InitialContext();
String JNDIName = "java:comp/env/ejb/greeter";
Object objref = initContext.lookup(JNDIName);
GreeterHome myGreeterHome =
   (GreeterHome)PortableRemoteObject.narrow(objref,
   GreeterHome.class);
```

注　iPlanet では、EJB へのすべての参照をアプリケーションコンポーネントの環境の ejb サブコンテキストで構成することをお勧めします (たとえば、java:comp/env/ejb サブコンテキスト)。

参照コンポーネントの web.xml ファイルの ejb-ref エントリは次のようになります。

```
<ejb-ref>
   <ejb-ref-name>ejb/greeter</ejb-ref-name>
   <ejb-ref-type>Session</ejb-ref-type>
   <home>samples.helloworld.ejb.GreeterHome</home>
   <remote>samples.helloworld.ejb.Greeter</remote>
   <ejb-link>TheGreeter</ejb-link>
</ejb-ref>
```

JNDI の命名では、web.xml の 2 つの属性が重要です。

- ejb-ref-name 属性は、ソースファイルで使う検索文字列を定義します。
- ejb-link 属性は、この参照をターゲット Enterprise JavaBeans に接続します。これは、ターゲット Enterprise JavaBeans の ejb-jar.xml ファイルの ejb-name 属性で定義されている名前です。

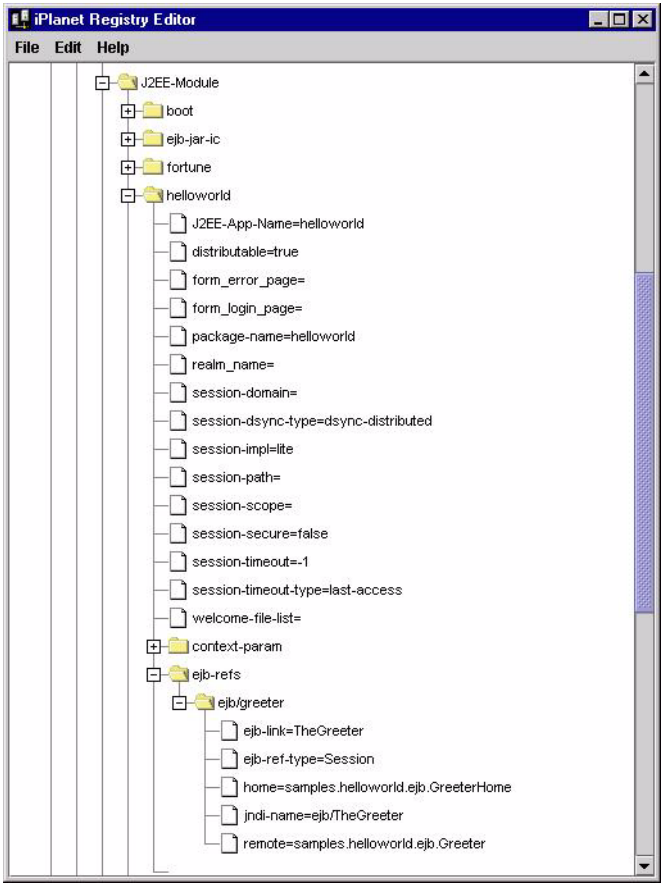
J2EE 仕様書によると、ターゲット Bean は同じ J2EE アプリケーションにある EJB JAR モジュールの一部でなければなりません。

アプリケーションが配置されると、参照は次の場所にあるレジストリ (LDAP 内) に保存されます。

SOFTWARE¥iPlanet¥Application Server¥6.5¥J2EE-Module¥*module_name*¥ejb-refs

図 4-1 は、レジストリエントリを示しています。

図 4-1　EJB 参照のレジストリエントリ



参照コンポーネントの `ias-web.xml` ファイルの `ejb-ref` エントリは次のようになります。

```
<ejb-ref>
   <ejb-ref-name>ejb/greeter</ejb-ref-name>
   <jndi-name>ejb/TheGreeter</jndi-name>
</ejb-ref>
```

iPlanet 固有の配置記述子 `ias-web.xml` の `ejb-ref` セクションでは、検索名 (`web.xml` の `ejb-ref-name` 属性と同じ) はターゲット Enterprise JavaBeans の JNDI 名に接続されます。Bean の JNDI 名は `ejb/`*bean_name* です。たとえば、Bean 名 (ターゲット Bean の `ejb-jar.xml` ファイルの `ejb-name` 属性によって定義される) が `TheGreeter` の場合、配置される Bean の JNDI 名は `ejb/TheGreeter` です。

さらに、ターゲット Enterprise JavaBeans の ejb-jar.xml ファイルは次のようになります。

```
<ejb-jar>
   <enterprise-beans>
   <session>
       <display-name>TheGreeter</display-name>
       <ejb-name>TheGreeter</ejb-name>
       <home>samples.helloworld.ejb.GreeterHome</home>
       <remote>samples.helloworld.ejb.Greeter</remote>
       <ejb-class>samples.helloworld.ejb.GreeterEJB</ejb-class>
       <session-type>Stateless</session-type>
       <transaction-type>Bean</transaction-type>
   </session>
   ...
</ejb-jar>
```

ターゲット Enterprise JavaBeans (ejb-name 属性) の配置記述子は、web.xml ファイルの ejb-link 属性、および参照コンポーネントの JNDI 名 (ias-web.xml の jndi-name) と同じです。また、参照コンポーネントとターゲット Bean の配置記述子の Bean タイプ、ホームおよびリモートインタフェースも同じです。

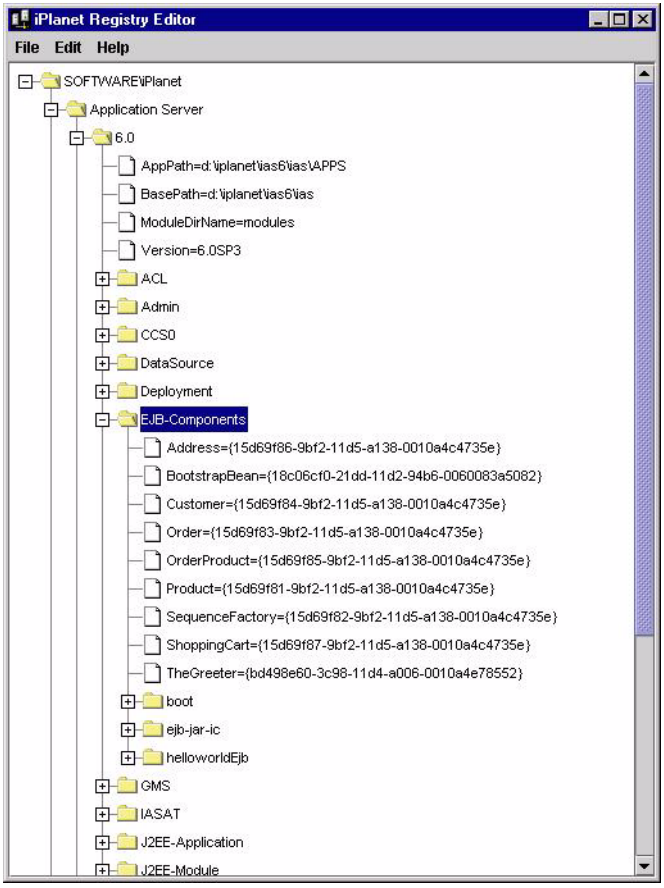
web.xml、ias-web.xml、および ejb-jar.xml の各ファイルの詳細については、第 11 章「配置のためのパッケージ化」を参照してください。

EJB が配置されると、すべての Bean 検索はレジストリの次のセクションに保存されます。

```
SOFTWARE¥iPlanet¥Application Server¥6.5¥EJB-Components
```

図 4-2 は、レジストリエントリを示しています。

図 4-2　EJB コンポーネントのレジストリエントリ



第 5 章

セッション EJB によるビジネスルール管理

この章では、アプリケーションのビジネスルールおよびビジネスロジックをカプセル化するセッション EJB の作成方法について説明します。この章では特に、セッション Beans を使って、1 人の特定のユーザの一時的なニーズである繰り返しタスク、時限タスク、およびユーザ依存のタスクをカプセル化する方法について説明します。

この章には次の節があります。

- セッション EJB の紹介
- セッション Beans のコンポーネント
- セッション Beans の付加的なガイドライン

セッション EJB の紹介

標準的な分散アプリケーションの多くは、繰り返しタスク、時限タスク、およびユーザ依存のタスクを実行するコードの論理的な単位から構成されます。これらのタスクには、単純なものと複雑なものがあり、さまざまなアプリケーションで必要です。たとえば、バンキングアプリケーションでは、トランザクションの前には必ず顧客のアカウント ID と残高を照合する必要があります。このようなタスクは、業務上必要なビジネスルールおよびビジネスロジックを定義します。このような個々のタスクは、本来は一時的なものであるためセッション EJB に適していると考えられます。

セッション EJB は、クライアント固有の一般的なオブジェクトのインスタンスを表す自己完結型コードの単位です。本来、これらのオブジェクトは一時的なものであり、アプリケーションの生存期間を通して必要に応じて作成および解放されます。たとえば、Web ベースのオンラインショッピングアプリケーションでよく利用されるショッピングカートは、典型的なセッション Beans です。これは、アイテムが選択されたときにだけ、オンラインショッピングアプリケーションによって作成されます。アイテ

ムの選択が終了すると、カート内のアイテムの価格が計算され、注文が出され、ショッピングカートオブジェクトが解放されます。ユーザはオンラインカタログで商品のブラウズを続けることができ、ユーザが別の商品を注文したい場合には新しいショッピングカートが作成されます。

セッション Beans には、ほかのアプリケーションオブジェクトとの依存関係やコネクションを持たないものもあります。たとえば、ショッピングカートの Bean には、アイテム情報を格納するデータリストメンバー、現在カート内にあるアイテムの総額を格納するデータメンバー、およびアイテムの追加、削除、レポート、総計を行うメソッドがあります。一方、ショッピングカートには、購入可能なすべてのアイテムのデータベースへのライブコネクションはありません。

セッション Beans は、状態があるかないかのどちらかです。状態のないセッション Beans は、限られた時間内に特定のクライアントに必要なビジネスロジックの一時的な部分をカプセル化します。状態のあるセッション Beans も一時的ですが、会話型ステート (状態) を使って、クライアントがコールしている間の内容および値に関する情報を保持します。会話型ステートによって、Bean のコンテナはセッション Beans のステート情報を管理し、必要に応じてプログラム実行中にそのステートを再現できます。

アプリケーション内でパーシスタントでない独立したステータスを使って、セッション Beans の特性を定義する必要があります。セッション Beans は、アプリケーションサーバで動作するクライアントアプリケーションの一時的な論理拡張と考えることもできます。次にセッション Beans の特性を示します。

- 1 つのクライアントを対象に実行する
- 基礎となっているデータベースのデータを更新する
- 短命である

通常、セッション Beans はデータベースの共有データを表しませんが、データのスナップショットを取得します。ただし、Bean はデータを更新できます。オプションで、セッション Beans はトランザクション認識も可能です。このオペレーションは、Bean によって管理されるトランザクションのコンテキストで実行できます。

クライアントは、Bean のリモートインタフェースである `EJBObject` を介してセッション Beans にアクセスします。EJB オブジェクトは、リモートオブジェクト呼び出しの標準 Java API を介してクライアントからアクセスできる、リモート Java プログラミング言語オブジェクトです。EJB は作成されてから破棄されるまでコンテナ内に存在し、コンテナは EJB のライフサイクルとサポートサービスを管理します。EJB が存在し、実行される場所はクライアントに対して透過的です。さらに、単一のコンテナに複数の EJB を組み込むことができます。コンテナの提供するサービスによって、クライアントは、組み込まれた EJB クラスのインタフェースを Java Naming and Directory Interface (JNDI) を介して検索することができます。

クライアントがセッション Beans のインスタンスに直接アクセスすることはありません。そうではなく、クライアントはセッション Beans のリモートインタフェースを使って Bean のインスタンスにアクセスします。セッション Beans のリモートインタフェースを実装する EJB オブジェクトクラスは、コンテナによって提供されます。EJB オブジェクトは少なくとも、`java.ejb.EJBObject` インタフェースのすべてのメソッドをサポートします。これらのメソッドには、セッション Beans のホームインタフェースの取得、オブジェクトのハンドルの取得、そのオブジェクトがほかのオブジェクトと重複していないかどうかを調べるテスト、およびオブジェクトの削除などがあります。これらのメソッドは EJB 仕様書に明記されています。さらに、多くの EJB オブジェクトでは、特定のビジネスロジックメソッドもサポートしています。これらのメソッドはアプリケーションの中枢です。

すべての仕様書については、*installdir*`/ias/docs/index.htm` から参照できます。*installdir* は iPlanet Application Server がインストールされている場所です。

セッション Beans のコンポーネント

セッション Beans をプログラミングするには、次のクラスファイルを準備する必要があります。

- `javax.ejb.EJBObject` を拡張する Enterprise JavaBeans リモートインタフェース
- Enterprise JavaBeans クラス定義
- `javax.ejb.EJBHome` を拡張する Enterprise JavaBeans ホームインタフェース
- Enterprise JavaBeans メタデータ (配置記述子 (DD) およびほかの設定情報)

リモートインタフェースの作成

セッション Beans のリモートインタフェースは、Bean メソッドへのユーザアクセスを定義します。すべてのリモートインタフェースが、`javax.ejb.EJBObject` を拡張します。次のようにします。

```
import javax.ejb.*;
import java.rmi.*;
public interface MySessionBean extends EJBObject {
// ここでビジネスメソッドを定義します。
}
```

リモートインタフェースは、クライアントが呼び出すセッション Beans のビジネスメソッドを定義します。リモートインタフェース内で定義されたビジネスメソッドは、実行時に Bean のコンテナによって実行されます。リモートインタフェースで定義した各メソッドには、Bean クラス内の対応するメソッドを指定する必要があります。Bean クラス内の対応するメソッドには同じシグネチャが必要です。

リモートインタフェースで定義するビジネスメソッド以外に、`EJBObject` インタフェースはいくつかの抽象的なメソッドを定義します。これらのメソッドを使うと、Bean のホームインタフェースを取得したり、固有の識別子である Bean のハンドルを取得できるだけでなく、重複していないことを確認するためにほかの Bean と比較したり、不要になった Bean を解放または削除できます。

これらの組み込みメソッドとその使用法の詳細については、EJB 仕様書を参照してください。この仕様書は *install_dir*`/ias/docs/index.htm` からアクセス可能です。*install_dir* は、iPlanet Application Server がインストールされている場所です。

リモートインタフェースの宣言と実装

Bean クラス定義には、Bean のリモートインタフェースで定義されるメソッドごとに、一致するメソッド名、引数、および戻り値タイプなどのマッチングメソッド定義が 1 つ以上必要です。EJB 仕様書では、Bean クラスへのリモートインタフェースの直接実装が許可されていますが、一方で、仕様書で述べられているクライアント - コンテナ - EJB プロトコルに違反して、クライアントに `this` を介して直接参照を不用意に渡さないようにするために、この方法を行わないように勧めています。

クラス定義の作成

セッション Beans のクラスは、`abstract` ではなく `public` として定義する必要があります。Bean クラスは、`javax.ejb.SessionBean` インタフェースを実装する必要があります。次のようにします。

```
import java.rmi.*;
import java.util.*;
import javax.ejb.*;
public class MySessionBean implements SessionBean {
// エンティティ Beans の実装。これらのメソッドは常に取り込む必要があります。
public void ejbActivate() throws RemoteException {
}
public void ejbPassivate() throws RemoteException {
}
public void ejbRemove() throws RemoteException{
}
public void setSessionContext(SessionContext ctx) throws
RemoteException {
```

```
}

// 他のコードは省略します。
}
```

セッション Beans は、1 つまたは複数の `ejbCreate(...)` メソッドも実装する必要があります。クライアントが Bean を呼び出すときは、必ずこのメソッドを 1 つ使います。次のようにします。

```
public void ejbCreate() {
String[] userinfo = {"User Name", "Encrypted Password"} ;
}
```

各 `ejbCreate(...)` メソッドは `public` として宣言し、`void` を返し、`ejbCreate` という名前を付ける必要があります。引数は、正しいタイプの Java RMI である必要があります。`throws` 句は、アプリケーション固有の例外を定義し、`java.rmi.RemoteException` や `java.ejb.CreateException` を含めることができます。

すべての有効なセッション Beans も 1 つまたは複数のビジネスメソッドを実装します。これらのメソッドは通常、各 Bean に対して固有であり、その特定の機能を表します。たとえば、セッション Beans がユーザログインを管理する場合、このセッション Beans は `ValidateLogin()` という名前の固有の関数を取り込みます。

任意のビジネスメソッド名を付けることができますが、この EJB アーキテクチャで使われるメソッド名と重複しないように注意してください。ビジネスメソッドは、`public` として宣言する必要があります。メソッドの引数および戻り値は、正しいタイプの Java RMI である必要があります。`throws` 句は、アプリケーション固有の例外を定義し、`java.rmi.RemoteException` を含める必要があります。

セッション Beans のクラス定義で許可される 1 つのインタフェース実装があります。特に、`javax.ejb.SessionSynchronization` によって、セッション Beans インスタンスはトランザクション境界を認識し、ステートとそれらのトランザクションの同期をとることができます。このインタフェースの詳細については、EJB 仕様書を参照してください。この仕様書は *install_dir*`/ias/docs/index.htm` からアクセス可能です。*install_dir* は、iPlanet Application Server がインストールされている場所です。

セッションタイムアウト

コンテナは、指定した (またはデフォルトの) 時間を過ぎてもアクティブにならないセッション Beans を削除します。タイムアウト値は、Bean の配置記述子で設定します。詳細については、326 ページの「EJB XML DTD」を参照してください。

不活性化と活性化

コンテナは、指定した (またはデフォルトの) 時間を過ぎてもアクティブにならないセッション Beans を不活性化します。タイムアウト値は、Bean の配置記述子で設定します。詳細については、326 ページの「EJB XML DTD」を参照してください。

不活性化の詳細については、EJB 仕様書を参照してください。この仕様書は *install_dir*/ias/docs/index.htm からアクセス可能です。*install_dir* は、iPlanet Application Server がインストールされている場所です。

ホームインタフェースの作成

ホームインタフェースは、クライアントがアプリケーションを使ってセッションオブジェクトの作成および削除を行うメソッドを定義します。ホームインタフェースは通常、javax.ejb.EJBHome を拡張します。次のようにします。

```
import javax.ejb.*;
import java.rmi.*;

public interface MySessionBeanHome extends EJBHome {
   MySessionBean create() throws CreateException, RemoteException;
}
```

この例のように、セッション Beans のホームインタフェースは 1 つまたは複数の create メソッドを定義します。各メソッドには create という名前を付け、セッション Beans クラスで定義された ejbCreate メソッドの数および引数のタイプと一致させる必要があります。ただし、各 create メソッドの戻り値タイプは、対応する ejbCreate メソッドの戻り値タイプとは一致しません。代わりに、セッション Beans のリモートインタフェースタイプを返します。

ejbCreate メソッドの throws 句で定義するすべての例外が、リモートインタフェースで一致している create メソッドの throws 句で定義する必要があります。さらに、ホームインタフェースの throws 句には常に、javax.ejb.CreateException を含める必要があります。

すべてのホームインタフェースが、不要になった EJB を破棄する 2 つの remove メソッドを自動的に定義します。

注	これらのメソッドはオーバーライドしないでください。

セッション Beans の付加的なガイドライン

セッション Beans として表現できるアプリケーションの各部を決める前に、セッション Beans について理解すべき事項がいくつかあります。これらの事項には、セッション Beans の EJB 仕様に関連する事項や、iPlanet Application Server および iPlanet Application Server によるセッション Beans のサポートに固有の事項などがあります。

状態のない、または状態のある Beans の作成

EJB 仕様書には、セッション Beans の 2 つのステート管理モードについて記述されています。

- **状態なし** : Bean はメソッド呼び出し間のステート情報を保持しないので、すべての Bean インスタンスがすべてのクライアントに対応します。
- **状態あり** : Bean はメソッドおよびトランザクション間のステート情報を保持するので、特定の Bean インスタンスを常に 1 つのクライアントと関連付ける必要があります。

状態のあるセッション Beans を使う場合は、状態のある Bean とそのクライアントを同じ場所に配置してください。また、特に頻繁に不活性化および活性化するセッション Beans や、データベースコネクションやハンドルなど多数のリソースを使うセッション Beans については、スティッキーロードバランスを使って RPC の回数を減らします。

iPlanet Application Server 機能へのアクセス

開発可能なセッション Beans には、EJB 仕様に厳密に準拠しているセッション Beans、その仕様と iPlanet Application Server の付加価値機能の両方を活用しているセッション Beans、および iPlanet Application Server 以外の環境における仕様に準拠してはいるが利用できる iPlanet Application Server 機能も活用するセッション Beans があります。希望の配置シナリオに最適なセッション Beans を選択します。

ハンドルと参照の直列化

Bean 参照の直列化を保証するには、EJB への直接参照ではなくハンドルを使うようにと、EJB 仕様書に記述されています。

iPlanet Application Server の直接参照も直列化が可能です。ただし、すべてのベンダーによってサポートされているわけではないので、この拡張機能を利用できない場合もあります。

トランザクションの管理

多くのセッション Beans がデータベースと対話します。Bean のプロパティファイルの設定値を使って、Bean のトランザクションを制御します。これにより、Bean 配置時にトランザクション属性を指定できます。

コンテナ管理トランザクションまたは Bean 管理トランザクションを選択することができます。

コンテナがトランザクションを管理する場合は、Bean のデータベースアクセスメッソド内で明示してトランザクションを「起動」、「ロールバック」、または「コミット」する必要はありません。

トランザクション管理をコンテナレベルに移すことによって、Bean のアクティビティがデータベースアクセスと直接結び付いていなくても、データベース呼び出しと同じトランザクション制御環境ですべての Bean アクティビティを配置できます。これにより、コンテナによって制御されるアプリケーションのすべての部分が、同じトランザクションの一部として動作しますが、失敗すると、コンテナが管理していたものはすべてコミットされるか、ロールバックされます。実際には、コンテナ管理トランザクションのステートによって、同期ルーチンをプログラミングせずにアプリケーションの同期をとることができます。

セッション Beans が Bean 管理トランザクションを使用する場合は、Bean メソッド内で明示してトランザクションの有効範囲をコーディングする必要があります。

データベースへのアクセス

多くのセッション Beans はデータにアクセスしてデータを更新します。セッション Beans は一時的なので、アクセスがどのように発生するかに注意してください。一般に、JDBC API を使って呼び出しを行い、第 8 章「EJB のトランザクション処理」に記述されているトランザクションおよびセキュリティ管理メソッドを使ってトランザクション分離レベルおよびトランザクション要件を Bean レベルで管理します。

トランザクションの詳細については、第 9 章「JDBC を使ったデータベースアクセス」を参照してください。

セッション Beans のフェールオーバー

サービスが提供されないために iPlanet Application Server を使用できない場合は、セッション Beans のフェールオーバー機能によって、状態のあるセッション Beans に会話型ステートを回復することができます。状態のあるセッション Beans のフェールオーバーは、iPlanet Application Server の機能です。iPlanet Application Server のフェールオーバー機能をサポートするために J2EE プログラムに変更を加える必要はありません。フェールオーバーはコンテナによって処理され、アプリケーション配置者によって配置記述子内に定義されています。

たとえば、企業の購買担当者が電子商取引の Web サイトでオンライン購入を実行するとします。数時間後、購買担当者はショッピングカート (状態のあるセッション Beans) に何百種類もの商品を入れました。ここで、システムに予期しない致命的な問題が発生し、iPlanet Application Server サーバのインスタンスを使用できなくなりました。フェールオーバー機能がない場合、購買担当者のショッピングカートは空になってしまいます。つまり、状態のあるセッション Beans のステートが失われます。しかし、フェールオーバー機能がある場合は、購買担当者はシステムエラーに気づくことはありません。フェールオーバーメカニズムによって、エラー前の Bean のステートを持つ実行中の iPlanet Application Server サーバのインスタンスに、クライアントがリダイレクトされるためです。購買担当者のショッピングカートには、エラー発生前に選択したものと同じ商品が入っています。

状態のあるセッション Beans のフェールオーバーサポートの注目すべき特徴は次のとおりです。

- フェールオーバーは、J2EE プログラムをサポートする付加価値機能である
- フェールオーバーはクライアントに対して透過的であり、特別な API は不要
- フェールオーバーはコンテナによって処理され、アプリケーション配置者によって設定される
- 分散ストア (DSync : Distrubuted Store) はシステムエラー後にステートを復元するメカニズムである
- フェールオーバーのサポートが不要な状態のあるセッション Beans のパフォーマンスへの悪影響が最小限に抑えられる

状態のある Bean のフェールオーバーを設定する方法

フェールオーバーが機能するように状態のあるセッション Beans を設定するには、フェールオーバー機能と分散ストア (DSync) を Bean に組み合わせる必要があります。

- インストール時または実行時に、DSync 用にサーバを設定する
- 配置時に、状態のあるセッション Beans のフェールオーバーを設定する

フェールオーバー機能を活用するために、Bean のフェールオーバーと DSync の両方を設定する必要があります。実行時、DSync メカニズムによってセッション Beans の会話型ステートが保存されます。さらに、フェールオーバーメカニズムによって、コンテナは、システムエラーを検出して、保存されたセッション Beans のステートが含まれた別の実行中の iPlanet Application Server インスタンスに接続できます。

配置時に状態のあるセッション Beans のフェールオーバーを設定する方法と、実行時に DSync を設定する方法の詳細については、『管理ガイド』を参照してください。インストール時に DSync を設定する方法の詳細については、『インストールガイド』を参照してください。

フェールオーバープロセスの動作

状態のある Bean のフェールオーバーは、スマートスタブと分散ストアの組み合わせによって実現されます。Bean をフェールオーバー Bean として配置すると、配置ツールによって特殊なスタブが生成されます。メソッドを起動すると、スマートスタブはエラーを検出し、Bean を透過的に新しい場所 (別のエンジン内など) に移動します。スタブは、無効 Bean からコネクション例外を取得することによって、Bean 参照が古くなっていないかどうかを調べます。スタブは、さらに Bean の場所を検索し、リモートインタフェースを取得します。Bean が再配置されると、スタブは、復元された Bean に対してメソッドを再度実行します。コンテナは、メソッドを試行し直した場合、セマンティックを一度限り保証します。

コンテナは、DSync に基づいた分散ストアを使って Bean のステートを維持します。Bean のステートは通常の間隔で保存され、リカバリプロセスの一環として自動的に復帰します。

フェールオーバーが設定された状態のあるセッション Beans によって使われる配置記述子の詳細については、第 11 章「配置のためのパッケージ化」を参照してください。

フェールオーバーのガイドライン

フェールオーバーを実装する場合は、次のガイドラインに従ってください。

- `ejbPassivate()` および `ejbActivate()` を簡潔にします。
- `home.remove(handle)` ではなく、`obj.remove()` を使って Bean を削除します。Bean とその元の場所の関連付けはフェールオーバー後も保持されないことがあります。
- Bean のフェールオーバーの利点とフェールオーバープロセスのコストパフォーマンスを慎重に比較検討した上で判断してください。

注 すべての状態のある Bean にフェールオーバー機能を設定しないようにしてください。

- セッション Beans のステートは会話型であることを覚えておいてください。トランザクションデータにはエンティティ Beans を使います。
- 状態のあるセッション Beans のステートを保存する間隔は、EJB タブの管理ツールを使って設定できます。デフォルトでは 10 秒です。
- Bean がトランザクションの場合、タイマーベースのステートの保存は、トランザクション時に自動的に無効になります。これにより、トランザクション時にサーバエンジンにエラーが発生してもトランザクションデータの安全性が確保されます。エラーが発生すると、トランザクションデータベースの更新は、データベースによりロールバックされます。回復した Bean のステートは、失敗したトランザクションの開始時のステートと同じです。ただし、トランザクションが問題なく処理された場合、Bean のステートはトランザクションの完了時に保存され、次のトランザクションが開始されるまでタイマーベースで再び保存されます。
- Bean が iPlanet Application Server が提供する `com.netscape.server.ejb.IEBFoStateModification` インタフェースを実装している場合、ステートセーバは、時間のかかる保存操作を行う前に Bean のステートが変更されているかどうかを確認できます。このインタフェースは次の 2 つのメソッドを定義します。

```
package com.netscape.server.ejb;

public interface IEBFoStateModification {

    /**
     ** Bean インスタンスが正しくない場合、コンテナによってこのメソッドが
     ** 呼び出され、確認を行います。
     **/

    boolean isDirty();

    /**
     ** コンテナが直接保存を実行する場合があります。次にコンテナは、
     ** 変更済みの Bean の不正なステートを呼び出し、リセットします。
     **/

    void setDirty(boolean dirty);

}
```

ユーザが提供する Bean の実装には、Bean の変更済みステートを追跡するブール変数があります。この変数は、ステートの保存前に参照されます。

ステートの保存間隔

フェールオーバーが設定されているコンテナは、実行時、Bean のステートを一定間隔で保存します。ステートを保存するプロセスは次のとおりです。

- 一定間隔 (間隔は再設定可能) で保存します。

- Bean がトランザクションに関わっている場合には、トランザクション境界上に保存します。

一定間隔は、管理ツールで設定します。

ステートの保存方法

ステートを保存するプロセスは次のとおりです。

- 最初に、状態のある各セッション Beans の `ejbPassivate()` メソッドを呼び出します。
- 次に、Bean の会話型ステートを直列化し、分散ストアに保存します。
- 最後に、Bean の `ejbActivate()` メソッドを呼び出します。

注	Bean のステート保存は、関連する操作の規模が大きいため時間がかかります。

第 6 章

エンティティ EJB のビルド

この章では、エンティティ EJB とエンティティ Beans の構成について説明します。また、アプリケーションでエンティティ Beans を作成したり、エンティティ Beans のニーズを決定する際に役立つ付加的なガイドラインも提供されます。

この章には次の節があります。

- エンティティ EJB の紹介
- エンティティ Beans のコンポーネント
- エンティティ Beans の付加的なガイドライン
- コンテナ管理パーシスタンス

この仕様書は *install_dir*`/ias/docs/index.htm` からアクセス可能です。*install_dir* は、iPlanet Application Server がインストールされている場所です。

エンティティ EJB の紹介

分散マルチユーザアプリケーションの中心部は、データベースや既存のレガシーアプリケーションなどのトランザクションのデータソースとも頻繁に対話します。ほとんどの場合、外部データソースあるいはビジネスオブジェクトはユーザにとって透過的か、またはダイレクトユーザ対話から隠されているか保護されています。これらの保護されたパーシスタントなトランザクション対話 (データベース、ドキュメント、およびほかのビジネスオブジェクトとの対話) はエンティティ EJB へのカプセル化に適しています。

EJB は、データメンバー、プロパティ、およびメソッドを持つ自己完結型の再利用可能なコンポーネントです。これらは、クライアント間での共有が可能な、トランザクション認識型のパーシスタントデータオブジェクトの一般的なインスタンスを表します。パーシスタンスとは、アプリケーションの生存期間を通じた Bean の作成と維持のことです。

パーシスタンス管理には次の 2 種類があり、iPlanet Application Server ではその両方をサポートしています。

- **コンテナ管理パーシスタンス** - コンテナが、Bean のパーシスタンスに対して責任を負う
- **Bean 管理パーシスタンス** - Bean が、Bean 自体のパーシスタンスに対して責任を負う

ユーザは開発者として、JDBC と SQL を介してデータベースアクセス呼び出しを Bean クラスのメソッドに直接提供することによって、Bean 管理エンティティ Beans をコード化します。データベースアクセス呼び出しは、`ejbCreate()`、`ejbRemove()`、`ejbFind`*XXX*`()`、`ejbLoad()`、および `ejbStore()` メソッドに存在する必要があります。Bean 管理パーシスタンスを使う利点は、コンテナがデータベースを呼び出さなくても、コンテナに Bean を入れることができる点にあります。

エンティティ Beans は、コンテナが管理するエンティティオブジェクトのセキュリティ、同時性、トランザクション、およびほかのコンテナ固有のサービスを管理するようにコンテナに要求します。複数のクライアントによるエンティティオブジェクトへの同時アクセスが可能で、コンテナは、トランザクションを介した同時アクセスを透過的に処理します。

アプリケーション開発者として、コンテナのエンティティ Beans サービスに直接アクセスすることはできず、その必要性もありません。コンテナは低レベルの実装詳細を管理するために存在するので、開発者は、アプリケーション全体でエンティティ Beans が果たす、より重要な役割の管理に専念できます。

クライアントは、Bean のリモートインタフェースを介してエンティティ Beans にアクセスします。リモートインタフェースを実装するオブジェクトを EJB オブジェクトと呼びます。通常、エンティティ EJB は複数のクライアントに共有され、データベースなどのデータリソースやビジネスオブジェクトに 1 つのエントリポイントを示します。指定された時間にどのクライアントがエンティティオブジェクトにアクセスするのかにかかわらず、各クライアントのオブジェクトのビューは場所に関係なく、ほかのクライアントに対して透過的です。

さらに、コンテナに組み込まれるエンティティ Beans の数に制限はありません。コンテナは各エンティティ Beans のホームインタフェースを実装します。ホームインタフェースによって、クライアントはエンティティオブジェクトの作成、検索、および削除ができます。クライアントは、Java Naming and Directory Interface (JNDI) を介してエンティティ Beans のホームインタフェースを検索します。

エンティティ Beans には次の属性があります。

- データベースのデータを表示する
- トランザクションをサポートする
- 複数のクライアントを対象に実行する

- すべてのクライアントにとって不要になるまで持続する
- サーバクラッシュを透過的に回避する

一般に、エンティティ Beans はデータベースの共有データを表し、トランザクション認識型です。通常、Bean のコンテナによって管理されるトランザクションのコンテキスト内で動作します。

エンティティ Beans へのアクセス

ブラウザや Servlet などのクライアントは、Bean のリモートインタフェースである `EJBObject` を介してエンティティ Beans にアクセスします。EJB オブジェクトは、リモートオブジェクト呼び出しの標準 Java API を介してクライアントからアクセスできる、リモート Java プログラミング言語オブジェクトです。EJB は作成されてから破棄されるまでコンテナ内に存在し、コンテナは EJB のライフサイクルとサポートサービスを管理します。

クライアントがエンティティ Beans のインスタンスに直接アクセスすることはありません。そうではなく、クライアントはエンティティ Beans のリモートインタフェースを使って Bean インスタンスにアクセスします。エンティティ Beans のリモートインタフェースを実装した EJB オブジェクトクラスは、コンテナによって提供されます。EJB オブジェクトは少なくとも、`java.ejb.EJBObject` インタフェースのすべてのメソッドをサポートします。これらのメソッドには、エンティティ Beans のホームインタフェースの取得、オブジェクトハンドルの取得、エンティティのプライマリキーの取得、そのオブジェクトがほかのオブジェクトと重複していないかどうかを調べるテスト、およびオブジェクトの削除などがあります。これらのメソッドは EJB 仕様書に明記されています。さらに、多くの EJB オブジェクトのリモートインタフェースは、特定のビジネスロジックメソッドもサポートしています。これらのメソッドがアプリケーションの中枢です。

この仕様書は *install_dir*`/ias/docs/index.htm` からアクセス可能です。*install_dir* は、iPlanet Application Server がインストールされている場所です。

エンティティ Beans のコンポーネント

エンティティ Beans を作成するには、次のクラスファイルを準備する必要があります。

- Enterprise JavaBeans クラス
- `javax.ejb.EJBHome` を実装する Enterprise JavaBeans ホームインタフェース
- `javax.ejb.EJBObject` を実装する Enterprise JavaBeans リモートインタフェース

クラス定義の作成

エンティティ Beans の Bean クラスは、`abstract` ではなく `public` として定義する必要があります。Bean クラスは、javax.ejb.EntityBean インタフェースを実装する必要があります。次のようにします。

```
import java.rmi.*;
import java.util.*;
import javax.ejb.*;
public class MyEntityBean implements EntityBean {
// エンティティ Beans の実装。これらのメソッドは常に取り込む必要があります。
public void ejbActivate() throws RemoteException {
}
public void ejbLoad() throws RemoteException {
}
public void ejbPassivate() throws RemoteException {
}
public void ejbRemove() throws RemoteException{
}
public void ejbStore() throws RemoteException{
}
public void setEntityContext(EntityContext ctx) throws
RemoteException {
}
public void unsetEntityContext() throws RemoteException {
}
// 他のコードは省略します。
}
```

これらのメソッドに加えて、エンティティ Beans のクラスは 1 つまたは複数の `ejbCreate()` メソッドと `ejbFindByPrimaryKey()` ファインダーメソッドを定義する必要があります。オプションで、各 `ejbCreate()` メソッドに `ejbPostCreate()` メソッドを定義できます。また、`ejbFind`*XXX* 形式の開発者定義ファインダーメソッドを追加することもあります。ここで、*XXX* はほかのメソッド名と重複しない連続した固有のメソッド名を示します (例 ：`ejbFindApplesAndOranges`)。

さらに、多くの有効なエンティティ Beans は、1 つまたは複数のビジネスメソッドを実装します。これらのメソッドは通常、各 Bean に対して固有であり、その特定の機能を表します。任意のビジネスメソッド名を付けることができますが、この EJB アーキテクチャで使われるメソッド名と重複しないように注意してください。ビジネスメソッドは、`public` として宣言する必要があります。メソッド引数および戻り値は Java RMI の正しいタイプである必要があります。`throws` 句は、アプリケーション固有の例外を定義し、`java.rmi.RemoteException` を含めることもできます。

エンティティ Beans を実装するには、次の 2 つのタイプのビジネスメソッド使用します。

- 内部ビジネスメソッドは、Bean 内のほかのビジネスメソッドに使われます。Bean の外部からアクセスされることはありません。
- 外部ビジネスメソッドは、エンティティ Beans のリモートインタフェースによって参照されます。

次の節では、エンティティ Beans のクラス定義のさまざまなメソッドについて説明します。

これらの節に示す、次のメンバー変数定義があると想定します。

```
private transient javax.ejb.EntityContext m_ctx = null;

// これらは、我々の Bean のステートを定義します。
private int m_quantity;
private int m_totalSold;
```

ejbActivate と ejbPassivate の使用法

サーバアプリケーションでエンティティ Beans のインスタンスが必要になると、Bean のコンテナは、`ejbActivate()` を呼び出して Bean インスタンスを準備します。同様に、アプリケーションでインスタンスが不要になると、Bean のコンテナは、`ejbPassivate()` を呼び出してアプリケーションから Bean を分離します。

アプリケーション用に初めて Bean を準備するときや Bean が不要になったときに、特定のアプリケーションタスクを実行する必要がある場合は、これらのメソッド内でそのような操作をプログラミングします。

Bean の活性化と作成は異なります。活性化は、すでに作成されている Bean に対して行います。同様に、Bean の不活性化と削除も異なります。不活性化とは、将来の使用に備えてコンテナプールに Bean インスタンスを返すことです。Bean インスタンスを削除するには、`ejbRemove()` を呼び出します。

コンテナは、指定した (またはデフォルトの) 時間を過ぎてもアクティブにならないエンティティ Beans を不活性化します。このタイムアウト値は、Bean のプロパティファイルで設定されます。詳細については、326 ページの「EJB XML DTD」を参照してください。

`ejbCreate()` および `ejbRemove()` の詳細については、156 ページの「ejbCreate メソッドの使用法」を参照してください。

`ejbActivate()` および `ejbPassivate()` の詳細については、EJB 仕様書を参照してください。この仕様書は *install_dir*`/ias/docs/index.htm` からアクセス可能です。*install_dir* は、iPlanet Application Server がインストールされている場所です。

ejbLoad と ejbStore の使用法

Bean 管理パーシスタンスを使用するエンティティ Beans は、コンテナが同期をとるためにデータベースに Bean のステート情報を格納することを許可します。`ejbStore()` を実装するとデータベースにステート情報が保存され、`ejbLoad()` を実装するとデータベースからステート情報が取り出されます。コンテナは、`ejbLoad()` を呼び出し、データベースからステート情報を読み込むことによって Bean のステートと同期をとります。

エンティティ Beans がコンテナ管理パーシスタンスを使う場合は、データベースアクセスレイヤをコーディングする必要はありません。

次の例は、アクティブデータの格納と取得を行う `ejbLoad()` および `ejbStore()` メソッドの定義を示します。

```
public void ejbLoad()
       throws java.rmi.RemoteException
{
   String itemId;
   DatabaseConnection dc = null;
   java.sql.Statement stmt = null;
   java.sql.ResultSet rs = null;

   itemId = (String) m_ctx.getPrimaryKey();

   System.out.println("myBean:Loading state for item " + itemId);

   String query =
       "SELECT s.totalSold, s.quantity " +
       " FROM Item s " +
```

```
        " WHERE s.item_id = " + itemId;

    dc = new DatabaseConnection();
    dc.createConnection(DatabaseConnection.GLOBALTX);
    stmt = dc.createStatement();
    rs = stmt.executeQuery(query);

    if (rs != null) {
        rs.next();
        m_totalSold = rs.getInt(1);
        m_quantity = rs.getInt(2);
    }
}

public void ejbStore()
        throws java.rmi.RemoteException
{
    String itemId;
    itemId = (String) m_ctx.getPrimaryKey();
    DatabaseConnection dc = null;
    java.sql.Statement stmt1 = null;
    java.sql.Statement stmt2 = null;

    System.out.println("myBean:Saving state for item = " + itemId);

    String upd1 =
        "UPDATE Item " +
        " SET quantity = " + m_quantity +
        " WHERE .item_id = " + itemId;

    String upd2 =
        "UPDATE Item " +
        " SET totalSold = " + m_totalSold +
        " WHERE .item_id = " + itemId;

    dc = new DatabaseConnection();
    dc.createConnection(DatabaseConnection.GLOBALTX);
    stmt1 = dc.createStatement();
    stmt1.executeUpdate(upd1);
    stmt1.close();
    stmt2 = dc.createStatement();

    stmt2.executeUpdate(upd2);
    stmt2.close();
}
```

ほかの Bean と同時にトランザクションにアクセスする Bean の分離レベルの詳細については、162 ページの「同時アクセスの処理」を参照してください。

setEntityContext と unsetEntityContext の使用法

コンテナは、エンティティ Beans のインスタンスを作成してコンテナに Bean のインタフェースを指定してから setEntityContext() を呼び出します。このメソッドを実装すると、インスタンス変数にコンテナの参照を保存できます。

```
public void setEntityContext(javax.ejb.EntityContext ctx)
{
m_ctx = ctx;
}
```

同様に、コンテナは unsetEntityContext() を呼び出して、インスタンスからコンテナ参照を削除します。これは、コンテナによって呼び出される最後の Bean クラスメソッドです。この呼び出しのあと、Java ガベージコレクションメカニズムは最終的にそのインスタンスに対して finalize() を呼び出し、そのインスタンスをクリーンアップして廃棄します。

```
public void unsetEntityContext()
{
m_ctx = null;
}
```

ejbCreate メソッドの使用法

エンティティ Beans は、1 つまたは複数の ejbCreate(...) メソッドを実装する必要もあります。クライアントが Bean を呼び出すたびに、必ずこのメソッドを 1 つ使います。次のようにします。

```
public integer ejbCreate() {
   string[] userinfo = {"User Name", "Encrypted Password"};
}
```

各 ejbCreate() メソッドは、public として宣言され、エンティティのプライマリキータイプまたはコレクションのどれかを返し、ejbCreate という名前を付ける必要があります。戻り値のタイプは、正しい Java RMI タイプであれば何であってもかまいません。引数はすべて、正しいタイプの Java RMI である必要があります。throws 句は、アプリケーション固有の例外を定義し、java.rmi.RemoteException や javax.ejb.CreateException を含めることができます。

各 ejbCreate() メソッドに対して、エンティティ Beans のクラスで作成後すぐにエンティティサービスを処理する ejbPostCreate() メソッドを定義することができます。各 ejbPostCreate() メソッドは public として宣言し、void を返し、ejbPostCreate という名前を付ける必要があります。メソッド引数がある場合は、対応する ejbCreate メソッドの数および引数のタイプと一致させる必要があります。throws 句は、アプリケーション固有の例外を定義し、java.rmi.RemoteException や java.ejb.CreateException を含めることができます。

さらに、エンティティ Beans は 1 つまたは複数の ejbRemove() メソッドを実装し、不要になった Bean を解放します。

ファインダーメソッドの使用法

エンティティ Beans はパーシスタントであり、複数のクライアント間で共有され、一度に複数回インスタンス化される可能性があります。したがって、エンティティ Beans は少なくとも 1 つの ejbFindByPrimaryKey() メソッドを実装する必要があります。このメソッドによって、クライアントおよび Bean のコンテナは特定の Bean インスタンスを探すことができます。すべてのエンティティ Beans には、識別署名として固有のプライマリキーを指定する必要があります。ejbFindByPrimaryKey() メソッドを Bean クラスに実装し、Bean がそのプライマリキーをコンテナに返すことができるようにします。

次の例は、FindByPrimaryKey() の定義を示しています。

```
public String ejbFindByPrimaryKey(String key)
       throws java.rmi.RemoteException
          javax.ejb.FinderException
{
   //System.out.println("@@@ myBean.ejbFindByPrimaryKey key = " +
key);
   return key;
}
```

特定のエンティティ Beans のインスタンスは、その Bean の動作内容、そのインスタンスが操作する値などの条件に基づいて検索される場合があります。特定の実装に関するこれらのファインダーメソッドの名前は ejbFindXXX の形式です。ここで、XXX は、ほかのメソッド名と重複しない連続した固有のメソッド名を表します (例 : ejbFindAppleSAndOranges)。

ファインダーメソッドは public として宣言され、その引数および戻り値は正しいタイプの Java RMI である必要があります。各ファインダーメソッドの戻り値のタイプは、エンティティ Beans のプライマリキータイプか、同じプライマリキータイプのオブジェクトのコレクションである必要があります。戻り値のタイプがコレクションのとき、次のいずれかである必要があります。

- JDK 1.1 の java.util.Enumeration インタフェース
- Java 2 の java.util.Collection インタフェース

ファインダーメソッドの throws 句は、アプリケーション特有の例外であり、java.rmi.RemoteException や javax.ejb.FinderException が含まれる場合があります。

リモートインタフェースの宣言と実装

Bean クラス定義には、Bean のリモートインタフェースで定義される各メソッドごとに、一致するメソッド名、引数、および戻り値タイプなどのマッチングメソッド定義が 1 つ以上必要です。EJB 仕様書では、Bean クラスへのリモートインタフェースのメソッドの実装が許可されています。ただし、一方では、仕様書に述べられているクライアント - コンテナ - EJB プロトコルに違反して、クライアントに `this` を介して直接参照を不用意に渡さないようするため、この方法を使用しないよう勧めています。

ホームインタフェースの作成

ホームインタフェースは、クライアントがアプリケーションにアクセスしてエンティティオブジェクトを作成したり削除したりするメソッドを定義します。ホームインタフェースは通常、`javax.ejb.EJBHome` を拡張します。次のようにします。

```
import javax.ejb.*;
import java.rmi.*;

public interface MyEntityBeanHome extends EJBHome {
   MyEntityBean create() throws CreateException, RemoteException;
}
```

この例では、1 つまたは複数の create メソッドを定義するエンティティ Beans のホームインタフェースについて説明します。ホームインタフェースは通常、Bean クラス内のファインダーメソッドに対応する 1 つまたは複数の find メソッドを定義します。

create メソッドの定義

各メソッドには create という名前を付け、エンティティ Beans クラスで定義された `ejbCreate` メソッドの数および引数のタイプと一致させる必要があります。ただし、各 create メソッドの戻り値タイプは、対応する `ejbCreate` メソッドの戻り値タイプとは一致しません。そうではなく、エンティティ Beans のリモートインタフェースタイプを返します。

`ejbCreate` メソッドの `throws` 句で定義するすべての例外を、ホームインタフェースで一致している create メソッドの `throws` 句で定義する必要があります。さらに、ホームインタフェースの `throws` 句には常に、`javax.ejb.CreateException` を含める必要があります。

find メソッドの定義

ホームインタフェースは、1 つまたは複数の find メソッドを定義できます。各メソッドには `findXXX` という名前を付ける必要があります (`findApplesAndOranges` など)。ここで、*XXX* は連続した固有のメソッド名を表します。各ファインダーメソッドは、エンティティ Beans のクラス定義で定義されたファインダーメソッドの 1 つと一致する必要があります。

その数および引数のタイプも Bean クラスのファインダーメソッド定義と一致する必要があります。しかし、戻り値タイプは異なる場合があります。ホームインタフェースのファインダーメソッドの戻り値のタイプは、エンティティ Beans のリモートインタフェースタイプか、インタフェースのコレクションである必要があります。さらに、ホームインタフェースの `throws` 句には常に、`javax.ejb.FinderException` を含める必要があります。

さらに、すべてのホームインタフェースが、不要になった EJB を破棄する 2 つの remove メソッドを自動的に定義します。

注	これらのメソッドはオーバーライドしないでください。

リモートインタフェースの作成

エンティティ Beans のリモートインタフェースは、Bean のメソッドへのユーザアクセスを定義します。すべてのリモートインタフェースが、`javax.ejb.EJBObject` を拡張します。次のようにします。

```
import javax.ejb.*;
import java.rmi.*;
public interface MyEntityBean extends EJBObject {
// ここでビジネスメソッドを定義します。
}
```

リモートインタフェースは、クライアントが呼び出すエンティティ Beans のビジネスメソッドを定義します。リモートインタフェース内で定義されたビジネスメソッドは、実行時に Bean のコンテナによって実行されます。リモートインタフェースで定義した各メソッドには、Bean クラス内の対応するメソッドを指定する必要があります。Bean クラス内の対応するメソッドには同じシグネチャが必要です。

リモートインタフェースで定義するビジネスメソッド以外に、`EJBObject` インタフェースはいくつかの抽象的なメソッドを定義します。これらのメソッドを使うと、Bean のホームインタフェースを取得したり、Bean のハンドルを取得できるだけでなく、Bean インスタンスを一意に識別する Bean のプライマリキーを取得したり、重複していないことを確認するためにほかの Bean と比較したり、不要になった Bean を削除したりすることもできます。

これらの組み込みメソッドとその使用法の詳細については、EJB 仕様書を参照してください。この仕様書は *install_dir*/ias/docs/index.htm からアクセス可能です。*install_dir* は、iPlanet Application Server がインストールされている場所です。

エンティティ Beans の付加的なガイドライン

エンティティ Beans として表すことができるアプリケーションの各部を決める前に、知っておくべき事項がいくつかあります。これらには、エンティティ Beans の EJB 仕様に関連する事項と、iPlanet Application Server および iPlanet Application Server によるエンティティ Beans のサポートに固有の事項があります。

iPlanet Application Server 機能へのアクセス

開発可能なエンティティ Beans には、EJB 仕様に厳密に準拠しているエンティティ Beans、その仕様と iPlanet Application Server の付加価値機能の両方を活用しているエンティティ Beans、および iPlanet Application Server 以外の環境での仕様に準拠してはいるが利用できる iPlanet Application Server 機能も活用するエンティティ Beans があります。希望の配置シナリオに最適な Bean を選択します。

iPlanet Application Server は、iPlanet Application Server コンテナを介していくつかの機能を提供します。また、iPlanet Application Server の API によって、アプリケーションは特定の iPlanet Application Server 環境での機能をプログラムで利用できます。iPlanet Application Server 環境だけでエンティティ Beans を使う予定であれば、これらの Bean に API 呼び出しを埋め込むことができます。

ハンドルと参照の直列化

Bean 参照の直列化を保証するには、EJB への直接参照ではなくハンドルを使うようにと、EJB 仕様書に記述されています。

iPlanet Application Server の直接参照も直列化が可能です。ただし、すべてのベンダーによってサポートされているわけではないので、この拡張機能を利用できない場合もあります。

トランザクションの管理

多くのエンティティ Beans はデータベースと対話します。Bean のプロパティファイルの設定値を使って、Bean のトランザクションを制御します。これにより、Bean 配置時にトランザクション属性を指定できます。

エンティティ Beans の場合は、コンテナ管理トランザクションが使用できます。

コンテナがトランザクションを管理する場合は、Bean のデータベースアクセスメソッド内で明示してトランザクションを「起動」、「ロールバック」、または「コミット」する必要はありません。

トランザクション管理をコンテナレベルに移すことによって、Bean のアクティビティがデータベースアクセスと直接結び付いていなくても、データベース呼び出しと同じトランザクション制御環境ですべての Bean アクティビティを配置できます。これにより、コンテナによって制御されるアプリケーションのすべての部分が、同じトランザクションの一部として動作しますが、失敗すると、コンテナが管理していたものはすべてコミットされるか、ロールバックされます。実際には、コンテナ管理トランザクションのステートによって、同期ルーチンをプログラミングせずにアプリケーションの同期をとることができます。

トランザクションのコミット

コミットが発生し、エンティティ Beans の有効な作業が終了したことがコンテナに伝わると、コンテナは、基礎となっているデータソースとステートの同期をとります。コンテナはトランザクションの終了を許可し、将来の使用に備えて Bean をプールに返します。コミットされたトランザクションに関連付けられたリザルトセットは無効になります。同じ Bean に対する連続したリクエストによって、基礎となっているデータソースとステートの同期をとる際にコンテナの負荷が発生します。

コンテナで開始されたトランザクションが暗黙的にコミットされることに注意してください。また、トランザクションに関連した Bean であれば、トランザクションをロールバックできます。トランザクションの詳細については、第 8 章「EJB のトランザクション処理」を参照してください。

コミットオプション C

コミットオプション C は、iPlanet Application Server によってサポートされます。コミットオプション C は、トランザクション開始時に空きプールから Bean のインスタンスを取得し、トランザクション終了時に空きプールにインスタンスを戻します。

コミットオプション C における各ビジネスメソッド起動のライフサイクルは次のようになります。

```
ejbActivate-> ejbLoad -> business method -> ejbStore -> ejbPassivate
```

同じエンティティ EJBObject に同時にアクセスしているトランザクションクライアントが 1 つ以上ある場合、最初のクライアントはレディインスタンスを取得し、次の同時アクセスのクライアントはプールから新規インスタンスを取得します。

同時アクセスの処理

エンティティ Beans の開発者は、複数のトランザクションからのエンティティ Beans への同時アクセスについて心配する必要はありません。このような場合、Bean のコンテナは自動的に同期をとります。iPlanet Application Server では、コンテナは、Bean を使う各同時発生トランザクションのエンティティ Beans のインスタンスを活性化します。トランザクション同期は、データベースアクセス呼び出し時に基礎となっているデータベースによって自動的に実行されます。

iPlanet Application Server の EJB コンテナ実装には、複数のトランザクションがエンティティ Beans にアクセスするときの同期メカニズムがありません。すべての新規トランザクションにエンティティ Beans の新規インスタンスを作成します。iPlanet Application Server コンテナはアプリケーションの同期の責任を委譲します。

通常は、基礎となっているデータベースやリソースと連携して同期をとります。Bean 管理パーシスタンスを使用している場合の手法の 1 つは、たとえば適切な分離レベルを選択したり、select for update 句を使うことによって、ejbLoad() メソッド内で対応するデータベースをロックすることです。その特性は、使われているデータベースによって異なります。詳細については、EJB 仕様書の同時アクセスに関連する節を参照してください。

次の例に示す ejbLoad() のコード抜粋は、データベースロックを取得するための select for update シンタックスを示しています。これにより、ほかのインスタンスが同時に読み込まれることを防ぎます。

```
public void ejbLoad() throws java.rmi.RemoteException
{
....
// 対応するデータベーステーブルのロックを取得します。
   try {
       java.sql.Connection dbConn = ds.getConnection();
       String query = "SELECT accountNum, balance FROM accounts "
          + "WHERE customerId = ?FOR UPDATE";
       prepStmt = dbConn.prepareStatement(query);
       prepStmt.setString(1, m_customerId);
       resultSet = prepStmt.executeQuery();
       if ((resultSet != null) && resultSet.next()) {
          acctNum = resultSet.getInt(1);
          acctBalance = resultSet.getInt(2);
       } else {
```

```
            throw new RemoteException("Database error. "
                   + "Couldn't find accout");
          }
          catch (java.sql.SQLException e) {
           throw new RemoteException("Database error. "
                 + "Couldn't load account");
      } finally {
          try {
              if (resultSet != null)
                 resultSet.close();
              if (prepStmt != null)
                 prepStmt.close();
              if (dbConn != null)
                 dbConn.disconnect();
              } catch (java.sql.SQLException e) {
              System.out.println("Unexpected exception while "
                 + "closing resources"); }
      }
}
```

コンテナ管理パーシスタンス

コンテナ管理パーシスタンス (CMP) を使うエンティティ Beans では、そのステート (またはパーシスタンス) の管理を iPlanet Application Server が行います。通常、CMP Bean はリレーショナルデータベースに従います。

開発者は CMP を使って、エンティティ Beans の作成作業を簡素化できます。CMP を使う開発者は、BMP エンティティ Beans の実装に必要な JDBC コードをすべて書き込む必要がなくなり、Bean 配置記述子を作成するためにツールを使うだけです。配置記述子には、リレーショナルデータベースのカラムを指す Bean にフィールドをマップするときにコンテナが使う情報が含まれています。

CMP の詳細については、EJB 1.1 仕様書の第 9.4 節を参照してください。

iPlanet Application Server には、CMP エンティティ Beans に関連する次のサポートが用意されています。

- J2EE v 1.2 仕様の CMP モデル (例 : EJB 1.1) のフルサポート
- サードパーティの O/R マッピングツールのサポート
- 独創的ライトウェイト CMP の実装。ライトウェイト CMP の特長は次のとおりです。

- iPlanet Application Server 配置ツール内の基本的なオブジェクトとリレーショナル (O/R) 間のマッピングツール。各 CMP Bean の XML 配置記述子を作成します。
- 複合 (マルチカラム) プライマリキーのサポート
- 高度なカスタムファインダーメソッドのサポート
- 標準ベースクエリ言語 (SQL92)

J2EE 完全サポート

iPlanet Application Server は、EJB 1.1 仕様書に定義されているエンティティ Beans コンポーネント規約を完全サポートしています。次に主な項目を示します。

- iPlanet Application Server は、EJB 1.1 仕様書に定義されているコミットオプション C を実装する
- プライマリキークラスは、`java.lang.Object` のサブクラスである必要がある。これは仕様書に準拠し、移植性を保証しますが、基本的なタイプ (`int` など) をプライマリキークラスとして一覧表示するベンダーもあるので記載しています。

サードパーティの O/R マッピングツール

iPlanet Application Server ではサードパーティのツールベンダーの使用が認定されています。一般に、EJB1.1 仕様を完全にサポートする、サードパーティの CMP ソリューションは、iPlanet Web Server とともに動作します。

たとえば、Thought, Inc. は、マッピング EJB の高度な O/R マッピングソリューションとして、リレーショナルデータベースに CocoBase Enterprise を提供しています。Cocobase を使うには、Cocobase の O/R マッピングツールを使って EJB をビルドし、iPlanet Application Server の配置ツールまたは Command Line Interface (CLI) を使って Bean を配置します。

ほかのいくつかのベンダーは、現在、認定中です。認定済みサードパーティの O/R マッピングツールに関する現在の情報については、次の Web サイト `developer.iplanet.com` で確認してください。

CMP エンティティ Beans の例

CMP エンティティ Beans の例については、次の場所にある『J2EE Developer's Guide』から製品サンプルアプリケーションを参照してください。

install_dir/ias/ias-samples/j2eeguide/product

ライトウェイト CMP 実装の使用法

iPlanet Application Server では、独創的なライトウェイト CMP 実装を提供しています。実装には iPlanet Application Server 配置ツールにあるマッピングツールと CMP 実行時環境が含まれています。CMP 実行時環境では、各 CMP Bean にパーシスタンスマネージャを作成します。パーシスタンスマネージャは XML 配置記述子で指定される情報を使います。CMP Bean で使われる 3 つの配置記述子は次のとおりです。

- `ejb-jar.xml` - 各 EJB モジュールに `ejb-jar.xml` ファイルが 1 つあります。この配置記述子は EJB 1.1 仕様書に詳しく記述されています。
- `ias-ejb-jar.xml` - `ejb-jar.xml` ファイルのように、EJB モジュールごとに `ias-ejb-jar.xml` ファイルが 1 つだけあります。ライトウェイト CMP を使うには、このファイルでプロパティを設定する必要があります。DTD の概要については、第 11 章「配置のためのパッケージ化」を参照してください。
- *property-file-name*`.xml` - さらに、各 CMP Bean には、独自の配置記述子があります。ファイルの名前は `ias-ejb-jar.xml` ファイルで指定されています (`properties-file-location` 要素による。詳細については、第 11 章「配置のためのパッケージ化」を参照)。このファイルの内容で、リファレンス実装のパーシスタンスマネージャがどのようにしてリレーショナルデータベースに各 Bean ステートを読み込み、保存するかが決まります。

これらのファイルを生成するには 2 つの方法があります。次の節で各メソッドを詳しく説明します。

- 手動による配置記述子の作成
- 配置ツールの使用法

手動による配置記述子の作成

iPlanet Application Server 配置ツールの動作を理解するには、その背景で起きている内容を理解する必要があります。そのため、まず手動の手順を説明します。

ejb-jar 配置記述子

`ejb-jar.xml` ファイルについては EJB 1.1 仕様書に詳しく記述されています。ejb-jar 配置記述子は、Bean のトランザクション属性やコンテナ管理される Bean フィールドなど、重要な情報を指定します。対応する `ias-ejb-jar.xml` ファイルを指定する場合、J2EE 互換の ejb-jar ファイルは iPlanet Application Server 上に配置可能です。

ias-ejb-jar 配置記述子

Enterprise JavaBeans の J2EE ベンダー固有の情報は、別の配置記述子 `ias-ejb-jar.xml` に保存されます。この XML ベースの配置記述子の Document Type Definition (DTD) の詳細は、第 11 章「配置のためのパッケージ化」に記述されています。

`<persistence-manager>` 要素内に、このファイルに格納される CMP Bean に固有の情報があります。

- パーシスタンスマネージャを作成するファクトリクラスの完全修飾クラス名は、`<factory-class-name>` 要素で指定されます。リファレンス実装のファクトリクラス名は、`com.netscape.server.ejb.SQLPersistenceManagerFactory` です。
- `ejb-jar.xml` ファイル内の CMP Bean 固有のプロパティファイルの相対パスは、`<properties-file-location>` 要素で指定されます。

配置記述子の関連要素を示すコードの一部を次に示します。

```
...
   <persistence-manager>
       <factory-class-name>
          com.netscape.server.ejb.SQLPersistenceManagerFactory
       </factory-class-name>
       <properties-file-location>
          META-INF/MyProduct-ias-cmp.xml
       </properties-file-location>
   </persistence-manager>
...
```

CMP Bean 配置記述子

CMP Bean 固有の配置記述子のファイル名は、ias-ejb-jar.xml ファイルで指定されます。前述の例で、プロパティファイル名は、MyProduct-ias-cmp.xml でした。ファイルのルート要素は <ias-persistence-manager> ノードですが、その他はシンプルな Bean プロパティファイルです。ファイルは、1 つの XML フォーマットを使い、さまざまなプロパティを記述します。この配置記述子の DTD ファイルは次の場所にあります。

install_dir/ias/dtd/IASPersistence_manager_1_0.dtd

XML ファイルのタグは、この基本的なフォーマットに従います。

```
<bean-property>
   <property>
       <name></name>
       <type></type>
       <value></value>
       <delimiter></delimiter>
   </property>
</bean-property>
```

次に <property> のサブ要素の記述子を示します。

name	有効な名前 dataSource, allFields、findByPrimaryKeySQL、findByPrimaryKeyParms、insertSQL、insertParms、deleteSQL、deleteParms、loadSQL、loadParms、loadResults, storeSQL、storeParms の一つ、またはカスタムファインダーの名前です。 これらの各プロパティについてはこの節の後半で説明します。
type	java.lang.String と java.util.Vector のどちらか一方です。Vector がタイプとして使われる場合、値はカンマで区切られたリストとして扱われます。
value	任意の文字列です。
delimiter	常に , (カンマ) です。

次のプロパティは、ライトウェイト CMP Bean の配置記述子で定義されます。

- データソース (dataSource)
- CMP フィールドの RDB カラムへのマッピング (allFields)
- パーシスタンスオペレーション

- findByPrimaryKey (`findByPrimaryKeySQL` および `findByPrimaryKeyParms`)
- insert (`insertSQL` および `insertParms`)
- delete (`deleteSQL` および `deleteParms`)
- load (`loadSQL`、`loadParms`、および `loadResults`)
- store (`storeSQL` および `storeParms`)
- カスタムファインダー (オプション)

データソース

XML ファイルで使われる最初のプロパティは、`dataSource` です。`dataSource` プロパティの値は、パーシスタントストアとして使われる JDBC データソースの JNDI 名です。次のようにします。

```
...
   <bean-property>
       <property>
          <name>dataSource</name>
          <type>java.lang.String</type>
          <value>j2eeguide/ProductDB</value>
          <delimiter>,</delimiter>
       </property>
   </bean-property>
...
```

CMP フィールドの RDB カラムへのマッピング

`allFields` プロパティは、O/R マッピングが指定される場所です。`value` 要素では、かっこで囲まれた文字列が CMP フィールドをデータベースカラムにマップします。CMP フィールドを = の左辺に位置し、データベースカラムを式の右辺に位置します。セミコロン ; で式を区切ります。次のようにします。

```
...
   <bean-property>
       <property>
          <name>allFields</name>
          <type>java.lang.String</type>
          <value>
       {description=DESCRIPTION;price=PRICE;productId=PRODUCTID;}
          </value>
          <delimiter>,</delimiter>
       </property>
   </bean-property>
...
```

パーシスタンスオペレーション

パーシスタンスオペレーションは 3 つのプロパティタイプから構成されます。これらのプロパティは、次のネーミングパターンに従います。

- **xxxxSQL** は、特定のパーシスタンスオペレーション (例 : insert) の SQL ステートメントです。xxxxSQL プロパティ内の SQL ステートメントは、java.sql.PreparedStatement を作成するときに使われるため、パラメータ化されたクエリに指定されたルールに従う必要があります (たとえば、パラメータを示すために ? を使用するなど)。SQL データタイプに CMP フィールドをマップする方法を理解するには、176 ページの「マッピングルール」を参照してください。
- **xxxxParms** は、パーシスタンスオペレーションに送られるパラメータのリストです。最初のフィールドは SQL ステートメント (? で表される) の最初のパラメータにマップされ、2 番目のフィールドは 2 番目のパラメータにマップされます。
- **xxxxResults** は、PreparedStatement の実行で返される ResultSet 内のフィールドのリストです。

名前の xxxx は次のどれかです。

- findByPrimaryKey (findByPrimaryKeySQL および findByPrimaryKeyResults)
- insert (insertSQL および insertParms)
- delete (deleteSQL および deleteParms)
- load (loadSQL、loadParms、および loadResults)
- store (storeSQL および storeParms)
- カスタムファインダー名

パーシスタンスオペレーションプロパティは、CMP Bean にシングルフィールドプライマリキーまたはマルチフィールドプライマリキーがあるかどうかによって異なります。異なる点を次の例に示します。

findByPrimaryKey

findByPrimaryKey プロパティは、findByPrimaryKeySQL および findByPrimaryKeyParms です。findByPrimaryKeyResults プロパティはすでにプライマリキークラスで定義済みなので、findByPrimaryKey プロパティに指定する必要はありません。このオペレーションは、EJB のホームインタフェース内の findByPrimaryKey() メソッドに該当します。

次にシングルフィールドプライマリキーの例を示します。

```
...
   <bean-property>
       <property>
          <name>findByPrimaryKeySQL</name>
```

```
            <type>java.lang.String</type>
            <value>
               SELECT PRODUCTID FROM PRODUCT WHERE PRODUCTID = ?
            </value>
            <delimiter>,</delimiter>
        </property>
    </bean-property>
    <bean-property>
        <property>
            <name>findByPrimaryKeyParms</name>
            <type>java.util.Vector</type>
            <value>productId</value>
            <delimiter>,</delimiter>
        </property>
    </bean-property>
...
```

次にマルチフィールドプライマリキーの例を示します。

```
...
    <bean-property>
        <property>
            <name>findByPrimaryKeySQL</name>
            <type>java.lang.String</type>
            <value>
     SELECT PRODUCTID, DESCRIPTION FROM PRODUCT WHERE PRODUCTID = ?AND DESCRIPTION = ?
            </value>
            <delimiter>,</delimiter>
        </property>
    </bean-property>
    <bean-property>
        <property>
            <name>findByPrimaryKeyParms</name>
            <type>java.util.Vector</type>
            <value>productId,description</value>
            <delimiter>,</delimiter>
        </property>
    </bean-property>
...
```

insert

insert プロパティは、`insertSQL` および `insertParms` です。insert オペレーションは、シングルおよびマルチフィールドプライマリキーでまったく同じです。このプロパティは、Bean のホームインタフェース内の `create()` メソッドに該当します。

```
...
   <bean-property>
       <property>
          <name>insertSQL</name>
          <type>java.lang.String</type>
          <value>
INSERT INTO PRODUCT ( DESCRIPTION,PRICE,PRODUCTID ) VALUES(?,?,?)
          </value>
          <delimiter>,</delimiter>
       </property>
   </bean-property>
   <bean-property>
       <property>
          <name>insertParms</name>
          <type>java.util.Vector</type>
          <value>description,price,productId</value>
          <delimiter>,</delimiter>
       </property>
   </bean-property>
...
```

delete

delete プロパティは、`deleteSQL` および `deleteParms` です。delete オペレーションは、Bean のホームインタフェース内の `remove()` 関数を有効にします。

次にシングルフィールドプライマリキーの例を示します。

```
...
   <bean-property>
       <property>
          <name>deleteSQL</name>
          <type>java.lang.String</type>
          <value>DELETE FROM PRODUCT WHERE PRODUCTID = ?</value>
          <delimiter>,</delimiter>
       </property>
   </bean-property>
   <bean-property>
       <property>
          <name>deleteParms</name>
          <type>java.util.Vector</type>
          <value>productId</value>
          <delimiter>,</delimiter>
       </property>
   </bean-property>
...
```

次にマルチフィールドプライマリキーの例を示します。

```
...
   <bean-property>
       <property>
          <name>deleteSQL</name>
          <type>java.lang.String</type>
          <value>
       DELETE FROM PRODUCT WHERE PRODUCTID = ?AND DESCRIPTION = ?
          </value>
          <delimiter>,</delimiter>
       </property>
   </bean-property>
   <bean-property>
       <property>
          <name>deleteParms</name>
          <type>java.util.Vector</type>
          <value>productId,description</value>
          <delimiter>,</delimiter>
       </property>
   </bean-property>
...
```

load

load プロパティは、loadSQL、loadParms、および loadResults です。load オペレーションは、シングルおよびマルチフィールドプライマリキーでほぼ同じです。したがって、loadSQL プロパティと loadParms プロパティに大きな違いはありません。load オペレーションは、EJB の ejbLoad() メソッドに該当します。

次にシングルフィールドプライマリキーの例を示します。

```
...
   <bean-property>
       <property>
          <name>loadSQL</name>
          <type>java.lang.String</type>
          <value>
SELECT DESCRIPTION,PRICE,PRODUCTID FROM PRODUCT WHERE PRODUCTID = ?
          </value>
          <delimiter>,</delimiter>
       </property>
   </bean-property>
   <bean-property>
       <property>
          <name>loadParms</name>
          <type>java.lang.String</type>
          <value>productId</value>
          <delimiter>,</delimiter>
       </property>
```

```
    </bean-property>
    <bean-property>
        <property>
           <name>loadResults</name>
           <type>java.util.Vector</type>
           <value>description,price,productId</value>
           <delimiter>,</delimiter>
        </property>
    </bean-property>
...
```

次にマルチフィールドプライマリキーの例を示します。

```
...
    <bean-property>
        <property>
           <name>loadSQL</name>
           <type>java.lang.String</type>
           <value>
SELECT DESCRIPTION,PRICE,PRODUCTID FROM PRODUCT WHERE PRODUCTID = ?AND DESCRIPTION = ?
           </value>
           <delimiter>,</delimiter>
        </property>
    </bean-property>
    <bean-property>
        <property>
           <name>loadParms</name>
           <type>java.util.Vector</type>
           <value>productId,description</value>
           <delimiter>,</delimiter>
        </property>
    </bean-property>
    <bean-property>
        <property>
           <name>loadResults</name>
           <type>java.util.Vector</type>
           <value>description,price,productId</value>
           <delimiter>,</delimiter>
        </property>
    </bean-property>
...
```

store

store プロパティは、storeSQL および storeParms です。プロパティを読み込む場合、storeSQL と storeParms プロパティは多少異なります。マルチフィールドプライマリキー内が正しい順序になるように確認してください。Bean 実装で、EJB コンテナが ejbStore() メソッドを呼び出すと、store オペレーションが実行されます。

次にシングルフィールドプライマリキーの例を示します。

```
...
   <bean-property>
       <property>
          <name>storeSQL</name>
          <type>java.lang.String</type>
          <value>
       UPDATE PRODUCT SET DESCRIPTION=?,PRICE=?WHERE PRODUCTID = ?
          </value>
          <delimiter>,</delimiter>
       </property>
   </bean-property>
   <bean-property>
       <property>
          <name>storeParms</name>
          <type>java.util.Vector</type>
          <value>description,price,productId</value>
          <delimiter>,</delimiter>
       </property>
   </bean-property>
...
```

次にマルチフィールドプライマリキーの例を示します。

```
...
   <bean-property>
       <property>
          <name>storeSQL</name>
          <type>java.lang.String</type>
          <value>
    UPDATE PRODUCT SET PRICE=?WHERE PRODUCTID = ?AND DESCRIPTION = ?
          </value>
          <delimiter>,</delimiter>
       </property>
   </bean-property>
   <bean-property>
       <property>
          <name>storeParms</name>
          <type>java.util.Vector</type>
          <value>price,productId,description</value>
          <delimiter>,</delimiter>
       </property>
   </bean-property>
...
```

カスタムファインダー

オプションで、カスタムファインダーを配置記述子に追加できます。カスタムファインダーオペレーションのルールは、ほかのオペレーションと少し異なります。

- カスタムファインダーの `xxxxSQL` プロパティでは、ホームインタフェースで定義されているファインダーメソッドへの最初の引数が、SQL ステートメント内の最初のパラメータにマップされ、2 番目の引数は 2 番目のパラメータにマップされます。
- カスタムファインダーの `xxxxResults` プロパティは、SQL ステートメントの `ResultSet` のカラムをプライマリキーのフィールド (マルチフィールドプライマリキーのフィールド) またはプライマリキー自体 (シングルフィールドプライマリキーのフィールド) にマップします。

たとえば、次に示すメソッドがエンティティ Beans のホームインタフェースで定義されると仮定します。

```
public Collection findInRange(double low, double high)
    throws FinderException, RemoteException;
```

プロパティ名は Bean のホームインタフェースに存在する名前です。この例のオペレーションでは、配置記述子内に 3 つのプロパティ `findInRangeSQL`, `findInRangeParms` および `findInRangeResults` (マルチフィールドプライマリキーだけに必要) を持つ可能性があります。

次に、このオペレーションを実装するシングルフィールドプライマリキーのプロパティを示します。

```
...
   <bean-property>
       <property>
          <name>findInRangeSQL</name>
          <type>java.lang.String</type>
          <value>
          SELECT PRODUCTID FROM PRODUCT WHERE PRICE BETWEEN ?AND ?
          </value>
          <delimiter>,</delimiter>
       </property>
   </bean-property>
   <bean-property>
       <property>
          <name>findInRangeParms</name>
          <type>java.lang.Vector</type>
          <value>low,high</value>
          <delimiter>,</delimiter>
       </property>
   </bean-property>
...
```

次に、このオペレーションを実装するマルチフィールドプライマリキーのプロパティを示します。

```
...
   <bean-property>
       <property>
          <name>findInRangeSQL</name>
          <type>java.lang.String</type>
          <value>
      SELECT PRODUCTID, DESCRIPTION FROM PRODUCT WHERE PRICE BETWEEN ?AND ?
          </value>
          <delimiter>,</delimiter>
       </property>
   </bean-property>
   <bean-property>
       <property>
          <name>findInRangeParms</name>
          <type>java.lang.Vector</type>
          <value>low,high</value>
          <delimiter>,</delimiter>
       </property>
   </bean-property>
   <bean-property>
       <property>
          <name>findInRangeResults</name>
          <type>java.util.Collection</type>
          <value>productid,description</value>
          <delimiter>,</delimiter>
       </property>
   </bean-property>
...
```

マッピングルール

ライトウェイト CMP は JDBC (特に `PreparedStatement` インタフェースのセッターメソッド) を使い、リレーショナルデータベーステーブルのカラムに CMP フィールドをマップします。そのため、標準 JDBC マッピングルールが CMP フィールドに適用されます。

たとえば、SQL カラムに `java.lang.String` をマップするには、ライトウェイト CMP が `PreparedStatement` インタフェース内の `setString` メソッドを使います。`PreparedStatement` インタフェースのドキュメントでは、`setString` が `VARCHAR` にマップすることを指定しています。

ライトウェイト CMP は、すべての固有の Java フィールドタイプ、固有のタイプを表すすべてのクラス (例 : Integer)、java.lang.String、java.sql.Date、java.sql.Time、java.sql.Timestamp、および任意の直列化可能なオブジェクトをサポートしています。表 6-1 では、Bean 属性とテーブルカラム間のマッピングを説明しています。

表 6-1 EJB/JDBC マッピング

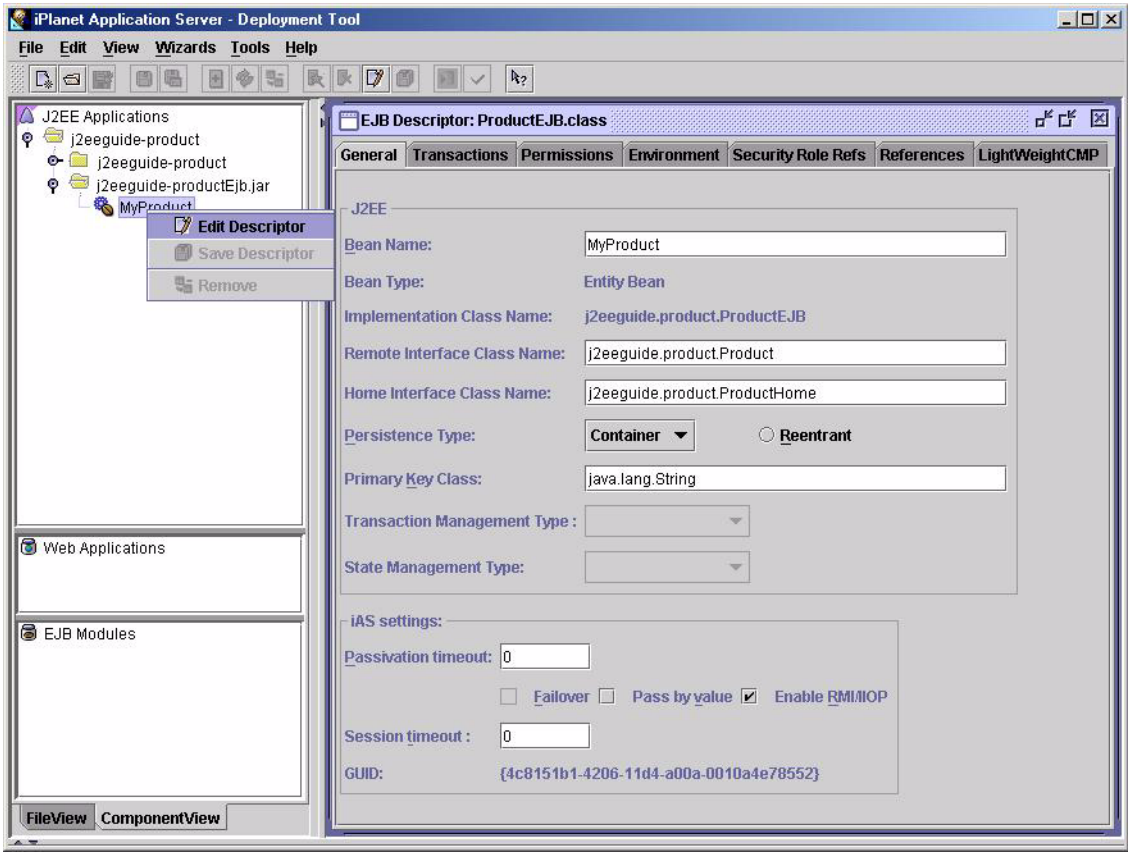
Java のタイプ	JDBC のタイプ	JDBC ドライバアクセスメソッド
boolean	BIT	getBoolean()、setBoolean()
byte	TINYINT	getByte()、setByte()
short	SMALLINT	getShort()、setShort()
int	INTEGER	getInt()、setInt()
long	BIGINT	getLong()、setLong()
float	FLOAT	getFloat()、setFloat()
double	DOUBLE	getDouble()、setDouble()
byte[]	VARBINARY または LONGVARBINARY(1)	getBytes()、setBytes()
java.lang.String	VARCHAR または LONGVARCHAR(1)	getString()、setString()
java.lang.Boolean	BIT	getObject()、setObject()
java.lang.Integer	INTEGER	getObject()、setObject()
java.lang.Long	BIGINT	getObject()、setObject()
java.lang.Float	REAL	getObject()、setObject()
java.lang.Double	DOUBLE	getObject()、setObject()
java.math.BigDecimal	NUMERIC	getObject()、setObject()
java.sql.Date	DATE	getDate()、setDate()
java.sql.Time	TIME	getTime()、setTime()
java.sql.Timestamp	TIMESTAMP	getTimestamp()、setTimestamp()
任意の直列化可能なクラス	VARBINARY または LONGVARBINARY(1)	getBytes()、setBytes()

配置ツールの使用法

CMP Bean の標準 ejb-jar 配置記述子を作成する簡単な方法は、iPlanet Application Server 配置ツールを使うことです。このツールの広範囲な組み込みを利用すると、配置記述子の作成方法についての詳細がわかります。

既存の EJB モジュールを開くか、または新しい EJB モジュールを作成することによって開始します。このツールを使った CMP Bean の作成方法の詳細については、ツール内のヘルプを参照してください。EJB クラスファイルに EJB モジュールが追加されていれば、図 6-1 のように、Bean 上で右クリックし、その記述子を編集できます。

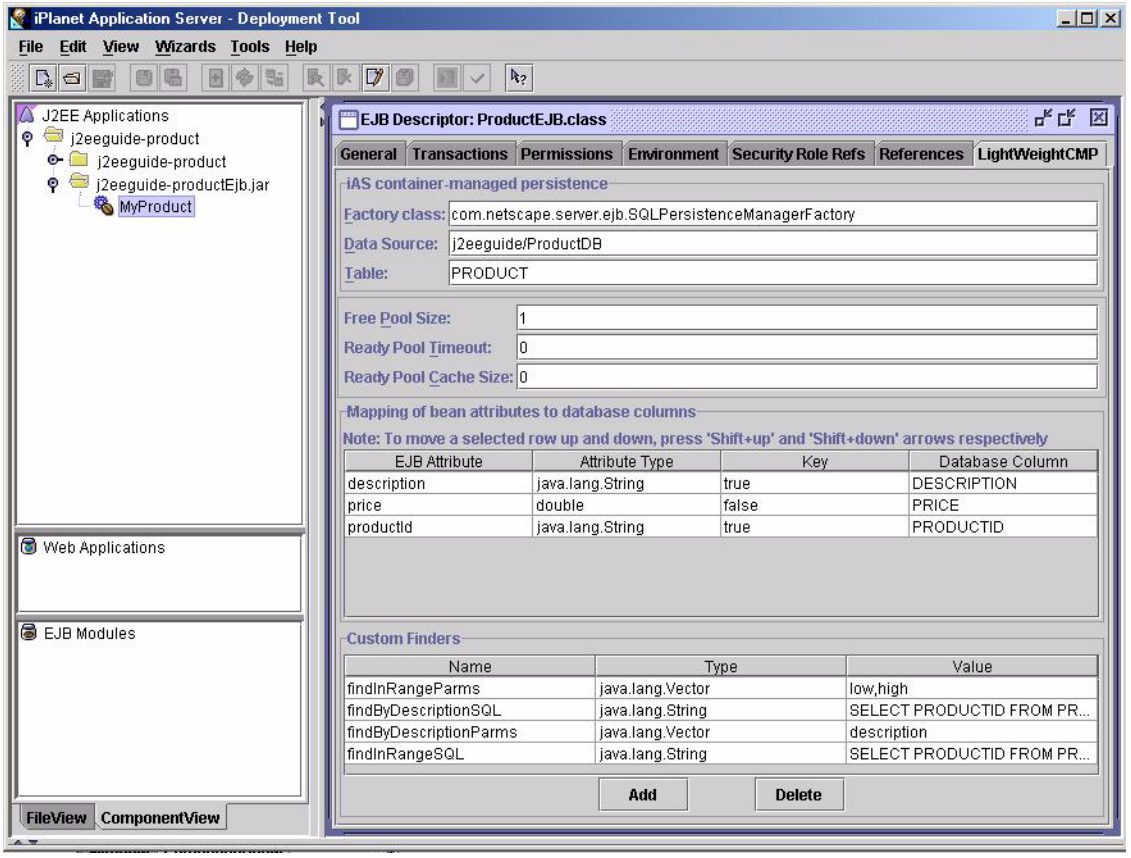
図 6-1　iPlanet Application Server 配置ツールで Bean を選択



配置記述子を開くと、ユーザインタフェースでの変更は、`ejb-jar.xml` 配置記述子の CMP EJB セクション、`ias-ejb-jar.xml` 配置記述子の CMP EJB セクション、および CMP Bean 固有の配置記述子に反映されます。CMP Bean のマッピング情報は、*ejbname*`-ias-cmp.xml` という名前のファイルに保存されます。ユーザインタフェースの要素の詳細については、ツールのヘルプを参照してください。

図 6-2 では「ライトウェイト CMP」タブを示します。

図 6-2　「ライトウェイト CMP」タブ



前述の節で手動による配置記述子の作成について理解すると、「ライトウェイト CMP」タブの内容がわかりやすくなります。例外を次に説明します。

- 「TABLE」テキストボックスは、指定のデータソースを介してアクセスするリレーショナルデータベーステーブルの入力フィールドです。

- 各 EJB 属性にキーフィールドを切り替えることができます。マルチフィールドプライマリキーを作成するには、2 つ以上の属性を `true` に設定するだけです。変更は Bean の対応する配置記述子に反映されます (マルチフィールドプライマリキーは、EJB 1.1 仕様書に定義されているプライマリキークラスの内容などのほかの変更も必要とします)。
- カスタムファインダーには、「Name」、「Type」、および「Value」フィールドを使います。これらのフィールドの使用法は前述の節で説明した方法と同じです。

注 iPlanet Application Server 配置ツール内で EJB の 配置記述子の作成を開始し、ツールにアプリケーションを保存し、手動でファイルを編集してツールに戻ることができます。ただし、この作業を行う場合、配置記述子を編集する前に、必ずツール内の EJB モジュールあるいは J2EE アプリケーションを再び開き、変更後にツールにアプリケーションを保存してください。失敗した場合は、ユーザインタフェースでの変更は配置記述子に反映されません。

第 7 章

メッセージ駆動 Beans の使用

この章では、メッセージ駆動 Beans とそのプロパティについて説明します。また、メッセージ駆動 Beans を作成して iPlanet Application Server に配置するための追加のガイドラインも用意してあります。

この章には次の節があります。

- メッセージ駆動 Beans のコンポーネント
- メッセージ駆動 Beans のガイドライン
- iPlanet Application Server 機能へのアクセス
- 配置ツールの使用法
- 手動による配置記述子の作成

注 iPlanet Application Server 内でのメッセージ駆動 Beans の機能は、開発者だけが使用します。運用環境では、テストも認定もされていません。

メッセージ駆動 Beans の概要

iPlanet Application Server は、JMS 仕様を実装するメッセージングミドルウェアのアプリケーションとして、iPlanet Message Queue for Java, 2.0 SP1 を使います。iPlanet Application Server でメッセージ駆動 Beans を使う前に、iMQ for Java, 2.0 SP1 をインストールしておく必要があります。

iMQ for Java, 2.0 SP1 は、iPlanet Application Server インストール CD に含まれています。このインストール CD がない場合は、`http://www.iplanet.com/products/iplanet_message_queue/home_message_queue.html` から無料の開発者用エディションをダウンロードできます。

メッセージ駆動 Beans にアクセスする方法

メッセージ駆動 Beans は、Java Message Service (JMS) インタフェースを使用します。つまり、メッセージ駆動 Beans は JMS リスナです。JMS ミドルウェアとメッセージ駆動 Beans のコンテナが一緒になって、JMS `MessageListener` オブジェクトへのメッセージの配信を制御します。

注　メッセージ駆動 Beans を使うには、まず JMS プロバイダを設定し、メッセージキューオブジェクトを設定する必要があります。JMS プロバイダを設定する方法については、『iPlanet Application Server 管理者ガイド』の 11 章「メッセージ駆動 Bean の管理」を参照してください。

次のステップで、クライアントリクエストに基づくアクションを説明します。

1. アプリケーションサーバが起動すると、配置されたすべてのメッセージ駆動 Beans が読み込まれ、メッセージリスナが起動します。

   アプリケーションサーバは、iPlanet Application Server の `ServerSessionPool` を使用して、デスティネーション固有の代理人を JMS で登録します。

2. ブラウザ、Servlet、スタンドアロンアプリケーションなどのクライアントは、JMS デスティネーションへメッセージを送信します。

3. JMS は、指定されたデスティネーションにリクエストを処理させるために、アプリケーションサーバのコールバックを呼び出します。

4. JMS セッションの `MessageListener` が、メッセージ駆動 Beans のインスタンスのコンテナになります。

   メッセージが着信すると、コンテナは、そのメッセージを処理するためにメッセージ駆動 Beans の `onMessage` メソッドを呼び出します。`onMessage` メソッドは通常、メッセージを 5 つの JMS メッセージタイプのうちの 1 つにキャストし、アプリケーションのビジネスロジックに従って処理します。`onMessage` メソッドは、ヘルパーメソッドを呼び出すことができ、また、セッションまたはエンティティ Beans を呼び出して、メッセージ内の情報を処理したりデータベースに格納したりすることもできます。

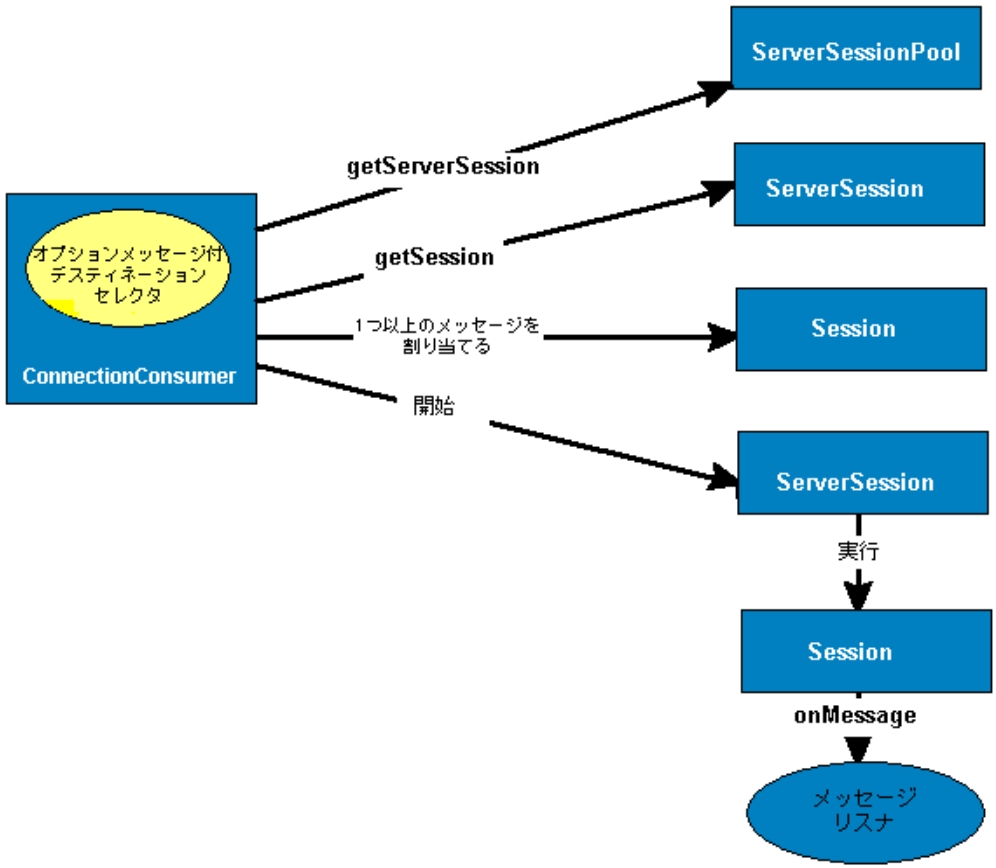
   メッセージをトランザクションコンテキスト内でメッセージ駆動 Beans に配信することもできるので、`onMessage` メソッド内のすべてのオペレーションは、1 つのトランザクションの一部です。メッセージ処理がロールバックされると、メッセージは再度送信されます。

5. iPlanet Application Server では、トランザクション、セキュリティなどのサポート対象のサービスが処理され、リクエストの処理は Bean のビジネスメソッドに委任されます。

6. JMS はメッセージをキューから削除し、メッセージ内に応答を求めるプロパティがあった場合は、クライアントに ACK を送信します。
7. 失敗した場合は、JMS が再度メッセージを送信します。

次の図は、配置された Bean のリクエストフローパスです。

図 7-1　リクエストフローパス



ConnectionConsumer と Session は、JMS プロバイダの一部です。ServerSessionPool、ServerSession およびメッセージリスナは、iPlanet Application Server の一部です。

メッセージ駆動 Beans のコンポーネント

メッセージ駆動 Beans を作成するには、次のクラスファイルを準備する必要があります。

- Enterprise JavaBeans クラス定義
- Enterprise JavaBeans メタデータ (配置記述子 (DD) およびほかの設定情報)

クラス定義の作成

メッセージ駆動 Beans の場合、Bean のクラスは、`abstract` ではなく `public` として定義する必要があります。Bean クラスは、`javax.ejb.MessageDrivenBean` インタフェースを実装する必要があります。次のようにします。

```
import javax.jms.*;
import javax.ejb.*;
public class MySessionBean implements MessageDrivenBean,
MessageListener {
// メッセージ駆動 Beans の実装。これらのメソッドは常に取り込む必要があります。
public void ejbRemove() throws RemoteException{
}
public void setMessageDrivenContext(MessageDrivenContext ctx) throws
RemoteException {
}

// 他のコードは省略します。
}
```

メッセージ駆動 Beans は、1 つまたは複数の `ejbCreate(...)` メソッドを実装する必要もあります。クライアントが Bean を呼び出すときは、必ずこのメソッドを 1 つ使います。次のようにします。

```
public void ejbCreate() {
}
```

各 `ejbCreate(...)` メソッドは `public` として宣言し、`void` を返し、`ejbCreate` という名前を付ける必要があります。throws 句には、`java.rmi.RemoteException` または `java.ejb.CreateException` を含めることができます。

メッセージ駆動 Beans のガイドライン

メッセージ駆動 Beans として表すことができるアプリケーションの各部を決める前に、メッセージ駆動 Beans について理解すべき事項がいくつかあります。これらには、メッセージ駆動 Beans の EJB 仕様に関連する事項と、iPlanet Application Server および iPlanet Application Server によるメッセージ駆動 Beans のサポートに固有の事項があります。

iPlanet Application Server 機能へのアクセス

この節には次のトピックがあります。

- トランザクションの管理
- トランザクションのコミット
- データベースへのアクセス

開発可能なメッセージ駆動 Beans には、EJB 仕様に厳密に準拠しているメッセージ駆動 Beans、その仕様と iPlanet Application Server の付加価値機能の両方を活用しているメッセージ駆動 Beans、および iPlanet Application Server 以外の環境における仕様に準拠してはいるが利用できる iPlanet Application Server 機能も活用するメッセージ駆動 Beans があります。希望の配置シナリオに最適な Bean を選択します。

iPlanet Application Server は、iPlanet Application Server コンテナを介していくつかの機能を提供します。また、iPlanet Application Server API によって、アプリケーションは特定の iPlanet Application Server 環境の機能をプログラムで利用できるようになります。iPlanet Application Server 環境だけでメッセージ駆動 Beans を使う予定であれば、これらの Bean に API 呼び出しを組み込みます。

たとえば、次の手順および例に従って `IAppEventMgr` インタフェースを使い、指定したアプリケーションイベントを EJB からトリガできます。

1. まず、`javax.ejb.SessionContext` または `javax.ejb.EntityContext` を `IServerContext` にタイプ変換して、`com.kivasoft.IContext` インスタンスを取得します。
2. 次に、`GXContext` クラスの `GetAppEventMgr()` メソッドを使って、`IAppEventMgr` オブジェクトを作成します。
3. さらに、`triggerEvent()` を使ってアプリケーションイベントをトリガします。

```
javax.ejb.SessionContext m_ctx;
....
 com.netscape.server.IServerContext sc;
 sc = (com.netscape.server.IServerContext) m_ctx;
 com.kivasoft.IContext kivaContext = sc.getContext();
 IAppEventMgr mgr = com.kivasoft.dlm.GXContext.GetAppEventMgr(ic);
 mgr.triggerEvent("eventName");
```

トランザクションの管理

メッセージ駆動 Beans はデータベースと対話します。Bean のプロパティファイルの設定値を使って、Bean のトランザクションを制御します。これにより、Bean 配置時にトランザクション属性を指定できます。Bean 操作のトランザクション管理があるので、Bean のデータベースアクセスメソッドのトランザクションを明示的に「起動」、「ロールバック」、または「コミット」する必要はありません。

トランザクション管理をコンテナレベルに移すことによって、Bean のアクティビティがデータベースアクセスと直接結び付いていなくても、データベース呼び出しと同じトランザクション制御環境ですべての Bean アクティビティを配置できます。これにより、メッセージ駆動 Beans によって制御されるアプリケーションのすべての部分が、同じトランザクションの一部として動作しますが、失敗すると、Bean が管理していたものはすべてコミットされるか、ロールバックされます。実際には、コンテナ管理トランザクションのステートによって、同期ルーチンをプログラミングせずにアプリケーションの同期をとることができます。

トランザクションのコミット

コミットが発生し、メッセージ駆動 Beans の有効な作業が終了したことがコンテナに伝わると、コンテナは、基礎となっているデータソースとステートの同期をとるように指示されます。コンテナはトランザクションの終了を許可し、Bean を解放します。

コンテナからのトランザクションは暗黙的にコミットされていることに注意してください。また、トランザクションに関連した Bean であれば、トランザクションをロールバックできます。トランザクションの詳細については、第 8 章「EJB のトランザクション処理」を参照してください。

データベースへのアクセス

多くのメッセージ駆動 Beans はデータにアクセスしてデータを更新します。メッセージ駆動 Beans は一時的なので、アクセスがどのように発生するかに注意してください。一般に、JDBC API を使って呼び出し、第 8 章「EJB のトランザクション処理」に記述されているトランザクションおよびセキュリティ管理メソッドを使って、トランザクション分離レベルおよびトランザクション要件を Bean レベルで管理します。

データベースへのアクセスについては、第 9 章「JDBC を使ったデータベースアクセス」を参照してください。

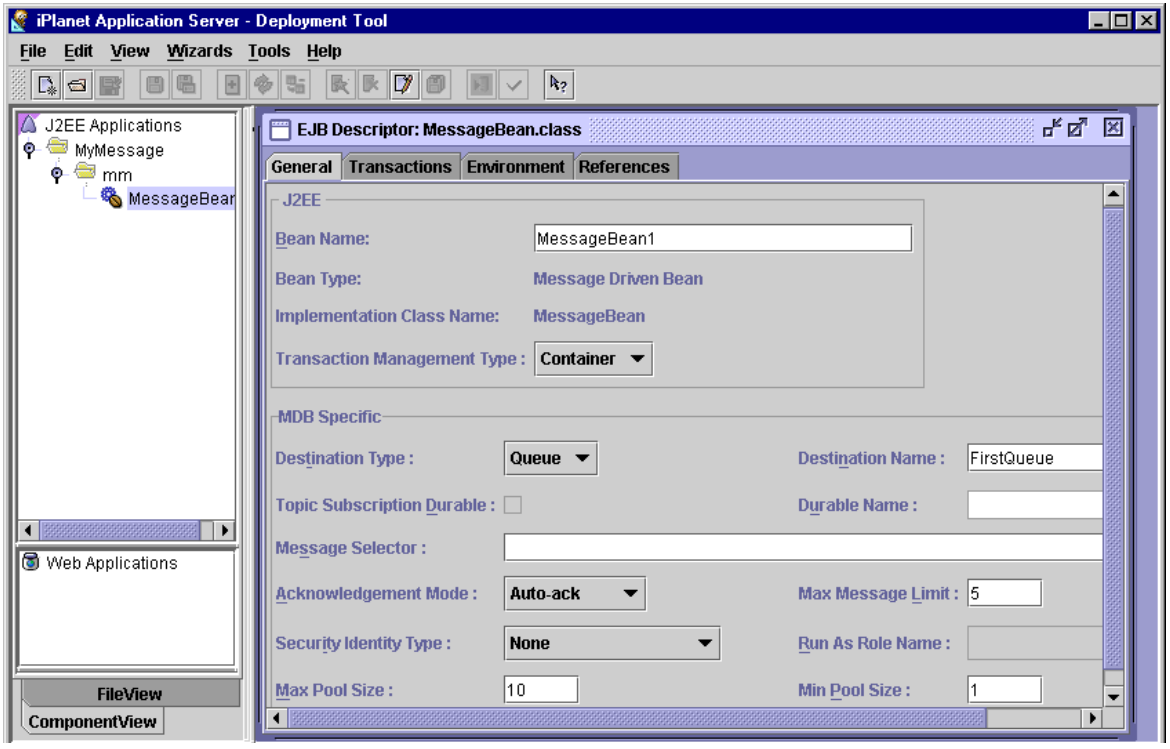
配置ツールの使用法

メッセージ駆動 Beans の標準 ejb-jar 配置記述子は、iPlanet Application Server の配置ツールを使って簡単に作成できます。

メッセージ駆動 Beans の配置は、配置ツールを使用したほかのアプリケーションの配置と似ています。

既存の EJB モジュールを開くか、または新しい EJB モジュールを作成することによって開始します。EJB クラスファイルに EJB モジュールが追加されていれば、次の図のように、Bean 上で右クリックし、その記述子を編集できます。

図 7-2　配置ツール内でのメッセージ駆動 Beans の選択



注　iPlanet Application Server 配置ツール内で EJB の 配置記述子の作成を開始し、ツールにアプリケーションを保存し、手動でファイルを編集してツールに戻ることができます。

ただし、この作業を行う場合、配置記述子を編集する前に、必ずツール内の EJB モジュールあるいは J2EE アプリケーションを再び開き、変更後にツールにアプリケーションを保存してください。失敗した場合は、ユーザインタフェースでの変更は配置記述子に反映されません。

配置ツールの配置記述子ダイアログボックスに次の情報を指定する必要があります。

J2EE 特有の配置記述子のフィールド

- Bean 名
- Bean タイプ (メッセージ駆動 Beans)

- 実装クラス名
- トランザクション管理タイプ (コンテナ管理または Bean 管理)

メッセージ駆動 Beans 特有のパラメータ

- デスティネーションタイプ (キューまたはトピック)
- デスティネーション名
- 永続名 (トピックサブスクリプション専用 -- オプション)
- 永続的トピックサブスクリプション (オプション)
- メッセージセレクタ (オプション)
- 通知モード (自動通知または重複自動通知)
- 最大メッセージ制限
- セキュリティ ID のタイプ (指定されたユーザとして実行が許可された場合のみ)
- ロール名として実行

注 ユーザロールがメッセージ駆動 Beans にアクセスされるデータに対して管理権限を持っている場合は、セキュリティ上のリスクがあります。メッセージ駆動 Beans を認証するユーザは、指定されたセキュリティロールの権限を受け継ぎます。

- 最大プールサイズ
- 最小プールサイズ
- トランザクションマネージャタイプ (ローカルまたはグローバル -- 指定しないと、このモジュールの `Transaction Manager Type` が使用される)

手動による配置記述子の作成

配置記述子ファイルの例

```
<ias-mdbs>
<!-- これは EJB 2.0 DTD ベースの配置記述子から抽出 --<&/tt>
 <ejb-jar>
  <enterprise-beans>
    <message-driven>
      <ejb-name>MyMDB1</ejb-name>
      <ejb-class>mycompany.mypackage.MyMDB1</ejb-class>
      <transaction-type>Container</transaction-type>
      <message-driven-destination>
       <jms-destination-type>javax.jms.Topic</jms-destination-type>
      </message-driven-destination>
       <security-identity>
        <run-as-specified-identity>
          <role-name>asmith</role-name>
        </run-as-specified-identity>
       </security-identity>
    </message-driven>
  </enterprise-beans>
 </ejb-jar>
<!-- これには ias 特有のすべての配置情報が含まれる --<&/tt>
 <ias-ejb-jar>
  <ias-enterprise-bean>
    <ejb-name>MyMDB1</ejb-name>
    <message-driven-descriptor>
      <jms-destination>
         <jndi-name>MyMDB1</jndi-name>
```

```
        <jms-topic-subscription>
            <durable>true</durable>
        </jms-topic-subscription>
      </jms-destination>
      <pool>
        <max-pool-size>100</max-pool-size>
        <min-pool-size>10</min-pool-size>
      </pool>
    </message-driven-descriptor>
  </ias-enterprise-bean>
 </ias-ejb-jar>
```

詳細については、*iASInstallDir*/ias/dtd 内の EJB JAR ファイル用 XML DTD (IASEjb_jar_1_1.dtd) を参照してください。

第 8 章

EJB のトランザクション処理

この章では、EJB プログラミングモデルに組み込まれているトランザクションサポートについて説明します。この章には次の節があります。

- トランザクションモデルを理解する
- EJB のトランザクション属性の指定
- Bean 管理トランザクションの使用法

トランザクションモデルを理解する

EJB の主な利点の 1 つは、宣言型トランザクションに提供するサポートです。宣言型トランザクションモデルでは、配置時に属性と Bean が関連付けられます。トランザクションコンテキストの境界を定めて透過的に伝播させることは、属性値をベースにしたコンテナの役割です。また、コンテナには、トランザクションマネージャと連係して、トランザクション内のすべての関係要素に一貫した結論を参照させる役割もあります。

宣言型トランザクションを使うと、プログラマはトランザクションの境界を定める作業をする必要がなくなります。宣言型トランザクションによって、潜在的に分散型で、さまざまなリソースを更新する複数のコンポーネントが、1 つのトランザクションを共有できるコンポーネントベースのアプリケーションを簡単に作成できるようになります。また、EJB 仕様書では、`javax.transactions.UserTransaction` を使ったプログラマによるトランザクションの境界設定もサポートしています。`UserTransaction` オブジェクトを取得するには、JNDI 検索を実行する必要があります。

iPlanet Application Server のトランザクション用サポートを理解するには、グローバルトランザクションとローカルトランザクションの違いを理解しておく必要があります。グローバルトランザクションは、トランザクションマネージャによって管理および調整され、複数のデータベースやプロセスにまたがることができます。トランザクションマネージャは通常、`XA` プロトコルを使って Enterprise Information System (EIS)

またはデータベースと対話します。ローカルトランザクションは、単一 EIS またはデータベースのみに固有であり、1 つのプロセス内に制限されます。ローカルトランザクションとグローバルトランザクションの両方とも、`javax.transaction.UserTransaction` インタフェースを使って境界が設定されます。クライアントはこのインタフェースを使う必要がありますが、内部的にローカルトランザクションは JDBC API で実装され、これによってグローバルトランザクションより高速で処理されます。

iPlanet Application Server はグローバルトランザクションモードまたはローカルトランザクションモードのどちらかで動作しますが、両方を混在させることはできません。

注 アプリケーションがグローバルトランザクションを使うには、対応する iPlanet Application Server Resource Manager を設定し、使用可能にします。詳細については、配置ツールのオンラインヘルプおよび『管理ガイド』を参照してください。

EJB 仕様書では、ネストとは対照的な単層型トランザクションのサポートを要求しています。このモデルでは、各トランザクションはシステムのほかのトランザクションから独立しており、依存していません。単層型トランザクション内では、現在のトランザクションが終了するまで同じスレッド内で別のトランザクションを開始できません。単層型トランザクションは、非常に普及したモデルであり、ほとんどの商用データベースでサポートされています。ネストされたトランザクションに対しては、より精細な制御が行われます。

注 サードパーティのドライバは、トランザクション内で使用できません。

EJB のトランザクション属性の指定

Bean のリモートインタフェースのトランザクション属性は、Bean 全体またはメソッドごとに指定します。属性を両方のレベルで指定すると、Bean 全体の値よりもメソッド固有の値が優先されます。制限に関する節で説明するように、無効な組み合わせがあるので、両レベルの属性を組み合わせるときは注意が必要です。

トランザクションの属性は Bean の XML DD ファイルの一部として指定されます。詳細については、327 ページの「EJB iPlanet Application Server XML DTD」を参照してください。

Bean 管理トランザクションの使用法

コンテナ管理のトランザクションの使用をお勧めしますが、アプリケーションの要件によっては、Bean 管理トランザクションの使用が必要になる場合があります。プログラムでのトランザクションの管理の詳細については、次の URL で、このインタフェースに関する Enterprise JavaBeans 仕様書バージョン 1.1 を参照してください。

```
http://java.sun.com/products/ejb/javadoc-1.1/javax/ejb/EJBContext.html
```

Bean 管理トランザクションへのポインタを用意することができます。たとえば、状態のあるセッション Beans のためにトランザクションを開始する場合、Bean は非活性化されません (フェールオーバーが影響を受けるので、トランザクション前のステートを調べます)。ただし、状態のないセッション Beans のためにトランザクションを開始する場合は、メソッドが復帰するとトランザクションは一度ロールバックされます。

第 9 章

JDBC を使ったデータベースアクセス

この章では、Java Database Connectivity (JDBC) API を使って iPlanet Application Server でデータベースをアクセスする方法について説明します。この章では、iPlanet Application Server を使った Servlet および EJB への高度な JDBC の実装について説明します。また、iPlanet Application Server の特定のリソースが明らかにプログラミング分岐を持つ場合に JDBC ステートメントの影響を受けるそれらのリソースについて説明します。

iPlanet Application Server では、EJB は基本的に JDBC API を介したデータベースアクセスをサポートします。iPlanet Application Server は、リザルトセットの拡張、バッチ更新、分散トランザクション、行セット、データソース名の検索用の Java Naming and Directory Interface (JNDI) サポートなどの、さまざまな JDBC 2.0 拡張だけでなく、JDBC 2.0 API 全体をサポートします。

注　コンテナ、ローカル、またはグローバルのトランザクション管理は、固有のドライバでサポートされなくなりました。

iPlanet Application Server 6.0 SP1 では固有の JDBC ドライバのサポートが廃止されていましたが、このリリースでは下位互換性を維持するためにサポートされています。

この章では、JDBC 2.0 を十分に理解していることを前提とし、プログラミング分岐を持つ可能性がある特定の実装に関する問題についても説明します。たとえば、JDBC 仕様では、何が JDBC リソースを構成するかについては明確にされていません。この仕様では、データベースコネクションを閉じる `Connection` クラスメソッドなどの JDBC ステートメントには、それらのリソースが何であるかを正確に指定せずにリソースを解放するものもあります。

この章には次の節があります。

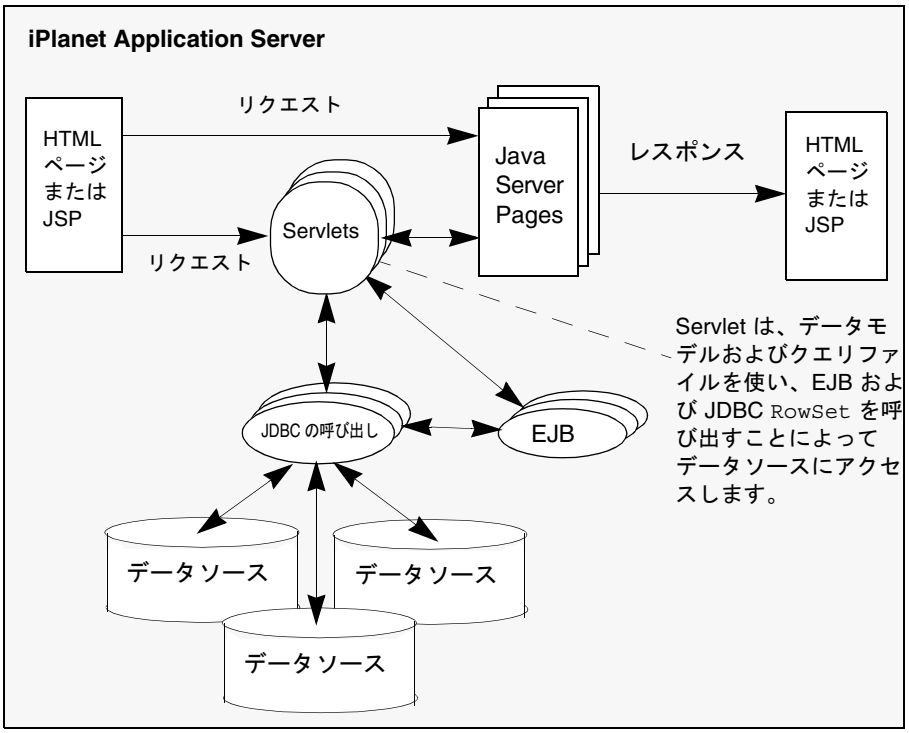
- JDBC の紹介
- 6.x DD XML ファイルの 6.5 への移行

- 新規の XML データソース記述子
- サーバアプリケーションでの JDBC の使用法
- コネクションの処理
- JDBC 機能の操作

JDBC の紹介

プログラミングの観点からすると、JDBC はサーバアプリケーションにデータベース呼び出しを埋め込むことができる Java クラスおよびメソッドのセットです。サーバアプリケーションで JDBC を使い始めるには、このことだけを知っていれば十分です。

より厳密に言うと、JDBC は一連のインタフェースであり、iPlanet などのすべてのサーバベンダーは、JDBC 仕様に従ってこれを実装する必要があります。iPlanet Application Server には、さまざまな Enterprise Information System (EIS) データベースをサポートする JDBC タイプ 2 ドライバがあります。このドライバは、アプリケーションで JDBC ステートメントを処理し、その JDBC ステートメントに含まれている SQL 引数をデータベースエンジンに渡します。



JDBC を使うと、低レベルのデータベース実装を十分に理解していなくても、さまざまなデータベースでシームレスに操作できる、高度で使いやすいプログラムを記述することができます。

サポートされている機能

注 この節では、サポートに制限のある、固有の JDBC ドライバの機能について説明します。サードパーティの JDBC ドライバの機能については、そのドライバベンダーのドキュメントを参照してください。

JDBC 仕様書は、データベースベンダーに依存しない幅広い一連のガイドラインです。ガイドラインには、簡単なフレームワークで可能な広範のデータベース機能が含まれています。JDBC では、データベースが少なくとも SQL-2 データベースアクセス言語をサポートしていることを前提としています。JDBC 仕様書は 3 つの部分に分かれています。

- JDBC 2.0 では、JDBC との互換性を維持するためにサーバベンダーが実装する必要のある**コアデータベースアクセスおよび機能**について説明しています。iPlanet Application Server はこの基準を完全に満たしています。データベースベンダーの観点から、JDBC 2.0 では、標準 SQL-2 言語、各ベンダーがサポートする標準言語部分、および各ベンダーが実装する言語の拡張へのフルアクセスを許可するデータベースアクセスモデルについて説明しています。
- JDBC 2.0 では、**追加のデータベースアクセスおよび機能**について説明しています。この機能には基本的に、新たに定義された SQL-3 機能、データタイプ、およびマッピングのサポートが含まれます。iPlanet Application Server への JDBC の実装によって、ほとんどの JDBC 機能拡張がサポートされますが、*BLOB*、*CLOB*、アレイなどの新しい SQL-3 データタイプについてはサポートされません。現在、リレーショナルデータベース管理システムでこれらを完全にサポートしているデータベースベンダーは多くありません。iPlanet Application Server 版の JDBC でも、SQL-3 データタイプマッピングはサポートされません。
- JDBC 2.0 Standard Extension API では、高度なサポート機能について説明しています。これらの機能によってデータベースのパフォーマンスが向上します。現在は、iPlanet Application Server への JDBC の実装によって、Java Naming and Directory Interface (JNDI) および行セットがサポートされます。

データベースの制約事項の理解

サーバアプリケーションで JDBC を使ったときに、望んだ結果や予期した結果が得られない場合があります。この場合、JDBC、または iPlanet Application Server への JDBC ドライバの実装に問題があると考えがちですが、この種の問題の原因の多くはデータベースエンジンの制約事項にあります。

JDBC は、可能なかぎり広範囲なデータベースサポートを扱うので、すべてのデータベースがサポートするとは限らないオペレーションを試みることもできます。たとえば、ほとんどのデータベースベンダーは SQL-2 言語の大部分をサポートしますが、どのベンダーも SQL-2 標準規格を制限付きでサポートします。ほとんどのベンダーは、既存の専用リレーショナルデータベース管理システムのトップに SQL-2 サポートを組み込みます。そして、専用のシステムが SQL-2 にない機能を提供するか、または SQL-2 が専用のシステムで利用できない機能を提供します。ほとんどのベンダーは、SQL の実装に非標準 SQL-2 拡張機能を追加することによって、ベンダー専用の機能をサポートしています。JDBC はベンダー固有の機能にアクセスする方法を提供しますが、使うデータベースによっては、これらの機能が利用できない場合があることを理解してください。

これは、複数のベンダーが提供するデータベースを使うアプリケーションを構築する場合に特に言えることです。つまり、すべてのベンダーが、利用可能な各 JDBC クラス、メソッド、およびメソッド引数のすべての機能を完全にサポートするとは限りません。さらに重要なことは、JDBC メソッド呼び出しに引数として埋め込まれた SQL

ステートメントのセットが、サーバアプリケーションが使うデータベースによってサポートされたりサポートされなかったりする可能性があることです。JDBC を最大限に使うためには、データベースのマニュアルで、SQL および JDBC のどの特性がサポートされるかについて調べる必要があります。したがって、データベースの問題について iPlanet のテクニカルサポートに問い合わせる前に、まず問題の原因となっているデータベースを削除してください。

iPlanet Application Server の制約事項の理解

iPlanet Application Server は、JDBC のように、さまざまなデータベースエンジンおよび機能をサポートします。iPlanet Application Server 自体または iPlanet Application Server の JDBC ドライバが、特定のデータベース機能を完全にサポートできない場合があります。つまり、間違った情報がレポートされることがあります。iPlanet Application Server アプリケーションからデータベース機能にアクセスできない場合に、そのデータベースを削除しても問題が解消されない場合は、この節の説明およびリリースノートを読んで、発生した問題が iPlanet Application Server の制約事項として記載されているかどうかを調べます。その制約事項が問題の原因でない場合は、その問題を完全に記録した上で iPlanet テクニカルサポートまでご連絡ください。

注 JDBC アクセス問題には、iPlanet Application Server の JDBC ドライバに部分的にしかサポートされていない、またはまったくサポートされていない JDBC 機能にアクセスすることで発生するものもあります。機能の制約事項のほとんどすべてが JDBC 2.0 に適用されます。

表 9-1 は、iPlanet Application Server で部分的にしかサポートされていない、またはまったくサポートされていない JDBC 機能を示しています。

表 9-1 JDBC 機能の制約事項

機能	制約事項
エスケープシーケンス	Oracle データベース以外では使用できない
`Connection.setTransactionIsolation`	データベースベンダーによってサポートされている分離レベルだけを操作する
`Connection.getTypeMap`	タイプマップはサポートされていない
`Connection.setTypeMap`	タイプマップはサポートされていない
`Connection.cancel`	サポートするデータベースだけを操作する
`PreparedStatement.setObject`	シンプルデータタイプだけを操作する

表 9-1　JDBC 機能の制約事項 (続き)

機能	制約事項
PreparedStatement.addBatch	変更されたレコードの数を返す、サポートされているデータ操作ステートメントだけを操作する
PreparedStatement.setRef	参照はサポートされていない
PreparedStatement.setBlob	BLOB はサポートされていない。その代わりに setBinaryStream() を使う
PreparedStatement.setClob	CLOB はサポートされていない。その代わりに setBinaryStream() を使う
PreparedStatement.setArray	ARRAY はサポートされていないその代わりに setBinaryStream() を使う
PreparedStatement.getMetaData	サポートされていない
CallableStatement.getObject	スカラタイプだけを操作する JDBC 2.0 は、マップ引数が含まれるこのメソッドのセカンドバージョンを提供する。マップ引数は無視される
CallableStatement.getRef	参照はサポートされていない
CallableStatement.getBlob	SQL3 スタイル BLOB はサポートされていない
CallableStatement.getClob	SQL3 スタイル CLOB はサポートされていない
CallableStatement.getArray	ARRAY はサポートされていない
CallableStatement	更新可能な ResultSet はサポートされていない
ResultSet.getCursorName	データベースによって動作が異なる
	Oracle では、ユーザが SetCursorName を使ってカーソル名を指定しない場合は空の文字列が返される
	Sybase では、リザルトセットが更新不可能な場合は iPlanet Application Server によってカーソル名が自動的に生成される。それ以外は、空の文字列が返される
	ODBC、Informix、および DB2 では、何も指定されない場合はドライバがカーソル名を返す
ResultSet.getObject	スカラタイプだけを操作する JDBC 2.0 は、マップ引数が含まれるこのメソッドのほかの 2 つのバージョンを提供する。マップ引数は無視される
ResultSet.updateObject	スカラタイプだけを操作する
ResultSet.getRef	参照はサポートされていない
ResultSet.getBlob	SQL3 スタイル BLOB はサポートされていない
ResultSet.getClob	SQL スタイル CLOB はサポートされていない

表 9-1　JDBC 機能の制約事項 (続き)

機能	制約事項
ResultSet.getArray	ARRAY はサポートされていない
ResultSetMetaData.getTableName	ODBC 以外のデータベースアクセスの場合は空の文字列を返す
DatabaseMetaData.getUDTs	サポートされていない
行挿入後の executeUpdate	DB2 の場合、1 の代わりに 0 を返す

ResultSet、ResultSetMetaData、および PreparedStatement の操作の詳細については、この章の後続の節を参照してください。

サポートされるデータベース

iPlanet Application Server でサポートされるデータベースは定期的に更新されるので、データベースベンダーのアップグレードに従って、『iPlanet Application Server インストールガイド』または最新情報については『リリースノート』を参照してください。

6.x DD XML ファイルの 6.5 への移行

iPlanet Application Server 6.5 には、データベースコネクションの基盤に関連する機能の拡張が含まれています。この新しい機能を利用し、旧来のデータソースの設定を使い続けるには、新しいデータソース XML DTD に移行する必要があります。

6.x 配置記述子 XML ファイルへの移行

1. 「ツール」>「データソースを登録」を選択します。

   データソースを登録するダイアログボックスが表示されます。

2. 「開く」をクリックし、データソースの記述が含まれた XML ファイルを選択します。

   配置ツールによって、以前の XML ファイルにあった値がこの XML ファイルにインポートされ、6.5 で利用できるようになった新しいフィールドにデフォルトの値が入れられます。

3. データソースのデフォルト値を受け入れるか、変更します。

4. データベースドライバのパラメータ、コネクションプールのパラメータ、およびコネクション妥当性のパラメータを入力します。

   各フィールドの詳細については、『iPlanet Application Server 管理者ガイド』の第 8 章「データベース接続の管理」を参照してください。

5. 「保存」をクリックします。

   配置ツールによって、指定した値のデータソース配置記述子が開いたファイルに上書きされます。

   注 : 更新された XML ファイルを別の名前で異なる場所に保存する場合は、「名前を付けて保存」を選択します。

6. 「登録」をクリックします。

   配置ツールによって、データソース配置記述子が指定した新しい値で更新されます。これで、データソースは iPlanet Application Server 6.5 の新しい機能を利用できるようになります。

新規の XML データソース記述子

iPlanet Application Server 6.5 のトランザクションマネージャの基盤の変更とともに、データソース記述子も変更されました。新しい記述子には、さらに多くの機能とオプションがあります。

以下は、サードパーティドライバ用の XML データソース記述子の例です。

ローカルトランザクション

以下の XML ファイルの例は、iPlanet Application Server の IASConnectionPoolDataSource を使うデータベース用です。IASConnectionPoolDataSource は、データベース特有のドライバマネージャをラップします。

- Oracle
- Sybase

Oracle

```
<ias-resource>

  <resource>

    <jndi-name>jdbc/estore/EstoreDB</jndi-name>
```

```
    <jdbc>
      <user>estore</user>
      <password>estore</password>
      <URL> jdbc:oracle:thin:@192.18.117.186:1521:orcl</URL>
      <driver-name>oracle_xa</driver-name>
      <conn-pooling>
        <initialPoolSize>1</initialPoolSize>
        <waitQueueEnabled>true</waitQueueEnabled>
        <reclaimTime>600</reclaimTime>
        <maxPoolSize>30</maxPoolSize>
        <maxIdleTime>120</maxIdleTime>
        <queueLength>30</queueLength>
        <trace>disable</trace>
        <stat>disable</stat>
        <waitTimeInQueue>120</waitTimeInQueue>
        <tableBasedSanity>false</tableBasedSanity>
        <isSanityRequired>true</isSanityRequired>
        <incrementPoolSize>1</incrementPoolSize>
         <minPoolSize>1</minPoolSize>
     </conn-pooling>
    </jdbc>
  </resource>
</ias-resource>
```

Sybase

```
<ias-resource>
      <resource>
        <jndi-name>jdbc/estore/EstoreDB</jndi-name>
        <jdbc>
          <URL> jdbc:sybase:Tds:192.138.151.39:4444</URL>
          <user>estore</user>
```

```
            <password>estore</password>
            <driver-name>jconnect</driver-name>
            <conn-pooling>
              <initialPoolSize>1</initialPoolSize>
              <waitQueueEnabled>true</waitQueueEnabled>
              <reclaimTime>600</reclaimTime>
              <maxPoolSize>30</maxPoolSize>
              <maxIdleTime>120</maxIdleTime>
              <queueLength>30</queueLength>
              <trace>disable</trace>
              <stat>disable</stat>
              <waitTimeInQueue>120</waitTimeInQueue>
              <tableBasedSanity>false</tableBasedSanity>
              <isSanityRequired>true</isSanityRequired>
              <incrementPoolSize>1</incrementPoolSize>
              <minPoolSize>1</minPoolSize>
            </conn-pooling>
          </jdbc>
        </resource>
</ias-resource>
```

グローバルトランザクション

以下に示す XML ファイルの例は、ドライバ付属の XADataSource / ConnectionPoolDataSource (JDBC 2.0 + 拡張機能) を使うデータベースドライバ用です。アプリケーションでグローバルトランザクションとローカルトランザクションの両方を使う場合は、これらの XML フォーマットの使用をお勧めします。

すべてのデータソース XML ファイル内で、コネクションプールの要素 (conn-pooling) はオプションです。省略した場合は、デフォルト値が使用されます。詳細については、『iPlanet Application Server 管理者ガイド』を参照してください。XADatasource は、グローバルトランザクションが有効な場合にだけ使用されます。グローバルトランザクションの使用法については、『管理者ガイド』を参照してください。

- DB2

- MSSQL
- Oracle
- Sequelink
- Sybase

DB2

```
<ias-resource>
    <resource>
        <jndi-name>jdbc/sample</jndi-name>
        <jdbc>
            <dataSourceName>friend</dataSourceName>
            <user>db2inst</user>
            <password>db2inst</password>
            <driver-name>db2_xa</driver-name>
            <databaseName>sample4</databaseName>
            <portNumber>50001</portNumber>
            <conn-pooling>
                <initialPoolSize>1</initialPoolSize>
                <waitQueueEnabled>true</waitQueueEnabled>
                <reclaimTime>600</reclaimTime>
                <maxPoolSize>30</maxPoolSize>
                <maxIdleTime>120</maxIdleTime>
                <queueLength>30</queueLength>
                <trace>disable</trace>
                <stat>disable</stat>
                <waitTimeInQueue>120</waitTimeInQueue>
                <tableBasedSanity>false</tableBasedSanity>
                <isSanityRequired>true</isSanityRequired>
                <incrementPoolSize>1</incrementPoolSize>
                <minPoolSize>1</minPoolSize>
            </conn-pooling>
```

```
        </jdbc>
    </resource>
</ias-resource>
```

Informix

```
<ias-resource>
  <resource>
    <jndi-name>jdbc/dshubble</jndi-name>
    <jdbc>
          <user>root</user>
          <datasourceName>rna_tcp</datasourceName>
          <databaseName>jts</databaseName>
          <serverName>rna_tcp</serverName>
          <portNumber>1528</portNumber>
          <ifxIFXHOST>rna</ifxIFXHOST>
          <password>abc123</password>
          <driver-name>ifx</driver-name>

<URL>jdbc:informix-sqli://rna:1528/sample:INFORMIXSERVER=rna_tcp</U
RL>
          <conn-pooling>
                <initialPoolSize>1</initialPoolSize>
                <waitQueueEnabled>true</waitQueueEnabled>
                <reclaimTime>600</reclaimTime>
                <maxPoolSize>30</maxPoolSize>
                <maxIdleTime>120</maxIdleTime>
                <queueLength>30</queueLength>
                <trace>disable</trace>
                <stat>disable</stat>
                <waitTimeInQueue>120</waitTimeInQueue>
                <tableBasedSanity>false</tableBasedSanity>
```

```
                <isSanityRequired>true</isSanityRequired>
                <incrementPoolSize>1</incrementPoolSize>
                <minPoolSize>1</minPoolSize>
            </conn-pooling>
        </jdbc>
    </resource>
</ias-resource>
```

MSSQL

```
<ias-resource>
    <resource>
        <jndi-name>jdbc/sample</jndi-name>
        <jdbc>
            <dataSourceName>lancer</dataSourceName>
            <user>sa</user>
            <password></password>
            <driver-name>mssql</driver-name>
            <databaseName>master</databaseName>
            <networkProtocol>Tds</networkProtocol>
            <resourceManagerName>testrm</resourceManagerName>
            <serverName>lancer</serverName>
            <conn-pooling>
                <initialPoolSize>1</initialPoolSize>
                <waitQueueEnabled>true</waitQueueEnabled>
                <reclaimTime>600</reclaimTime>
                <maxPoolSize>30</maxPoolSize>
                <maxIdleTime>120</maxIdleTime>
                <queueLength>30</queueLength>
                <trace>disable</trace>
                <stat>disable</stat>
```

```
                <waitTimeInQueue>120</waitTimeInQueue>
                <tableBasedSanity>false</tableBasedSanity>
                <isSanityRequired>true</isSanityRequired>
                <incrementPoolSize>1</incrementPoolSize>
                <minPoolSize>1</minPoolSize>
            </conn-pooling>
        </jdbc>
    </resource>
</ias-resource>
```

Oracle

```
<ias-resource>
    <resource>
        <jndi-name>jdbc/sample</jndi-name>
        <jdbc>
            <URL>jdbc:oracle:oci8:@hubble</URL>
            <user>estore</user>
            <password>estore</password>
            <databaseName>hubble</databaseName>
            <driver-name>oracle_xa</driver-name>
            <conn-pooling>
                <initialPoolSize>1</initialPoolSize>
                <waitQueueEnabled>true</waitQueueEnabled>
                <reclaimTime>600</reclaimTime>
                <maxPoolSize>30</maxPoolSize>
                <maxIdleTime>120</maxIdleTime>
                <queueLength>30</queueLength>
                <trace>disable</trace>
                <stat>disable</stat>
                <waitTimeInQueue>120</waitTimeInQueue>
                <tableBasedSanity>false</tableBasedSanity>
```

```
                <isSanityRequired>true</isSanityRequired>
                <incrementPoolSize>1</incrementPoolSize>
                <minPoolSize>1</minPoolSize>
            </conn-pooling>
        </jdbc>
    </resource>
</ias-resource>
```

Sequelink

```
<ias-resource>
    <resource>
        <jndi-name>jdbc/sample</jndi-name>
        <jdbc>
            <datasourceName>mig</datasourceName>
            <user>kdemo</user>
            <password>kdemo</password>
            <driver-name>sequelink</driver-name>
            <databaseName>mig</databaseName>
            <serverName>mig</serverName>
            <portNumber>23003</portNumber>
            <URL>jdbc:sequeliik://mig:23003</URL>
            <conn-pooling>
                <initialPoolSize>1</initialPoolSize>
                <waitQueueEnabled>true</waitQueueEnabled>
                <reclaimTime>600</reclaimTime>
                <maxPoolSize>30</maxPoolSize>
                <maxIdleTime>120</maxIdleTime>
                <queueLength>30</queueLength>
                <trace>disable</trace>
```

```
                <stat>disable</stat>
                <waitTimeInQueue>120</waitTimeInQueue>
                <tableBasedSanity>false</tableBasedSanity>
                <isSanityRequired>true</isSanityRequired>
                <incrementPoolSize>1</incrementPoolSize>
                <minPoolSize>1</minPoolSize>
            </conn-pooling>
        </jdbc>
    </resource>
</ias-resource>
```

Sybase

```
<ias-resource>
    <resource>
    <jndi-name>jdbc/sample</jndi-name>
    <jdbc>
        <dataSourceName>prodigy</dataSourceName>
        <user>iplanet</user>
        <password>iplanet</password>
        <driver-name>sybase</driver-name>
        <databaseName>iplanet</databaseName>
        <networkProtocol>Tds</networkProtocol>
        <portNumber>4100</portNumber>
        <serverName>prodigy</serverName>
        <URL>jdbc:sybase:Tds:prodigy:4100</URL>
        <conn-pooling>
            <initialPoolSize>1</initialPoolSize>
            <waitQueueEnabled>true</waitQueueEnabled>
            <reclaimTime>600</reclaimTime>
            <maxPoolSize>30</maxPoolSize>
            <maxIdleTime>120</maxIdleTime>
```

```
                <queueLength>30</queueLength>
                <trace>disable</trace>
                <stat>disable</stat>
                <waitTimeInQueue>120</waitTimeInQueue>
                <tableBasedSanity>false</tableBasedSanity>
                <isSanityRequired>true</isSanityRequired>
                <incrementPoolSize>1</incrementPoolSize>
                <propertyCycle>0</propertyCycle>
                <minPoolSize>1</minPoolSize>
            </conn-pooling>
        </jdbc>
        </resource>
</ias-resource>
```

サーバアプリケーションでの JDBC の使用法

JDBC は iPlanet Application Server ランタイム環境の一部です。これは、Java を使ってアプリケーションをプログラミングするときには、常に JDBC が利用できることを意味します。典型的な多層サーバアプリケーションでは、Servlet 内および EJB 内で JDBC を使うと、クライアントやプレゼンテーションレイヤから EIS データベースにアクセスできます。

ただし、実際問題として、多層サーバアプリケーションの中間レイヤへのデータベースアクセスを制限することは、セキュリティおよび移植性に対して効果的です。iPlanet Application Server プログラミングモデルでは、これは EJB に対してプリファレンスを持つ Servlet および EJB に、すべての JDBC 呼び出しを配置することを意味します。

このプログラミングプリファレンスには 2 つの理由があります。

- すべての JDBC 呼び出しを EJB 内部に配置すると、アプリケーションがさらにモジュール化され移植性が向上するため
- EJB がトランザクション制御にビルトインメカニズムを提供するため

適切に設計された EJB に JDBC 呼び出しを配置すると、JDBC、または JDBC で低レベルトランザクションをサポートする `java.transaction.UserTransaction` を使った明示的なトランザクション制御をプログラミングする必要がありません。

注	EJB トランザクションマネージャがトランザクションを制御できるように、グローバルに利用できるデータソースを常に使って、グローバルな (Bean ワイドな) コネクションを作成してください。

EJB での JDBC の使用法

JDBC 呼び出しを EJB に配置すると、サーバアプリケーションの移植性が確実に向上します。これによって、明示的な JDBC 呼び出しを使ってトランザクション制御を管理する必要もなくなります。EJB はコンポーネントであるため、多数のアプリケーションでは、EJB をほとんど変更しないか、またはまったく変更しないビルディングブロックとして使って、EIS データベースへの共通のインタフェースを管理します。

JDBC または javax.transaction.UserTransaction によるトランザクションの管理

トランザクションの管理には、EJB トランザクション属性プロパティを使うことをお勧めしますがこれは必須ではありません。JDBC または `javax.transaction.UserTransaction` を使うトランザクション管理の明示的なプログラミングが、アプリケーションに適している場合があります。これらの場合、Bean 本体でトランザクション管理をプログラミングします。EJB で明示的なトランザクションを使うことを Bean 管理トランザクションと呼びます。

トランザクションは特定のメソッドに対してローカル (メソッド固有) になったり、または Bean 全体 (`Bean` ワイド) を含んだりします。

Bean 管理トランザクションを作成するには次の 2 つの手順があります。

1. Bean の配置記述子で EJB の `Transaction Type` プロパティを `Bean` に設定します。
2. トランザクションを起動、コミットまたはロールバックするステートメントを含む、適切な JDBC またはトランザクション管理ステートメントを Bean でプログラミングします。

`Transaction Type` プロパティが `Bean` 以外の場合、EJB に明示的なトランザクション処理をプログラミングしないでください。JDBC を使ったトランザクション処理の詳細については、JDBC 2.0 API 仕様書を参照してください。

トランザクションの分離レベルの指定

setTransactionIsolation() および getTransactionIsolation() メソッドをそれぞれ使って、コネクションのトランザクションレベルの指定または確認を行います。トランザクションの途中で setTransactionIsolation() を呼び出すことはできないので注意してください。

表 9-2 では、次のようにトランザクション分離レベルを定義します。

表 9-2 トランザクション分離レベル

トランザクション分離レベル	説明
TRANSACTION_NONE	トランザクションはサポートされない。Connection.getTransactionIsolation() だけで使われる
TRANSACTION_READ_COMMITTED	不正な読み込みを防ぐ。再現しない読み込みとファントム読み込みが発生する可能性がある
TRANSACTION_READ_UNCOMMITTED	不正な読み込み、再現しない読み込み、およびファントム読み込みが発生する可能性がある
TRANSACTION_REPEATABLE_READ	不正な読み込みと再現しない読み込みを防ぐ。ファントム読み込みが発生する可能性がある
TRANSACTION_SERIALIZABLE	不正な読み込み、再現しない読み込み、およびファントム読み込みを防ぐ

Bean のトランザクション分離レベルを指定する前に、データベース管理システムがそのレベルをサポートしていることを確認してください。すべてのデータベースがすべての分離レベルをサポートするとは限りません。次の例のように、java.sql.DatabaseMetaData の supportsTransactionIsolationLevel() メソッドを使うことによって、プログラムでデータベースをテストできます。

```
java.sql.DatabaseMetaData db;
if (db.supportsTransactionIsolationLevel(TRANSACTION_SERIALIZABLE) {
   Connection.setTransactionIsolation(TRANSACTION_SERIALIZABLE);
}
```

これらの分離レベルとその意味の詳細については、JDBC 2.0 API 仕様書を参照してください。

Servlet 内での JDBC の使用法

Servlet は iPlanet Application Server アプリケーションの中枢です。Servlet は、ブラウザ上の HTML ページや HTML を生成する JSP などのクライアントインタフェースと、アプリケーションの大部分の動作を実行する EJB の間に位置しています。

iPlanet Application Server アプリケーションは、EJB に埋め込まれた JDBC を使ってほとんどのデータベースにアクセスします。これは、iPlanet Application Server を使ったデータベースアクセスに最適なメソッドです。なぜなら、これによって EJB およびそのコンテナに組み込まれているトランザクション制御を利用できるからです。しかし、Servlet を使っても、JDBC を介してデータベースにアクセスできます。

Servlet から直接データベースにアクセスすると、EJB からデータベースにアクセスするよりも高速になる場合があります。EJB は Java Remote Method Interface (RMI) 経由でだけアクセス可能なため、アプリケーションが複数のサーバに分散している場合は、呼び出しのオーバーヘッドが小さくなります。Servlet からの直接データベースサービスを多用しないでください。Servlet からのデータベースアクセスを提供する場合は、アクセス時間が非常に短く、トランザクションが読み取り専用で、JDBC 2.0 の `RowSet` クラスを利用できる状況へのアクセスに制限されます。

データベースへのアクセスが Servlet からの場合は、JDBC 2.0 の `RowSet` インタフェースを使ってデータベースと対話してください。行セットは、データベースまたはスプレッドシートなどのほかの表形式データソースから取得した一連の行をカプセル化する Java オブジェクトです。`RowSet` インタフェースは、データソースに接続して一連の行を取得するように `RowSet` インスタンスを設定できる JavaBeans プロパティを提供します。行セットの操作方法の詳細については、229 ページの「RowSet の操作」の節を参照してください。

コネクションの処理

iPlanet Application Server は、JDBC 2.0 互換インタフェース `java.sql.Connection` を実装しています。コネクションの動作は、コネクションがローカルか、グローバルか、またはコンテナ管理ローカルコネクションかによって異なります。

ローカルコネクション

`Connection` オブジェクトは、トランザクションコンテキストが EJB コンテナによって管理されていない場合はローカルコネクションと呼ばれます。ローカルコネクションのトランザクションコンテキストは、複数のプロセスまたはデータソース全体に伝播させることができません。つまり、現在のプロセスおよび現在のデータソースに対してローカルです。

このコネクションのタイプのトランザクションコンテキストは、setAutoCommit()、commit()、および rollback() メソッドを使って管理されます。

ローカルデータソースの登録

ローカルコネクションを作成するには、まず iPlanet Application Server にデータソースを登録します。データソースを登録すると、登録したそのデータソースを使って、getConnection() を使って一覧表示されたデータベースに接続できます。

データソースの登録は、データソースのプロパティを記述する XML リソース記述子ファイルを作成することによって行います。次に、管理ツール、または resreg ユーティリティを使って、iPlanet Application Server にプロパティを登録します。resreg はデータソースを記述するリソース記述子ファイル名を引数と見なします。

注	実行中、resreg は既存のエントリを上書きします。

たとえば、ユーザ名 kdemo、パスワード kdemo、データベース ksample、およびサーバ ksample を使って、Oracle データベースに接続する SampleDS と呼ばれるデータソースを登録するには、次のような XML 記述子ファイルを作成して、SampleDS.xml という名前を付けます (この XML ファイルは iPlanet Application Server 配置ツールを使って作成します)。

```
<ias-resource>
   <resource>
       <jndi-name>jdbc/SampleDS</jndi-name>
       <jdbc>
          <database>ksample</database>
          <datasource>ksample</datasource>
          <username>kdemo</username>
          <password>kdemo</password>
          <driver-type>ORACLE_OCI</driver-type>
       </jdbc>
   </resource>
</ias-resource>
```

次に、次のコマンドによって、このリソース記述子ファイルを使ってデータソースを登録します。

```
resreg SampleDS.xml
```

リソース記述子ファイルの詳細については、第 11 章「配置のためのパッケージ化」を参照してください。iPlanet Application Server 管理ツールの詳細については、『管理者ガイド』を参照してください。

グローバルコネクション

Connection オブジェクトは、トランザクションコンテキストが EJB コンテナによって管理されている場合はグローバルコネクションと呼ばれます。グローバルコネクションのトランザクションコンテキストは、データソース全体に伝播させることができます。コンテナ管理トランザクションの場合、トランザクションコンテキストは EJB コンテナによって暗黙的に管理され、Bean 管理トランザクションの場合は明示的に管理されます。トランザクションの詳細については、第 8 章「EJB のトランザクション処理」を参照してください。

たとえば、setAutoCommit()、commit()、rollback() などのトランザクション管理メソッドは、グローバルコネクションでは無効です。

グローバルデータソースの登録

グローバルコネクションを作成するには、まず iPlanet Application Server にデータソースを登録します。データソースを登録したら、そのデータソースを使って、getConnection() を使って一覧表示されるデータベースに接続します。

データソースの登録は、データソースのプロパティを記述する XML リソース記述子ファイルを作成することによって行います。次に、管理ツール、または resreg ユーティリティを使って、iPlanet Application Server にプロパティを登録します。resreg はデータソースを記述するリソース記述子ファイル名を引数と見なします。

注	実行中、resreg は既存のエントリを上書きします。

たとえば、ユーザ名 kdemo、パスワード kdemo、データベース ksample、およびサーバ ksample を使って、Oracle データベースに接続する GlobalSampleDS と呼ばれるデータソースを登録するには、次のような XML 記述子ファイルを作成して、GlobalSampleDS.xml という名前を付けます (この XML ファイルは iPlanet Application Server 配置ツールを使って作成)。

```
<ias-resource>
   <resource>
       <jndi-name>jdbc/GlobalSampleDS</jndi-name>
       <jdbc>
          <database>ksample</database>
          <datasource>ksample</datasource>
          <username>kdemo</username>
          <password>kdemo</password>
          <driver-type>ORACLE_OCI</driver-type>
```

```
            <resource-mgr>ksample_rm</resource-mgr>
        </jdbc>
    </resource>
</ias-resource>
```

次のコマンドによって、このリソース記述子ファイルを使ってデータソースを登録します。

```
resreg GlobalSampleDS.xml
```

リソース記述子ファイルの詳細については、第 11 章「配置のためのパッケージ化」を参照してください。iPlanet Application Server 管理ツールの詳細については、『管理者ガイド』を参照してください。

グローバルコネクションの作成

次のプログラムは、データソースを検索し、そのデータソースからコネクションを作成する方法を示します。ここに示されているように、検索される文字列はリソース記述子ファイル内の <jndi-name> タグに指定されるものと同じです。

```
InitialContext ctx = null;
String dsName1 = "jdbc/GlobalSampleDS";
DataSource ds1 = null;

try
{
    ctx = new InitialContext();
    ds1 = (DataSource)ctx.lookup(dsName1);

    UserTransaction tx = ejbContext.getUserTransaction();

    tx.begin();

    Connection conn1 = ds1.getConnection();

    // データベースの作業には conn1 を使います。conn1.commit()、
    // conn1.rollback()、および conn1.setAutoCommit() はここでは使用できないので注意してください。

    tx.commit();

} catch(Exception e) {
    e.printStackTrace(System.out);
}
```

コンテナ管理ローカルコネクション

トランザクションコンテキストが EJB コンテナによって管理され、グローバルトランザクションが無効になっている場合、Connection オブジェクトはコンテナ管理ローカルコネクションであると考えられます。コンテナ管理トランザクションの場合、トランザクションコンテキストは EJB コンテナによって暗黙的に管理され、Bean 管理トランザクションの場合には明示的に管理されます。

Connection オブジェクトメソッド `setAutoCommit()`、`commit()`、および `rollback()` は、このタイプのコネクションでは無効です。

EJB コンテナ内でグローバルトランザクションを有効または無効にする方法については、『管理者ガイド』を参照してください。

コンテナ管理ローカルデータソースの登録

コンテナ管理ローカルデータソースの登録プロセスは、ローカルおよびグローバルデータソースと同じです。詳細については、217 ページの「ローカルデータソースの登録」を参照してください。

JDBC 機能の操作

この章では JDBC については説明しませんが、iPlanet Application Server で、EJB に JDBC を使う方法を紹介します。次の節では、さまざまな JDBC インタフェースおよびクラスについて説明します。これらのインタフェースおよびクラスには、iPlanet Application Server 環境に特定の必要条件があるか、または iPlanet Application Server アプリケーションの開発時に特に有益な新しい JDBC 2.0 機能があります。

たとえば、221 ページの「コネクションの操作」では JDBC の実装ごとに情報が異なるため、コネクションを閉じるときに iPlanet Application Server が解放するリソースについて説明します。また、222 ページの「コネクションのプール」および 229 ページの「RowSet の操作」では、能力、柔軟性、およびサーバアプリケーションの速度を高める新しい JDBC 2.0 機能についてより詳しく説明します。

この節には次のトピックがあります。

- コネクションの操作
- コネクションのプール
- ResultSet の操作
- ResultSetMetaData の操作
- PreparedStatement の操作

- CallableStatement の操作
- バッチ更新の操作
- 分散トランザクションの作成
- RowSet の操作
- JNDI を使ったデータベースドライバ

コネクションの操作

JDBC コネクションを開くと、iPlanet Application Server はそのコネクションリソースを割り当てます。コネクションが不要になったときに `Connection.close()` を呼び出すと、そのコネクションリソースが解放されます。`Connection.close()` を呼び出したあとに継続してデータベース操作を行うには、その前に必ずコネクションを確立し直してください。

`Connection.isClose()` を使って、コネクションが閉じているかどうかをテストします。このメソッドは、コネクションが開いていると false を返し、`Connection.close()` が呼び出された場合だけ true を返します。閉じたコネクションで JDBC オペレーションを行うとスローされる例外を見つけることによって、データベースコネクションが不正かどうかを調べます。

最後に、コネクションの開閉には時間がかかります。アプリケーションが複数のコネクションを使う場合や、コネクションを頻繁に開いたり閉じたりする場合、iPlanet Application Server は自動的にコネクションをプールします。コネクションをプールすると、必要に応じて自動的に閉じるコネクションのキャッシュが生成されます。

注	コネクションのプールは iPlanet Application Server の自動的な機能です。API は公開されていません。

setTransactionIsolation

すべてのデータベースベンダーが、JDBC で利用できるすべてのトランザクション分離レベルをサポートしているわけではありません。iPlanet Application Server を使うと、ユーザのデータベースサポートの任意の分離レベルを指定できますが、iPlanet Application Server はユーザのデータベースがサポートしていない値に対する例外をスローします。詳細については、215 ページの「トランザクションの分離レベルの指定」を参照してください。

getTypeMap、*setTypeMap*

固有の JDBC ドライバが実装された iPlanet Application Server は、新しい SQL-3 機能であるタイプマッピングをサポートしません。この機能は、ほとんどのデータベースベンダーがサポートしていません。

cancel

`cancel()` は、`cancel()` をサポートするすべてのデータベースでサポートされています。

コネクションのプール

JDBC で実行する 2 つのデータベース操作、データベースコネクションの作成および破棄には時間がかかります。コネクションをプールすると、1 つのコネクションキャッシュをコネクションリクエストに使用できます。コネクションは実際には破棄されずに、あとで再利用するためにプールに戻されます。コネクションを作成するためにあとで呼び出すと、プールから利用可能なコネクションを取得します。

JDBC 呼び出しを作成すると、iPlanet Application Server は常に自動的に JDBC コネクションをプールします。データベースコネクションをプールするプロセスは、コネクションのタイプによって動作が異なります。

- **ローカルコネクション**の場合、データベースコネクションはアプリケーションによって閉じられたときにプールされます。
- **グローバルコネクション**の場合、データベースコネクションはトランザクションを開始したスレッドに接続されます。これらのコネクションはスレッドで実行するトランザクションによってあとで再利用されます。
- **コンテナ管理ローカルコネクション**の場合、`connection.close()` メソッドはコネクションをコネクションプールにすぐに解放しません。コネクションが関係しているトランザクションが終了すると、コネクションは iPlanet Application Server によって解放され、コネクションプールに戻されます。

それぞれの Java エンジンには、各ドライバ (Oracle、Sybase、Informix、および DB2) にそれぞれのコネクションプールがあります。それぞれのコネクションプールのサイズは、アプリケーションの要件によって異なります。コネクションプールの設定 (コネクションの最大数やコネクションタイムアウトなど) の詳細については、『管理者ガイド』を参照してください。

ResultSet の操作

ResultSet は、データベースクエリによって返されるデータをカプセル化するクラスです。このクラスに関連する次の動作または制約事項に注意してください。

注 この節では、固有の JDBC ドライバの機能について説明します。サードパーティの JDBC ドライバによってサポートされるオプションについては、そのドライバベンダーのドキュメントを参照してください。

同時性のサポート

iPlanet Application Server は、FORWARD-ONLY READ-ONLY および SCROLL-INSENSITIVE READ-ONLY リザルトセットの同時性をサポートしています。呼び出し可能なステートメントでは、iPlanet Application Server は FORWARD-ONLY UPDATABLE リザルトセットの同時性もサポートしています。

SCROLL-SENSITIVE の同時性はサポートされていません。

更新可能なリザルトセットのサポート

iPlanet Application Server では、更新可能なリザルトセットの作成は 1 つのテーブルのクエリに限定されます。更新可能なリザルトセットの SELECT クエリには、FOR UPDATE 句を含む必要があります。

```
SELECT...FOR UPDATE [OF column_name_list]
```

注 ジョインを使って複数のテーブルに対して読み取り専用のリザルトセットを作成できますが、これらのリザルトセットは更新できません。

Sybase では、選択リストに固有のインデックス列が必要です。Sybase を使うと、execute() または executeQuery() を呼び出して更新可能なリザルトセットも作成できます。ただし、ほかの SQL ステートメントを実行する前に、このステートメントを閉じる必要があります。

Oracle 8 で更新可能なリザルトセットを使うには、次のようにトランザクションでリザルトセットクエリをラップする必要があります。

```
conn.setAutoCommit(false);
ResultSet rs =
   stmt.executeQuery("SELECT...FOR UPDATE...");
...
rs.updateRows();
...
conn.commit();
```

Microsoft SQL Server では、リザルトセットの同時性が CONCUR_UPDATABLE の場合、execute() または executeQuery() メソッドの SELECT ステートメントに ORDER BY 句を含めないでください。

getCursorName

リザルトセットメソッドの 1 つである getCursorName() を使うと、リザルトセットをフェッチするために使われるカーソル名を調べることができます。カーソル名がクエリ自体で指定されていない場合は、さまざまなデータベースベンダーがさまざまな情報を返します。iPlanet Application Server はこれらの情報をできるだけ透過的に処理しようとします。表 9-3 では、開始クエリでカーソル名が指定されていない場合に各データベースベンダーが返すカーソル名を示しています。

表 9-3 カーソル名

データベースのベンダー	getCursorName の戻り値
Oracle	カーソル名が setCursorName() を使って指定されていない場合は、空の文字列を返す
Sybase	カーソル名が setCursorName() を使って指定されていない場合、およびリザルトセットを更新できない場合は、iPlanet Application Server によって固有のカーソル名が自動的に生成される。それ以外は、空の文字列が返される
Informix、DB2、ODBC	カーソル名が setCursorName() を使って指定されていない場合は、ドライバによって固有のカーソル名が自動的に生成される

getObject

iPlanet Application Server はこの JDBC メソッドを実装し、スカラデータタイプを使った場合だけ動作します。JDBC 2.0 はマップ引数を含む補足のメソッドバージョンを追加します。iPlanet Application Server はマップを実装せず、マップ引数を無視します。

getRef、*getBlob*、*getClob*、および *getArray*

参照、BLOB、CLOB、およびアレイは新しい SQL-3 データタイプです。iPlanet Application Server はこれらのデータオブジェクトまたはメソッドを実装しません。しかし、`getBinaryStream()` および `setBinaryStream()` を使って、参照、BLOB、CLOB、およびアレイを操作できます。

ResultSetMetaData の操作

`getTableName()` メソッドは、OBDC 互換のデータベースにとって意味のある情報だけを返します。ほかのすべてのデータベースには、空の文字列を返します。

PreparedStatement の操作

`PreparedStatement` はデータのフェッチに繰り返し使われるクエリ、更新、または挿入ステートメントをカプセル化するクラスです。このクラスに関連する次の動作または制約事項に注意してください。

注	iPlanet Application Server 機能 `SqlUtil.loadQuery()` を使って、コンパイル済みステートメントで `iASRowSet` を読み込むことができます。詳細については、『Foundation Class Reference (Java)』の `SqlUtil` クラスの項目を参照してください。

setObject

このメソッドは、スカラデータタイプとともに使います。

addBatch

このメソッドを使うと、データ操作ステートメントのセットをまとめて、1 つのステートメントとしてデータベースに渡すことができます。`addBatch()` は、更新された行数または挿入された行数のカウントを返す SQL データ操作ステートメントだけ操作します。JDBC 2.0 仕様書の要求に反して、`addBatch()` は CREATE TABLE などの SQL データ定義ステートメントを操作しません。

setRef、*setBlob*、*setClob*、*setArray*

参照、BLOB、CLOB、およびアレイは新しい SQL-3 データタイプです。iPlanet Application Server は、これらのデータオブジェクト、またはデータオブジェクトを操作するメソッドを実装しません。しかし、`getBinaryStream()` および `setBinaryStream()` を使って、参照、BLOB、CLOB、およびアレイを操作できます。

getMetaData

すべてのデータベースシステムが、完全なメタデータ情報を返すとは限りません。ご使用のデータベースがどのタイプのメタデータをクライアントに提供するかを調べるには、データベースのマニュアルを参照してください。

CallableStatement の操作

`CallableStatement` は、ストアドプロシージャからリザルトセットを返すことをサポートするデータベースの、データベースプロシージャまたは関数呼び出しをカプセル化するクラスです。このクラスに関連する次の制約事項に注意してください。JDBC 2.0 仕様書には、呼び出し可能なステートメントは更新可能なリザルトセットを返すことができると記述されています。この機能は iPlanet Application Server ではサポートされていません。

getRef、*getBlob*、*getClob*、*getArray*

参照、BLOB、CLOB、およびアレイは新しい SQL-3 データタイプです。iPlanet Application Server は、これらのデータオブジェクト、またはデータオブジェクトを操作するメソッドを実装しません。しかし、`getBinaryStream()` および `setBinaryStream()` を使って、参照、BLOB、CLOB、およびアレイを操作できます。

バッチ更新の操作

JDBC 2.0 仕様書は、アプリケーションが 1 つのデータベースリクエストにある複数の SQL 更新ステートメント (`INSERT`、`UPDATE`、`DELETE`) を渡す準備をするバッチ更新機能について規定しています。ステートメントを 1 つにまとめると、大量の更新ステートメントが保留されている場合のパフォーマンスを著しく向上します。

`Statement` クラスには、バッチ更新を実行する 2 つの新しいメソッドがあります。

- `addBatch()` を使うと、実行する前に SQL 更新ステートメント (`INSERT`、`UPDATE`、`DELETE`) をステートメントのグループに追加できます。このメソッドを使って、簡単な更新カウントを返す更新ステートメントだけを 1 つにまとめることができます。
- `executeBatch()` を使うと、1 つのデータベースリクエストとして SQL 更新ステートメントのコレクションを実行できます。

バッチ更新を使うには、アプリケーションは自動コミットオプションを次のように無効にする必要があります。

```
...
// 自動コミットを無効にして、各ステートメントが別々にコミットすることを防ぎます。
con.setAutoCommit(false);

Statement stmt = con.createStatement();

stmt.addBatch("INSERT INTO employees VALUES(4671, 'James
Williams')");
stmt.addBatch("INSERT INTO departments VALUES(560, 'Produce')");
stmt.addBatch("INSERT INTO emp_dept VALUES( 4671, 560)");

// 一連のアップデートを実行するために送信します。
int[] updateCounts = stmt.executeBatch();
con.commit();
```

`executeBatch()` の前に、エラーが検出されたなどの理由でバッチオペレーションから 1 つにまとめられたすべてのステートメントを削除するには、`clearBatch()` を呼び出します。

注 JDBC 2.0 仕様書では、誤って、バッチ更新に `CREATE TABLE` などのデータ定義言語 (DDL) ステートメントが含まれる可能性があることを暗示しています。DDL ステートメントは簡単な更新カウントを返さず、バッチオペレーションに対してグループ化できません。また、トランザクションでデータ定義ステートメントを使用できないデータベースもあります。

分散トランザクションの作成

JDBC 2.0 仕様書は、分散トランザクションを処理する機能について規定しています。分散トランザクションは、別々のサーバマシンにある複数の異なるデータベースに適用する 1 つのトランザクションです。

分散トランザクションのサポートは、すでに iPlanet Application Server EJB コンテナに組み込まれています。EJB が `TX_BEAN_MANAGED` トランザクション属性を指定しない場合は、アプリケーションの分散トランザクションの自動サポートが有効になります。

Servlet および `TX_BEAN_MANAGED` トランザクション属性を指定する EJB でも、分散トランザクションを使用できますが、JTS の `UserTransaction` クラスを使ってトランザクションを管理する必要があります。次のようにします。

```
InitialContext ctx = null;
String dsName1 = "jdbc/SampleDS1";
String dsName2 = "jdbc/SampleDS2";
DataSource ds1 = null;
DataSource ds2 = null;
```

```
   try {
       ctx = new InitialContext();
       ds1 = (DataSource)ctx.lookup(dsName1);
       ds2 = (DataSource)ctx.lookup(dsName2);

   } catch(Exception e) {
 e.printStackTrace(System.out);
   }

UserTransaction tx = ejbContext.getUserTransaction();

tx.begin();

Connection conn1 = ds1.getConnection();
Connection conn2 = ds2.getConnection();

// いくつかの作業をここで行います。

tx.commit();
```

この例では、ds1 および ds2 は iPlanet Application Server を使ってグローバルデータソースとして登録する必要があります。つまり、データソースプロパティファイルには、値をインストール時に設定しなければならない ResourceMgr エントリを含める必要があります。

```
DataBase=ksample
DataSource=ksample
UserName=kdemo
PassWord=kdemo
DriverType=ORACLE_OCI
ResourceMgr=orarm
```

この例では、orarm は有効な ResourceMgr エントリである必要があり、グローバルコネクションを確実に取得できる必要があります。ResourceMgr エントリ を有効にするには、リソースマネージャは CCS0¥RESOURCEMGR にレジストリを一覧表示する必要があります。エントリ自体には次のプロパティが必要です。

```
DatabaseType (string key)
IsEnabled (integer type)
Openstring ( string type key)
ThreadMode ( string type key)
```

RowSet の操作

RowSet は、データベースやスプレッドシートなど、他の表形式データストアから取得した一連の行をカプセル化するオブジェクトです。RowSet を実装するには、プログラムが javax.sql をインポートして、RowSet インタフェースを実装する必要があります。RowSet は java.sql.ResultSet インタフェースを拡張して、JavaBean コンポーネントとしての役割を果たすことができます。

RowSet は JavaBean なので、RowSet のイベントを実装して、RowSet にプロパティを設定できます。さらに、RowSet は ResultSet のエクステンションなので、ResultSet を繰り返す場合と同様に、RowSet を繰り返すことができます。

RowSet を埋めるには、RowSet.execute() メソッドを呼び出します。execute() メソッドは、プロパティ値を使ってデータソースを調べてデータを取得します。設定および確認する必要があるプロパティは、起動する RowSet の実装によって異なります。

RowSet インタフェースの詳細については、『JDBC 2.0 Standard Extension API Specification』を参照してください。

iASRowSet の使用

iPlanet Application Server には、便宜上、iASRowSet という名前の RowSet クラスがあります。iASRowSet は ResultSet を拡張するので、呼び出しメソッドを ResultSet オブジェクトから継承します。iASRowSet は、ResultSet の getMetaData() および close() メソッドをオーバーライドします。iASRowSet は、ドライバレベルのクラスではないので、ResultSet より使いやすくなっています。

RowSet インタフェースは、表 9-4 に示すものを除いて完全にサポートされています。

表 9-4 RowSet インタフェースサポートの例外

「Method」	引数	スローされる例外	理由
setReadOnly()	false	SQLException	iASRowSet はすでに読み取り専用である
setType()	TYPE_SCROLL_INSENSITIVE	SQLException	SCROLL_INSENSITIVE はサポートされていない
setConcurrency()	CONCUR_UPDATABLE	SQLException	iASRowSet は読み取り専用である
addRowSetListener()	任意	なし	サポートされていない

表 9-4　RowSet インタフェースサポートの例外 (続き)

「Method」	引数	スローされる例外	理由
removeRowSetListener()	任意	なし	サポートされていない
setNull()	すべてのタイプの名前	引数は無視される	サポートされていない
setTypeMap()	java.util.Map	なし	マップは、現在サポートされていない JDBC 2.0 の機能である

RowSetReader

iASRowSet は RowSetReader クラスを完全に実装しています。

RowSetWriter

iASRowSet は読み取り専用です。しかし、このクラスのインタフェースは将来の拡張に備えて提供されています。現在は、唯一のメソッドである writeData() が SQLException をスローします。

RowSetInternal

この内部クラスは、RowSet についての情報を取得するために RowSetReader によって使われます。1 つのメソッド getOriginalRow() がありますが、1 つの行の代わりに元の ResultSet を返します。

CachedRowSet の使用法

JDBC の仕様書では、CachedRowSet と呼ばれる RowSet クラスについて規定しています。CachedRowSet を使うとデータソースからデータを取得でき、その後データの確認および変更を行う場合にデータソースから取り除きます。キャッシュした行セットは取得した元のデータ、およびアプリケーションによるデータの変更を記録します。アプリケーションが元のデータソースを更新しようとすると、行セットはデータソースに再び接続され、変更された行だけがデータベースにマージされます。

RowSet の作成

iPlanet Application Server アプリケーションに行セットを作成するには、次のコマンドを入力します。

```
iASRowSet rs = new iASRowSet();
```

JNDI を使ったデータベースドライバ

iPlanet Application Server に実装されている JDBC ドライバマネージャなどのすべての JDBC ドライバマネージャが、データベースに接続する JDBC ドライバおよび JDBC URL を調べて、そのドライバにアクセスする必要があります。ただし、JDBC URL は特定のベンダーの JDBC 実装に固有なだけでなく、特定のマシンおよびポート番号に固有な場合があります。このようなハードコードされた依存性によって、あとで異なる JDBC 実装およびマシンに簡単に移すことができる移植可能なアプリケーションの作成が難しくなります。

JDBC 2.0 には JNDI を使って、アプリケーションがネットワーク上でリモートサービスを見つけてアクセスするための、プラットフォームおよび JDBC ベンダーに依存しない同一の方法を提供することが明記されています。このハードコードされた情報の代わりに、JNDI を使うと特定のデータソースに論理名を割り当てることができます。論理名を確立すると、配置およびアプリケーションの位置を変更するには論理名を 1 回変更するだけで済みます。

JDBC 2.0 には、すべての JDBC データソースを JNDI ネーム空間の `jdbc` ネーミングサブコンテキストに、または子サブコンテキストの一つに登録することが明示されています。JNDI ネーム空間はファイルシステムのディレクトリ構造のように階層的であるため、簡単に参照を見つけてネストできます。データソースは、論理 JNDI 名にバインドされます。その名前は、ルートコンテキストのサブコンテキスト `jdbc` および論理名を識別します。データソースを変更するには、アプリケーションを変更せずに JNDI ネーム空間のエントリを変更するだけです。

JNDI の詳細については、JDBC 2.0 Standard Extension API を参照してください。

この節の残りの部分では、データソース検索の例を使って、リソースファクトリを参照する方法について説明します。以下の参照方法は、すべてのリソースに適用できます (JavaMail 参照など)。

アプリケーションコード内でリソースを検索するには、次のように行います。

```
String dsName = "java:comp/env/HelloDbDataSource";
DataSource ds = (javax.sql.DataSource)initContext.lookup(dsName);
Connection conn = ds.getConnection();
```

照会するリソースは、`web.xml` ファイルの `res-ref-name` 属性に次のように指定します。

```
<resource-ref>
   <description>Datasource Reference</description>
   <res-ref-name>HelloDbDataSource</res-ref-name>
   <res-type>javax.sql.DataSource</res-type>
   <res-auth>Container</res-auth>
</resource-ref>
```

iPlanet 固有の配置記述子 ias-web.xml の resource-ref セクションでは、res-ref-name (アプリケーションコードでクエリの対象になる名前) をデータソースの JNDI 名に割り当てます。JNDI 名には、リソースをサーバに登録するときに、リソースファイルに定義したデータソースの名前を指定します。

```
<resource-ref>
   <res-ref-name>HelloDbDataSource</res-ref-name>
   <jndi-name>jdbc/hellodb/HelloDbDB</jndi-name>
</resource-ref>
```

リソース登録ファイルは XML ファイルです。データソースの JNDI 名を指定し、iPlanet サーバに登録されているドライバに割り当てます。JNDI 名は、ias-web.xml ファイルの resource-ref セクションの jndi-name 属性に指定されていなければなりません。

```
<ias-resource>
   <resource>
       <jndi-name>jdbc/hellodb/HelloDbDB</jndi-name>
       <jdbc>
          <driver-type>PointBaseDriver</driver-type>
          <database-url>
              jdbc:pointbase://localhost/iassamples
          </database-url>
          <username>hellodb</username>
          <password>hellodb</password>
       </jdbc>
   </resource>
</ias-resource>
```

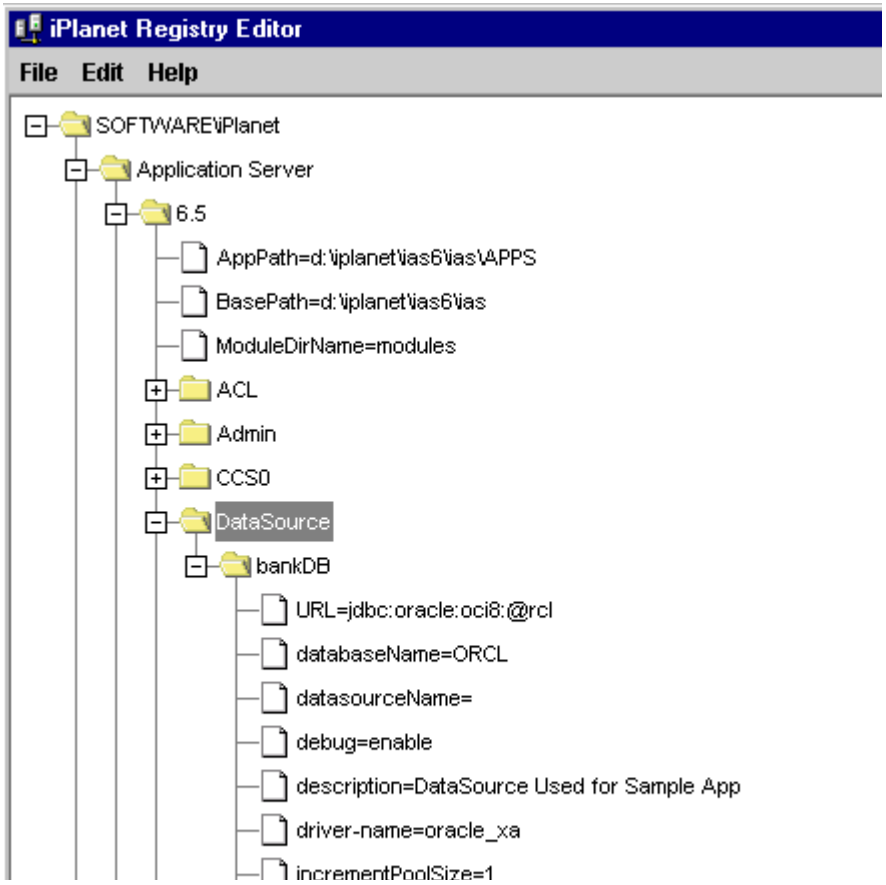
web.xml、ias-web.xml、およびリソース XML ファイルの詳細については、第 11 章「配置のためのパッケージ化」を参照してください。

登録したリソースは、LDAP ネーム空間のレジストリにある次のセクションに挿入されます。

```
SOFTWARE¥iPlanet¥Application Server¥6.5¥DataSource
```

図 9-1 は、レジストリエントリを示しています。

図 9-1　データソースのレジストリエントリ



第 10 章

CORBA ベースクライアントの開発と配置

この章では、iPlanet Application Server 環境内で IIOP (RMI over IIOP または IDL over IIOP) プロトコルを介して EJB にアクセスする方法について説明します。

この章には次の節があります。

- CORBA クライアントサポートの概要
- RMI/IIOP クライアントアプリケーション
- C++ IIOP クライアントアプリケーション (UNIX のみ)

CORBA クライアントサポートの概要

iPlanet Application Server では、『Enterprise JavaBeans Specification, V1.1』および『Enterprise JavaBeans to CORBA Mapping』仕様書で指定されている IIOP プロトコルを経由した EJB へのアクセスがサポートされています。CORBA クライアントは、JNDI を使って EJB を検索し、Java RMI/IIOP または IIOP とともに C++ IDL を使って、リモート EJB のビジネスメソッドにアクセスします。

この章には次の節があります。

- シナリオ
- アーキテクチャの概要
- iPlanet の付加価値機能
- 制約事項
- ORB の選択

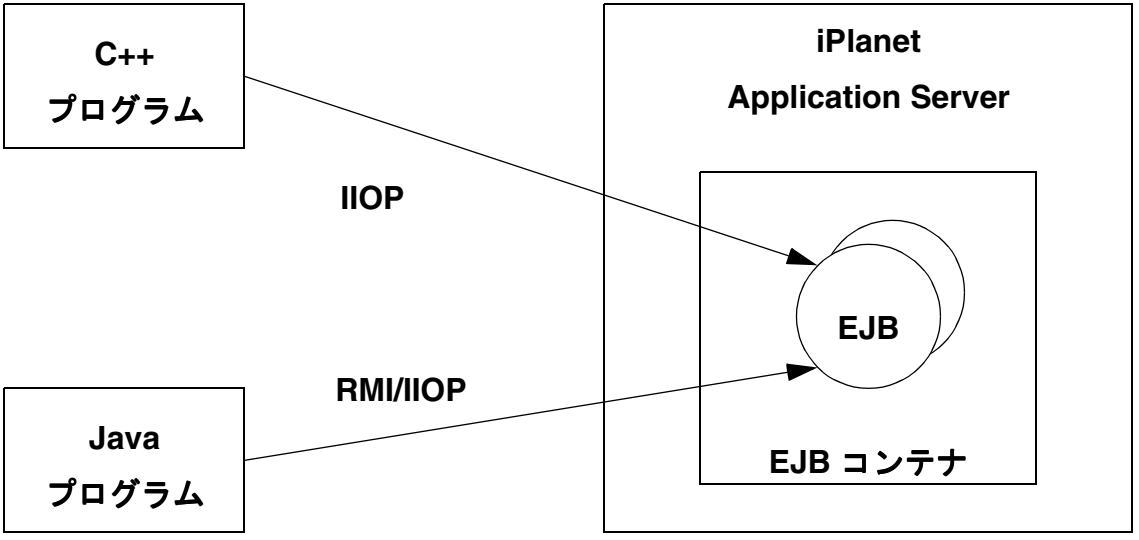
シナリオ

CORBA クライアントを採用するもっとも一般的なシナリオは、スタンドアロンプログラムまたは別のアプリケーションサーバを、iPlanet Application Server に配置した EJB のクライアントとして動作させるケースです。

スタンドアロンのプログラム

もっとも単純なケースの場合、図 10-1 のように、さまざまな OS で動作するスタンドアロンのプログラムは、IIOP を使ってバックエンド EJB コンポーネントに配置されているビジネスロジックにアクセスします。

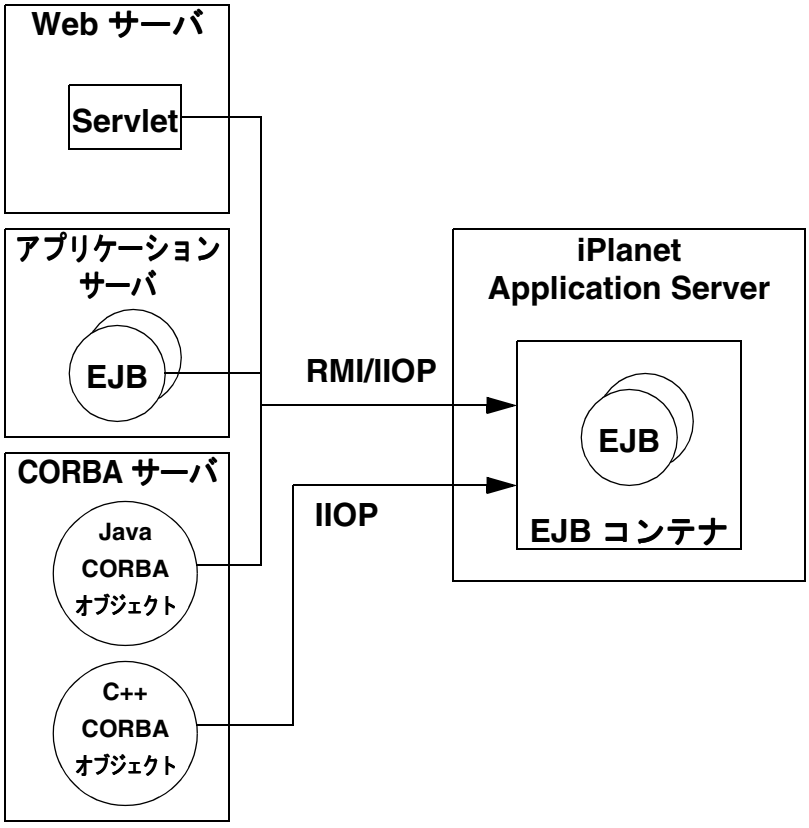
図 10-1　スタンドアロンのプログラム



サーバ間

図 10-2 の、Web サーバや CORBA オブジェクト、その他のアプリケーションサーバも、IIOP を使って iPlanet Application Server に配置された EJB にアクセスできます。

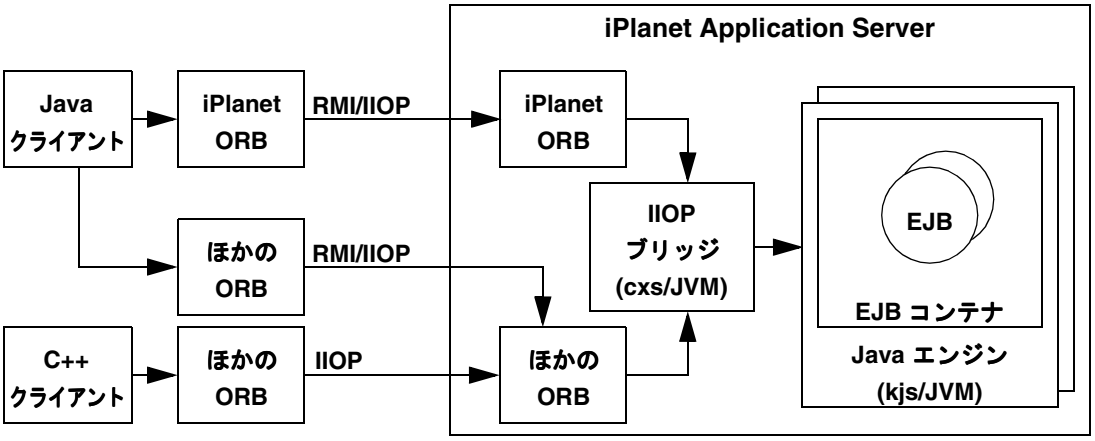
図 10-2　サーバ間



アーキテクチャの概要

iPlanet Application Server での CORBA クライアントサポートには、CORBA Executive Server (CXS) という特殊な Java エンジンプロセスが関与します。CXS は、IIOP を使う Java または C++ クライアントと、EJB コンテナとしての役割を果たす 1 つまたは複数の Java エンジンに配置された EJB 間のブリッジとして機能します。この IIOP ブリッジプロセスでは、図 10-3 のように、CORBA クライアントがアクセスする EJB ごとに、着信する IIOP ベースのリクエストが処理されます。これらのリクエストは EJB コンテナ内に配置された EJB への内部呼び出しにマッピングされます。

図 10-3　アーキテクチャ



このリリースの iPlanet Application Server では、iPlanet Application Server に組み込まれている ORB やサードパーティの ORB (ORBIX 2000) を使うことができます。

注　バンドルされている iPlanet ORB は、JDK 1.2 以前、または iPlanet Application Server の初期のバージョンでは機能しません。

iPlanet の付加価値機能

CORBA クライアントの iPlanet 実装では、次の付加価値機能が提供されるため、仕様以上の機能が実現します。

- ネーミングサービス
- C++ クライアントサポート
- 組み込み ORB とサードパーティ ORB のサポート
- 基本認証と EJB コンテナの統合
- クライアントサイド認証
- ロードバランス
- スケーラビリティ
- 利用度の向上

- ファイヤウォールで開くポートの数の最小化

ネーミングサービス

CORBA クライアントは、標準 CORBA COS Naming Service を使って EJBHome オブジェクトを解決します。EJB が iPlanet Application Server に配置されると、EJB は、ネーミングサービスに自動的かつダイナミックに登録されます。

C++ クライアントサポート

このリリースの iPlanet Application Server では、UNIX システム上の C++ クライアントで IIOP を使用できます。詳細については、279 ページの「C++ IIOP クライアントアプリケーション (UNIX のみ)」を参照してください。

組み込み ORB とサードパーティ ORB のサポート

iPlanet には、EJB への IIOP アクセスをサポートする組み込み ORB が用意されています。iPlanet Application Server で IIOP を使うために、サードパーティ ORB (ORBIX 2000) をインストールして設定することもできます。詳細については、243 ページの「ORB の選択」を参照してください。

基本認証と EJB コンテナの統合

CORBA 標準では、CORBA クライアントと EJB サーバ間で基本認証を実行する方法はまだ定義されていませんが、iPlanet のアプリケーションサーバでは、このサポートが提供されています。この機能によって、EJB 配置者は Web および CORBA クライアントの両方に適用される標準の宣言およびプログラム制御を使って EJB へのアクセスを制御できます。

CORBA クライアントが iPlanet Application Server に認証を求めると、標準 EJB セキュリティメカニズムに基づいて、認証に必要な主要な情報が EJB コンテナに自動的に伝播します。iPlanet には、クライアントのユーザ名およびパスワードのコレクションをトリガするクライアントサイドコールバックメカニズムが用意されています。このメカニズムによって、アプリケーションは、アプリケーション固有の方法でユーザ名とパスワードを取得します。iPlanet CORBA インフラストラクチャがユーザ名およびパスワード情報を収集すると、この情報は、IIOP を介してアプリケーションサーバに伝播します。

クライアントサイド認証

ORBIX 2000 が RMI/IIOP に使用される ORB の場合、移植可能なインターセプタは、セキュリティを実装するためにフック (インターセプトポイント) を設定します。インターセプトポイントは、リクエストにステージを定義し、シーケンスを返信します。ネーミングサービスは、これらのインターセプトポイントを使って、リクエストの照会とデータの返信を行い、クライアントとサーバ間のサービスコンテキストを転送します。

インターセプタは、クライアントサイドとサーバサイドに 1 つずつ設定されます。クライアントインターセプタは、サーバに送信される前にリクエストをインターセプトし、Principal クラス (com.netscape.ejb.client.IUserPrincipal を実装するクラス) が設定されているかどうかを確認します。設定されている場合は、そのクラスをインスタンス化し、setPrincipal メソッドを呼び出します。setPrincipal メソッドには、userId および password インスタンス変数を設定する必要があります。このメソッド内でカスタムコードを実装し、ユーザ ID とパスワードを取得するためです。クライアントインターセプタは、対応するアクセサを使って、これらの値を取得します。

取得した値は、PICurrent オブジェクト (スロットのテーブル) に格納されます。PICurrent オブジェクト内のスロットは、ユーザ ID とパスワードごとに作成されます。次に、クライアントインターセプタは、ユーザ ID とパスワードごとにサービスコンテキストを作成し、それらを追加したリクエストを送信します。ユーザ ID とパスワードは、最初のリクエストの PICurrent オブジェクトに格納されます。後続のリクエストのユーザ ID とパスワードは、PICurrent オブジェクトから取得されます。

Principal クラスが設定されていない場合、リクエストはそのまま送信され、サービスコンテキストデータは追加されません。

サーバサイドでは、リクエストを受信すると、サーバインターセプタを呼び出します。サーバインターセプタは、サービスコンテキストデータがリクエストに存在するかどうかを確認します。存在しない場合、リクエストの処理を継続します。サービスコンテキストデータが存在する場合、サーバインターセプタはユーザ ID とパスワードを抽出して、セキュリティマネージャの login メソッドを呼び出します。このメソッドは、ユーザを認証します。認証に失敗した場合は、例外をクライアントに返します。認証に成功した場合は、リクエストの処理を継続します。

注 認証 / 認可に失敗すると、java.rmi.AccessException がクライアントへスローされます。java.rmi.AccessException は java.rmi.RemoteException から派生します。

この機能を有効にするには、ORBInitializer クラスを使って、サーバサイドおよびクライアントサイドにインターセプタを登録する必要があります。クライアントサイドの ORBInitilaizer クラスでも、PICurrent オブジェクトを作成します。

アクセスを制御するロールは、メソッドまたは Bean レベルで割り当てられます。EJB コンテナは、セキュリティマネージャからロールマッピング情報を取得して、メソッドまたは Bean へのアクセスを認可します。ユーザが認可された場合は、そのメソッドが実行されます。認可されない場合は、`java.rmi.AccessException` がスローされます。

ロードバランス

新しい IIOP リクエストが iPlanet Application Server のインスタンスに着信すると、iPlanet Application Server は、EJB コンテナとして機能する 1 つまたは複数の JVM 間でこれらのリクエストをロードバランスします。ロードバランスは単純なラウンドロビン方式で実装されています。アプリケーションサーバを起動すると、使用可能な EJB コンテナプロセス (Java エンジン) のリストを取得します。ホーム検索リクエストを CORBA クライアントから受信すると、アプリケーションサーバは、エンジンのリストから、EJB ホームを管理しているターゲットエンジンを選択します。それに続く EJB ホームの検索、そのホームでの Bean の作成、および作成した Bean でのビジネスメソッドの起動は、同じターゲットエンジンに関連します。

クライアントサイドロードバランス

組み込みの iPlanet ORB が使用される場合、クライアントアプリケーションは、使用可能な CORBA プロセスのリストを繰り返したり、ラウンドロビン DNS を使って基本的なクライアントサイドロードバランスを実装したりできます。

ORBIX 2000 が使用される ORB の場合は、ほかの方式のクライアントサイドロードバランスを利用できます。ネーミングサービスでは、名前とオブジェクトを対応付けたリポジトリを定義します。名前とオブジェクトは、1 対 1 で対応付けます。ORBIX 2000 では、ネーミングサービスモデルを拡張し、1 つの名前をオブジェクトのグループに対応付けることができます。オブジェクトグループは、オブジェクトの集合で、サイズを動的に調節できます。オブジェクトグループごとに、選択アルゴリズムを指定します。このアルゴリズムは、オブジェクトグループに関連付けられた名前をクライアントが解決するときに適用されます。次の 3 つのアルゴリズムがサポートされています。

- ラウンドロビン選択
- ランダム選択
- アクティブロードバランス選択

オブジェクトグループを利用すれば、頻繁に要求されるオブジェクトを複製して、リクエストの処理負荷を分散することができます。ネーミングサービスは、オブジェクトグループの選択アルゴリズムに従って、クライアントリクエストを複製されたオブジェクトに転送します。オブジェクトグループは、クライアントに対して透過的です。クライアントは、ほかの名前と同様に、オブジェクトグループの名前を解決します。

スケーラビリティ

アプリケーションサーバの各インスタンスに、複数の CORBA プロセスを設定できます。この機能を使って、システム管理者は受信する IIOP リクエストを専門的に処理する任意の数の JVM を設定できます。また管理者は、各 CORBA および EJB コンテナプロセスが使用可能な処理スレッドの数を変更して、システムの予測される負荷に適合させることもできます。

利用度の向上

次の機能によって、利用度が向上します。

- **Java エンジンの自動再起動** : アプリケーションサーバは、EJB コンテナをサポートする Java エンジンだけでなく、ブリッジプロセスも監視します。プロセスが失敗しても、管理サービスによって自動的にプロセスが再起動します。
- **状態のあるセッション Beans のフェールオーバー** : CORBA クライアントは、iPlanet Application Server に組み込まれた EJB の状態のあるセッション Beans のレプリケーション機能を利用できます。EJB コンテナを配置している Java エンジンが失敗しても、Java エンジンが再起動し、状態のあるセッション Beans に対する後続のリクエストが引き続き処理されます。
- **EJB ハンドルおよびオブジェクト参照フェールオーバー** : ブリッジプロセスが失敗しても、プロセスは自動的に再起動され、CORBA クライアントは、引き続き EJB にアクセスできます。

ファイヤウォールで開くポートの数の最小化

組み込みの iPlanet ORB が使用されると、ブリッジプロセスは、共通の固定 IP ポート番号を使って行われるネーミングサービスメソッドおよびビジネスメソッドの両方を呼び出します。この方法によって、CORBA クライアントと、ブリッジプロセスが設定されている iPlanet Application Server インスタンス間に配置されているファイヤウォールで開くポートの数を最小限に抑えることができます。

制約事項

iPlanet Application Server で CORBA クライアントを使う場合は、次の制約事項があります。

- EJB へのアクセスに限られる
- 一般的な RMI オブジェクトには RMI/IIOP 経由ではアクセスできない
- Java RMI/IIOP クライアントからのトランザクション伝播はサポートされない
- クライアントサイドで JDK 1.3.*x* を使用している場合は、基本的なデータタイプのみを交換できます。

ORB の選択

iPlanet には、EJB への IIOP アクセスをサポートする組み込み ORB が用意されています。iPlanet Application Server で IIOP を使うために、サードパーティ ORB (ORBIX 2000) をインストールして設定することもできます。

会社が ORBIX 2000 を標準 ORB として使っている場合、または EJB と通信する C++ クライアントを開発する場合は、iPlanet Application Server を ORBIX 2000 用に設定する必要があります。ORBIX 2000 では、追加の認証およびロードバランスも利用できます。ORBIX 2000 のインストール、および ORBIX 2000 と iPlanet Application Server の統合の詳細については、『管理者ガイド』を参照してください。

ORBIX 2000 用に RMI/IIOP アプリケーションを設定する方法については、266 ページの「ORBIX 用に RMI/IIOP アプリケーションを設定する」を参照してください。
ORBIX 2000 を使用するために C++ IIOP アプリケーションを設定する方法については、280 ページの「ORBIX 用 C++ IIOP アプリケーションの設定」を参照してください。

RMI/IIOP クライアントアプリケーション

iPlanet Application Server での RMI/IIOP ベースクライアントアプリケーションの使用法は、ほかの J2EE 認定アプリケーションサーバでのクライアントの使用法とほぼ同じです。クライアントの JNDI 検索部分に最小限の変更を加えるだけで、Java クライアントを再利用してさまざまな J2EE アプリケーションサーバと連動させることができます。

この節には次の節があります。

- RMI/IIOP クライアントアプリケーションの開発
- RMI/IIOP クライアントアプリケーションのパッケージング
- RMI/IIOP サポートの設定
- RMI/IIOP クライアントアプリケーションの配置
- ORBIX 用に RMI/IIOP アプリケーションを設定する
- RMI/IIOP クライアントアプリケーションの実行
- RMI/IIOP クライアントアプリケーションのトラブルシューティング
- RMI/IIOP のパフォーマンスチューニング
- RMI/IIOP のファイヤウォールの設定
- RMI/IIOP ログメッセージの表示

- RMI/IIOP サンプルアプリケーション

RMI/IIOP クライアントアプリケーションの開発

この節には次の節があります。

- EJB ホームインタフェースの JNDI 検索
- クライアント認証
- クライアントサイドのロードバランスおよびフェールオーバー

EJB ホームインタフェースの JNDI 検索

RMI/IIOP クライアントのコードを作成するには、最初に EJB のホームインタフェースを検索します。ホームインタフェースの JNDI を検索する準備として、まず、`InitialContext` の環境プロパティをいくつか設定する必要があります。次に、EJB の検索名を指定します。

次の節では、手順と例を示します。

- ネーミングファクトリクラスの指定
- ターゲット IIOP ブリッジの指定
- EJB の JNDI 名の指定
- JNDI サンプル

ネーミングファクトリクラスの指定

RMI/IIOP 仕様書に従って、クライアントは、`Properties` オブジェクトのインスタンス内の `java.naming.factory.initial` エントリの値として、`com.sun.jndi.cosnaming.CNCtxFactory` を指定する必要があります。さらに、このオブジェクトは、EJB のホームインタフェースを検索する前に JNDI `InitialContext` コンストラクタに渡されます。次のようにします。

```
...
Properties env = new Properties();
env.put("java.naming.factory.initial", "com.sun.jndi.cosnaming.CNCtxFactory");
env.put("java.naming.provider.url", "iiop://" + host + ":"+port);
Context initial = new InitialContext(env);
Object objref = initial.lookup("java:comp/env/ejb/MyConverter");
...
```

ターゲット IIOP ブリッジの指定

RMI/IIOP 仕様書に従って、クライアントは、`java.naming.provider.url` プロパティを次の形式の値に設定する必要があります。

`iiop://`*server*`:`*port*

server は、iPlanet Application Server インスタンスが配置されているホストを示します。*port* は、アプリケーションサーバホスト上で実行される IIOP ブリッジプロセスを示します。

`java.naming.factory.initial` プロパティとともに、`java.naming.provider.url` プロパティを、コマンドラインで、またはクライアントアプリケーションのコードで指定できます。

次に、Java コマンドライン (このコマンドはすべて 1 行で指定すること) で IIOP URL を設定する例を示します。

```
java -Djava.naming.provider.url="iiop://127.0.0.1 :9010"
-Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory
j2eeguide.cart.CartClient
```

この場合、クライアントアプリケーションは、`Properties` オブジェクトをインスタンス化する必要はありません。

```
...
public static void main(String[] args) {
   Context initial = new InitialContext();
   Object objref = initial.lookup("java:comp/env/ejb/MyConverter");
   ...
}
```

代わりの方法として、クライアントアプリケーション内に IIOP URL を設定できます。次の例では、クライアントのメインクラスに 2 つのコマンドライン引数を渡します。

```
...
public static void main(String[] args) {
   String host = args[0];
   String port = args[1];
   Properties env = new Properties();

   env.put("java.naming.factory.initial",
       "com.sun.jndi.cosnaming.CNCtxFactory");

   env.put("java.naming.provider.url", "iiop://" + host + ":"+port);

   Context initial = new InitialContext(env);
   Object objref = initial.lookup("java:comp/env/ejb/MyConverter");
...
}
```

EJB の JNDI 名の指定

新しい JNDI `InitialContext` オブジェクトが作成されると、クライアントは、`InitialContext` に対して `lookup` メソッドを呼び出して EJB のホームインタフェースを検索します。EJB の名前は、`lookup` の呼び出しで指定されています。RMI/IIOP を使ってリモート EJB にアクセスする場合、パラメータは、EJB の「JNDI 名」として参照されます。クライアントアプリケーションのパッケージ方法によって、サポートされている JNDI 名の値は異なります。

アプリケーションクライアントコンテナを使わない JNDI 名

クライアントがアプリケーションクライアントコンテナ (ACC) の一部としてパッケージされていない場合は、JNDI 検索で、EJB の絶対名を指定する必要があります。iPlanet では、ACC の外部で JNDI 検索を実行するために次の方法がサポートされています。

```
initial.lookup("ejb/ejb-name");
```

```
initial.lookup("ejb/module-name/ejb-name");
```

ejb-name は、EJB の配置記述子の `<ejb-name>` 要素内に存在するときの EJB の名前です。たとえば、次に値 MyConverter を使った検索を示します。

```
initial.lookup("ejb/MyConverter");
```

この検索では、次のように、EJB 配置記述子が `MyConverter` を `<ejb-name>` として指定する必要があります。

```
<ejb-jar>
   <enterprise-beans>
       <session>
          <ejb-name>MyConverter</ejb-name>
          <home>j2eeguide.converter.ConverterHome</home>
          <remote>j2eeguide.converter.Converter</remote>
          ...
       </session>
   </enterprise-beans>
</ejb-jar>
```

RMI/IIOP クライアント上で JNDI 検索に EJB 名だけを使った場合は、この名前を持つ EJB が 1 つだけアプリケーションサーバに登録されている場合に限り、適切に動作します。この名前を持つ複数の EJB が登録されている場合は、対象となる EJB が存在する EJB JAR モジュールの名前で、EJB 名を限定する必要があります。それには、JNDI 検索で EJB 名の前に EJB JAR モジュール名を付けます。EJB JAR モジュール名は、EJB JAR ファイルの名前から `.jar` 拡張子を除いた名前です。

Converter サンプルアプリケーションの EJB JAR モジュール名は `j2eeguide-converterEjb` (`j2eeguide-converterEjb.jar` の EJB JAR ファイル名に基づく) となるため、モジュール名をベースとした検索は次のようになります。

```
initial.lookup("ejb/j2eeguide-converterEjb/MyConverter");
```

アプリケーションクライアントコンテナのパッケージングを使わない RMI/IIOP クライアントから JNDI 検索を実行する場合は、必ずモジュール名修飾子を使うようにすると安全です。モジュール名を使う方法の唯一の欠点は、クライアントが、EJB 絶対名だけでなくサーバサイド環境の配置構造について余分な情報を認識してしまうことです。

Service Pack 3 では、絶対参照で検索を実行する場合、プレフィックス `java:comp/env/ejb/` も使用できます。たとえば、Converter サンプルでの検索は次のように記述できます。

```
initial.lookup("java:comp/env/ejb/MyConverter");
```

また、モジュール名を使う場合は次のように記述します。

```
initial.lookup("java:comp/env/ejb/j2eeguide-converterEjb/MyConverter");
```

このプレフィックスを指定した場合と最初の 2 つの方法の間には、機構的な違いはありません。この表記法は EJB 間接参照を使う場合にも利用されるので、EJB 絶対参照とともに `java:comp/env/ejb/` を使う場合、混乱しないよう注意が必要です。

アプリケーションクライアントコンテナを使う場合の JNDI 名

アプリケーションクライアントコンテナ (ACC) を使ってクライアントを収容する場合、JNDI 名には、ACC 配置記述子の `<ejb-ref-name>` 要素で指定されている EJB の論理名を使用できます。EJB の JNDI 名を指定するこの方法は、ACC のコンテキストにおけるクライアントのパッケージングおよび実行方法によって異なりますが、アプリケーションサーバ内に配置されている Servlet または EJB 内で使われている方法とほぼ同じです。

EJB で検索を実行する Servlet および EJB の場合と同様に、検索の形式は次の例のようになります。

```
initial.lookup("java:comp/env/ejb/ejb-ref-name");
```

ejb-ref-name は、ACC 配置記述子の `<ejb-ref-name>` 要素で指定されている値です。

次の例では、`SimpleConverter` は、ACC 配置記述子の `<ejb-ref-name>` 要素内で指定されているので、`SimpleConverter` の値は JNDI 検索で使われます。

```
initial.lookup("java:comp/env/ejb/SimpleConverter");
```

`application-client.xml` ファイルは次のようになります。

```
<application-client>
   <display-name>converter-acc</display-name>
   <description>
       Currency Converter Application Client Container Sample
   </description>
   <ejb-ref>
```

```
        <ejb-ref-name>SimpleConverter</ejb-ref-name>
        <ejb-ref-type>Session</ejb-ref-type>
        <home>j2eeguide.converter.ConverterHome</home>
        <remote>j2eeguide.converter.Converter</remote>
        <ejb-link>Test</ejb-link>
    </ejb-ref>
</application-client>
```

ACC パッケージングを使う利点は、クライアントアプリケーションで指定されている JNDI 名が、EJB の JNDI 絶対名に間接的にマッピングされることです。ACC を使う利点はほかにありません。詳細については、254 ページの「アプリケーションクライアントコンテナ (ACC) の使用」を参照してください。

JNDI サンプル

次のクライアントプログラムは、iPlanet Application Server にバンドルされている『J2EE 開発者ガイド』にあるサンプルの一部である Currency Converter アプリケーションからの抜粋です。アプリケーションサーバに含まれる RMI/IIOP の例については、278 ページの「RMI/IIOP サンプルアプリケーション」を参照してください。

```
package j2eeguide.converter;

import java.util.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;

import j2eeguide.converter.Converter;
import j2eeguide.converter.ConverterHome;

public class ConverterClient {

public static void main(String[] args) {
    try {

        if (args.length != 2) {
            System.out.println("Wrong number of arguments to client");
            System.exit(1);
        }
        String host = args[0];
        String port = args[1];
        Properties env = new Properties();
        env.put("java.naming.factory.initial",
            "com.sun.jndi.cosnaming.CNCtxFactory");
        env.put("java.naming.provider.url", "iiop://" + host
            + ":"+port);

        Context initial = new InitialContext(env);
        Object objref = initial.lookup("ejb/MyConverter");
```

```
// あるいは、モジュール名は修飾子として使用できます。
// Object objref =
// initial.lookup("ejb/j2eeguide-converterEjb/MyConverter");

        ConverterHome home
           =(ConverterHome)PortableRemoteObject.narrow(objref,
           ConverterHome.class);

        Converter currencyConverter = home.create();

        double amount = currencyConverter.dollarToYen(100.00);

        System.out.println(String.valueOf(amount));

        amount = currencyConverter.yenToEuro(100.00);

        System.out.println(String.valueOf(amount));

    }

    catch (Exception ex) {
        System.err.println("Caught an unexpected exception!");
        ex.printStackTrace();
    }
}
}
```

クライアント認証

RMI/IIOP クライアントのオプションの認証メカニズムを利用するには、`com.netscape.ejb.client.IUserPrincipal` インタフェースを実装するセキュリティプリンシパルクラスを指定する必要があります。JNDI `lookup` メソッドが呼び出されると、このクラスは、クライアントサイドの iPlanet RMI/IIOP インフラストラクチャによって一度インスタンス化されます。クライアントサイドの RMI/IIOP インフラストラクチャは、JNDI 検索によってリモートネーミングサービスへの呼び出しがトリガされる前に、このインタフェースの `setPrincipal` メソッドを呼び出します。

クライアント実行時に RMI/IIOP インフラストラクチャがクラスを読み込むには、このセキュリティプリンシパルクラスをクライアントのプロパティで指定し、クラスをクライアントの CLASSPATH 内に挿入する必要があります。

たとえば、Converter サンプルアプリケーションでは、JNDI 検索を実行する際に、セキュリティプリンシパルクラスのインスタンス化を指定する 3 番目のプロパティを追加できます。

```
...
Properties env = new Properties();
env.put("java.naming.factory.initial",
   "com.sun.jndi.cosnaming.CNCtxFactory");
env.put("java.naming.provider.url", "iiop://" + host + ":"+port);
env.put("com.netscape.ejb.client.PrincipalClass",
   "j2eeguide.converter.RmiPrincipal");

Context initial = new InitialContext(env);
Object objref = initial.lookup("ejb/MyConverter");
...
```

RmiPrincipal クラスは、com.netscape.ejb.client.IUserPrincipal インタフェースを実装するためにユーザが開発するクラスです。

プリンシパルクラスのサンプル

IUserPrinicpal インタフェースはさまざまな方法で実装できます。もっとも簡単な方法は setPrinciapl コールバックでダイアログをポップアップし、ユーザとパスワードの組み合わせを取得し、それらをユーザ名とパスワードの文字列フィールドに保存することです。これによって、クライアントが EJB を起動するたびに getUserId() および getPassword() メソッドが使われ、クライアントによって伝播されたセキュリティコンテキストが設定されます。

IIOP ブリッジは、iPlanet Application Server セキュリティマネージャを使ってユーザとパスワードを認証しようとします。ブリッジに認証例外が発生すると、クライアントサイド ORB に通知され、setPrincipal メソッドが呼び出されて正しいユーザ / パスワード情報を取得します。認証例外がクライアントサイドで発生すると、クライアントサイド RMI/IIOP インフラストラクチャは自動的にリクエストを 3 回試行します。

```
...
import com.netscape.ejb.client.IUserPrincipal;

public class Principal implements IUserPrincipal {

   private String username;
   private String password;

   public void setPrincipal() {
       //GUI をポップアップし、ユーザ名とパスワードを取得します。
   }

   public String getUserId() {
       return username;
   }
```

```
    public String getPassword() {
        return password;
    }
}
```

IUserPrinicpal のもう一つの有効な実装方法は、同じクライアントの JVM で複数のユーザ ID をサポートすることです。それには、ThreadLocal 変数を使って、ユーザ名とパスワードを保存します。この場合、IUserPrincipal 実装のメソッドに、ThreadLocal を認識させる必要があります。

クライアントサイドのロードバランスおよびフェールオーバー

iPlanet Application Server には RMI/IIOP アクセスのため、サーバサイドのロードバランスおよびフェールオーバーが用意されていますが、アプリケーションのパフォーマンスおよび利用度をさらに向上させるため、クライアントサイドの方法を実装することも検討してください。

iPlanet ORB の設定

組み込みの iPlanet ORB を使用している場合は、クライアントサイドのロードバランスを次のような方法で設定できます。

- 既知のブリッジのリストから手動で選択する

  クライアントビジネスアプリケーションの代わりに、一連の既知のブリッジホスト名およびポートの組み合わせをラウンドロビン方式で使う wrapper クラスを作成できます。ホスト名 / ポートの組み合わせの一つで通信例外が発生した場合、wrapper クラスは、リスト内の次のホスト名 / ポートの組み合わせを使います。

  たとえば、リモート IIOP ブリッジに接続できない場合、基本となるクライアントクラスによって次の例外がスローされます。

  ```
  javax.naming.CommunicationException:Cannot connect to ORB.Root
  exception is org.omg.CORBA.COMM_FAILURE:
  ```

  クライアントの wrapper コードはこの例外を受け取り、次に使用可能な *host_name*:*port* ペアを選択して EJB にアクセスし直します。

- ラウンドロビン DNS

  DNS のラウンドロビン機能を利用すると、クライアントのソースコードを変更せずに簡単なロードバランス方法を実装できます。この方法では、IIOP ブリッジプロセスがリッスンするときに使う複数の物理 IP アドレスを表す仮想ホスト名を 1 つ定義します。共通 IIOP ポート番号を使ってリッスンするようにすべての IIOP ブリッジプロセスを設定する場合、クライアントアプリケーションは、JNDI 検索時に 1 つの *host_name*:*IIOP_port* を使用できます。DNS サーバは、クライアントが実行されるたびに別の IP アドレスからホスト名を取得します。

  クライアントアプリケーションを開発した後、配置の準備としてアプリケーションをパッケージングする必要があります。

ORBIX の設定

ORBIX 2000 が使用される ORB の場合は、ほかの方式のクライアントサイドロードバランスを利用できます。ネーミングサービスでは、名前とオブジェクトを対応付けたリポジトリを定義します。名前とオブジェクトは、1 対 1 で対応付けます。ORBIX 2000 では、ネーミングサービスモデルを拡張し、1 つの名前をオブジェクトのグループに対応付けることができます。オブジェクトグループは、オブジェクトの集合で、サイズを動的に調節できます。オブジェクトグループごとに、選択アルゴリズムを指定します。このアルゴリズムは、オブジェクトグループに関連付けられた名前をクライアントが解決するときに適用されます。次の 3 つのアルゴリズムがサポートされています。

- ラウンドロビン選択
- ランダム選択
- アクティブロードバランス選択

オブジェクトグループを利用すれば、頻繁に要求されるオブジェクトを複製して、リクエストの処理負荷を分散することができます。ネーミングサービスは、オブジェクトグループの選択アルゴリズムに従って、クライアントリクエストを複製されたオブジェクトに転送します。オブジェクトグループは、クライアントに対して透過的です。クライアントは、ほかの名前と同様に、オブジェクトグループの名前を解決します。

UNIX の場合、フラグ `ORBIX_LOADBALANCING=true` または `false` を `iasenv.ksh` ファイルに設定すると、Java 引数を次のように設定できます。

```
-DORBIXLoadBalancing=$ORBIX_LOADBALANCING
```

Windows の場合、レジストリの Java 引数を次のように設定できます。

```
HKEY_LOCAL_MACHINE¥SOFTWARE¥iPlanet¥Application
Server¥6.5¥Java¥JavaArgs=-DORBIXLoadBalancing=true
```

RMI/IIOP クライアントアプリケーションのパッケージング

RMI/IIOP クライアントアプリケーションのパッケージング方法は次のとおりです。

- アセンブリツール GUI の使用法
- Ant を使った再組立の自動化
- アプリケーションクライアントコンテナ (ACC) の使用

アセンブリツール GUI の使用法

EJB に IIOP 経由でアクセス可能であることを指示すると、iPlanet Application Server 配置ツールによって、EJB 固有のホームおよびリモートインタフェースとスタブクラスを含む JAR ファイルが自動的に生成されます。個々のクラスファイルをクライアントにコピーする代わりに、この JAR ファイルをクライアントアプリケーションの一部として配置できます。

配置ツールでは、アプリケーションクライアントコンテナの一部として配置されるアプリケーションのパッケージングはサポートされていません。

Ant を使った再組立の自動化

RMI/IIOP クライアントアプリケーションのパッケージングをコマンドラインを使って行う場合は、サンプルアプリケーションの一部として提供されている Ant ベースの `build.xml` ファイルを参照することをお勧めします。RMI/IIOP ベースサンプルの `build.xml` ファイルには、`install_client` ターゲットが含まれています。配置ツールがクライアント指向クラスの JAR ファイルを生成する方法と同じように、このターゲットを簡単に強化して、自己完結型クライアント JAR ファイルを組み立てることができます。

アプリケーションクライアントコンテナ (ACC) の使用

iPlanet では、アプリケーションクライアントコンテナにクライアントアプリケーションを配置することをお勧めしませんが、この配置および実行時メソッドは J2EE 仕様の一部としてサポートされています。ただし、現時点の ACC 仕様では、ACC の使い方は複雑で、利点も少ないため、この方法はお勧めしません。また、J2EE v 1.2 では、ACC の定義が制限されているため、ACC のサポートは J2EE アプリケーションサーバによって大きく異なります。

iPlanet Application Server で ACC を試す場合は、次の配置手順を考慮してください。

- iPlanet Application Server の一部として提供されている `iasacc.jar` ファイルは、クライアントの CLASSPATH に含める必要があります。このファイルは、次の場所からクライアント環境にコピーできます。

  install_dir`/ias/classes/java/iasacc.jar`

  このファイルを CLASSPATH に含めると、クライアントの環境に `iasclient.jar` ファイルを含める必要はなくなります。

- J2EE v1.2 に準拠した EAR ファイルを作成する必要があります。この EAR ファイルには、次のものが必要です。

  - RMI/IIOP　クライアントアプリケーションクラス、ホームおよびリモートインタフェース、スタブ

- app-client.xml という名前の J2EE v1.2 XML 記述子ファイル。次に例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE application-client PUBLIC '-//Sun Microsystems,
Inc.//DTD J2EE Application Client 1.2//EN'
'http://java.sun.com/j2ee/dtds/application-client_1_2.dtd'>

<application-client>
   <display-name>converter-acc</display-name>
   <description>
   Currency Converter Application Client Container Sample
   </description>
   <ejb-ref>
          <ejb-ref-name>SimpleConverter</ejb-ref-name>
          <ejb-ref-type>Session</ejb-ref-type>
          <home>j2eeguide.converter.ConverterHome</home>
          <remote>j2eeguide.converter.Converter</remote>
          <ejb-link>Test</ejb-link>
   </ejb-ref>
</application-client>
```

- iPlanet Application Server 固有の XML 記述子ファイル (通常の名前は ias-app-client.xml)。この記述子によって、EJB 参照が EJB 絶対名にマッピングされます。

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE ias-java-client-jar PUBLIC '-//Sun Microsystems, Inc.//DTD iAS
Enterprise JavaBeans 1.0//EN'
'http://developer.iplanet.com/appserver/dtds/IASjava_client_jar_1_0.dtd'>

<ias-java-client-jar>
   <ejb-ref>
          <ejb-ref-name>SimpleConverter</ejb-ref-name>
          <jndi-name>ejb/MyConverter</jndi-name>
   </ejb-ref>
</ias-java-client-jar>
```

- J2EE v1.2 XML 記述子ファイル (application.xml)。

RMI/IIOP クライアントの EAR ファイルの構造に関する詳細については、第 11 章「配置のためのパッケージ化」を参照してください。

注 EJB JAR 名は、.jar 拡張子ではなく、ファイル名の最初の部分によって識別されます。Application Server に配置する EJB JAR 名は、一意でなければなりません。ejb-jar.xml ファイルの <ejb-name> 部分に指定する EAR ファイル名および EJB 名には、Java パッケージ方式の命名規則を使ってください。Java パッケージ方式の命名規則を使えば、名前の衝突は発生しません。この命名規則は、iPlanet Application Server だけでなく、ほかの J2EE アプリケーションサーバでも使うことをお勧めします。

アプリケーションクライアントコンテナを介してクライアントを呼び出すコマンドは、次のとおりです。

```
java com.netscape.ejb.client.AppContainer client_ear_file -iasXml ias_xml_file
```

RMI/IIOP サポートの設定

iPlanet Application Server に配置された EJB への RMI/IIOP アクセスを有効にするには、次の節で説明するようにアプリケーションサーバおよびクライアントの両方の環境を設定する必要があります。

- サーバの設定
- クライアントの設定

次の設定手順は一度限り必要です。EJB およびクライアントアプリケーションを配置する際に繰り返す必要はありません。

サーバの設定

iPlanet Application Server をインストールする時点で、IIOP ブリッジプロセスが設定されていない場合は、iPlanet Application Server Administrative Tool を起動して、IIOP ブリッジプロセスをアプリケーションサーバ環境に追加する必要があります。

1. iPlanet Application Server 管理ツールを起動します。

   UNIX の場合

   install_dir/ias/bin/ksvradmin

   Windows の場合

   「スタート」>「プログラム」>「iPlanet Application Server」>「iAS Administration Tool」を選択します。

2. アプリケーションサーバインスタンスに接続し、サーバ名アイコンをダブルクリックして、アプリケーションサーバのインスタンスに定義されているプロセスを一覧表示します。少なくとも kjs プロセスが 1 つ、および kxs プロセスが 1 つ表示されます。EJB への RMI/IIOP アクセスには、kxs プロセスは不要です。cxs プロセスが表示された場合は、アプリケーションサーバインスタンスにすでに IIOP ブリッジプロセスが定義されています。この場合は、cxs プロセスエントリをダブルクリックし、IIOP ポート番号を書き留め、次の節に進んでください。ブリッジプロセスが表示されない場合は、次の手順に進んでプロセスを定義してください。
3. 任意の既存プロセスエントリを選択し、「ファイル」>「新規」>「プロセス」を選択します。
4. プロセスタイプのプルダウンリストから cxs を選択し、kjs および kxs プロセスによってすでに使われているほかのポート番号と競合しないポート番号 (ポート 10822 など) を入力します。システム環境でほかに割り当てられているポートと競合しないかぎり、デフォルトの IIOP ポート番号 (9010) を選択します。「OK」をクリックしてプロセスをインスタンス化します。
5. 数秒後、アプリケーションサーバ環境で IIOP ブリッジプロセスが動作している状態が表示されます。このプロセスは、Administrative Tool に一覧表示されているその他のすべてのアプリケーションサーバプロセスとともに、アプリケーションサーバの再起動時に自動的に開始されます。
6. UNIX の場合、コマンドラインから IIOP ブリッジプロセスの存在も確認できます。たとえば、次のコマンドを入力します (各コマンドはすべて 1 行で指定する)。

```
ps -ef | grep iiop
```

```
root 1153 1 0 17:00:15 ?0:00 /bin/sh /usr/iPlanet/ias6/ias/bin/kjs -cset CCS0 -eng
3 -iiop -DORBinsPort=9010
```

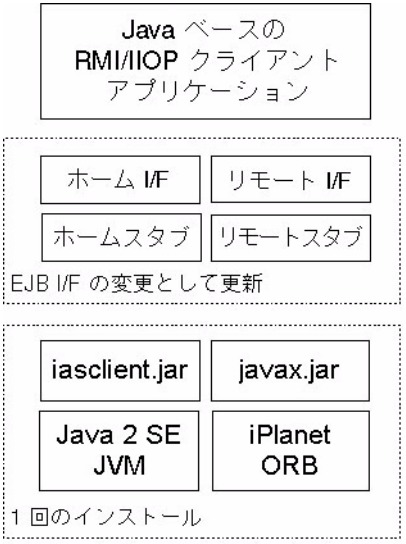
この出力には、-iiop オプションで開始された iPlanet Java エンジンプロセスが表示されます。このオプションは、Java エンジンのこのインスタンスに、J2EE Web および EJB コンテナプロセスではなく、IIOP ブリッジプロセスとして開始するように指示します。

cxs プロセスをインスタンス化すると、RMI/IIOP サポートのサーバサイドの設定が完了します。

クライアントの設定

iPlanet に配置されている EJB に Java アプリケーションクライアントがアクセスできるようにするには、図 10-4 のように、適切な Java 2 環境、iPlanet ORB、およびいくつかの JAR ファイルがクライアントシステム上で使用可能になっている必要があります。

図 10-4　クライアントの設定



手順については次の節で説明します。

- Java 2 環境と iPlanet ORB の設定
- RMI/IIOP クライアントサポートクラスのインストール

Java 2 環境と iPlanet ORB の設定

RMI/IIOP 経由でリモート EJB への通信をサポートするには、Java 2 環境と iPlanet ORB がクライアント上に存在する必要があります。iPlanet Application Server の一部としてバンドルされている Java 2 環境、または、259 ページの「既存の JDK の使用法」の節で説明しているテスト済みの改良型環境のどれかをクライアント上で使う必要があります。ほかの Java 2 環境も正しく動作する場合はありますが、これらの環境は iPlanet ではサポートされていません。

バンドルされた JDK の使用法

iPlanet は、そのほとんどの RMI/IIOP テストを、クライアントサイドの RMI/IIOP ベースアプリケーション向けに推奨されている Java 2 プラットフォーム上で行っています。したがって、Java 2 環境をアプリケーションサーバの一部としてバンドルしています。クライアントサイドでこの JVM を使うには、Java 2 環境を iPlanet インス

トールからクライアント環境にコピーし、該当する java 実行可能ファイルを含めるように PATH を適切に設定します。バンドルされている Java 2 環境には iPlanet ORB が含まれているため、クライアントサイドにコピーしたあとに Java 2 環境を変更する必要はありません。

バンドルされている Java 2 プラットフォームは、アプリケーションサーバをインストールした次の場所にあります。

install_dir/ias/usr/java/

サーバの JVM 環境をクライアントにコピーするには、次の手順を実行します。

1. *install_dir*/ias/usr/ に移動します。
2. java/ ディレクトリ全体をクライアント環境にコピーします。java/ ディレクトリを zip または tar 形式で圧縮し、そのアーカイブファイルをクライアントシステムに転送して選択したディレクトリ内で展開できます。
3. クライアントの PATH を、*client_side_JVM_directory*/java/bin を含めるように設定します。
4. java -fullversion を実行して、適切な JDK (1.3.1) が使われていることを確認します。UNIX の場合は、which java を実行してこの動作を確認します。

これで、バンドルされている JDK が iPlanet ORB とともにインストールされました。次に、クライアント環境にいくつかの JAR サポートファイルをインストールする必要があります。262 ページの「RMI/IIOP クライアントサポートクラスのインストール」に進み、これらの JAR ファイルをインストールします。

既存の JDK の使用法

Java 2 環境のいくつかの配布版で基本的なテストを行った結果、簡単なセットアップ手順で、既存の Java 2 環境を活用しながら、iPlanet Application Server に配置されている EJB に RMI/IIOP クライアントがアクセスできることが証明されました。この場合は、iPlanet Application Server 環境からクライアントシステムの既存の JVM に iPlanet ORB ファイルをコピーする必要があります。

iPlanet Application Server では、オペレーティングシステムと Java 2 プラットフォームの次の組み合わせがテストされています。

- Solaris と JDK 1.3.1
- Linux と Java 1.3.1
- Windows 98、NT、または 2000 と Java 1.3.1

オペレーティングシステムと Java 2 プラットフォームのほかの組み合わせでも、RMI/IIOP および iPlanet Application Server が正しく連動する場合もありますが、テストは実施されていません。どのような組み合わせを選択しても、設定をテストしてから実際の運用に適した選択を行ってください。

Solaris と JDK 1.3.1

このシナリオでは、Solaris システム上にすでに Java 2 1.2 環境がインストールされており、この JVM を RMI/IIOP クライアントのプラットフォームとして使うことを前提としています。

次の手順では、`JAVA_HOME` は JDK 1.3.1 配布版をインストールしたディレクトリです。次に例を示します。

```
export JAVA_HOME=/usr/java1.3
```

1. iPlanet Application Server の Solaris をインストールしたディレクトリから Solaris クライアントシステムに j2eeorb.jar をコピーします。

   j2eeorb.jar ファイルをコピーします。

   install_dir`/ias/usr/java/jre/lib/ext/j2eeorb.jar`

   Solaris クライアントの JDK をインストールしたディレクトリにコピーします。

   `$JAVA_HOME/jre/lib/ext`

   (削除されるようにするには) 必ず、共有オブジェクトファイルが含まれている `sparc/` ディレクトリをこの手順でコピーします。この手順では、iPlanet ORB、固有の直列化ファイル、およびほかのサポートファイルがクライアントにコピーされます。

2. `orb.properties` ファイルを、iPlanet をインストールした次のディレクトリから

   install_dir`/ias/usr/java/jre/lib/orb.properties`

   クライアントの JDK をインストールした次のディレクトリにコピーします。

   `$JAVA_HOME/jre/lib/`

3. クライアントアプリケーションが DLL にアクセスできるように PATH を設定します。

   (削除対象) `export PATH=$JAVA_HOME/bin:$JAVA_HOME/jre/lib/ext/i386:$PATH`

これで、iPlanet ORB を使えるように既存の JDK が設定されました。次は、クライアント環境にいくつかの JAR サポートファイルをインストールする必要があります。262 ページの「RMI/IIOP クライアントサポートクラスのインストール」に進み、これらの JAR ファイルをインストールします。

Linux と Java 1.3.1

このシナリオでは、Linux システム上にすでに Java 1.3.1 環境がインストールされており、この JVM を RMI/IIOP クライアントのプラットフォームとして使うことを前提としています。次の方法は、RedHat 6.2 でテスト済みです。

注 JDK 1.2 と JDK 1.3 の固有の直列化ライブラリに互換性がないため、クライアントサイドで JDK 1.3 を使う場合は、クライアントとサーバ間で基本データタイプの値のみが交換できます。

1. iPlanet ORB を保持するディレクトリをクライアント上に作成します。次のようにします。

   ```
   mkdir -p /opt/iplanet/orb
   ```

2. Linux システムにインストールされた iPlanet Application Server から、クライアントシステム上の適切なディレクトリ、たとえば、/opt/iplanet/orb/ に次の JAR ファイルをコピーします。

   install_dir/ias/usr/java/jre/lib/ext/j2eeorb.jar

3. 環境を設定します。次のようにします。

   ```
   JAVA_HOME=/opt/jdk1.3

   PATH=:$JAVA_HOME/bin:$JAVA_HOME/jre/lib/i386:$PATH

   CLASSPATH=/opt/iplanet/orb/j2eeorb.jar

   LD_LIBRARY_PATH=$JAVA_HOME/jre/lib:$JAVA_HOME/jre/lib/i386

   export JAVA_HOME PATH CLASSPATH LD_LIBRARY_PATH
   ```

4. iPlanet ORB クラスは、コマンドライン (このコマンドはすべて 1 行で指定する) でプロパティとして指定できます。

```
java -Dorg.omg.CORBA.ORBClass=com.netscape.ejb.client.ClientORB
-Dorg.omg.CORBA.ORBSingletonClass=com.sun.corba.ee.internal.corba.ORBSingleton
j2eeguide.converter.ConverterClient ias_host 9010
```

これで、iPlanet ORB を使えるように既存の JDK が設定されました。次は、クライアント環境にいくつかの JAR サポートファイルをインストールする必要があります。262 ページの「RMI/IIOP クライアントサポートクラスのインストール」に進み、これらの JAR ファイルをインストールします。

Windows 98、NT、または 2000 と Java 1.3.1

このシナリオでは、Windows システム上にすでに Java 1.3.1 環境がインストールされており、この JVM を RMI/IIOP クライアントのプラットフォームとして使うことを前提としています。

次の手順では、JAVA_HOME は JDK 1.3.1 配布版をインストールしたディレクトリです。次に例を示します。

```
set JAVA_HOME=c:¥JDK1.3.1
```

1. iPlanet Application Server の Windows をインストールしたディレクトリから Windows クライアントシステムに `j2eeorb.jar` をコピーします。

   `j2eeorb.jar` ファイルを、次のディレクトリから

   install_dir¥ias¥usr¥java¥jre¥lib¥ext¥j2eeorb.jar

   クライアントの JDK をインストールした次のディレクトリにコピーします。

   `%JAVA_HOME%¥jre¥lib¥ext`

2. `orb.properties` ファイルを、iPlanet をインストールした次のディレクトリから

   install_dir¥ias¥usr¥java¥jre¥lib¥orb.properties

   クライアントの JDK をインストールした次のディレクトリにコピーします。

   `%JAVA_HOME%¥jre¥lib¥`

3. 固有の直列化 DLL を、iPlanet をインストールした次のディレクトリから

   install_dir¥ias¥usr¥java¥jre¥bin¥ioser12.dll

   クライアントの JDK をインストールした次のディレクトリにコピーします。

   `%JAVA_HOME%¥jre¥bin¥`

4. クライアントアプリケーションが DLL にアクセスできるように PATH を設定します。

   ```
   set PATH=%JAVA_HOME%¥bin;%PATH%
   ```

これで、iPlanet ORB を使えるように既存の JDK が設定されました。次は、クライアント環境にいくつかの JAR サポートファイルをインストールする必要があります。262 ページの「RMI/IIOP クライアントサポートクラスのインストール」に進み、これらの JAR ファイルをインストールします。

RMI/IIOP クライアントサポートクラスのインストール

クライアントサイドで Java 2 プラットフォームが使われていても、クライアントの CLASSPATH にはファイル `iasclient.jar` を含める必要があります。このファイルは、iPlanet のクライアントの認証機能をサポートするいくつかのセキュリティ関連クラスを含んでいる iPlanet 固有の JAR ファイルです。ACC を使う場合は、`iasclient.jar` は、`iasacc.jar` に置き換わります。標準 `javax.jar` ファイルも、クライアントの CLASSPATH に含める必要があります。このファイルには、ネーミングサービスおよびその他の Java エクステンションの標準 Java インタフェースが含まれています。

これらの JAR ファイルは、iPlanet をインストールしたディレクトリからクライアント環境にコピーし、クライアントの CLASSPATH に追加できます。UNIX の場合、これらのファイルは、iPlanet Application Server をインストールしたディレクトリである次の場所にあります。

install_dir/ias/classes/java/iasclient.jar

install_dir/ias/classes/java/javax.jar

Windows の場合、これらのファイルは、iPlanet Application Server をインストールしたディレクトリである次の場所にあります。

install_dir/ias/classes/java/iasclient.jar

install_dir/ias/lib/java/javax.jar

これらのサポートファイルをクライアント環境にコピーしたあと、JAR ファイルを含めるようにクライアントの CLASSPATH を設定する必要があります。

同じシステム上の EJB への RMI/IIOP クライアントアクセス

アプリケーションサーバと同じマシン上にあるクライアントを使って RMI/IIOP クライアントアクセスを行う場合は、PATH および CLASSPATH 変数のセットアップ手順を短縮できます。既存のインストール済み *javax.jar*、iasclient.jar のコピーと *install_dir*usr/java/bin/ 内の JVM を参照するだけです。たとえば、ローカルで RMI/IIOP アクセスをテストするには、次のように CLASSPATH 変数を設定します。

Windows の場合

```
set CLASSPATH=d:¥iplanet¥ias6¥ias¥lib¥java¥javax.jar;
d:¥iplanet¥ias6¥ias¥classes¥java¥iasclient.jar;%CLASSPATH%
```

(Windows システムの PATH 環境変数には、バンドルされている JDK の *install_dir*/usr/java/bin/ がすでに含まれているため、Windows 上で再び設定する必要はありません。)

Windows システムの CLASSPATH は、手動で変数を設定する必要がないように設定できます。

UNIX の場合

```
export
CLASSPATH=/usr/iplanet/ias6/ias/classes/java/javax.jar:/usr/iplanet
/ias6/ias/classes/java/iasclient.jar:$CLASSPATH
```

UNIX の場合、バンドルされている JDK ディレクトリも含めるように PATH を変更する必要があります。

```
export PATH=/usr/iplanet/ias6/ias/usr/java/bin:$PATH
```

リモートシステムからの EJB への RMI/IIOP クライアントアクセス

リモートクライアントシステムを使う場合は、次の手順を実行して適切な PATH および CLASSPATH を設定します。

UNIX の場合

適切な Java 2 bin/ ディレクトリを含めるように PATH 環境変数を設定します。

```
export PATH=Java2_install_dir/usr/java/bin:$PATH
```

標準 Java 拡張機能クラスと iPlanet RMI/IIOP クライアントサポート JAR を含めるように CLASSPATH を設定します。

```
export
CLASSPATH=/opt/rmi-client/iasclient.jar:/opt/rmi-client/javax.jar:$
CLASSPATH
```

正しく設定されているかどうか CLASSPATH を再び確認します。CLASSPATH は、次に示す設定と異なる場合があります。

```
echo $CLASSPATH
```

```
/opt/rmi-client/iasclient.jar:/opt/rmi-client/javax.jar:
```

Windows の場合

適切な JDK の bin/ ディレクトリを含めるように PATH 環境変数を設定します。

```
set PATH=Java2_install_dir¥usr¥java¥bin;%PATH%
```

標準 Java 拡張機能クラス (javax.jar) および iPlanet クライアントサポート JAR (iasclient.jar) を含めるように CLASSPATH を設定します。

```
set
CLASSPATH=d:¥rmi-client¥javax.jar;d:¥rmi-client¥iasclient.jar;%CLASS
PATH%
```

RMI/IIOP クライアントアプリケーションの配置

クライアントアプリケーションを開発する場合、開発環境からクライアントシステムに多数のファイルを配置する必要があります。この節の次の項で、RMI/IIOP 対応クライアントアプリケーションの配置に必要な基本手順について説明します。

- クライアントの配置
- 配置ツール
- サーバの CLASSPATH の設定 (SP2 以前)

クライアントの配置

クライアントアプリケーションクラスがクライアントシステム上で使用可能になっていることを確認するだけでなく、EJB 固有のホームおよびリモートインタフェースとそれらの対応スタブがクライアントシステムに配置されていることを確認する必要があります。たとえば、Converter サンプルアプリケーションでは、次のクラスをクライアントシステムにコピーする必要があります。

ホームおよびリモートインタフェースクラス

`ConverterHome.class`

`Converter.class`

EJB 固有の iPlanet クライアントスタブ

`_Converter_Stub.class`

`_ConverterHome_Stub.class`

配置ツール

配置ツールは、ホームおよびリモートインタフェースと RMI/IIOP スタブクラスだけが含まれている JAR ファイルを生成します。現時点では、その他のクライアントアプリケーションクラスおよびリソースはパッケージされません。

Java ベースの Ant ビルド機能を使うと、クライアントアプリケーションの組み立てを簡単に自動化できます。Ant を使ってクライアントアプリケーションのパッケージおよび配置を行う場合の例については、RMI/IIOP サンプルアプリケーションを参照してください。

サーバの CLASSPATH の設定 (SP2 以前)

この節の記述は、iPlanet Application Server 6.0 Service Pack 2 (SP2) 以前に適用されます。Service Pack 3 以降には、次の設定手順は不要です。SP 3 以降を使う場合は、270 ページの「RMI/IIOP クライアントアプリケーションの実行」に進んでください。

iPlanet Application Server Service Pack 2 以前で EJB クラスを読み込むには、IIOP ブリッジプロセスが、アプリケーションサーバの CLASSPATH を使って EJB スタブとホーム / リモートインタフェースにアクセスできるようになっている必要があります。RMI/IIOP ベースの Java アプリケーションクライアントを SP 2 以前で初めて実行する場合は、まずアプリケーションサーバの CLASSPATH を変更する必要があります。

iPlanet Application Server 6.0 SP2 以降、EJB ベースアプリケーションを登録すると EJB JAR ファイルがアプリケーションサーバの配置ディレクトリに展開されるようになりました。デフォルトでは、`j2eeguide-converter.ear` などの J2EE アプリケーションをアプリケーションサーバに配置すると、`j2eeguideEjb.jar` などの組み込み EJB JAR ファイルが次のディレクトリに展開されます。

install_dir`/ias/APPS/j2eeguide-converter/j2eeguide-converterEjb/`

スタンドアロン EJB JAR モジュール (または WAR モジュール) を iPlanet Application Server に配置する場合、このスタンドアロンモジュールのデフォルトの展開場所は次のディレクトリになります。

install_dir`/ias/APPS/modules/j2eeguide-converterEjb/`

RMI/IIOP クライアントを実行する前に、適切なモジュールディレクトリをアプリケーションサーバの CLASSPATH に追加する必要があります。

ORBIX 用に RMI/IIOP アプリケーションを設定する

『管理者ガイド』で説明しているように、ORBIX 2000 をインストールして iPlanet Application Server と統合すれば、使用する ORB (組み込みまたは ORBIX) を RMI/IIOP クライアントアプリケーションごとに選択することができます。この節では、ORBIX を使用するために、RMI/IIOP クライアントを設定する方法について説明します。

ORBIX とクライアントサイドのロードバランスについては、252 ページの「クライアントサイドのロードバランスおよびフェールオーバー」を参照してください。

参照資料

- ORBIX 2000 Programmer's Guide, Java Edition

  `http://www.iona.com/docs/orbix2000/1.2.1/pguide_java/html/index.html`

- ORBIX 2000 1.2.1 Documentation

  `http://www.iona.com/docs/orbix2000/1.2.1/index.html`

- OpenORB RMI over IIOP

  `http://www.openorb.org/rmioveriiop.html`

- Java 2 SDK 1.4 のダウンロード

  `http://java.sun.com/j2se/1.4/index.html`

設定の手順

設定の手順は、ファイルパスの形式を除いて、UNIX と Windows で違いはありません。次の例では、UNIX のファイルパスを使用しています。

1. ORBIX 2000 バージョン 1.2.1 をインストールします。ライセンスファイルを `/etc/opt/iona/` にコピーし、ライセンスをインストールします。『管理者ガイド』および ORBIX のマニュアルを参照してください。
2. ORBIX ドメインコンフィグレーションファイルを、サーバ上の ORBIX のインストール先 (ネーミングサービスを実行している場所) から別のディレクトリ (たとえば、次のディレクトリ) にコピーします。

/etc/opt/iona/domains

このファイルには、サーバ上の ORBIX にアクセスするために必要な設定情報が含まれています。クライアントプログラムは、このファイルを使ってネーミングサービスに接続します。

クライアントとサーバが同じマシン上で動作している場合、このファイルは必要ありません。その場合は、localhost 設定 (デフォルト) が使われます。

3. 事前にパッケージされているサンプルアプリケーションのいずれかをインストールする場合は、ejbc ユーティリティを使用して、スタブとスケルトンを再生成します。ejbc については、130 ページの「ejbc コンパイラの使用法」を参照してください。
4. クライアントアプリケーションをインストールします。
5. アプリケーションに関連付けられたスタブ (*Stub*.class) をクライアントディレクトリにコピーします。たとえば、次の方法で実行します (改行しない)。

```
cp
ias_inst_dir/ias/APPS/j2eeguide-converter/j2eeguide-converterEjb/j2eeguide/converter/
_Converter_Stub.class client_inst_dir/j2eeguide/converter
```

次の処理も実行します (改行しない)。

```
cp
ias_inst_dir/ias/APPS/j2eeguide-converter/j2eeguide-converterEjb/j2eeguide/converter/
_ConverterHome_Stub.class client_inst_dir/j2eeguide/converter
```

6. クラスパスを設定します。たとえば、次の方法で実行します (改行しない)。

```
export
CLASSPATH=orbix_inst_dir/orbix_art/1.2/classes/orbix2000.jar:orbix_inst_dir/orbix_art/1.2
/classes/omg.jar:orbix_license_file_path/licenses.txt:server_orbix_config_file_path:
ias_inst_dir/classes/java/javax.jar:ias_inst_dir/ias/classes/java/iasclient.jar
```

iPlanet Application Server がクライアントマシンにインストールされていない場合は、任意の iPlanet Application Server (バージョン 6.5) の javax.jar ファイルをクライアントマシンにコピーして、クラスパスに設定します。

7. ORBdomain_name プロパティをサーバの ORBIX コンフィグレーションファイルのドメイン名に設定します。このファイルについては、ORBIX のマニュアルを参照してください。この設定は、次の 2 つの方法で行うことができます。
   - クライアントを実行するときに、Java コマンドラインで ORBdomain_name プロパティを設定する

     この場合、*jdk_inst_dir*/jre/lib/orb.properties ファイル内の org.omg.CORBA.ORBClass プロパティは com.iplanet.ias.iona.clientorb.IONAorb に設定する必要がある。さらに、iasclient.jar ファイルを手順 6 に示してあるクラスパスに含めなければならない

- クライアントプログラムで、ORBdomain_name プロパティを文字列の配列として ORB.init 呼び出しの最初のパラメータに渡す。これにより、ORB が初期化される。初期化された ORB は、JNDI 呼び出しに渡すことができる。つまり、後続の CORBA 初期化呼び出しは、その ORB 上で呼び出すことができる。

  この場合、*jdk_inst_dir*/jre/lib/orb.properties ファイル内の org.omg.CORBA.ORBClass プロパティは com.iona.corba.art.artimpl.ORBImpl に設定する必要があります。

初期化された ORB を JNDI 呼び出しに渡す方法については、JDK のマニュアルで JNDI に関する説明を参照してください。

IONAorb クラスを使うこともできます。IONAorb は com.iona.corba.art.artimpl.ORBImpl (ORBIX ORB クラス) から派生し、set_parameters メソッドをオーバーロードします。set_parameters メソッドでは、ORBname プロパティおよび ORBdomain_name プロパティが読み込まれ、ORB.init 呼び出しに渡されます。

8. orb. ファイル内の次のプロパティを変更します。ファイルは、*jdk_inst_dir*/jre/lib にあります。

```
org.omg.CORBA.ORBSingletonClass=com.iona.corba.art.artimpl.ORBSingleton
```

9. クライアントを実行します。

注

組み込み ORB は、IIOP URL (ホスト名とポート番号を含む) を使って iPlanet Application Server に接続します。ORBIX ORB は、CXS への接続にこの URL を使いません。

組み込み ORB が使われている場合、ネーミングサービスは CXS への in-process です。ORBIX ORB が使われている場合、ネーミングサービスは out-of-process です。

ORBIX Java クライアントが java.util.vector などの Java コンテナ内部に埋め込まれたユーザ定義の直列化可能なオブジェクトを渡そうとすると、MARSHALLING 例外が発生します。

セキュリティの有効化

セキュリティを有効にするには、クライアントを起動する前に次の手順を実行します。

1. 次のプロパティに値が設定されていないことを確認します。等号の右辺には、空白にしてください。

```
org.omg.PortableInterceptor.ORBInitializerClass.com.iplanet.ias.iona.
orbinitializers.IONAClientORBInitializerImpl=
```

2. ORBname プロパティを、client_interceptor 名が登録されている範囲に含まれる orbname に設定します。この設定は、次の 2 つの方法で行うことができます。

- Java コマンドラインで ORBname プロパティを設定する

  この場合、org.omg.CORBA.ORBClass プロパティは、com.iplanet.ias.iona.clientorb.IONAorb に設定しなければならない。

- クライアントプログラムで、ORBname プロパティを文字列の配列として ORB.init 呼び出しの最初のパラメータに渡す。これにより、ORB が初期化される。初期化された ORB は、JNDI 呼び出しに渡すことができる。つまり、後続の CORBA 初期化呼び出しは、その ORB 上で呼び出すことができる。

  この場合、org.omg.CORBA.ORBClass プロパティは、com.iona.corba.art.artimpl.ORBImpl に設定しなければならない。

初期化された ORB を JNDI 呼び出しに渡す方法については、JDK のマニュアルで JNDI に関する説明を参照してください。

IONAorb クラスを使うこともできます。IONAorb は com.iona.corba.art.artimpl.ORBImpl (ORBIX ORB クラス) から派生し、set_parameters メソッドをオーバーロードします。set_parameters メソッドでは、ORBname プロパティが読み込まれ、ORB.init 呼び出しに渡されます。

3. クライアントのインターセプタ名 iASClientInterceptor を、ORBIX のコンフィグレーションファイルの client_binding_list に適切な範囲で追加します。追加したインターセプタ名の範囲がグローバルでない場合は、プロパティ ORBname を値 orbname に設定します。詳細は、ORBIX のマニュアルを参照してください。次のようにします。

```
binding:client_binding_list = ["OTS+POA_Coloc", "POA_Coloc",
"OTS+TLS_Coloc+POA_Coloc", "TLS_Coloc+POA_Coloc",
"iASClientInterceptor+GIOP+IIOP", "OTS+GIOP+IIOP", "GIOP+IIOP",
"OTS+GIOP+IIOP_TLS", "GIOP+IIOP_TLS"];
```

注 "iASClientInterceptor+GIOP+IIOP" エントリは、"OTS+GIOP+IIOP" エントリの前に指定する必要があります。

4. portable_interceptor を orb_plugins リストに適切な範囲で追加します。

   追加する前は、次のようになっています。

   ```
   orb_plugins=["iiop_profile","giop" ,"iiop", "ots"]
   ```

   追加した後は、次のようになります。

```
orb_plugins=["iiop_profile","giop" ,"iiop", "ots", "portable_interceptor"]
```

   インターセプタ名と portable_interceptor は、同じ範囲で追加する必要があります。

5. セキュリティを有効にするには、com.netscape.ejb.client.IUserPrincipal インタフェースを実装して、クラスファイル名をプロパティ com.netscape.ejb.client.PrincipalClass に指定する必要があります。

注　　org.omg.CORBA.ORBClass プロパティが com.iona.corba.art.artimpl.ORBImpl に設定されている場合は、システムプロパティに com.netscape.ejb.client.PrincipalClass プロパティを設定する必要があります。このプロパティは、プログラムから渡すことはできません。

RMI/IIOP クライアントアプリケーションの実行

クライアントが Java main プログラムであり、クライアント環境が正しく設定され、互換性のある JVM を使う限り、main クラスを実行するだけで十分です。IIOP URL コンポーネント (ホストおよびポート番号) をコマンドラインで渡すか、またはプロパティファイルからこの情報を取得するかによって、main プログラムの実行方法は異なります。たとえば、ConverterClient サンプルは次の方法で実行します。

```
java j2eeguide.converter.ConverterClient host_name port
```

host_name は、IIOP ブリッジが指定された *port* でリッスンするときのホストの名前です。

RMI/IIOP クライアントアプリケーションのトラブルシューティング

RMI/IIOP クライアントを実行すると、クライアント上でエラーが発生する場合があります。IIOP ブリッジログを表示するには、277 ページの「RMI/IIOP ログメッセージの表示」を参照してください。表 10-1 には、共通する RMI/IIOP 設定上の問題の一般的な症状および修正方法が一覧表示されています。

負荷状態の RMI/IIOP クライアントアプリケーションを実行中に問題が発生した場合は、負荷関連の問題のトラブルシューティング方法を調べて、273 ページの「パフォーマンス問題の認識」を参照してください。

表 10-1 トラブルシューティング

症状	問題の原因	対処法
クライアントが JNDI 検索時に次の例外をスローする `org.omg.CORBA.INITIALIZE:can't instantiate default ORB implementation`	クライアントの CLASSPATH に `iasclient.jar` ファイルが含まれていない クライアントの PATH が適切な Java コマンドを取得しない。アプリケーションサーバにバンドルされた JVM、または適切な既存の JVM を使う	クライアントの設定手順が正しく行われていることを確認する。257 ページの「クライアントの設定」を参照
クライアントに CORBA 通信失敗例外が発生する `javax.naming.CommunicationException :Cannot connect to ORB.Root exception is org.omg.CORBA.COMM_FAILURE:`	次のどれかの原因による IIOP ブリッジへのコネクションの失敗 • IIOP ホストまたはポート番号が不正である • IIOP ブリッジプロセスが開始されていない	IIOP ブリッジプロセスが設定され、開始されていることを確認する。256 ページの「サーバの設定」を参照
	• IIOP ブリッジプロセスは開始されているが、初期化が行われていない • クライアントマシンがネットワークにアクセスできない • ファイヤウォールルールにより、アプリケーションサーバシステムにアクセスできない	クライアントマシン上にネットワークアクセスが設定され、中間ファイヤウォールによってアクセスが阻止されていないことを確認する

表 10-1　トラブルシューティング (続き)

症状	問題の原因	対処法
a) クライアントがハングしたようになり、メモリ不足例外が発生する `Exception in thread "main" java.lang.OutOfMemoryError.`	クライアントアプリケーションで指定されている JNDI 名が正しくない または	クライアントが使う JNDI 名を修正する または
b) IIOP ブリッジが次のどれかの例外を繰り返しスローする **名前が見つからない** `[01/May/2001 08:20:14:4] info:GDS-007:finished a registry load [01/May/2001 08:20:14:6] info:PROT-006:new connection established SendRemoteReq status=0x0 javax.naming.NameNotFoundException: EjbContext:exception on getHome(), com.nets cape.server.eb.UncheckedException:u nchecked exception thrown by impl com.kivasoft.eb.boot.EBBootstrapImp l@1fca24a; nested exception is:` **クラスが見つからない** `[24/Jan/2001 12:25:52:9] error:EBFP-unserialize:error during unserialization of method, exception = java.lang.ClassNotFoundException:j2 eeguide.confirmer.ejb_stub_Confirme rHome java.lang.ClassNotFoundException:j2 eeguide.confirmer.ejb_stub_Confirme rHome at java.lang.Throwable.fillInStackTrac e(Native Method)` **クラスタイプ変換の例外**	(SP3 以前) 展開された EJB JAR ディレクトリがサーバの CLASSPATH に追加されていない。または、EJB JAR ディレクトリを CLASSPATH に追加してからサーバを再起動していない	アプリケーションサーバの CLASSPATH を設定する
クライアントアプリケーションでネーミングの通信例外が発生する `javax.naming.CommunicationException`	アプリケーションサーバに関連付けられている Directory Server が動作しない	Directory Server を起動する

RMI/IIOP のパフォーマンスチューニング

RMI/IIOP パスで多数の同時ユーザに対応しなければならない配置環境の場合は、この節で説明するチューニングガイドラインを試してください。RMI/IIOP を使う場合、JVM のデフォルト設定とその基本 OS だけでは最適なパフォーマンスおよび容量を達成できません。

この節には次のトピックがあります。

- 負荷テストの方法
- パフォーマンス問題の認識
- 基本的なチューニング方法
- スケーラビリティの向上

負荷テストの方法

RMI/IIOP 用の負荷テストツールはわずかしかないため、基本的な負荷テスト用の比較的単純なドライバを独自に作成することができます。次の Java メインプログラムは、Converter EJB の単純な負荷テストクライアントの例を示しています。

サンプルコード . . .

パフォーマンス問題の認識

RMI/IIOP クライアントアプリケーションを負荷状態で実行する前に、基本構造のテストが成功していることを確認してください。

負荷状態のクライアントアプリケーションの実行を開始する際、RMI/IIOP クライアントに次の例外が発生する場合があります。

```
org.omg.CORBA.COMM_FAILURE

java.lang.OutOfMemoryError

java.rmi.UnmarshalException
```

アプリケーションの基本構造は正しく動作することが確認されている場合に、アプリケーションの負荷テスト時にこれらの例外が発生した場合は、次の節で説明する RMI/IIOP 環境のチューニング方法を参照してください。

基本的なチューニング方法

次に説明するチューニング方法を試し、ユーザの環境に最適なバランスを見つけてください。

Solaris ファイル記述子の設定

Solaris の場合、`ulimit` を使って、開いているファイル数のプロパティを最大に設定すると、サポートできる RMI/IIOP クライアントの最大数に影響を与えます。このプロパティのデフォルト値は、Solaris 2.6 または Solaris 8 のどちらを実行しているかによって、`64` または `1024` となります。数を増やすには、次のコマンドを `/etc/system` に追加し、再起動します。

```
set rlim_fd_max = 8192
```

次のコマンドを使うと、この使用制限を確認できます。

```
ulimit -a -H
```

上記の使用制限を設定後、次のコマンドを使うと、このプロパティの値をこの制限まで明示的に増やすことができます。

```
ulimit -n 8192
```

次のコマンドを使うと、この制限を確認できます。

```
ulimit -a
```

たとえば、ulimit がデフォルトの 64 の場合、1 つのテストドライバがサポートできる同時クライアントは 25 ですが、`ulimit` を `8192` に設定すると、同じテストドライバで 120 の同時クライアントをサポートできます。このテストドライバでは複数のスレッドが生成されました。これらの各スレッドは JNDI 検索を実行し、ビジネスメソッド呼び出し間の思考 (遅延) 時間が 500 ミリ秒で同じビジネスメソッドを繰り返し呼び出し、約 100 KB のデータを送受信できました。

これらの設定値は RMI/IIOP クライアント (Solaris)、および Solaris システムにインストールされた IIOP ブリッジに適用されます。ファイル記述子の制限の設定については、Solaris のマニュアルを参照してください。

Java ヒープ設定値

ファイル記述子の容量をチューニングするだけでなく、クライアントおよびブリッジ JVM の両方について異なるヒープ値も設定できます。デフォルトのヒープサイズの変更については、JDK 1.3.1 のマニュアルを参照してください。

スケーラビリティの向上

1 つのブリッジプロセスおよびクライアントシステムの容量をチューニングするだけでなく、複数の IIOP ブリッジプロセスを使うことによって、RMI/IIOP 環境のスケーラビリティを向上させることができます。同じアプリケーションサーバインスタンス上で複数のブリッジプロセスを設定すると、アプリケーション配置のスケーラビリティが向上します。場合によっては、それぞれ 1 つまたは複数のブリッジプロセスを使って設定した多数のアプリケーションサーバインスタンスを使うこともできます。

複数のブリッジプロセスがアクティブな設定では、一連のクライアントをさまざまなブリッジにスタティックにマッピングするか、またはクライアントサイドに独自のロジックを実装して既知のブリッジプロセスと照らし合わせてロードバランスを行うことによって、クライアント負荷を分割できます。

RMI/IIOP のファイヤウォールの設定

RMI/IIOP クライアントがファイヤウォールを通過して iPlanet Application Server と通信する場合は、IIOP ブリッジプロセスが使うクライアントシステムから IIOP ポートへのアクセスを有効にする必要があります。クライアントのポート番号はダイナミックに割り当てられるため、RMI/IIOP トラフィックがクライアントシステムからファイヤウォールを通過してアプリケーションサーバのインスタンスに達するように、ソースポートの範囲を広げ、1 つのデスティネーションポートを開く必要があります。

さらに、Converter サンプルアプリケーションを一度実行すると、2 つのシステム間の IIOP トラフィックが巡回ベースで追跡されます。ホスト `swatch` は RMI/IIOP クライアントで、ホスト `mamba` は、デスティネーションシステムまたはアプリケーションサーバシステムです。IIOP ブリッジプロセスに割り当てられたポート番号は `9010` です。ダイナミックに割り当てられた 2 つのポート (`33046` および `33048`) は RMI/IIOP クライアントで使われます。一方、ブリッジプロセスとの通信にはポート 9010 が使われます。

```
swatch -> mamba.red.iplanet.com TCP D=9010 S=33046 Syn Seq=140303570 Len=0
Win=24820
Options=<nop,nop,sackOK,mss 1460>
mamba.red.iplanet.com -> swatch TCP D=33046 S=9010 Syn Ack=140303571
Seq=1229729413 Len=0 Win=8760
Options=<mss 1460>
swatch -> mamba.red.iplanet.com TCP D=9010 S=33046 Ack=1229729414 Seq=140303571
Len=0 Win=24820
swatch -> mamba.red.iplanet.com TCP D=9010 S=33046 Ack=1229729414 Seq=140303571
Len=236 Win=24820
mamba.red.iplanet.com -> swatch TCP D=33046 S=9010 Ack=140303807 Seq=1229729414
Len=168 Win=8524
swatch -> mamba.red.iplanet.com TCP D=9010 S=33046 Ack=1229729582 Seq=140303807
Len=0 Win=24820
swatch -> mamba.red.iplanet.com TCP D=9010 S=33048 Syn Seq=140990388 Len=0
Win=24820
Options=<nop,nop,sackOK,mss 1460>
mamba.red.iplanet.com -> swatch TCP D=33048 S=9010 Syn Ack=140990389
Seq=1229731472 Len=0 Win=8760
Options=<mss 1460>
swatch -> mamba.red.iplanet.com TCP D=9010 S=33048 Ack=1229731473 Seq=140990389
Len=0 Win=24820
```

```
swatch -> mamba.red.iplanet.com TCP D=9010 S=33048 Ack=1229731473 Seq=140990389
Len=285 Win=24820
mamba.red.iplanet.com -> swatch TCP D=33048 S=9010 Ack=140990674 Seq=1229731473
Len=184 Win=8475
swatch -> mamba.red.iplanet.com TCP D=9010 S=33048 Ack=1229731657 Seq=140990674
Len=0 Win=24820
swatch -> mamba.red.iplanet.com TCP D=9010 S=33048 Ack=1229731657 Seq=140990674
Len=132 Win=24820
mamba.red.iplanet.com -> swatch TCP D=33048 S=9010 Ack=140990806 Seq=1229731657
Len=25 Win=8343
swatch -> mamba.red.iplanet.com TCP D=9010 S=33048 Ack=1229731682 Seq=140990806
Len=0 Win=24820
swatch -> mamba.red.iplanet.com TCP D=9010 S=33048 Ack=1229731682 Seq=140990806
Len=124 Win=24820
mamba.red.iplanet.com -> swatch TCP D=33048 S=9010 Ack=140990930 Seq=1229731682
Len=0 Win=8219
mamba.red.iplanet.com -> swatch TCP D=33048 S=9010 Ack=140990930 Seq=1229731682
Len=336 Win=8219
swatch -> mamba.red.iplanet.com TCP D=9010 S=33048 Ack=1229732018 Seq=140990930
Len=120 Win=24820
mamba.red.iplanet.com -> swatch TCP D=33048 S=9010 Ack=140991050 Seq=1229732018
Len=0 Win=8099
mamba.red.iplanet.com -> swatch TCP D=33048 S=9010 Ack=140991050 Seq=1229732018
Len=32 Win=8099
swatch -> mamba.red.iplanet.com TCP D=9010 S=33048 Ack=1229732050 Seq=140991050
Len=120 Win=24820
mamba.red.iplanet.com -> swatch TCP D=33048 S=9010 Ack=140991170 Seq=1229732050
Len=0 Win=7979
mamba.red.iplanet.com -> swatch TCP D=33048 S=9010 Ack=140991170 Seq=1229732050
Len=32 Win=7979
swatch -> mamba.red.iplanet.com TCP D=9010 S=33046 Fin Ack=1229729582
Seq=140303807 Len=0 Win=24820
mamba.red.iplanet.com -> swatch TCP D=33046 S=9010 Ack=140303808 Seq=1229729582
Len=0 Win=8524
mamba.red.iplanet.com -> swatch TCP D=33046 S=9010 Fin Ack=140303808
Seq=1229729582 Len=0 Win=8524
swatch -> mamba.red.iplanet.com TCP D=9010 S=33048 Fin Ack=1229732082
Seq=140991170 Len=0 Win=24820
swatch -> mamba.red.iplanet.com TCP D=9010 S=33046 Ack=1229729583 Seq=140303808
Len=0 Win=24820
mamba.red.iplanet.com -> swatch TCP D=33048 S=9010 Ack=140991171 Seq=1229732082
Len=0 Win=7979
mamba.red.iplanet.com -> swatch TCP D=33048 S=9010 Fin Ack=140991171
Seq=1229732082 Len=0 Win=7979
swatch -> mamba.red.iplanet.com TCP D=9010 S=33048 Ack=1229732083 Seq=140991171
Len=0 Win=24820
```

RMI/IIOP ログメッセージの表示

RMI/IIOP パスによって生成されたログメッセージは、IIOP ブリッジプロセスによって生成されたログファイルをレビューすることによって監視できます。IIOP ブリッジプロセスは Java エンジン (kjs) 形式なので、Web および EJB コンテナをサポートする Java エンジンを監視する方法と同じ方法で、これらのログを監視します。該当するログファイルを表示するには、IIOP ブリッジの役割を果たす Java エンジンを指定する必要があります。

Windows 上でのログの監視

iPlanet Application Server を Windows にインストールしている場合、デフォルトでは、アプリケーションサーバの起動時、Java エンジンログファイルは自動的には表示されません。コンソールログ情報の自動表示を有効にすると便利です。これを行うには、次の手順を実行します。

1. 「スタート」>「設定」>「コントロールパネル」を選択します。
2. 「サービス」をダブルクリックします。
3. 「iPlanet Application Server 6.5」エントリを選択します。
4. 「スタートアップ」をクリックします。
5. 「デスクトップとの対話をサービスに許可」、「OK」の順にクリックします。
6. アプリケーションサーバを停止するには、「停止」をクリックします。
7. アプリケーションサーバを起動するには、「開始」をクリックします。

アプリケーションサーバが起動すると、多数の MS DOS 出力ウィンドウがデスクトップに表示されます。アプリケーションサーバの物理プロセスごとに 1 つの出力ウィンドウが表示されます。エンジンが起動したら、Java エンジンを検索し、CXS (ブリッジ) プロセスで定義されているポート番号を指定するエンジンを検索します。

これらの出力ウィンドウの垂直スクロールバーを有効にするには、次の手順を実行します。

1. 出力ウィンドウの左上隅の MS DOS アイコンを選択します。
2. 「プロパティ」を選択します。
3. 「レイアウト」を選択します。
4. 「画面バッファのサイズの高さ」を 200 または希望の数値に設定します。
5. このウィンドウを起動するたびにこれらの変更を適用するかどうかを尋ねるプロンプトが表示された場合は、「Yes」をクリックします。

UNIX 上でのログの監視

UNIX では、開発者の多くは、`tail -f` コマンドを使って、対象プロセスのアプリケーションサーバログファイルを監視します。この方法で Java エンジンログを監視するには、次の手順を実行します。

1. 次のログディレクトリに移動します。

   ```
   cd install_dir/ias/logs
   ```

2. Java エンジン (`kjs`) の一つおよび実行サービス (`kxs`) プロセスに対して `tail` コマンドを実行します。

   ```
   tail -f kjs_2*
   ```

   監視する適切な Java エンジンログファイルを選択する必要があります。Java エンジンは、管理ツールで定義されている方法に従って番号付けされます。通常、CXS (ブリッジ) プロセスがもっとも番号の大きい Java エンジンログファイルですが、CXS プロセスによって生成されたログファイルを確認するために、ログファイルのポート番号情報を再び確認してください。

3. `tail` コマンドを無効にするには、Control + C キーを押します。

RMI/IIOP サンプルアプリケーション

RMI/IIOP 指向サンプルのリストは、Web サーバのドキュメントルートの次の場所、またはアプリケーションサーバのインストールディレクトリの下にあります。

`http://webserver_host/ias-samples/ -> RMI/IIOP`

`install_dir/ias/ias-samples/index.html -> RMI/IIOP`

Converter サンプルアプリケーション

Sun の『J2EE 開発者ガイド』の Currency Converter サンプルアプリケーションは iPlanet Application Server にバンドルされています。このサンプルには、アプリケーションを iPlanet Application Server に配置するための詳しいセットアップ手順が追加されています。ほかの RMI/IIOP ベースアプリケーションを配置する前に、このサンプルの詳しいセットアップ手順に従って Converter サンプルを実行することをお勧めします。Currency Converter のセットアップマニュアルおよびソースコードは次の場所にあります。

`install_dir/ias/ias-samples/j2eeguide/docs/converter.html`

`install_dir/ias/ias-samples/j2eeguide/converter/src/`

その他の RMI/IIOP サンプルアプリケーション

iPlanet Application Server にバンドルされている『J2EE 開発者ガイド』のほとんどのサンプルには、RMI/IIOP クライアントプログラムが含まれています。これらのプログラムは、EJB 仕様のさまざまな局面を示した比較的簡単なサンプルです。これらのサンプルは次の場所にあります。

install_dir/ias/ias-samples/j2eeguide/docs/index.html

C++ IIOP クライアントアプリケーション (UNIX のみ)

iPlanet Application Server で C++ IIOP ベースのクライアントアプリケーションを使用する方法は、ほかの J2EE 認定アプリケーションサーバでこれらのクライアントを使用する場合とほぼ同じです。クライアントの検索部分に最小限の変更を加えるだけで、クライアントを再利用してさまざまな J2EE アプリケーションサーバと連動させることができます。

この節には次の節があります。

- ORBIX 用 C++ IIOP アプリケーションの設定
- C++ クライアント開発の準備
- データタイプの前提条件と制約事項
- IDL ファイルの生成
- IDL ファイルから CPP ファイルへの変換
- C++ IIOP アプリケーションのセキュリティの有効化
- EJB ホームインタフェースの検索
- クライアントサイドのロードバランスおよびフェールオーバー
- IIOP ブリッジの設定
- C++ IIOP クライアントアプリケーションの配置
- IIOP のパフォーマンスチューニング
- IIOP ログメッセージの表示
- C++ IIOP サンプルアプリケーション

ORBIX 用 C++ IIOP アプリケーションの設定

『管理者ガイド』で説明しているように、iPlanet Application Server で C++ IIOP クライアントを使う前に、ORBIX 2000 をインストールして iPlanet Application Server と統合する必要があります。この節では、ORBIX のソフトウェア要件とその他のマニュアルを示します。

要件

ソフトウェア要件は次のとおりです。

- Solaris 2.6 +
- ORBIX 2000 C++ 開発キット、バージョン 1.2+
- Sun Workshop 6.2 (C++ 5.2)
- iPlanet Application Server 6.5
- Java 用 IDL コンパイラ (J2SE 1.3 以前の rmic にはいくつかの問題があるため、J2SE 1.4 Beta またはその他の Java 用 IDL コンパイラの rmic を使用すること)
- 値渡しのユーザ定義と Java ネイティブの複合データタイプを、C++ に実装する。EJB および C++ クライアント間で値が交換されるデータタイプを、異なる言語間で正しく変換させるには、上記のデータタイプを C++ に実装しなければならない。これは、Java ネイティブおよびユーザ定義のデータタイプに適用される。

参照資料

- OMG IDLto Java Language Mapping

  `ftp://ftp.omg.org/pub/docs/ptc/00-01-08.pdf`
- ORBIX 2000 Programmer's Reference, C++ Edition

  `http://www.iona.com/docs/orbix2000/1.2.1/pguide_cpp/html/index.html`
- ORBIX 2000 Programmer's Guide, Java Edition

  `http://www.iona.com/docs/orbix2000/1.2.1/pguide_java/html/index.html`
- ORBIX 2000 1.2.1 Documentation

  `http://www.iona.com/docs/orbix2000/1.2.1/index.html`
- OpenORB RMI over IIOP (Java 用 IDL コンパイラがある)

  `http://www.openorb.org/rmioveriiop.html`
- Java 2 SDK 1.4 のダウンロード

  `http://java.sun.com/j2se/1.4/index.html`

C++ クライアント開発の準備

C++ クライアントを開発する前に、次の手順を実行します。

1. 必要なソフトウェアがすべてインストールされていることを確認します。280 ページの「要件」を参照してください。
2. EAR ファイルのスタブとスケルトンが `ejbc` の `-iiop` オプションを使って生成されていることを確認します。生成されていない場合は、`build.xml` ファイルを編集して `-iiop` を `ejbc` オプションに追加し、EAR ファイルを再生成します。
3. EJB を配置します。iPlanet Application Server に組み込まれている J2EE のサンプル (通貨の変換と確認用のサンプルなど) を使うことができます。

注　「OMG IDL to Java Language Mapping」に準拠しているため、Java パッケージ名の大文字と小文字は区別されません。パッケージ内のクラス名またはインタフェース名でも、大文字と小文字は区別されません。大文字と小文字だけが異なる Java パッケージ名、クラス名、およびインタフェース名は、エラーとして処理されます。大文字と小文字だけが異なるパッケージ名とクラス名は、Bean に配置しないでください。J2EE のサンプルは、この点に考慮していないため、配置する前に変更する必要があります。この例については、293 ページの「C++ クライアント用に Converter サンプルを再配置する」を参照してください。

データタイプの前提条件と制約事項

クライアントとサーバ間での値の受け渡しのテストが完了しているデータタイプは、double、int、long、short、float、char、boolean、および byte データタイプだけです。その他の CORBA 標準データタイプでは、IDL から Java/C++ 言語へのマッピングが完了していることを前提としています。`java.lang.BigDecimal` を渡すと、`NO_IMPLEMENT` 例外がスローされます。

その他の値渡しのデータタイプ (`HashTable` などのカスタム Java クラス) の場合は、C++ 固有の実装を使用するか、既存のクラス (STL など) の C++ 実装を Java クラス用に生成された IDL に変換するラッパーを作成する必要があります。

IDL ファイルの生成

次の 2 つの方法で、IDL ファイルを生成できます。

- J2SE 1.4 rmic 2 を使用する
- OpenORB JavaToIDL コンパイラを使用する

J2SE 1.4 rmic 2 を使用する

J2SE 1.4 の `rmic` を使うには、次の手順で行います。

1. C++ クライアントの開発用に新しいディレクトリを作成します。次のようにします。

   ```
   mkdir cppclient
   cd cppclient
   ```

2. `rmic` を実行します。たとえば、次の方法で実行します (改行しない)。

```
rmic -classpath ias_inst_dir/ias/APPS/j2eeguide-myconverter/j2eeguide-myconverterEjb:
ias_inst_dir/ias/classes/java/javax.jar -idl j2eeguide.myconfirmer.Confirmer
```

   次の処理も実行します (改行しない)。

```
rmic -classpath ias_inst_dir/ias/APPS/j2eeguide-myconverter/j2eeguide-myconverterEjb:
ias_inst_dir/ias/classes/java/javax.jar -idl j2eeguide.myconfirmer.ConfirmerHome
```

3. IDL ファイルを移動します。次のようにします。

   ```
   mv j2eeguide/myconverter/Converter.idl
   mv j2eeguide/myconverter/ConverterHome.idl
   ```

4. 生成された 2 つの IDL を結合します。次のようにします。

   a. `cat ConverterHome.idl >> Converter.idl`

   b. `Converter.idl` を編集して、`j2eeguide` と `myconverter` モジュール、および `Converter` と `ConverterHome` インタフェースを宣言する行と、それらに対応する `#pragma` 宣言以外の行をすべて削除します。

   c. IDL ファイルの先頭に次の行を追加します。

      ```
      #include <omg/orb.idl>
      #include "ejb.idl"
      #include "_std_java.idl"
      ```

   最終 IDL ファイルの出力です。確認してください。

   ```
   #include <omg/orb.idl>
   #include "ejb.idl"
   #include "_std_java.idl"
   module j2eeguide {
      module myconverter {
   ```

```
            interface Converter : ::javax::ejb::EJBObject {
               double dollarToYen(in double arg0) ;
               double yenToEuro(in double arg0) ;
            };
            #pragma ID Converter
            "RMI:j2eeguide.myconverter.Converter:0000000000000000"
            interface ConverterHome : ::javax::ejb::EJBHome  {
               Converter create() raises(::javax::ejb::CreateEx);
            };
            #pragma ID ConverterHome
         "RMI:j2eeguide.myconverter.ConverterHome:0000000000000000"
         };
   };
```

5. 生成された _std_java.idl ファイルと ejb.idl ファイルを cppclient ディレクトリにコピーします。

OpenORB JavaToIDL コンパイラを使用する

openorb JavaToIdl ツールを使って、生成された openorb JAR ファイル (openorb_rmi-1.0.1.jar および openorb_tools-1.0.1.jar) を現在のディレクトリにコピーします。たとえば、次の方法で実行します (改行しない)。

```
java -cp openorb_rmi-1.0.1.jar:openorb_tools-1.0.1.jar:ias_inst_dir/ias/APPS/
j2eeguide-myconverter/j2eeguide-myconverterEjb:
ias_inst_dir/ias/classes/java/javax.jar org.openorb.rmi.compiler.JavaToIdl
j2eeguide.myconverter.Converter
```

次の処理も実行します (改行しない)。

```
java -cp openorb_rmi-1.0.1.jar:openorb_tools-1.0.1.jar:ias_inst_dir/ias/APPS/
j2eeguide-myconverter/j2eeguide-myconverterEjb:
ias_inst_dir/ias/classes/java/javax.jar org.openorb.rmi.compiler.JavaToIdl
j2eeguide.myconverter.ConverterHome
```

IDL ファイルから CPP ファイルへの変換

.idl ファイルから .cpp ファイルを生成するには、次の手順で行います。

1. 次のコマンドを実行して、ORBIX 環境設定スクリプトを生成します。

   . *orbix_inst_dir*/bin/*domain*_env

   次のようにします。

   ```
   . /opt/iona/bin/localhost_env
   ```

2. 次のコマンドを実行します (改行しない)。

orbix_inst_dir/bin/idlgen cpp_poa_genie.tcl -ns -all -complete Confirmer.idl -I. -I *orbix_inst_dir*/orbix_art/1.2/idl

3. makefile を編集して、値 IT_PRODUCT_DIR をインストールに適用できる値に変更します。
4. CXXFLAGS に -I を設定します。
5. PATH をエクスポートして、パスの先頭に workshop6 bin ディレクトリを含めます。
6. 次のコマンドを実行します。エラーが発生しますが、後の手順で修正します。

   ```
   make -e
   ```

7. ejb.hh ファイルの CORBA を ::CORBA に変更します。ただし、ネーム空間 javax::rmi::CORBA に限ります。
8. EJBMetaData を javax::ejb::EJBMetaData に変更します。
9. makefile および client.cxx で、EJBMetaDataImpl を含む行を削除するかコメントにして、コンパイルエラーが発生しないようにします。
10. client.cxx を次のように編集します。

    a. EJBMetaData を登録する行を、次のようにコメントにします。

    ```
    javax_ejb_EJBMetaDataFactory::_register_with_orb(orb);
    ```

    b. 次の行を削除します。

    ```
    tmp_ref = default_context->resolve_str("IT_GenieDemo");
    CosNaming::NamingContext_var demo_context =
        CosNaming::NamingContext::_narrow(tmp_ref);
    assert(!CORBA::is_nil(demo_context));
    ```

    c. 生成されたコードから、Converter、EJBObject、および EJBHome の検索を削除します。次のコメントで識別できます。

    ```
    //Exercise interface j2eeguide::myconverter::Converter
    //Exercise interface javax::ejb::EJBObject
    //Exercise interface javax::ejb::EJBHome
    ```

    d. 次の行を変更します。

```
name = default_context->to_name("j2eeguide_myconfirmer_Confirmer");
tmp_ref = demo_context->resolve(name);
```

次のようにしてください。

```
name = default_context->to_name("ejb/MyMyConfirmer");
tmp_ref = default_context->resolve(name);
```

    e. 生成されたコードから、関数呼び出し (call_j2eeguide_ で始まる) をコメントにし、create とビジネスメソッドを呼び出すコードを挿入します。次のようにします。

```
j2eeguide::myconverter::Converter_var converter =
    ConverterHome4->create();
CORBA::Double yen = 4000;
CORBA::Double euro = converter->yenToEuro(yen);
```

11. ejbC.cxx を編集して、すべての CORBA を ::CORBA に変更します。たとえば、次のような正規表現の構文で変更します。

```
s/^CORBA/::CORBA/g

s/ CORBA/ ::CORBA/g

s/namespace ::CORBA/namespace CORBA/g

s/¥!CORBA/¥!::CORBA/g

s/(CORBA/(::CORBA/g

s/ EJBMetaData/::javax::ejb::EJBMetaData/g

s/IT_CONST_CAST(::CORBA/IT_CONST_CAST(CORBA/g
```

12. ConverterC.cxx ファイルを編集して、操作名を次のパターンの名前に変更します。連続した下線はリテラル文字列です。

    function-name__return-type(pkg1_pkg2_class)__argument-type

    データタイプは Java タイプでなければなりません。たとえば、Java タイプが int の場合、IDL タイプは long ですが、操作で表現されるデータタイプは int でなければなりません。生成された Java スタブ (_Converter_Stub.java など。build.xml の ejbc に -gs オプションが指定されている場合に生成される) で _request メソッドのパラメータを参照すれば、正確な操作名を取得できます。次のようにします。

```
s/"create"/"create__j2eeguide_myconverter_Converter__void"/g
s/"yenToEuro"/"yenToEuro__double__double"/g
s/"dollorToYen"/"dollorToYen__double__double"/g
```

13. make を実行します。

```
make -e client
```

14. クライアントを実行します。

```
./client
```

C++ IIOP アプリケーションのセキュリティの有効化

セキュリティを有効にするには、クライアントアプリケーションを次のライブラリに接続する必要があります。

- iPlanet Application Server 6.5 に組み込まれている `libgxorbixclientinterceptor.so` ライブラリ
- ORBIX に組み込まれている `it_portable_interceptor` ライブラリ

次の手順で接続します。

1. クライアントの makefile にある CLIENT_LIBS 行に、次の行を挿入します。

   ```
   -lit_portable_interceptor -lgxorbixclientinterceptor
   ```

2. `libgxorbixclientinterceptor.so` へのパスを makefile の LDLIBS 設定に挿入します。次のようにします。

   ```
   -L/space/interceptor ¥
   ```

3. `libgxorbixclientinterceptor.so` へのパスを LD_LIBRARY_PATH に挿入します。次のようにします。

   ```
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/space/interceptor
   ```

4. クライアントをリビルドします。

   ```
   make -e client
   ```

5. 環境変数 IAS_RC_USERID および IAS_RC_PASSWORD を LDAP ユーザに対応するユーザ ID とパスワードに設定します。シェルから設定するか、putenv 関数を使用します。これらの変数を NULL に設定したり、設定しなかった場合、ユーザ認証が行われず、ロールが割り当てられている Bean にアクセスできません。次のようにします。

   ```
   export IAS_RC_USERID=j2ee
   export IAS_RC_PASSWORD=j2ee
   ```

6. ORBIX を統合するための管理者ガイドで説明しているように、クライアントインターセプタとサーバインターセプタを有効にします。

7. クライアントを実行します。

   ```
   ./client
   ```

EJB ホームインタフェースの検索

IIOP クライアントのコードを作成するには、最初に EJB のホームインタフェースの検索を実行します。ホームインタフェースの検索を実行する準備として、環境プロパティをいくつか設定する必要があります。次に、EJB の検索名を指定します。次の例に、設定手順を示します。

```
//ORB を初期化します。
global_orb = CORBA::ORB_init(argc, argv);

// genie デモのネーミングコンテキストを取得します。
tmp_ref = global_orb->resolve_initial_references("NameService");
CosNaming::NamingContext_var default_context =
   CosNaming::NamingContext::_narrow(tmp_ref);
assert(!CORBA::is_nil(default_context));

// ホームを検索します。
name = new CosNaming::Name(2);
name->length(2);
name[0].id = CORBA::string_dup("ejb");
name[0].kind = CORBA::string_dup("");
name[1].id = CORBA::string_dup("MyMyConfirmer");
name[1].kind = CORBA::string_dup("");
tmp_ref = default_context->resolve(name);

j2eeguide::myconfirmer::ConfirmerHome_var
ConfirmerHome=j2eeguide::myconfirmer::ConfirmerHome::_narrow(tmp_re
f);

//create を呼び出します。
j2eeguide::myconfirmer::Confirmer_var
Confirmer=ConfirmerHome->create();

// ビジネスメソッドを呼び出します。
Confirmer->sendNotice(chars);
```

クライアントサイドのロードバランスおよびフェールオーバー

iPlanet Application Server には IIOP アクセス用のサーバサイドのロードバランスおよびフェールオーバーが用意されていますが、アプリケーションのパフォーマンスおよび可用性をさらに向上させるために、クライアントサイドの方法を実装することも検討してください。

ネーミングサービスでは、名前とオブジェクトを対応付けたリポジトリを定義します。名前とオブジェクトは、1 対 1 で対応付けます。ORBIX 2000 では、ネーミングサービスモデルを拡張し、1 つの名前をオブジェクトのグループに対応付けることができます。オブジェクトグループは、オブジェクトの集合で、サイズを動的に調節できます。オブジェクトグループごとに、選択アルゴリズムを指定します。このアルゴリズムは、オブジェクトグループに関連付けられた名前をクライアントが解決するときに適用されます。次の 3 つのアルゴリズムがサポートされています。

- ラウンドロビン選択
- ランダム選択
- アクティブロードバランス選択

オブジェクトグループを利用すれば、頻繁に要求されるオブジェクトを複製して、リクエストの処理負荷を分散することができます。ネーミングサービスは、オブジェクトグループの選択アルゴリズムに従って、クライアントリクエストを複製されたオブジェクトに転送します。オブジェクトグループは、クライアントに対して透過的です。クライアントは、ほかの名前と同様に、オブジェクトグループの名前を解決します。

フラグ `ORBIX_LOADBALANCING=true` または `false` を `iasenv.ksh` ファイルに設定すると、Java 引数を次のように設定できます。

```
-DORBIXLoadBalancing=$ORBIX_LOADBALANCING
```

IIOP ブリッジの設定

iPlanet Application Server をインストールする時点で、IIOP ブリッジプロセスが設定されていない場合は、iPlanet Application Server Administrative Tool を起動して、IIOP ブリッジプロセスをアプリケーションサーバ環境に追加する必要があります。

1. iPlanet Application Server 管理ツールを起動します。

   install_dir`/ias/bin/ksvradmin`

2. アプリケーションサーバインスタンスに接続し、サーバ名アイコンをダブルクリックして、アプリケーションサーバのインスタンスに定義されているプロセスを一覧表示します。少なくとも `kjs` プロセスが 1 つ、および `kxs` プロセスが 1 つ表示されます。EJB への IIOP アクセスには、`kxs` プロセスは不要です。`cxs` プロセスが表示された場合は、アプリケーションサーバインスタンスにすでに IIOP ブリッジプロセスが定義されています。この場合は、`cxs` プロセスエントリをダブルクリックし、IIOP ポート番号を書き留め、次の節に進んでください。ブリッジプロセスが表示されない場合は、次の手順に進んでプロセスを定義してください。

3. 任意の既存プロセスエントリを選択し、「ファイル」>「新規」>「プロセス」を選択します。

4. プロセスタイプのプルダウンリストから cxs を選択し、kjs および kxs プロセスによってすでに使われているほかのポート番号と競合しないポート番号 (ポート 10822 など) を入力します。システム環境でほかに割り当てられているポートと競合しないかぎり、デフォルトの IIOP ポート番号 (9010) を選択します。「OK」をクリックしてプロセスをインスタンス化します。
5. 数秒後、アプリケーションサーバ環境で IIOP ブリッジプロセスが動作している状態が表示されます。このプロセスは、Administrative Tool に一覧表示されているその他のすべてのアプリケーションサーバプロセスとともに、アプリケーションサーバの再起動時に自動的に開始されます。
6. コマンドラインから IIOP ブリッジプロセスの存在も確認できます。たとえば、次のコマンドを入力します (各コマンドはすべて 1 行で指定する)。

```
ps -ef | grep iiop
```

```
root 1153 1 0 17:00:15 ?0:00 /bin/sh /usr/iPlanet/ias6/ias/bin/kjs -cset CCS0 -eng
3 -iiop -DORBinsPort=9010
```

この出力には、-iiop オプションで開始された iPlanet Java エンジンプロセスが表示されます。このオプションは、Java エンジンのこのインスタンスに、J2EE Web および EJB コンテナプロセスではなく、IIOP ブリッジプロセスとして開始するように指示します。

cxs プロセスをインスタンス化すると、IIOP サポートのサーバサイドの設定が完了します。

C++ IIOP クライアントアプリケーションの配置

クライアントアプリケーションを開発する場合、開発環境からクライアントシステムに多数のファイルを配置する必要があります。この節では、次の節で説明する IIOP 対応クライアントアプリケーションの配置に必要な、基本手順について説明します。

- クライアントの配置
- サーバの CLASSPATH の設定 (SP2 以前)

クライアントの配置

EJB 固有のホームおよびリモートインタフェースとそれらに対応するスタブが、クライアントシステムに配置されていなければなりません。たとえば、Converter サンプルアプリケーションでは、次のクラスをクライアントシステムにコピーする必要があります。

ホームおよびリモートインタフェースクラス

```
ConverterHome.class
```

```
Converter.class
```

EJB 固有の iPlanet クライアントスタブ

```
_Converter_Stub.class
```

```
_ConverterHome_Stub.class
```

サーバの CLASSPATH の設定 (SP2 以前)

この節の記述は、iPlanet Application Server 6.0 Service Pack 2 (SP2) 以前に適用されます。Service Pack 3 以降には、次の設定手順は不要です。SP 3 以降を使う場合は、次の節に進んでください。

iPlanet Application Server Service Pack 2 以前で EJB クラスを読み込むには、IIOP ブリッジプロセスが、アプリケーションサーバの CLASSPATH を使って EJB スタブとホーム / リモートインタフェースにアクセスできるようになっている必要があります。IIOP ベースの Java アプリケーションクライアントを SP 2 以前で初めて実行する場合は、まずアプリケーションサーバの CLASSPATH を変更する必要があります。

iPlanet Application Server 6.0 SP2 以降、EJB ベースアプリケーションを登録すると EJB JAR ファイルがアプリケーションサーバの配置ディレクトリに展開されるようになりました。デフォルトでは、`j2eeguide-converter.ear` などの J2EE アプリケーションをアプリケーションサーバに配置すると、`j2eeguideEjb.jar` などの組み込み EJB JAR ファイルが次のディレクトリに展開されます。

install_dir`/ias/APPS/j2eeguide-converter/j2eeguide-converterEjb/`

スタンドアロン EJB JAR モジュール (または WAR モジュール) を iPlanet Application Server に配置する場合、このスタンドアロンモジュールのデフォルトの展開場所は次のディレクトリになります。

install_dir`/ias/APPS/modules/j2eeguide-converterEjb/`

C++ IIOP クライアントを実行する前に、適切なモジュールディレクトリをアプリケーションサーバの CLASSPATH に追加する必要があります。

IIOP のパフォーマンスチューニング

IIOP パスで多数の同時ユーザに対応しなければならない配置環境の場合は、この節で説明するチューニングガイドラインを試してください。IIOP を使う場合、JVM のデフォルト設定とその基本 OS だけでは最適なパフォーマンスおよび容量を達成できません。

この節には次のトピックがあります。

- 基本的なチューニング方法
- スケーラビリティの向上

基本的なチューニング方法

次に説明するチューニング方法を試し、ユーザの環境に最適なバランスを見つけてください。

Solaris ファイル記述子の設定

Solaris の場合、`ulimit` を使って、開いているファイル数のプロパティを最大に設定すると、サポートできる IIOP クライアントの最大数に影響を与えます。このプロパティのデフォルト値は、Solaris 2.6 または Solaris 8 のどちらを実行しているかによって、`64` または `1024` となります。数を増やすには、次のコマンドを `/etc/system` に追加し、再起動します。

```
set rlim_fd_max = 8192
```

次のコマンドを使うと、この使用制限を確認できます。

```
ulimit -a -H
```

上記の使用制限を設定後、次のコマンドを使うと、このプロパティの値をこの制限まで明示的に増やすことができます。

```
ulimit -n 8192
```

次のコマンドを使うと、この制限を確認できます。

```
ulimit -a
```

たとえば、ulimit がデフォルトの 64 の場合、1 つのテストドライバがサポートできる同時クライアントは 25 ですが、`ulimit` を `8192` に設定すると、同じテストドライバで 120 の同時クライアントをサポートできます。このテストドライバでは複数のスレッドが生成されました。これらの各スレッドは JNDI 検索を実行し、ビジネスメソッド呼び出し間の思考 (遅延) 時間が 500 ミリ秒で同じビジネスメソッドを繰り返し呼び出し、約 100 KB のデータを送受信できました。

これらの設定値は IIOP クライアント (Solaris)、および Solaris システムにインストールされた IIOP ブリッジに適用されます。ファイル記述子の制限の設定については、Solaris のマニュアルを参照してください。

スケーラビリティの向上

1 つのブリッジプロセスおよびクライアントシステムの容量をチューニングするだけでなく、複数の IIOP ブリッジプロセスを使うことによって、IIOP 環境のスケーラビリティを向上させることができます。同じアプリケーションサーバインスタンス上で複数のブリッジプロセスを設定すると、アプリケーション配置のスケーラビリティが向上します。場合によっては、それぞれ 1 つまたは複数のブリッジプロセスを使って設定した多数のアプリケーションサーバインスタンスを使うこともできます。

複数のブリッジプロセスがアクティブな設定では、一連のクライアントをさまざまなブリッジにスタティックにマッピングするか、またはクライアントサイドに独自のロジックを実装して既知のブリッジプロセスと照らし合わせてロードバランスを行うことによって、クライアント負荷を分割できます。

IIOP ログメッセージの表示

IIOP パスによって生成されたログメッセージは、IIOP ブリッジプロセスによって生成されたログファイルをレビューすることによって監視できます。IIOP ブリッジプロセスは Java エンジン (`kjs`) 形式なので、Web および EJB コンテナをサポートする Java エンジンを監視する方法と同じ方法で、これらのログを監視します。該当するログファイルを表示するには、IIOP ブリッジの役割を果たす Java エンジンを指定する必要があります。

開発者の多くは、`tail -f` コマンドを使って、対象プロセスのアプリケーションサーバログファイルを監視します。この方法で Java エンジンログを監視するには、次の手順を実行します。

1. 次のログディレクトリに移動します。

   ```
   cd install_dir/ias/logs
   ```

2. Java エンジン (`kjs`) の一つおよび実行サービス (`kxs`) プロセスに対して `tail` コマンドを実行します。

   ```
   tail -f kjs_2*
   ```

   監視する適切な Java エンジンログファイルを選択する必要があります。Java エンジンは、管理ツールで定義されている方法に従って番号付けされます。通常、CXS (ブリッジ) プロセスがもっとも番号の大きい Java エンジンログファイルですが、CXS プロセスによって生成されたログファイルを確認するために、ログファイルのポート番号情報を再び確認してください。

3. `tail` コマンドを無効にするには、Control + C キーを押します。

C++ IIOP サンプルアプリケーション

Sun の『J2EE 開発者ガイド』の Currency Converter サンプルアプリケーションは iPlanet Application Server にバンドルされています。このサンプルには、アプリケーションを iPlanet Application Server に配置するための詳しいセットアップ手順が追加されています。ほかの IIOP ベースのアプリケーションを配置する前に、このサンプルの詳しいセットアップ手順に従って Converter サンプルを実行することをお勧めします。Currency Converter のセットアップマニュアルおよびソースコードは次の場所にあります。

install_dir/ias/ias-samples/j2eeguide/docs/converter.html

install_dir/ias/ias-samples/j2eeguide/converter/src/

C++ クライアント用に Converter サンプルを再配置する

Bean を配置するときには、パッケージ名とクラス名の大文字と小文字は区別されません。このため、次の手順に従って、C++ IIOP クライアント用に Converter サンプルを再配置する必要があります。ほかのサンプルを再配置するときにも、この手順を適用できます。

1. cd *ias_inst_dir*/ias/ias-samples/j2eeguide
2. cp -R converter myconverter
3. cd myconverter/src
4. 次の表に示すように、build.xml、ejb-jar.xml、web.xml、application.xml、および schema/*.xml ファイルの、パッケージなどの名前を変更します。

表 10-2　Converter サンプルの XML ファイルの変更

名前を変更する要素	変更前	変更後
パッケージ名	converter	myconverter
appname、display-name、および context-root	j2eeguide-converter	j2eeguide-myconverter
ejb-name および ejb-link	MyConverter	MyMyConverter

5. 次のコマンドを実行します。

   ias_inst_dir/ias/bin/kguidgen
6. 生成された guid をコピーし、ias-ejb-jar.xml ファイルの <guid> セクションにある guid 値を置換します。
7. 次のコマンドを再度実行します。

   ias_inst_dir/ias/bin/kguidgen
8. 生成された guid をコピーし、ias-web.xml ファイルの <guid> セクションにある guid 値を置換します。
9. mv j2eeguide/converter j2eeguide/myconverter
10. cd j2eeguide/myconverter

11. Java ファイルをすべて変更して、パッケージ名の変更 (converter から myconverter への変更など) を反映し、ConverterClient.java の検索名を MyConverter から MyMyConverter に変更します。
12. cd ../..
13. *ias_inst_dir*/ias/bin/build
14. cd ../assemble/ear
15. *ias_inst_dir*/ias/bin/iasdeploy deployapp j2eeguide-myconveter.ear
16. iPlanet Application Server 6.0 SP2 以前のバージョンの場合は、以下の追加手順を実行してください。
    a. *ias_inst_dir*/ias/bin/kjs スクリプトを編集して、新しいディレクトリ *ias_inst_dir*/ias/APPS/j2eeguide-myconverter/j2eeguide-myconverterEjb をクラスパスに追加します。
    b. iPlanet Application Server を再起動します。

第 11 章

配置のためのパッケージ化

この章では、iPlanet Application Server モジュールの内容と、アプリケーションとモジュールを個別にまたはすべてパッケージ化する方法について説明します。

パッケージ化に関連する設計上の考慮事項については、30 ページの「アプリケーションのモジュール化」を参照してください。

iPlanet Application Server のモジュールとアプリケーションには、J2EE 標準の要素と iPlanet Application Server 固有の要素が組み込まれています。この章では、iPlanet Application Server 固有の要素についてだけ詳細に説明します。

この章には次の節があります。

- パッケージと配置の概要
- モジュールおよびアプリケーションのアセンブリ
- モジュールおよびアプリケーションの配置
- XML DTD について
- Web アプリケーション XML DTD
- EJB XML DTD
- RMI/IIOP クライアント XML DTD
- リソース XML DTD

パッケージと配置の概要

アプリケーションアセンブリ (パッケージ化とも呼ばれる) は、アプリケーションの個別のコンポーネントを、J2EE に準拠するアプリケーションサーバに配置できる単位に結合するプロセスです。パッケージは、モジュールまたは独立したアプリケーションとして利用できます。

モジュール

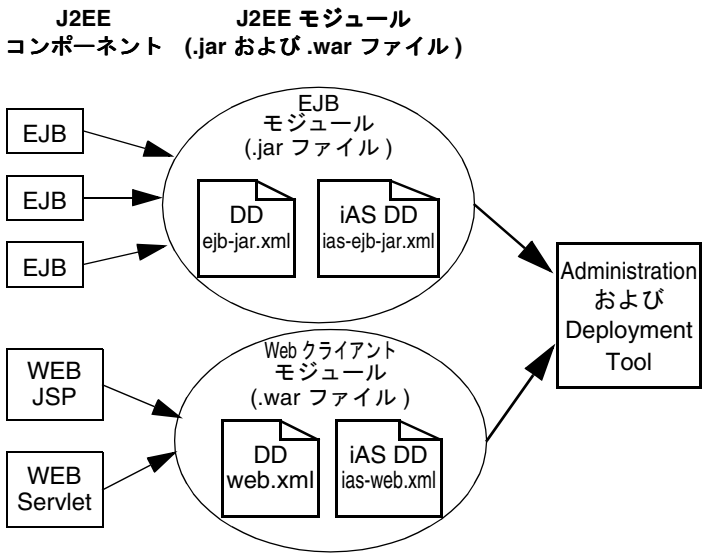
J2EE モジュールは、J2EE コンポーネントの集合で、同一コンテナタイプの 2 つの配置記述子 (DD) を持ちます。一方の DD は J2EE 標準で、もう一方の DD は iPlanet Application Server 固有です。J2EE モジュールのタイプは次のとおりです。

- **Web アプリケーションアーカイブ (WAR)** : Web アプリケーションは、Servlet、HTML ページ、クラスなどのリソースの集合で、いくつかの J2EE アプリケーションサーバに含めて配置することができます。WAR ファイルは、Servlet、JSP、ユーティリティクラス、静的ドキュメント、クライアントサイドアプレット、Bean と Bean クラス、および配置記述子 (`web.xml` および `ias-web.xml`) から構成されます。
- **EJB JAR ファイル** : EJB JAR ファイルは、Enterprise JavaBeans をパッケージ化するときの標準フォーマットです。このファイルには、Bean クラス (ホーム、リモート、および実装)、すべてのユーティリティクラス、および配置記述子 (`ejb-jar.xml` および `ias-ejb-jar.xml`) が含まれています。
- **RMI/IIOP クライアント JAR ファイル** : RMI/IIOP クライアントは、iPlanet Application Server 固有の J2EE クライアントです。RMI/IIOP クライアントでは、J2EE 標準のアプリケーションクライアント仕様がサポートされているだけでなく、iPlanet Application Server に直接アクセスすることができます。RMI/IIOP クライアントの配置記述子は、`app-client.xml` と `ias-app-client.xml` です。
- **リソース JAR ファイル** : リソースには、JDBC データソース、Java Mail、JMS などがあります。各 iPlanet Application Server リソースには、リソース XML ファイルがあります。

モジュールを配置した後にクラスローダが正しいクラスを検索できるように、すべてのモジュールのソースコードでパッケージ定義を使う必要があります。

DD 内の情報は宣言型であるため、ソースコードを変更しなくても変更できます。J2EE サーバは、実行時に読み込んだ DD 内の情報に従って動作します。

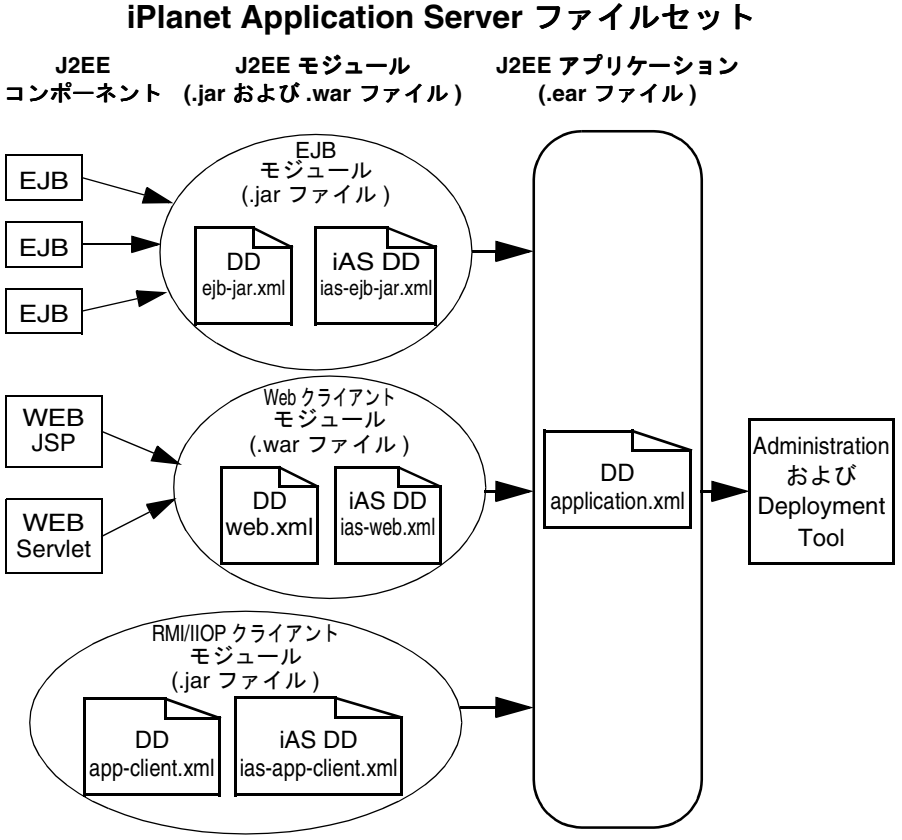
また、EJB JAR および Web モジュールは、次の図に示すように、`.war` ファイルまたは `.jar` ファイルとして個別にパッケージ化し、アプリケーションの外部に個別に配置することもできます。



アプリケーション

J2EE アプリケーションは、1 つまたは複数の J2EE モジュールの論理集合で、アプリケーション配置記述子によって関連付けられています。コンポーネントは、モジュールレベルまたはアプリケーションレベルでアセンブルできます。また、モジュールレベルまたはアプリケーションレベルで配置することもできます。

次の図は、配置する準備として、モジュールにコンポーネントをパッケージングして、iPlanet Application Server アプリケーションの `.ear` ファイルにまとめる方法を示しています。



各モジュールには、iPlanet Application Server DD および J2EE DD が定義されています。iPlanet Application Server Deployment Tool (iASDT) は、DD を使って、アプリケーションコンポーネントを配置し、iPlanet Application Server にリソースを登録します。

アプリケーションは、1 つまたは複数のモジュールと J2EE アプリケーション DD から構成されます。これらのすべてのアイテムが、Java ARchive (`.jar`) ファイル形式で、拡張子 `.ear` を持つ 1 つのファイルにパッケージングされます。

命名規則

EJB JAR および WAR モジュール名は、.war および .jar 拡張子ではなく、ファイル名の最初の部分によって識別されます。Application Server に配置する EJB JAR および WAR モジュール名は、一意でなければなりません。ejb-jar.xml ファイルの <ejb-name> 部分に指定するモジュールファイル名、EAR ファイル名、および EJB 名には、Java パッケージ方式の命名規則を使ってください。Java パッケージ方式の命名規則を使えば、名前の衝突は発生しません。この命名規則は、iPlanet Application Server だけでなく、ほかの J2EE アプリケーションサーバでも使うことをお勧めします。

モジュールおよびアプリケーションのアセンブリ

iPlanet Application Server 内でモジュールとアプリケーションをアセンブリ (パッケージ化) するときは、従来のあらゆる J2EE 定義仕様に準拠します。ただし、iPlanet Application Server 内でアセンブリするときは、iPlanet Application Server 固有の配置記述子 (ias-web.xml、ias-ejb-jar.xml など) を含めて、アプリケーションサーバの機能を拡張する必要があります。たとえば、iPlanet Application Server には、ロードバランス (クラスタ内のサーバ間にタスクを均等に分散する) やフェールオーバー (障害が発生したサーバのタスクを別のサーバに割り当てる) などの機能があります。

この節には次のトピックがあります。

- サンプルファイル
- WAR モジュールのアセンブリ
- EJB JAR アプリケーションのアセンブリ
- RMI/IIOP アプリケーションのアセンブリ

iPlanet Application Server では、モジュールまたはアプリケーションのパッケージ化を 3 つの方式で行うことができます。ここでは、3 つのツールについて簡単に説明し、詳細については各トピックで説明します。

- CLI ツール : コマンドラインインタフェースをアセンブリツールとして使うときは、JAR ファイルと、Ant の自動アセンブリ機能を使います。Ant は、Jakarta Apache に組み込まれている Java ベースのビルドツールです。

  http://jakarta.apache.org/ant/

- Deployment Tool : iPlanet Application Server に組み込まれている Deployment Tool (配置ツール) を使って、J2EE のアプリケーションとモジュールをアセンブリおよび配置することができます。

- Visual Café プラグイン : iPlanet の Visual Café プラグインは、WebGain® Visual Café ツールと iPlanet Application Server を統合します。

http://www.iplanet.com/products/application_server_plug/home_2_1_1aj.html

Visual Café の開発機能を使って、iPlanet Application Server 固有の DD、WAR ファイル、および JAR ファイルの作成を自動化できます。

サンプルファイル

J2EE サンプルアプリケーションをアセンブリする前に、次の Java ソースをコンパイルする必要があります。

- Servlet (`GreeterServlet.java`)
- EJB (`GreeterHome.java`、`GreeterEJB.java`、`Greeter.java`)
- JSP (`GreeterView.jsp`)
- スタティックファイル (`HWSample`)

これらのファイルはすべて、次の場所にあります。

http://developer.iplanet.com/docs/articles/packaging/packaging_print.jsp

コンパイルプロセスは、Ant ツールを使って自動化できます。関連するコーディングについては、次の場所の「Compile」の節を参照してください。

http://developer.iplanet.com/docs/articles/packaging/AntCompile.html

完成した Ant ビルドファイルについては、次の Ant XML ファイルを参照してください。

http://developer.iplanet.com/docs/articles/packaging/Ant.xml

WAR モジュールのアセンブリ

この節では、WAR モジュールの 3 つのアセンブリ手順について説明します。

- コマンドラインインタフェース (CLI) の使用法
- 配置ツールの使用法
- Visual Café プラグインの使用法

コマンドラインインタフェース (CLI) の使用法

CLI を使って WAR モジュールを作成するときは、次の手順で行います。

1. 作業ディレクトリ *working_dir*/cli を作成します。
2. web.xml および ias-web.xml という名前の 2 つの配置記述子を作成します。次の例を参照してください。

   http://developer.iplanet.com/docs/articles/packaging/web.xml

   http://developer.iplanet.com/docs/articles/packaging/ias-web.xml

ヒント 配置記述子を初めて作成するときは、配置ツールを使うこともできます。生成された WAR ファイルから配置記述子を作成できます。

3. 手順 1 で作成したディレクトリに WAR ファイルの内容を移動します。詳細については、次のサイトを参照してください。

   http://developer.iplanet.com/docs/articles/packaging/war.html

4. ディレクトリ *working_dir*/cli/assemble/war に移動します。
5. 次のコマンドを実行します。

   ```
   jar -cvf helloworldWar.war *
   ```

   WAR ファイル helloworldWar.war が作成されます。

ヒント CLI アセンブリプロセスは、Ant ツールを使って自動化できます。詳細については、次の URL を参照してください。

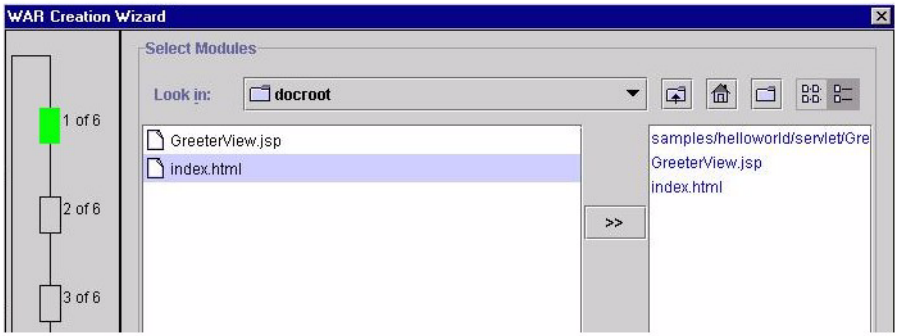
http://developer.iplanet.com/docs/articles/packaging/AntCompile.html

配置ツールの使用法

iPlanet Deployment Tool を使って WAR モジュールを組み立てるときは、次の手順で行います。

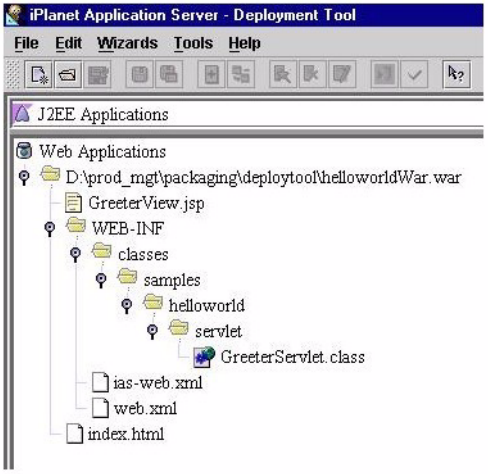
1. 配置ツールを使って、ディレクトリ *working_dir*/deploytool に helloworld.war という名前の新しい WAR ファイルを作成します。

2. Deployment Tool ウィザードを使って、GreeterView.jsp、index.html、GreeterServlet.class という Web ファイルを挿入します。

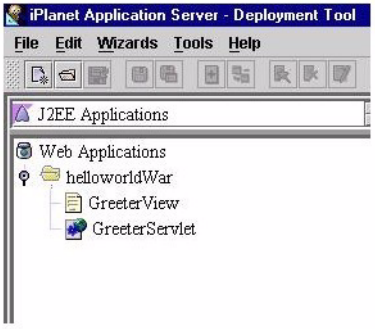


3. 画面の右下の「Resolve」ボタンをクリックして、ファイルを解決します。
4. 「完了」をクリックします。war ファイル *working_dir*/deploytool/helloworldWar.war が作成されます。

   記述子 (web.xml および ias-web.xml) が Deployment Tool によって作成されています。
5. 以下の図は、WAR モジュールをファイル表示で表示しています。



以下の図は、コンポーネント表示で表示しています。



ヒント CLI を使ってアセンブリする場合でも、初めは Deployment Tool を使ってパッケージ化することをお勧めします。

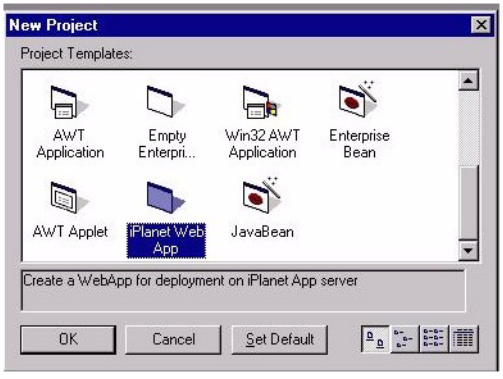
Visual Café プラグインの使用法

iPlanet Visual Café プラグインを使って WAR モジュールをアセンブリするときは、次の手順で行います。

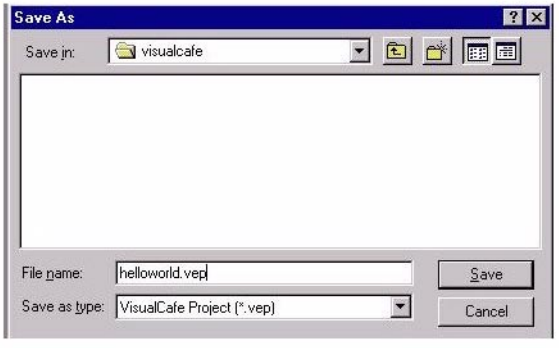
1. iPlanet Application Server 用の Visual Café プラグインをダウンロードしてインストールします。

```
http://www.iplanet.com/products/application_server_plug/home_2_1_1aj.html
```

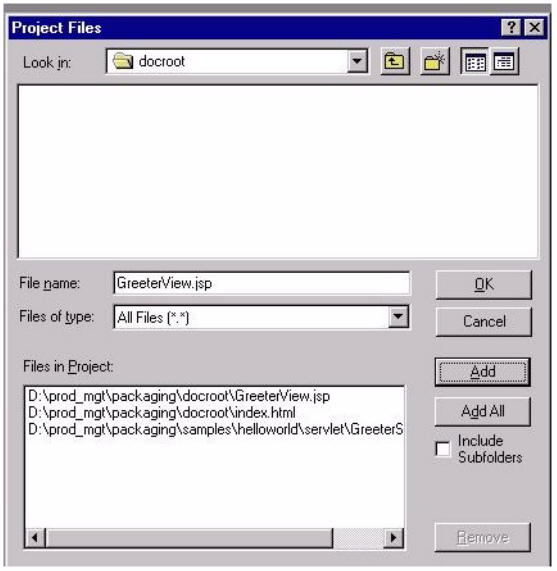
2. 新しい iPlanet Application Server Web Application を *working_dir*/visualcafe に作成します。



このファイルに helloworld.vep という名前を付けます。



3. これらの Web ファイルをアプリケーション GreeterServlet.java、GreeterView.jsp、index.html に挿入します。



4. 必要なヘルパークラスを追加します。
5. ソース (`GreeterServlet.java`) をコンパイルして、アプリケーションを配置します。配置記述子は、モジュールを配置したときに作成されます。

注 iPlanet Application Server 用の Visual Café プラグインの詳細については、ダウンロードしたプラグインに付属しているマニュアルを参照してください。

EJB JAR アプリケーションのアセンブリ

この節では、EJB JAR モジュールの 2 つのアセンブリ手順について説明します。

- コマンドラインインタフェース (CLI) の使用法
- 配置ツールの使用法

コマンドラインインタフェース (CLI) の使用法

CLI を使って J2EE アプリケーションを作成するときは、次の手順で行います。

1. *working_dir*`/cli` という名前の作業ディレクトリを作成します。
2. 配置記述子 `application.xml` を作成します。次の例を参照してください。

http://developer.iplanet.com/docs/articles/packaging/application.xml

3. 手順 1 で作成したディレクトリに配置記述子、WAR ファイル、および EJB JAR ファイルを移動します。移動するファイルの一覧は、次のサイトを参照してください。

   http://developer.iplanet.com/docs/articles/packaging/app.html

4. 作業ディレクトリに移動します。
5. 次のコマンドを実行します。

   ```
   jar -cvf helloworld.ear *
   ```

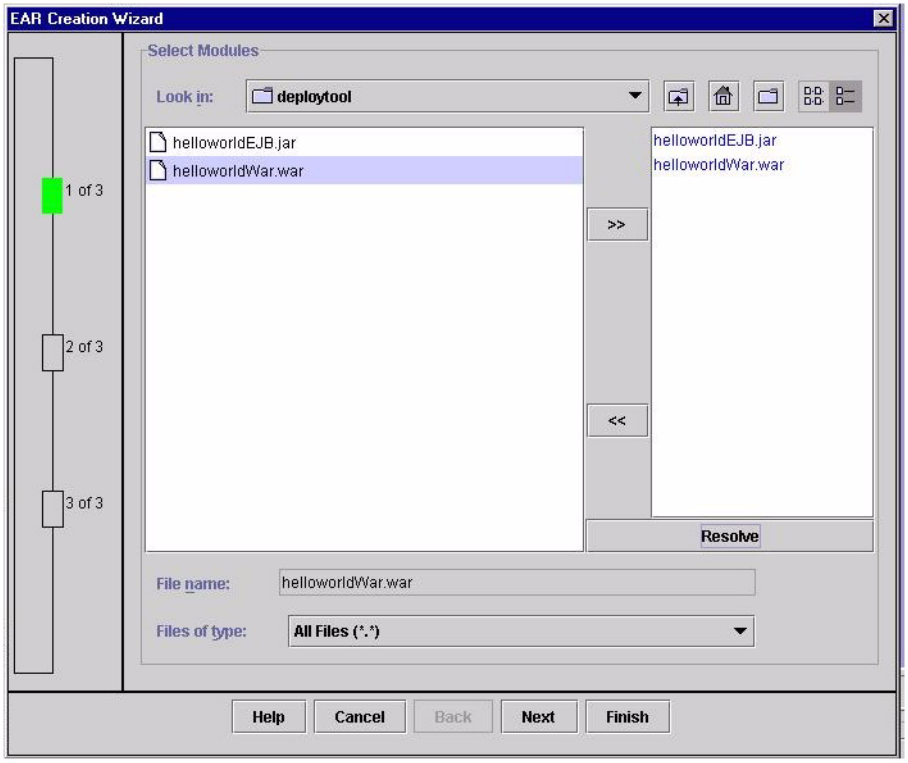
   J2EE アプリケーション helloworld.ear が作成されます。

配置ツールの使用法

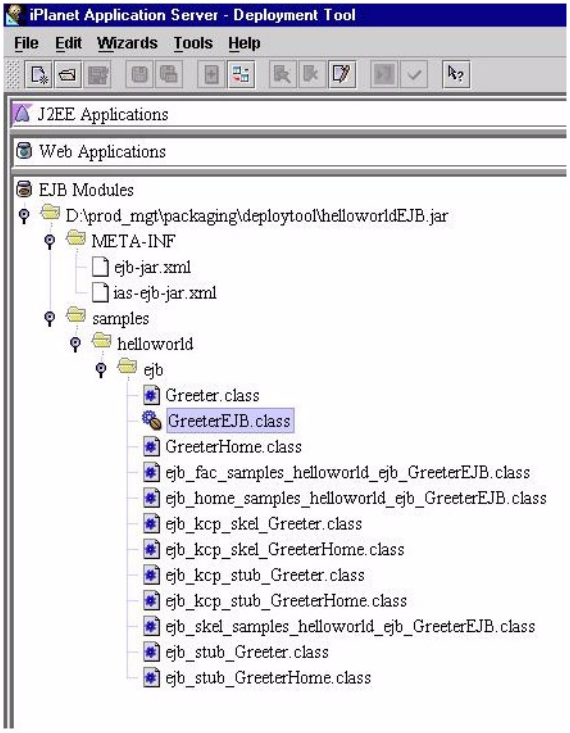
iPlanet Deployment Tool を使って J2EE アプリケーションを組み立てるときは、次の手順で行います。

1. 配置ツールを使って、ディレクトリ *working_dir*/deploytool に helloworld.ear という名前の新しい EAR ファイルを作成します。

2. Deployment Tool ウィザードを使って、EJB JAR ファイル helloworldEJB.jar および WAR ファイル helloworldWAR.war を挿入します。



3. 画面の右下の「Resolve」ボタンをクリックして、ファイルを解決します。
4. アプリケーションのルートまでのパスを削除して、「更新」ボタンをクリックします。
5. 「完了」をクリックします。EAR ファイル *working_dir*`/deploytool/helloworld.ear` が作成されます。配置記述子 (`application.xml`) が作成されています。
6. 以下の図は、アプリケーションを EAR ファイル表示で表示しています。



以下の図は、EAR コンポーネント表示で表示しています。

7. 「ファイル」メニューの「検証」を選択して、結果を確認します。

ヒント CLI を使ってアセンブリする場合でも、初めは Deployment Tool を使ってパッケージ化することをお勧めします。

RMI/IIOP アプリケーションのアセンブリ

この節では、RMI/IIOP アプリケーションのアセンブリについて簡単に説明します。ただし、第 10 章「CORBA ベースクライアントの開発と配置」の内容を理解していることを前提としています。

RMI/IIOP アプリケーションの簡単なパッケージ化と配置の例については、次のサイトを参照してください。

`http://developer.iplanet.com/appserver/samples/pkging/docs/sampleD.html`

RMI/IIOP アプリケーションは、2 つのタイプに分類できます。

- **単純なクライアント** : iasacc.jar から提供されるコンテナサービス、およびアプリケーション xml を持たないクライアントです。
- **アプリケーションクライアントコンテナ** : 配置記述子を含み、追加サービスにアクセスできる J2EE 1.2 に準拠したクライアントです。

iPlanet Application Server では、RMI/IIOP アプリケーションをパッケージ化するときに、次の点に準拠する必要があります。

サーバサイド

- iPlanet Application Server Administration Tool から CXS (Corba eXecutive Service) を設定する
- Deployment Tool を使って、RMI/IIOP スタブおよびスケルトンを生成する

クライアントサイド

- 必要な JAR ファイル (`iasclient.jar`、`javax.jar`、`jms.jar`、`mail.jar`、および `servlet.jar`) をクライアントのクラスパスに含める
- EJB のクライアントサイドスタブを含める。クライアントサイドスタブは、Deployment Tool によって設定される。`iasacc.jar` (ACC のみ) を含める
- `application-client.xml` (ACC のみ) を含める

ヒント これらのファイルはすべて、`iasclient.tar` (UNIX の場合) または `iasclient.zip` (NT の場合) にあります。

モジュールおよびアプリケーションの配置

この節では、J2EE のアプリケーションおよびモジュールを iPlanet Application Server に配置する方法について説明します。この節には、次のトピックがあります。

- モジュールによる配置
- アプリケーションによる配置
- RMI/IIOP クライアントの配置
- スタティックコンテンツの配置
- 配置ツール
- 配置に関する一般的な規則

モジュールによる配置

モジュールとアプリケーションは、個別に配置できます。次の要素が共有コンポーネントにアクセスする場合は、モジュールベースで登録および配置することをお勧めします。

- ほかのモジュール
- J2EE アプリケーション
- RMI/IIOP クライアント (モジュールベースで登録すると、RMI/IIOP クライアント、Servlet、または EJB から Bean に共有アクセスできる)

モジュールを登録するには、次のコマンドを実行します。

```
iasdeploy deploymodule module_name
```

複数のモジュールを 1 つの EAR ファイルに結合すれば、上のコマンドを使って 1 つのモジュールとして配置できます。この操作は、EAR のモジュールを個別に配置する場合に似ています。モジュールベースで登録したときの実行時レジストリおよびファイルシステムについては、付録 B「実行時の注意事項」を参照してください。

別のモジュール配置方法については、311 ページの「配置ツール」を参照してください。

アプリケーションによる配置

J2EE アプリケーションを登録するには、次のコマンドを実行します。

```
iasdeploy deployapp app_name
```

モジュールベースで登録したときの実行時レジストリおよびファイルシステムについては、付録 B「実行時の注意事項」を参照してください。

別のアプリケーション配置方法については、311 ページの「配置ツール」を参照してください。

RMI/IIOP クライアントの配置

RMI/IIOP クライアントは、2 つの手順で配置します。

1. RMI/IIOP クライアントからアクセスする EJB JAR をインストールします。
2. 必要なクライアントファイルをパッケージ化し (309 ページの「RMI/IIOP アプリケーションのアセンブリ」を参照)、クライアントコードを作成して、クライアントマシンからクライアントを実行します。

スタティックコンテンツの配置

スタティックコンテンツ (HTML、イメージなど) は、Web サーバ上および iPlanet Application Server 上で管理できます。ただし、WAR が登録されているときは、スタティックコンテンツはアプリケーションサーバに配置されます。iPlanet Application Server に付属するパッケージサンプルではすべて、アプリケーションサーバ上でスタティックコンテンツを管理します。

たとえば、アプリケーションサーバ上のスタティックファイル `index.html` にアクセスするには、次のパスを使います。

`http://`*`server`*`:`*`port`*`/NASApp/&lt`*`context_root`*`/index.html`

配置ツール

この節では、モジュールとアプリケーションを配置するときに使用するツールについて説明します。次の配置ツールがあります。

- iasdeploy コマンド
- iPlanet Deployment Tool
- iPlanet Visual Café プラグイン

iasdeploy コマンド

`iasdeploy` コマンドは、CLI ツール の 1 つで、モジュールおよびアプリケーションをローカルサーバに登録および配置するときに使用できます。モジュールを配置するときは、次のコマンドを使います。

`iasdeploy deploymodule` *module_name*

アプリケーションを配置するときは、次のコマンドを使います。

`iasdeploy deployapp` *app_name*

iPlanet Deployment Tool

iPlanet Deployment Tool を使用して、モジュールとアプリケーションをローカルおよびリモートの iPlanet Application Server サイトに配置できます。このツールを使うには、次の手順で行います。

1. 配置する WAR、JAR、または EAR ファイルを開きます。これらのファイルは、個別に配置できます。
2. 「ファイル」メニューから「配置」を選択します。
3. 「登録」ボタンをクリックします。
4. 配置するファイルを登録します。
5. 登録先のサーバを強調表示し、「配置」ボタンをクリックします。

iPlanet Visual Café プラグイン

Visual Café の iPlanet プラグインを使用して、モジュールとアプリケーションを統合開発環境 (IDE) コンテキストに配置できます。このツールでは、アセンブリと配置が同時に行われます。詳細については、プラグインをダウンロードしたときに提供されるマニュアルを参照してください。

配置に関する一般的な規則

モジュールとアプリケーションを配置するときは、いくつかの一般的な規則に準拠する必要があります。ここでは、それらの規則について説明します。

アプリケーションまたはモジュールの再配置

アプリケーションまたはモジュールを再配置するときに、一部のファイルシステムの内容と Application Server のレジストリ設定が上書きまたは削除されないことがあります。この場合、再配置した後でも古い設定が残ることがあります。クリーンな環境に再配置するには、アプリケーションまたはモジュールを再配置する前に削除します。

iPlanet Application Server クラスタへの配置

アプリケーションを iPlanet Application Server サーバのクラスタに配置するときは、そのアプリケーションを各サーバに個別に登録する必要があります。共有情報は LDAP 上に保存されるため、クラスタ内のすべてのサーバからアクセスできますが、ファイルシステムエントリは各サーバ上になければなりません。

共有フレームワークへのアクセス

J2EE のアプリケーションとモジュールで共有フレームワーククラス (コンポーネント、ライブラリなど) を使用する場合、それらのクラスはアプリケーションやモジュールではなくシステムクラスパスに配置できます。共有ライブラリのサイズが大きい場合、そのライブラリを使用するすべてのモジュールにパッケージ化するときに、サーバへの登録に多くの時間が必要になります。また、同一クラスの複数のインスタンスが独自のクラスローダを使用すると、リソースの浪費になります。

システムクラスローダの詳細については、付録 B「実行時の注意事項」を参照してください。

iPlanet Application Server に付属する Cocoon サンプル (XML サンプルの一部) は、フレームワークの使用例です。

XML DTD について

DTD (Document Type Definition : ドキュメントタイプ定義) は、配置記述子 (DD) の XML 書式を定義しています。DD には、アプリケーションレベル記述子と、コンポーネントレベル記述子の 2 つのレベルがあります。

iPlanet Application Server では、アプリケーションの実行に DD が必要です。DD は、XML ファイルで、アプリケーションを構成する J2EE モジュール (Servlet、JSP、EJB など) の配置情報について記述したメタデータを含みます。各 XML ファイルの情報は、iPlanet Application Server 内部のレジストリに格納されています。

各アプリケーションモジュールには J2EE DD ファイルが必要です。さらに、各アプリケーションコンポーネントはグローバルな固有識別子、つまり、`GUID` に関連付けられている必要があります。

iPlanet Application Server によってサポートされている DD のタイプは次のとおりです。

- アプリケーション DD
- Web アプリケーション DD と iPlanet Application Server Web アプリケーション DD
- EJB DD と iPlanet Application Server EJB DD

- アプリケーションクライアント DD と iPlanet Application Server RMI/IIOP クライアント DD
- iPlanet Application Server リソース DD

J2EE 標準記述子

J2EE プラットフォームでは、パッケージングおよび配置機能が提供されます。これらの機能では、コンポーネントおよびアプリケーションの標準パッケージとして JAR ファイルが使われ、パラメータのカスタマイズには XML ベースの DD が使われます。J2EE パッケージングおよび配置プロセスの詳細については、『Developing Enterprise Applications with the J2EE, v 1.0』の第 7 章を参照してください。

J2EE 標準 DD については、J2EE 仕様書のバージョン 1.1 に規定されています。J2EE 標準 DD の詳細は、次の仕様書を参照してください。

- 『Java 2 Platform Enterprise Edition Specification, v1.2』の第 8 章「Application Assembly and Deployment - J2EE:application XML DTD」
- 『Java 2 Platform Enterprise Edition Specification, v1.2』の第 9 章「Application Clients - J2EE:application-client XML DTD」
- 『JavaServer Pages Specification, v1.1』の第 7 章「JSP Pages as XML Documents」
- 『JavaServer Pages Specification, v1.1』の第 5 章「Tag Extensions」
- 『Java Servlet Specification, v2.2』の第 13 章「Deployment Descriptor」
- 『Enterprise JavaBeans Specification, v1.1』の第 16 章「Deployment Descriptor」

配置記述子を作成する

iPlanet Application Server アプリケーション用のすべての DD は、配置ツールを使って作成します。その手順の詳細については、配置ツールのオンラインヘルプを参照してください。

ドキュメントタイプ定義 (DTD)

DTD は、DD ファイルの構造とクラス プロパティを記述します。各 DD には、その他のすべての要素 (またはサブ要素) を完全に含む 1 つの要素があります。

XML ファイルにある要素の記述はテーブル形式で示されます。これらの要素テーブルには、要素の目的と設定パラメータを記述するいくつかのフィールドがあります。一部の要素は階層化されています。つまり、パラメータがほかの要素 (サブ要素) を持っている場合があります。パラメータに要素が含まれている場合、その要素の記述は、その要素について記述している別のテーブルにあります。表 11-1 はサポートされている DTD エントリを示しています。

表 11-1 ドキュメントタイプ定義 (DTD)

タイプ	説明
要素	XML ファイルに表示される要素の名前と要素の説明
サブ要素	この要素に含まれる要素を一覧表示する

iPlanet Application Server レジストリ

iPlanet Application Server レジストリは、ツリー構造のアプリケーションメタデータのコレクションであり、アクティブメモリ内またはすぐにアクセス可能な Directory Server 上で連続して使用可能です。iPlanet Application Server が、Servlet、EJB、およびその他のアプリケーションリソースへのアクセスを増やすプロセスを登録と呼びます。これは、各アイテムの iPlanet Application Server レジストリへのエントリの配置がこのプロセスに関連するためです。

iPlanet Application Server Administrator Tool を使うと、実行時にレジストリの一部の情報を変更できます。レジストリ および Administrator Tool の詳細については、iPlanet Application Server 配置ツールのオンライン ヘルプおよび『管理者ガイド』を参照してください。

グローバルに固有な識別子

GUID は EJB、Servlet、またオプションで JSP にも割り当てられる 128 ビットの 16 進数です。GUID は配置ツールによって自動的に生成されます。

GUID はグローバルに固有であることが保証されているため、iPlanet Application Server アプリケーションなどの大規模な異機種システムのコンポーネントを識別するには最適です。

GUID は通常、配置ツールによって自動的に割り当てられます。GUID は kguidgen という名前のユーティリティを使って手動で作成できます。kguidgen は、デフォルトで BasePath/bin ディレクトリにインストールされます。このディレクトリは検索パス (GUID を生成するための PATH 環境変数) に一覧表示する必要があります。

新しい GUID を生成するには、コマンドラインまたはウィンドウから kguidgen を実行するだけです。

Web アプリケーション XML DTD

この節では、Web アプリケーション、Web アプリケーションモジュール、および Web アプリケーション DD について説明します。DD 配置ツールを使って作成されます。詳細については、iPlanet Application Server 配置ツールのオンラインヘルプおよび『管理者ガイド』を参照してください。

Web アプリケーションの概要

Web サーバで実行される Web アプリケーションは、Servlet、JSP、JSP Tag ライブラリ、HTML ページ、クラス、およびその他のリソースから構成されます。Web アプリケーションの場所は、Web サーバ内の特定のパスでそのルートが決められます。DD によって Web アプリケーションが分散可能とマークされていないかぎり、その Web アプリケーションのインスタンスは、指定した時刻に 1 台の仮想マシン (VM) でだけ実行する必要があります。分散可能とマークされている場合、そのアプリケーションは指定した時刻に複数の VM で実行できますが、Java Servlet 仕様書バージョン 2.2 に示されたより厳しい規則に従う必要があります。

Web アプリケーションは、以下のアイテムから構成されます。

- Servlet
- JSP
- ユーティリティクラス
- スタティックドキュメント (HTML、イメージ、サウンドなど)
- クライアントサイドアプレット、Bean、およびクラス
- 上記のすべてのアイテムをバンドルした記述的メタ情報

Web アプリケーションを作成するには、まず、モジュール DD とともに、必要なすべての Web コンポーネントを 1 つの Web アプリケーションモジュールにまとめます。次に、その Web アプリケーションモジュールを、アプリケーション DD とともに J2EE アプリケーションが利用するほかのすべてのモジュールと一緒にパッケージングして、そのまま配置できる最終的な Web アプリケーションにします。J2EE の組み立てと配置の詳細については、J2EE 仕様書の第 8 章を参照してください。

Web アプリケーション XML DTD

この節では、iPlanet Application Server 固有の Web アプリケーション DD の XML DTD について説明します。標準の J2EE アプリケーション DD については、J2EE 仕様書の第 8.4 節を参照してください。

Web アプリケーション DD は、次の情報を指定する要素の定義をサポートしています。

- Servlet 情報
- セッション情報
- EJB 参照情報
- リソース参照情報
- Servlet 情報の指定

iPlanet Application Server Web アプリケーションを指定する要素

表 11-2 は、iPlanet Application Server Web アプリケーション DD の ルート要素とともに使う `<ias-web-app>` 要素およびサブ要素を示しています。

表 11-2 `<ias-web-app>` サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
`Servlet`	0 回以上	要素	なし	Servlet の設定情報を含む
`session-info`	0 回または 1 回	要素	なし	セッション情報を指定する
`ejb-ref`	0 回以上	要素	なし	J2EE XML ファイルの `ejb-ref` エントリと一致する JNDI 絶対名の保存場所を指定する
`resource-ref`	0 回以上	要素	なし	J2EE XML ファイルの `ejb-ref` エントリと一致する `ejb-link` である JNDI 絶対名の保存場所を指定する
`nlsinfo`	0 回または 1 回	要素	なし	NLS 設定記述子
`role-mapping`	0 回または複数回	要素	なし	LDAP ロールマッピング記述子

Servlet 設定情報を指定する要素

表 11-3 は、Servlet についての設定情報を持つ `servlet` サブ要素を示しています。

表 11-3　Servlet サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
servlet-name	1 回だけ	文字列	なし	Servlet 名。この名前は、J2EE Web アプリケーション XML の servlet-name パラメータと完全に一致する必要がある
guid	1 回だけ	文字列	なし	Servlet の guid を表す文字列
servlet-info	0 回または 1 回	要素	なし	Servlet のオプション特性
validationRequired	0 回または 1 回	ブール値	false	入力パラメータを確認する必要があるかどうかを指定する
error-handler	0 回または 1 回	文字列	なし	Servlet のエラーハンドラを記述する
パラメータ	0 回以上	要素	なし	確認するすべての入力パラメータを記述する
param-group	0 回以上	要素	なし	各パラメータグループは、イベントソース名とそれに関連付けられているパラメータによって示される

Servlet 特性を指定する要素

表 11-4 は、Servlet のオプションの特性を記述する servlet-info サブ要素を示しています。

表 11-4　servlet-info サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
sticky	0 回または 1 回	ブール値	false	sticky が true である場合、Servlet によって、セッションアフィニティが示され、セッションが存在しない場合だけロードバランスが実行される。あるエンジンでセッションが作成されると、sticky Servlet に対する後続のリクエストがその同じエンジンに引き続きルートされる
encrypt	0 回または 1 回	ブール値	false	Servlet への通信が暗号化されている (true) か、暗号化されていない (false) かを示すオプションのフラグ
caching	0 回または 1 回	要素	なし	Servlet のキャッシュの基準を指定する

表 11-4　servlet-info サブ要素 (続き)

サブ要素	繰り返し規則	内容	デフォルト	説明
number-of-singles	0 回または 1 回	整数	10	SingleThread モードが使われたときの Servlet プールのオブジェクト数
disable-reload	0 回または 1 回	ブール値	false	ダーティなとき、Servlet の再読み込みを無効にするために使う。論理値は true または false
server-info	0 回または複数回	要素	なし	サーバ、ロードバランスの有効 / 無効などの、サーバのオプション情報
server-ip	1 回だけ	文字列	なし	サーバの IP アドレス
server-port	1 回だけ	文字列	なし	Executive Server のポート番号
sticky-lb	0 回または複数回	ブール値	servlet-info sticky 設定	スティッキーロードバランスを設定する。論理値は true または false 設定すると、servlet-info の設定が無効になる
enable	0 回または複数回	ブール値	true	サーバが有効かどうかを指定する。論理値は true または false

Servlet 確認を指定する要素

表 11-5 は Servlet についての入力を確認する必要があるかどうかの調査に使う validation-required サブ要素を示しています。

表 11-5　validation-required サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
validation-required	1 回だけ	ブール値	false	入力パラメータを確認する必要があるかどうかを指定する

Servlet のキャッシュを指定する要素

表 11-6 は、Servlet のキャッシュ基準を記述する caching サブ要素を示しています。キャッシュ要素を定義しないと、caching は無効になります。

表 11-6　caching サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
cache-timeout	1 回だけ	整数	なし	Servlet のキャッシュのタイムアウトを設定する (単位は秒)。値が 0 の場合、キャッシュは無効
cache-size	1 回だけ	整数	なし	キャッシュのサイズを設定する。値が 0 の場合、キャッシュは無効
cache-criteria	1 回だけ	シンタックスが、入力パラメータリストの任意の arg の値である場合の文字列。詳細については、381 ページの「Servlet の結果のキャッシュ」を参照してください。	なし	カンマ区切りの記述子の文字列を含んでいるキャッシュ基準式。各記述子が、Servlet に対する入力パラメータの一つとの比較を定義している
cache-option	1 回だけ	TIMEOUT_CREATE または TIMEOUT_LASTACCESS の文字列	TIMEOUT_LASTACCESS	キャッシュのタイムアウトオプションを設定する

キャッシュ基準設定とキャッシュオプションの例

次の例は、共通の使用法とキャッシュ基準要素の設定値を示しています。

例 1

```
<cache-criteria>EmployeeCode</cache-criteria>
```

これは、EmployeeCode が入力パラメータリストにあれば、キャッシュが有効であることを意味します。

例 2

```
<cache-criteria>stock=NSCP</cache-criteria>
```

これは、stock 入力パラメータ値が NSCP であれば、キャッシュが有効であることを意味します。

例 3

```
<cache-criteria>*</cache-criteria>
```

これは、入力パラメータの値がキャッシュされる値と同じであれば、キャッシュが有効であることを意味します。

例 4

```
<cache-criteria>dept=sales|marketing|support</cache-criteria>
```

dept パラメータの値が営業、マーケティング、またはサポートであれば、キャッシュが有効であることを意味します。

例 5

```
<cache-criteria>salary=40000-60000</cache-criteria>
```

これは、入力パラメータ salary の値が 40000 ～ 60000 であれば、キャッシュが有効であることを意味します。

例 6

```
<cache-option>TIMEOUT_CREATE</cache-option>
```

これは、作成時刻からキャッシュタイムアウト値を決めることを意味します。

例 7

```
<cache-option>TIMEOUT_LASTACCESS</cache-option>
```

これは、前回アクセスした時刻に基づいてキャッシュタイムアウトを決めることを意味します。

Servlet パラメータを指定する要素

表 11-7 は、確認する入力パラメータを記述する parameters 要素を示しています。

表 11-7 parameters サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
param	0 回以上	要素	なし	確認のために適用される名前と規則によって各パラメータを指定する

Servlet サブパラメータを指定する要素

表 11-8 は、確認のために適用される名前と規則によって各パラメータが示される param サブ要素を示しています。

表 11-8 param サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
param-name	1 回だけ	文字列	なし	入力パラメータの名前

表 11-8　param サブ要素 (続き)

サブ要素	繰り返し規則	内容	デフォルト	説明
input-fields	1 回だけ	要素	なし	入力パラメータの詳細を記述する

Servlet 入力フィールドを指定する要素

表 11-9 は、入力パラメータの詳細を記述する input-field サブ要素を示しています。

表 11-9　input-field サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
input-required	0 回または 1 回	ブール値	なし	入力パラメータが必要かどうか、つまり、フィールドを入力リストの一部とするかどうかを指定する
input-rule	0 回または 1 回	文字列	なし	入力パラメータの確認のために適用されている入力規則を指定する
format	0 ～ 1 回	日付 / 時刻形式の文字列	なし	入力パラメータの確認のために適用される日付 / 時刻の形式を指定する
in-session	0 ～ 1 回	文字列	なし	確認のためのパラメータがキャッシュ (セッション) にあるかどうかを指定する
param-error-handler	0 回または 1 回	文字列	なし	パラメータのエラーハンドラを指定する

Servlet パラメータグループを指定する要素

表 11-10 は、各パラメータグループがイベントソース名とそれに関連付けられているパラメータによって示される param-group サブ要素を示しています。

表 11-10　param-group サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
param-group-name	1 回だけ	文字列	なし	パラメータグループの名前
param-input	1 回または複数回	文字列	なし	パラメータグループに関連付けられたパラメータ入力の名前

セッション情報を指定する要素

表 11-11 は、セッション情報を指定する session-info 要素を示しています。

表 11-11 session-info サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
impl	1 回だけ	distributed または lite の文字列	なし	セッションは、分散フォールトトレランスセッション、またはライトウェイトローカル専用セッションのどちらか
timeout-type	0 回または 1 回	last-access または creation の文字列	last-access	通常、セッションタイムアウトは「最終アクセスからの時間」で測定される。または、絶対タイムアウトを「セッション作成からの時間」として指定することも可能
タイムアウト	0 回または 1 回	分を表す正の整数	30 分	タイムアウト前のセッションタイムアウト分数。指定しないと、システム全体のデフォルトセッションタイムアウトが使われる この値と web.xml の <session-timeout> 値は、同じ場所に内部的に保存される。一方の値を変更すると、もう一方の値も変更される
secure	0 回または 1 回	ブール値	false	保護されている (HTTPS) サーバからだけセッションが見えるように指定する
domain	0 回または 1 回	cookie を設定するドメインの文字列名	なし	セッションドメイン cookie の設定に使われるアプリケーションドメインを指定する ドメインの文字列引数は、少なくとも 2 つ、または 3 つのピリオドを持つ必要がある (3 ピリオドドメインは acme.co.uk などのドメインに適用) ドメインが acme.com に設定されると、セッションは Who.acme.com、bar.asme.com などから見えるようになる

表 11-11　session-info サブ要素 (続き)

サブ要素	繰り返し規則	内容	デフォルト	説明
path	0 回または 1 回	/ で始まるセッション cookie の URL の文字列値	cookie を作成した URL	セッション cookie のパスを指定する。パスが存在しない場合、cookie で設定されたものと同じパスが使われていることを意味する たとえば、パス /phoenix は /phoenix/types/bird.html や /phoenix/birds.html と一致する
scope	0 回または 1 回	ほかのアプリケーションを識別する文字列	なし	このセッションにアクセスできるほかのアプリケーションを選択するグループ化の名前 たとえば、ドメインが acme.com に設定されていると、セッションは Who.acme.com、bar.acme.com などから見えるようになる
dsync-type	0 回または 1 回	dsync-local または dsync-distributed の文字列	なし	DSync セッションのタイプを指定する

EJB 参照情報を指定する要素

表 11-12 は、J2EE XML ファイルの ejb-ref エントリと一致する ejb-link の絶対名 jndi-name の保存場所である ejb-ref サブ要素を示しています。

表 11-12　ejb-ref サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
ejb-ref-name	1 回だけ	文字列	なし	対応する J2EE XML ファイルの ejb-ref エントリの ejb-link
jndi-name	1 回だけ	文字列	なし	絶対 jndi-name

リソースの参照情報を指定する要素

表 11-13 は、J2EE XML ファイルの resource-ref エントリと一致する resource-ref の絶対名 jndi-name の保存場所である resource-ref サブ要素を示しています。

表 11-13　resource-ref サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
resource-ref-name	1 回だけ	文字列	なし	対応する J2EE XML ファイルの resource-ref エントリの resource-ref 名
jndi-name	1 回だけ	文字列	なし	絶対 jndi-name

NLS 設定を指定する要素

表 11-14 は、アプリケーションの NLS 設定についての情報を持つ nlsinfo サブ要素を示しています。

表 11-14　nlsinfo サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
locale-charset-map	0 回または複数回	要素	なし	ロケールとそれに対応する文字セットを含む
default-locale	1 回だけ	文字列	なし	デフォルトロケール

ロケール文字セットを指定する要素

表 11-15 は、ロケールとそれに対応する文字セットの記述子情報を持つ locale-charset-map サブ要素を示しています。

表 11-15　locale-charset-map サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
locale	1 回だけ	文字列	なし	ロケール名
charset	1 回だけ	文字列	なし	デフォルトロケール

ロールマッピングを指定する要素

表 11-16 は、ロールを LDAP ユーザ、グループなどにマップする記述子情報を持つ role-mapping サブ要素を示しています。

表 11-16　role-mapping サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
role-name	1 回だけ	文字列	なし	<security-role> 要素で参照されるロールの名前
role-impl	1 回だけ	要素	なし	特定の role-name を構成する LDAP グループおよびユーザの表現に使われる文字列。role-impl には、任意の数のグループやユーザを指定可能

ロール IMPL を指定する要素

表 11-17 は、ロール実装の記述子情報を持つ role-impl サブ要素を示しています。

表 11-17　role-impl サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
group	0 回または複数回	文字列	なし	特定の LDAP グループに対応する LDAP 固有の文字列
user	0 回または複数回	文字列	なし	特定の LDAP ユーザに対応する LDAP 固有の文字列

EJB XML DTD

この節では、EJB 配置記述子によって使われる EJB DTD について説明します。DD は、配置ツールを使って作成されます。DD 作成の詳細については、配置ツールのオンライン ヘルプを参照してください。

EJB JAR ファイルの内容

Enterprise JavaBeans をパッケージするときに使われる標準フォーマットは EJB-JAR ファイルです。このフォーマットは Bean プロバイダとアプリケーション編成者間、およびアプリケーションの編成者と配置者間の規約です。

EJB-JAR ファイルには、DD だけでなく、次のすべてのクラスファイルが含まれている必要があります。

- Enterprise JavaBeans クラス
- Enterprise Helper クラス
- Enterprise JavaBeans のホームおよびリモートのインタフェース
- Bean がエンティティ Beans である場合はプライマリキークラス

さらに、EJB-JAR ファイルには、Enterprise JavaBeans クラスとリモートおよびホームインタフェースが依存しているすべてのクラスとインタフェース用のクラスファイルが含まれている必要があります。

パラメータのパス規則

Servlet または EJB が、同じプロセス内にある別の Bean を呼び出すと、デフォルトでは、iPlanet Application Server は呼び出したすべてのパラメータを整理しません。この最適化によって、同じプロセス内の Bean の呼び出しは、`by-value` のセマンティックが厳密に使われる場合よりも効率が向上します。Bean に渡されたパラメータが常に値によって渡されていることを確認する場合もあります。iPlanet Application Server では、Bean または Bean 内の特定のメソッドを `pass-by-value` セマンティックを要求しているものとしてマークできます。EJB によって使われるパラメータを渡すメソッドは、`pass-by-value` 要素によって定義されます。詳細については、`session` (表 11-20) または `entity` 要素 (表 11-21) にある `pass-by-value` 要素の説明を参照してください。このオプションでは呼び出しのオーバーヘッドが大幅に増大してパフォーマンスが低下するので、デフォルト値は `false` になります。

EJB iPlanet Application Server XML DTD

EJB-JAR ファイル用の iPlanet Application Server 固有の XML DTD は次のとおりです。

EJB-JAR を指定する要素

表 11-18 は、iPlanet Application Server Web アプリケーション DD ルート要素である `ias-ejb-jar` 要素を示しています。

表 11-18 `ias-ejb-jar` 要素

サブ要素	繰り返し規則	内容	デフォルト	説明
`enterprise-beans`	1 回だけ	要素	なし	`enterprise-beans` 要素には 1 つまたは複数の Enterprise JavaBeans の宣言がある

Enterprise JavaBeans を指定する要素

表 11-19 は、1 つまたは複数の Enterprise JavaBeans の宣言を持つ enterprise-beans サブ要素を示しています。

表 11-19 enterprise-beans サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
session	1 回またはそれ以外	要素	なし	iPlanet Application Server 固有のすべてのセッション Beans に関連する配置情報を宣言する要素
エンティティ	1 回またはそれ以外	要素	なし	iPlanet Application Server 固有のすべてのエンティティ Beans に関連する配置情報を宣言する要素

セッションを指定する要素

表 11-20 は、iPlanet Application Server 固有のすべてのセッション Beans に関連する配置情報を宣言する session サブ要素を示しています。ejb-name は、J2EE XML ファイルで宣言された ejb-name と 1 対 1 で一致する必要があります。

表 11-20 session サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
ejb-name	1 回だけ	文字列	なし	EJB の名前
guid	1 回だけ	文字列	なし	該当する EJB の guid
pass-timeout	1 回だけ	正の整数	なし	コンテナによって使われる受動型タイムアウト (単位は秒)。この値は管理ツールの実行時に変更可能
pass-by-value	1 回だけ	ブール値	なし	「true」の場合、EJB に対するすべての呼び出しパラメータが整理される。「false」であり、Bean が同じ場所にある場合、by-value のセマンティックは厳密に保証されない
session-timeout	1 回だけ	正の整数	なし	セッションタイムアウト (単位は分)
ejb-ref	0 回以上	要素	なし	J2EE XML ファイルの ejb-ref エントリと一致する ejb-link である絶対名 jndi-name の保存場所
resource-ref	0 回以上	要素	なし	J2EE XML ファイルの resource-ref エントリと一致する resource-ref である絶対名 jndi-name の保存場所

表 11-20　session サブ要素 (続き)

サブ要素	繰り返し規則	内容	デフォルト	説明
failoverrequired	0回または1回	ブール値	なし	フェールオーバーが必要かどうかを示す

エンティティを指定する要素

表 11-21 は、iPlanet Application Server 固有のすべてのエンティティ Beans に関連する配置情報を宣言する entity サブ要素を示しています。ejb-name は、J2EE XML ファイルで宣言された ejb-name と 1 対 1 で一致する必要があります。

表 11-21　entity サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
ejb-name	1 回だけ	文字列	なし	EJB の名前
guid	1 回だけ	文字列	なし	該当する EJB の guid
pass-timeout	1 回だけ	正の整数	なし	コンテナによって使われる受動型タイムアウト (単位は秒)。この値は管理ツールの実行時に変更可能
pass-by-value	1 回だけ	ブール値	なし	「true」の場合、EJB に対するすべての呼び出しパラメータが整理される。「false」であり、Bean が同じ場所にある場合、by-value のセマンティックは厳密に保証されない
persistence-manager	0回または1回	要素	なし	パーシスタンス情報を指定する
pool-manager	0回または1回	要素	なし	キャッシュのプール属性の記述子
ejb-ref	0 回以上	要素	なし	J2EE XML ファイルの ejb-ref エントリと一致する ejb-link である絶対名 jndi-name の保存場所
resource-ref	0 回以上	要素	なし	J2EE XML ファイルの resource-ref エントリと一致する resource-ref である絶対名 jndi-name の保存場所
failover-required	0回または1回	ブール値	false	フェールオーバーが必要かどうかを示す

表 11-21　entity サブ要素 (続き)

サブ要素	繰り返し規則	内容	デフォルト	説明
iiop	0回または1回	ブール値	false	Bean で RMI/IIOP クライアントが有効になっているかどうかを示す
role-mapping	0 回または複数回	要素	なし	ロールマッピングを作成する記述子

パーシスタンスマネージャを指定する要素

表 11-22 は、パーシスタンスマネージャ固有のすべての情報を定義する persistence-manager サブ要素を示しています。

表 11-22　persistence-manager サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
factory-class-name	1 回だけ	文字列	なし	パーシスタンスマネージャ名のファクトリクラス
properties-file-location	1 回だけ	文字列	なし	プロパティファイルの JAR ファイル内の場所

プールマネージャを指定する要素

表 11-23 は、プールマネージャ固有のすべての情報を定義する pool-manager サブ要素を示しています。

表 11-23　pool-manager サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
commit-option	1 回だけ	COMMIT_OPTION_C の文字列値	COMMIT_OPTION_C	オプション C: トランザクション間で、コンテナは「ready」インスタンスをキャッシュしない。詳細については、EJB 仕様書バージョン 1.1 の第 9.1.10 節を参照
ready-pool-timeout	1 回だけ	正の整数	無期限	コンテナによって使われるレディープールタイムアウト。この値は管理ツールの実行時に変更可能

表 11-23　pool-manager サブ要素 (続き)

サブ要素	繰り返し規則	内容	デフォルト	説明
ready-pool-maxsize	1 回だけ	正の整数または無期限を示す「0」	無期限	エントリ数で示したレディーキャッシュの最大サイズ。この値は管理ツールの実行時に変更可能
free-pool-maxsize	1 回だけ	正の整数または無期限を示す「0」	無期限	エントリの数で示したインスタンスの使用可能なプールの最大サイズ。この値は管理ツールの実行時に変更可能

EJB 参照を指定する要素

表 11-24 は、J2EE XML ファイルの ejb-ref エントリと一致する ejb-link の絶対名 jndi-name の保存場所である ejb-ref サブ要素を示しています。

表 11-24　ejb-ref サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
ejb-ref-name	1 回だけ	文字列	なし	対応する J2EE XML ファイルの ejb-ref エントリの ejb-link
jndi-name	1 回だけ	文字列	なし	絶対 jndi-name

リソースの参照を指定する要素

表 11-25 は、J2EE XML ファイルの resource-ref エントリと一致する resource-ref の絶対名 jndi-name の保存場所である resource-ref サブ要素を示しています。

表 11-25　resource-ref サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
resource-ref-name	1 回だけ	文字列	なし	対応する J2EE XML ファイルの resource-ref エントリの resource-ref 名
jndi-name	1 回だけ	文字列	なし	絶対 jndi-name

ロールマッピングを指定する要素

表 11-26 は、LDAP ユーザ、グループなどのマッピングのロール記述子である role-mapping サブ要素を示しています。

表 11-26 role-mapping サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
role-name	1 回だけ	文字列	なし	<security-role> 要素で参照されるロールの名前
role-impl	1 回だけ	要素	なし	特定の role-name を構成する LDAP グループやユーザの表現に使われる文字列。role-impl には、任意の数のグループやユーザを指定可能

ロール実装を指定する要素

表 11-27 は、ロール実装の記述子である role-impl サブ要素を示しています。

表 11-27 role-impl サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
group	0 回または複数回	文字列	なし	特定の LDAP グループの LDAP 固有の文字列
user	0 回または複数回	文字列	なし	特定の LDAP ユーザの LDAP 固有の文字列

RMI/IIOP クライアント XML DTD

RMI/IIOP クライアントは、iPlanet Application Server 固有の J2EE クライアントです。RMI/IIOP クライアントでは、標準の J2EE アプリケーションクライアント仕様がサポートされているだけでなく、iPlanet Application Server への直接アクセスがサポートされています。RMI/IIOP クライアントの詳細については、第 10 章「CORBA ベースクライアントの開発と配置」を参照してください。

RMI/IIOP クライアント JAR ファイルには、配置ツールによって生成された 2 つの DD があります。このうち一つの DD は、J2EE アプリケーションクライアント XML DTD によって指定され、J2EE 仕様書バージョン 1.0 の第 9 章「Application Clients」で説明しています。もう一つの DD には iPlanet Application Server 固有の RMI/IIOP クライアント要素が含まれます。詳細については、333 ページの「iPlanet Application Server RMI/IIOP クライアント XML DTD」を参照してください。

サンプル RMI/IIOP クライアント DD ファイルについては、433 ページの「RMI/IIOP Client DD XML ファイル」を参照してください。

iPlanet Application Server RMI/IIOP クライアント XML DTD

`ias-java-client-jar` 要素は、RMI/IIOP クライアント DD のルート要素です。

EJB 参照情報を指定する要素

表 11-28 は、J2EE XML ファイルの `ejb-ref` エントリと一致する `ejb-link` の絶対名 `jndi-name` の保存場所である `ejb-ref` サブ要素を示しています。

表 11-28 `ejb-ref` サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
`ejb-ref-name`	1 回だけ	文字列	なし	対応する J2EE XML ファイルの `ejb-ref` エントリの `ejb-link`
`jndi-name`	1 回だけ	文字列	なし	絶対 `jndi-name`

リソースの参照情報を指定する要素

表 11-29 は、J2EE XML ファイルの resource-ref エントリと一致する resource-ref の絶対名 jndi-name の保存場所である resource-ref サブ要素を示しています。

表 11-29 resource-ref サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
resource-ref-name	1 回だけ	文字列	なし	対応する J2EE XML ファイルの resource-ref エントリの resource-ref 名
jndi-name	1 回だけ	文字列	なし	絶対 jndi-name

リソース XML DTD

各 iPlanet Application Server リソースには、リソース XML ファイルがあります。リソースには、JDBC データソース、Java Mail、JMS などがあります。この XML ファイルには、iPlanet Application Server でリソースを登録するために使われるエントリがあります。これらのエントリによって、そのリソースへの iPlanet Application Server の接続方法が定義されます。これらのファイルは配置ツールによって生成されます。この節では、リソース XML ファイルエントリについて説明します。これらのファイルの作成方法については、配置ツールのオンラインヘルプを参照してください。

データソース XML DTD

この節では、iPlanet Application Server データソースの XML DTD について説明します。

データソースを指定する要素

表 11-30 は、リソース DD ルート要素である ias-Datasource-jar サブ要素を示しています。

表 11-30 ias-Datasource-jar サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
ias-resource	1 回だけ	要素	なし	すべてのリソース DD の共通要素

iPlanet Application Server リソースを指定する要素

表 11-31 は、すべてのリソースに使われる記述子である ias-resource サブ要素を示しています。

表 11-31 ias-resource サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
resource	1 回だけ	要素	なし	すべてのリソース DD の共通要素

リソースを指定する要素

表 11-32 は、すべてのリソースに使われる記述子である resource サブ要素を示しています。

表 11-32 resource サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
jndi-name	1 回だけ	文字列	なし	リソースファクトリの絶対 jndi-name (jdb/Who など)
jdbc	1 回またはそれ以外	要素	なし	JDBC データソースの記述子
jms	1 回またはそれ以外	文字列	なし	JMS データソースの記述子
mail	1 回またはそれ以外	文字列	なし	mail データソースの記述子
url	1 回またはそれ以外	文字列	なし	URL データソースの記述子

JDBC データソースを指定する要素

表 11-33 は、JDBC データソースに使われる記述子である jdbc サブ要素を示しています。

表 11-33 jdbc サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
database	1 回だけ	文字列	なし	接続先のデータベースの名前
datasource	1 回だけ	文字列	なし	割り当てられたデータソースの名前

表 11-33　jdbc サブ要素 (続き)

サブ要素	繰り返し規則	内容	デフォルト	説明
username	1 回だけ	文字列	なし	有効なデータベースユーザ名
password	1 回だけ	文字列	なし	有効なユーザ名パスワード
driver-type	1 回だけ	次のどれかを含む文字列フィールド ORACLE_OCI (Oracle) DB2_CLI (DB2) INFORMIX_CLI (Informix) SYBASE_CTLIB (Sybase) ODBC (ODBC)	なし	EIS 固有の JDBC ドライバ
resource-mgr	0 回または 1 回	文字列	なし	この属性を設定すると、一覧表示されたリソースマネージャによる分散トランザクションにデータソースを利用可能 この属性を指定しない場合、データソースはローカルなデータベースでだけ利用可能 その値は、RESOURCEMGR キーの下のリソースマネージャに作成する名前である必要がある

RMI/IIOP クライアントデータソース XML DTD

この節では、RMI/IIOP クライアントデータソースの XML DTD について説明します。

Java クライアントリソースを指定する要素

表 11-34 は、RMI/IIOP クライアントのデータソース XML DD ルート要素である `ias-javaclient-resource` サブ要素を示しています。

表 11-34 `ias-javaclient-resource` サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
`jdbc`	1 回またはそれ以外	要素	なし	RMI/IIOP クライアント JDBC 設定の記述子
`jms`	1 回またはそれ以外	文字列	なし	未定義
`jndi-name`	1 回だけ	文字列	なし	絶対 `jndi-name`

JDBC 設定を指定する要素

表 11-35 は JDBC 設定記述子である `jdbc` サブ要素を示しています。

表 11-35 `jdbc` サブ要素

サブ要素	繰り返し規則	内容	デフォルト	説明
`driverClass`	1 回だけ	要素	なし	有効なドライバクラス
`connectUrl`	1 回だけ	文字列	なし	接続先の有効な URL
`userName`	1 回だけ	文字列	なし	有効なユーザ名
`password`	1 回だけ	文字列	なし	有効なユーザ名パスワード

第 12 章

ユーザセッションの作成と管理

この章では、ユーザおよびトランザクションの情報を対話間で継続的に維持できるようにするセッションを作成し、管理する方法について説明します。

この章には次の節があります。

- セッションについて
- セッションの使用法

セッションについて

ユーザセッションという用語は、サーバが記録するユーザとアプリケーション間の一連の対話を意味します。セッションは、パーシスタントオブジェクト (EJB やデータベースリザルトセットへのハンドルなど) や認証されたユーザ ID などのユーザ固有のステートを多数の対話間で維持するために使われます。たとえば、確認されたユーザログインと、そのあとにそのユーザに対して指示された一連のアクティビティを記録するためにセッションを使用できます。

セッション自体はサーバ内に存在します。クライアントは、リクエストごとに、cookie 内のセッション ID を送信します。または、ブラウザが cookie を許可しない場合、サーバは自動的にセッション ID を URL に書き込みます。

iPlanet Application Server はすべてのセッションアクティビティに対して、`HttpSession` と呼ばれる Servlet の標準セッションインタフェースをサポートします。このインタフェースによって移植可能で安全な Servlet を作成できます。

さらに、iPlanet Application Server には `HttpSession2` という名前の補足インタフェースが用意されています。このインタフェースは、Servlet のセキュリティフレームワークだけでなく、Servelet と古い iPlanet Application Server コンポーネント (AppLogic) 間でのセッションの共有をサポートします。

その背景には、分散可能セッションとローカルセッションの 2 つのセッションスタイルがあります。2 つのスタイルの主な違いとして、分散可能セッションはその名前が示すようにクラスタ内の複数のサーバ間で分散でき、ローカルセッションはスティッキーで、個々のサーバにバインドされています。ローカルセッションモデルを使うように設定されているアプリケーションの Servlet には、スティッキーロードバランスが自動的に設定されます。アプリケーションのコンフィグレーションファイル内でどのセッションスタイルを使うかを決めます。アプリケーションのコンフィグレーションファイル内にあるセッション関連要素の詳細については、第 11 章「配置のためのパッケージ化」を参照してください。

セッションと cookie

cookie は、サーバが同じクライアントからの呼び出しを認識できるように、呼び出し側のブラウザに送信され、そのブラウザからの後続の呼び出しのたびに取り出される小さな情報の集合です。cookie は、期限切れになるまで、それを作成したサイトを呼び出すたびに返されます。

セッションは、その最初の作成時にクライアントに送信されるセッション cookie によって自動的に維持されます。セッション cookie には、継続的な各対話でブラウザに接続するクライアントを識別するセッション ID が含まれています。クライアントが cookie をサポートしない場合や許可しない場合、サーバはセッション ID がそのクライアントからの URL 内に現れている部分の URL を書き換えます。

セッションと URL の書き換え

iPlanet Application Server が暗黙的に URL を書き換える状況には次の 2 つがあります。

- iPlanet Application Server から応答が戻って来るとき。暗黙的に URL を書き換えるよう指定されている場合、プラグインは応答をクライアントに渡す前に URL を書き換えます。
- クライアントからのリクエストが iPlanet Application Server に送信される必要がなく、Web サーバサイドで処理できるとき。このようなリクエストはセッションの途中で発生する可能性があり、応答が必要ない場合があります。

この節には次のトピックがあります。

- サポートされるタグと属性
- URL の書き換えプロセス
- ロケーションヘッダ
- cookie の順序

サポートされるタグと属性

URL の書き換えに関連して、以下のタグと属性がサポートされます。このプラグインに関しては、これらすべてのタグと属性で大文字と小文字は区別されません。

表 12-1 URL の書き換えに関連してサポートされるタグと属性

タグまたは属性	言語	例
A	HTML	`<a href="http://www.sun.com"> Sun </a>`
		`<a href="/index.html"> Index </a>`
AREA	HTML	`<area shape=circle cords="50,50,25" href="http://docs.sun.com">`
FORM	HTML	
FRAME	HTML	
GO	WML	`<go href="/help.wml">`
IMG	HTML	
ONENTERBACKWARD	WML	`<card onenterbackward="/url"> xyz </card>`
ONENTERFORWARD	WML	`<card onenterforward="/url"> Hello </card>`
ONPICK	WML	`<select>`
		`<option onpick="/a.wml"> A </option>`
		`<option onpick="/b.wml"> B </option>`
		`</select>`
ONTIMER	WML	`<card ontimer="/next">`

次の節では、各タグについて詳しく説明します。

A

- このタグの `href` 属性で指定された URL が cookie とともに書き換えられる
- URL にクエリ文字列を持たせることができる
- URL は単一引用符または二重引用符で囲む必要がある
- URL の先頭に「#」があってはならない
- cookie は URL の直後に書き換えられる。URL にすでにクエリ文字列がある場合は、cookie の後に置かれる

AREA
- このタグの href 属性で指定された URL が書き換えられる
- nohref の場合は何も行われない
- URL にクエリ文字列を持たせることができる
- URL は単一引用符または二重引用符で囲む必要がある

FORM
- cookie は隠しフィールドの形式でエンコードされる
- POST および GET の両方が処理される

FRAME
- このタグの SRC 属性でポイントされた URL が書き換えられる
- FRAMESET および NOFRAMES は処理されない
- URL にクエリ文字列を持たせることができる
- URL は単一引用符または二重引用符で囲む必要がある

GO
- このタグの href 属性でポイントされた URL が cookie とともに書き換えられる
- URL にクエリ文字列を持たせることができる
- URL は単一引用符または二重引用符で囲む必要がある
- URL の先頭に「#」があってはならない
- cookie は URL の直後に書き換えられる。URL にすでにクエリ文字列がある場合は、cookie の後に置かれる

IMG
- このタグの SRC 属性でポイントされた URL が書き換えられる
- URL にクエリ文字列を持たせることができる
- URL は単一引用符または二重引用符で囲む必要がある

ONENTERBACKWARD
これはタグではなく、CARD や ONEVENT のような WML タグの属性です。
- ONENTERBACKWARD にポイントされた URL が書き換えられる
- URL にクエリ文字列を持たせることができる
- URL は単一引用符または二重引用符で囲む必要がある

ONENTERFORWARD

これはタグではなく、CARD や ONEVENT のような WML タグの属性です。

- ONENTERFORWARD にポイントされた URL が書き換えられる
- URL にクエリ文字列を持たせることができる
- URL は単一引用符または二重引用符で囲む必要がある

ONPICK

これはタグではなく、WML タグ OPTION の属性です。

- ONPICK にポイントされた URL が書き換えられる
- URL にクエリ文字列を持たせることができる
- URL は単一引用符または二重引用符で囲む必要がある

ONTIMER

これはタグではなく、CARD や ONEVENT のような WML タグの属性です。

- ONTIMER にポイントされた URL が書き換えられる
- URL にクエリ文字列を持たせることができる
- URL は単一引用符または二重引用符で囲む必要がある

URL の書き換えプロセス

プラグインの URL の書き換えプロセスは、次の 2 段階で行われます。

- 応答のヘッダ処理
- 応答のボディ処理

iPlanet Application Server へ送信されないリクエストの場合は、応答のボディ処理だけが行われます。

プラグインがリクエストを受信すると、クエリ文字列かどうかをチェックします。クエリ文字列がある場合は、前の応答にエンコードされた iPlanet Application Server の cookie が取り出されます。cookie は、`GXHC_` という接頭辞で始まっています。これらの cookie は、2 つのデータ構造体に格納されます。ここでは、QueryCookies および FormCookies が使用されています。前者には、cookie がクエリ文字列の形式で格納されます。後者には、HTML FORM タグに適した形式で格納されます。現在のところ、FORM 以外のすべてのタグの書き換えに、QueryCookies 内の cookie が使用されます。

以下に、QueryCookies 内の cookie の例を示します。

```
GXHC_GX_jst=d1f1943e55096164&amp;gx_session_id_=74cd83f757b5c8f6;
```

次は、FormCookies 内の cookie の例です。

```
<INPUT NAME=" GXHC_GX_jst" TYPE=HIDDEN VALUE="d1f1943e55096164"
</INPUT><INPUT NAME=" GXHC_ gx_session_id_" TYPE=HIDDEN
VALUE="74cd83f757b5c8f6" </INPUT>
```

これらの cookie は、後続の応答の書き換えで使うために取り出され、格納されます。

応答のヘッダ処理

iPlanet Application Server からの応答は、HTTP 応答の形式でプラグインに戻ります。プラグインでは、次のアルゴリズムを使用して、この応答のヘッダをまず処理します。

1. プラグインは応答内の Set-Cookie ヘッダの数をカウントし、この数を使用して ResponseCookies というデータ構造体に領域を割り当てます。応答内の各 Set-Cookie ヘッダに対して残りの手順が実行されます。
2. `domain` 属性がある場合は、その値が取り出されます。
3. QueryCookies または FormCookies 内にすでに cookie がある場合は、次のいずれかが発生します。
   - これから取り出す cookie に `domain` が関連付けられている場合は、QueryCookies または FormCookies 内の cookie もドメインに関連付けられていた可能性があり、その関連付けは URL の書き換え時に失われています。その cookie は、ドメインに関連付けられているため、新しいものとして処理されます。`treatAsNew` フラグが `TRUE` に設定され、QueryCookies または FormCookies 内の cookie が削除されます。
   - これから取り出す cookie に `domain` が関連付けられていない場合は、その値が QueryCookies または FormCookies 内の同じ cookie の値と比較されます。
     - 値が同じ場合は、新しい cookie が削除され、`treatAsNew` は `FALSE` に設定されます。
     - 2 つの値が異なる場合は、cookie が QueryCookies または FormCookies から削除され、`treatAsNew` は `TRUE` に設定されます。
4. cookie が QueryCookies または FormCookies に存在しない場合、その cookie はまったく新しいもので、`treatAsNew` が `TRUE` に設定されます。
5. これから取り出す cookie に `domain` が関連付けられている場合は、次のいずれかが発生します。
   - 次の RFC 2109 の規則に従ってドメインが検証されます。
     - ドットで始まっていること
     - 少なくとも 1 つのドットが文字列の途中にあること

     `domain` がこれらの規則に従っていれば、cookie は URL の書き換えに使用され、`domainOK` フラグは `TRUE` に設定されます。そうでない場合は、`domainOK` フラグが `FALSE` に設定されます。

- これから取り出す cookie に domain が関連付けられていません。cookie は URL の書き換えに使用され、domainOK フラグは TRUE に設定されます。

6. domainOK フラグが TRUE であり、treatAsNew が TRUE の場合、これから取り出す cookie はまったく新しいものか、新しい値を持つ古い cookie です。その名前、値、およびドメインが、ResponseCookies データ構造体に追加されます。

ヘッダ処理が終了した時点で、古い cookie はすべて QueryCookies または FormCookies 内に、新しい cookie はすべて ResponseCookies 内にあります。前者は応答 URL 内にエンコードされる準備が整っていますが、後者の場合はそのような形式に変換する必要があります。

応答のボディ処理

応答ヘッダがクライアントへ送信された後に、この段階に達します。応答のボディが解析されます。プラグインは、341 ページの「サポートされるタグと属性」の節に説明されているタグを検索します。これらのタグに対して、次のチェックが行われます。

1. URL が、絶対 URL か相対 URL かチェックされます。絶対 URL はプロトコル名で始まり、http://machine.website.com のような形式です。絶対 URL の場合は、http://machine.website.com などのホスト名が取り出されます。
2. プラグインは、ResponseCookies から cookie を選択し、NewCookies と呼ばれる、URL の書き換えに使用できる形式に変換する必要があります。FORM 以外のすべてのタグで使用されるこの形式は、QueryCookies 形式と同一です。FORM の場合は、FormCookies 形式が使用されます。
3. ResponseCookies 内の各 cookie は、次のデシジョンツリーに従って NewCookies に追加されるか、または追加されません。
   - 書き換えられる応答 URL が絶対 URL の場合は、次のいずれかが発生します。
     - cookie にドメインが指定され、それが応答 URL のホスト名の一部である場合は、NewCookies に追加されます。
     - cookie にドメインが指定されていない場合は、応答 URL 内のホスト名がリクエスト内のホスト名と比較されます。同じであれば、NewCookies に追加されます。
   - 書き換えられる応答 URL が相対 URL の場合は、次のいずれかが発生します。
     - cookie にドメインが指定されていない場合は、NewCookies に追加されます。
     - cookie にドメインが指定され、それがリクエスト内のホスト名の一部である場合は、cookie が NewCookies に追加されます。
4. NewCookies 内の cookie は、応答 URL 内に常にエンコードされます。QueryCookies と FormCookies 内の cookie も、次の条件に従って応答 URL 内にエンコードされます。
   - 書き換えられる応答 URL が絶対 URL で、応答 URL 内のホスト名がリクエスト内のホスト名と一致するとき

- 書き換えられる応答 URL が相対 URL のとき

ロケーションヘッダ

応答内の HTTP ヘッダ `Location` を送り返すことによって、リクエストを別の URL にリダイレクトすることもできます。このヘッダに関連する URL も書き換えられます。URL をエンコードするために使用される技術は、応答のボディ処理で使用される技術と同じです。ただしこの書き換えは、応答のヘッダ処理の一部として行われます。`Location` ヘッダが検出されるときまでに、有効なすべての cookie は ResponseCookies 内に収集されています。URL 内にクエリ文字列がある場合は、取り出されて保存されます。必要な場合は、QueryCookies 内の cookie が最初に追加されます。次に、ResponseCookies から選択された cookie が追加されます。最後に、元のクエリ文字列が追加されます。

cookie の順序

エンコードされるすべての cookie は、クエリ文字列の前にあります。

応答 URL 内に cookie がエンコードされる順序は、リクエストの一部として cookie が到着した順序になります。後続のリクエスト内の新しい cookie は、cookie リストの最後に追加されます。ただし、後続のリクエスト内で cookie が再定義された場合は、リストの元の位置から削除され、リストの最後に追加されます。

たとえば、リクエストに対する応答が次のような cookie 付きで到着したと想定します。

```
Set-Cookie c1=v1
Set-Cookie c2=v2
Set-Cookie c3=v3
```

これらの cookie は次のようにエンコードされます。

```
c1=v1&amp;c2=v2&amp;c3=v3
```

この順序は、後続のすべての応答に使用されます。新しい cookie はすべて、最後に追加されていきます。

ただし、後続のリクエストで `c2` が次のように変更されたとします。

```
Set-Cookie c2=v22
```

この場合、形式は次のように変更されます。

```
c1=v1&amp;c3=v3&amp;c2=v22
```

セッションとセキュリティ

iPlanet Application Server のセキュリティモデルは、認証されたユーザセッションをベースにしています。セッションが作成されると、使う場合はアプリケーションユーザを認証し、そのセッションにログインします。EJB リクエストを受け取る Servlet の対話の各ステップで、出力をフォーマット化する内容を JSP に対して作成し、ユーザが正しく認証されていることを認識します。

さらに、セッション cookie がセキュアコネクション (HTTPS) だけに渡されるように指定できます。したがって、安全な経路上に限りセッションをアクティブな状態で維持できます。

セキュリティの詳細については、第 13 章「安全なアプリケーションの作成」を参照してください。

セッションの使用法

セッションを使うには、まず `HttpServletRequest` の `getSession()` メソッドを使ってセッションを作成します。セッションが確立したら、所定のメソッドを使ってそのプロパティを調べたり、設定したりします。必要に応じて、非アクティブな状態が一定時間続いたあとでタイムアウトになるようにセッションを設定したり、セッションを手動で無効にしたりします。ほかのコンポーネントも使用できるように、オブジェクトを保存するセッションにバインドすることもできます。

この節では、次のトピックについて説明します。

- セッションの作成またはセッションへのアクセス
- セッションプロパティの調査
- セッションへのデータのバインド
- セッションの無効化
- セッションタイプの制御
- 分散環境でのセッションの共有
- AppLogic とのセッションの共有

セッションの作成またはセッションへのアクセス

新しいセッションを作成したり、既存のセッションにアクセスしたりするには、次の例に示すように HttpServletRequest の getSession() メソッドを使います。

```
HttpSession mySession = request.getSession();
```

getSession() は、リクエストに関連付けられた正当なセッションオブジェクトを返します。このセッションオブジェクトは、リクエストオブジェクト内にカプセル化されているセッション cookie 内で識別されます。引数を指定せずにこのメソッドを呼び出すと、リクエストに関連付けられているセッションがまだ存在していない場合にはセッションが作成されます。さらに、ブール値の引数でメソッドを呼び出すと、その引数が true の場合だけ、セッションが作成されます。

次の例は、セッションが存在する場合に、Servlet の主な関数だけを実行する Servlet の doPost() メソッドを示しています。getSession() に false パラメータを指定すると、セッションがまだ存在しない場合でも Servlet は新しいセッションを作成しないので注意してください。

```
public void doPost (HttpServletRequest req,
                    HttpServletResponse res)
             throws ServletException, IOException
{
   if ( HttpSession session = req.getSession(false) )
   {
       // セッションが取り出され、Servlet オペレーションとともに継続します。
   }
   else
       // セッションがないので、エラーページが返されます。

   }
}
```

注　getSession() メソッドは、レスポンスストリームに書き込みが行われる前に呼び出す必要があります。そうでないと、SetCookie 文字列は、HTTP ヘッダーではなく HTTP レスポンスの本体に配置されます。

getSession() の詳細については、Java Servlet 仕様書バージョン 2.2 を参照してください。

セッションプロパティの調査

セッション ID を確立したら、HttpSession インタフェース内のメソッドを使って、セッションのプロパティを調べ、HttpServletRequest インタフェース内のメソッドを使ってそのセッションに関連するリクエストプロパティを調べます。

表 12-2 は、セッションのプロパティを調べるメソッドを示しています。

表 12-2 HttpSession メソッド

HttpSession メソッド	説明
getCreationTime()	セッション時刻を返す (1970 年 1 月 1 日 00:00:00 GMT 以降の時刻でミリ秒単位)
getId()	割り当てられたセッション識別子を返す。HTTP のセッションの識別子は、サーバが作成し、維持するユニークな文字列
getLastAccessedTime()	割り当てられたセッション識別子を持つリクエストをクライアントが送信した最後の時刻を返す (1970 年 1 月 1 日 00:00:00 GMT 以降の時刻でミリ秒単位)。新しいセッションの場合は -1 を返す
isNew()	このセッションが新規と見なされるかどうかを示すブール値を返す。サーバがセッションを作成し、クライアントがそのセッションにリクエストを送信していない場合は、新規のセッションになる。つまり、クライアントはセッションを「認識」または「結合」しておらず、次のリクエストを出すときに正しいセッション識別情報を返さない可能性がある

次のようにします。

```
String mySessionID = mySession.getId();
if ( mySession.isNew() ) {
   log.println(currentDate);
   log.println("client has not yet joined session " + mySessionID);
}
```

表 12-3 は、そのセッションに関連するリクエストオブジェクトプロパティを調べるメソッドを示しています。

表 12-3 HttpServletRequest メソッド

HttpServletRequest メソッド	説明
getRemoteUser()	リクエストを出したユーザの名前を取得する (HTTP 認識によって情報を取得)。リクエストにユーザ名の情報がない場合には NULL を返す

表 12-3　HttpServletRequest メソッド (続き)

HttpServletRequest メソッド	説明
getRequestedSessionId()	このリクエストとともに指定されたセッション ID を返す。クライアントが指定したセッション ID が無効で新しいセッションが作成された場合は、現在のセッション内のセッション ID と異なる場合がある。リクエストに関連付けられたセッションがない場合は NULL を返す
isRequestedSessionIdValid()	このリクエストが現在有効なセッションに関連付けられているかどうかを確認する。リクエストされたセッションが有効でない場合、getSession() メソッドからは返されない
isRequestedSessionIdFromCookie()	クライアントから指定されたリクエストのセッション ID が cookie である場合は true を返し、それ以外のときは false を返す
isRequestedSessionIdFromURL()	クライアントから指定されたリクエストのセッション ID が URL の一部である場合は true を返し、それ以外のときは false を返す

次のようにします。

```
if ( request.isRequestedSessionIdValid() ) {
   if ( request.isRequestedSessionIdFromCookie() ) {
      // このセッションはセッション cookie 内で維持されます。
   }
   // 有効なセッションを必要とするほかのタスク
} else {
   // アプリケーションエラーを記録します。
}
```

セッションへのデータのバインド

複数のユーザ対話間で利用できるように、オブジェクトをセッションにバインドできます。次の HttpSession メソッドはセッションオブジェクトへのオブジェクトのバインドをサポートします。

表 12-4　HttpSession メソッド

HttpSession メソッド	説明
getValue()	セッション内の所定の名前にバインドされたオブジェクトを返す。バインドされたものがなければ NULL を返す

表 12-4　HttpSession メソッド (続き)

HttpSession メソッド	説明
getValueNames()	セッションにバインドされたすべての値の名前の配列を返す
putValue()	指定された名前を使って、指定されたオブジェクトをセッションにバインドする。同じ名前でバインドされている既存のオブジェクトは上書きされる。セッションにバインドされたオブジェクトを分散するには、serializable インタフェースを実装する必要がある。iPlanet Application Server の RowSets および JDBC ResultSets は serializable インタフェースではないので、分散させることはできない
removeValue()	指定した名前を持つセッション内のオブジェクトのバインドを解除する。指定した名前のオブジェクトがバインドされていなければ、このメソッドの影響はない

HttpSessionBindingListener によるバインドの通知

オブジェクトによっては、セッションに入れられたとき、またはセッションから削除されたときにユーザがそれを認識する必要があるものもあります。この情報を取得するには、これらのオブジェクト内に HttpSessionBindingListener インタフェースを実装します。アプリケーションがセッションにデータを保存したり、セッションからデータを削除したりするとき、Servlet エンジンはバインドまたはバインド解除されているオブジェクトが HttpSessionBindingListener を実装しているかどうかを確認します。実装している場合は、HttpSessionBindingListener インタフェースを通じて、セッションにバインドされているかまたはバインドされていないかを、対象のオブジェクトに通知します。

セッションの無効化

非アクティブな状態が一定時間続いたあとで、セッションが自動的に無効になるように指定します。または、HttpSession の invalidate() メソッドを使って手動でセッションを無効にします。

ヒント　セッションの API には明示的なセッションログアウト API はありません。したがって、ログアウトを実行するには session.invalidate() API を呼び出す必要があります。

手動によるセッションの無効化

手動でセッションを無効にするには、次のメソッドを呼び出します。

```
session.invalidate();
```

セッションにバインドされたオブジェクトはすべて削除されます。

セッションタイムアウトの設定

セッションタイムアウトは、ias-specific 配置記述子を使って設定します。詳細については、第 11 章「配置のためのパッケージ化」の session-info 要素を参照してください。

セッションタイプの制御

iPlanet Application Server には、lite (ライト) と distributed (分散) の、セッションタイプがあります。

- lite セッションは、HttpSession の高速で単一プロセスの実装です。このセッションは、速度が最優先事項であり、セッションデータを分散する必要がないあらゆる状況で使用されます。これは、HttpSession のもっとも単純な形式です。
- distributed セッションは、HttpSession API の堅牢でスケーラブルな実装です。Application Server の分散機能が使用されるので、フェールオーバーとロードバランス機能が有効にされます。ネットワークをバックアップするオーバーヘッドがあるため、lite セッションよりは若干遅くなります。

セッションのタイプを制御するには、iPlanet Application Server 固有の XML ファイル内で適切な要素を設定します。詳細については、第 11 章「配置のためのパッケージ化」の session-info 要素を参照してください。

分散環境でのセッションの共有

iPlanet Application Server 6.5 では、同じ JVM 内の同時リクエストで同じセッションオブジェクトを共有することができます。次に、iPlanet Application Server が行うプロセスを説明します。

1. リクエストがセッションにアクセスするたびに、カウンタの値が増加します。
2. セッションへの最初の可変アクセスがあるたびに、Dsync ロックがトリガされ、ロックしているスレッドへの参照がセッション内に格納されます。
3. セッションのステートは、ロック直後に Dsync から更新されます。
4. 一方、リクエストが出力されるたびに、カウンタの値が減少します。セッションを保存する場合も同様です。

5. 出力されるリクエストが最初にセッションをロックしたリクエストの場合は、`servletrunner.execute()` の完了前にほかのすべてのリクエストが出力されるまで待機します。

   このリクエストは、待機が終わるまでにストリーム出力されます。ロックしているスレッドは、ロックを解除できる唯一のスレッドであるため、待機する必要があります。

6. セッションのすべてのリクエストが出力されると、ロックしているスレッドがセッションのロックを解除して終了します。

7. セッションが途中で無効化された場合、その時点でバックエンドの一貫性は必要なくなるため、ロックしているスレッドは直ちにロックを解除するよう求められます。

注

- 同時リクエストの場合、ロックしているスレッドは、セッションにアクセスしているすべてのリクエストが終了するまで待機状態にあるため、多少のオーバーヘッドがあります。
  大量の同時アクセスが行われる場合は、パフォーマンス向上のために調整する際に、このオーバーヘッドを考慮に入れる必要があります。
- 属性としてオブジェクトを相互参照している場合は、分散セッションの実行時にオブジェクトが相互参照を保持しなくなります。
  これは、各属性が個別に直列化されて BLOB として格納されるためです。そのため、参照されているすべてのオブジェクトもこの BLOB に格納されます。取得中に、オブジェクトグラフ全体が属性ごとに別々に直列化解除されます。

AppLogic とのセッションの共有

Servlet のプログラマは、iPlanet Application Server 機能のインタフェース `HttpSession2` を使って、AppLogic と Servlet 間で分散可能セッションを共有できます。セッションの共有は、アプリケーションを NAS 2.x から iPlanet Application Server 6.5 に移行するときに役立ちます。`HttpSession2` インタフェースを使うと、セキュリティを確保し、分散可能セッションを直接操作できます。

さらに、`loginSession()` を使って AppLogic 内でセッションを確立し、Servlet からそのセッションにアクセスする場合は、`AppLogic` クラス内で `setSessionVisibility()` メソッドを呼び出して、Servlet だけでなく AppLogic にも送信するようにセッション cookie に指示する必要があります。また、この作業は `saveSession()` を呼び出す前に行う必要があります。

AppLogic 内の例

```
domain=".mydomain.com";
path="/"; // すべてのドメインに表示します
isSecure=true;
if ( setSessionVisibility(domain, path, isSecure) == GXE.SUCCESS )
   { // セッションはすべてのドメインに表示されています }
```

`setSessionVisibility()` の詳細については、『Foundation Class Reference (Java)』の `AppLogic` クラスを参照してください。AppLogics と Servlet 間のセッションの共有の詳細については、『移行ガイド』を参照してください。

第 13 章

安全なアプリケーションの作成

この章では、ユーザの認証を実行し、Servlet と EJB ビジネスロジックへの認可にアクセスするコンポーネントを持つ、iPlanet Application Server の安全な J2EE アプリケーションを作成する方法について説明します。

この章には次の節があります。

- iPlanet Application Server のセキュリティの目標
- iPlanet Application Server 固有のセキュリティ機能
- iPlanet Application Server のセキュリティモデル
- セキュリティの責任の概要
- セキュリティの一般的な用語
- コンテナセキュリティ
- プログラムによるセキュリティ
- 宣言によるセキュリティ
- Servlet によるユーザ認証
- Servlet によるユーザ認可
- EJB によるユーザ認可
- シングルサインオンでのユーザ認証
- RMI/IIOP クライアントのユーザ認証
- セキュリティ情報のガイド
- Web サーバからアプリケーションサーバのコンポーネントのセキュリティ

iPlanet Application Server のセキュリティの目標

企業のコンピューティング環境には、多くのセキュリティ上のリスクがあります。iPlanet Application Server の目標は、J2EE セキュリティモデルをベースとして、高度に安全で相互利用可能な分散コンポーネントコンピューティング環境を実現することです。iPlanet Application Server のセキュリティの目標は次のとおりです。

- J2EE v1.2 セキュリティモデル (J2EE 仕様書バージョン 1.2 の第 3 章「Security」を参照) への完全準拠
- EJB v1.1 セキュリティモデル (Enterprise JaveBeans 仕様書バージョン 1.1 の第 15 章「Security Management」を参照) への完全準拠。EJB ロールベースの認可も含まれます
- Java Servlet v2.2 セキュリティモデル (Java Servlet 仕様書バージョン 2.2 の第 11 章「Security」を参照) への完全準拠。Servlet ロールベースの認可も含まれます
- iPlanet Application Server アプリケーション全体のシングルサインオンをサポート
- RMI/IIOP クライアントのセキュリティをサポート
- LDAP をセキュリティのバックエンドとして使い、実行時のユーザ管理を実現
- iPlanet Application Server 固有の XML ベースの明確なロールマッピング情報を実装
- iPlanet Application Server 配置ツールによって作成された、宣言によるセキュリティを持つ iPlanet Application Server 固有の XML ファイル
- AppLogic セキュリティ API との互換性

iPlanet Application Server 固有のセキュリティ機能

iPlanet Application Server は、次の iPlanet Application Server 固有の機能だけでなく、J2EE v1.2 セキュリティモデルをサポートします。

- iPlanet Application Server アプリケーション全体にわたるシングルサインオン
- RMI/IIOP クライアントのセキュリティ
- iPlanet Application Server 固有の XML ベースのロールマッピング情報
- GUI ベースの配置ツールを、セキュリティ情報を保持する XML ファイルの作成に使用
- 実行時のユーザ管理 LDAP

- LDAP をセキュリティのバックエンドとして使用

iPlanet Application Server のセキュリティモデル

安全なアプリケーションでは、クライアントを有効なアプリケーションユーザとして認証する必要があり、EJB ビジネスロジックにアクセスする認可を持っています。iPlanet Application Server は Web クライアントと RMI/IIOP クライアントの両方のセキュリティをサポートします。

Web クライアントはブラウザと Web サーバを使い、HTTP を使って iPlanet Application Server 上で実行中の Servlet と通信します。これらのクライアントには、Web サーバの機能を拡張するために Servlet および JSP を使った通信が必要です。

安全な Web コンテナと安全な EJB コンテナを使ったアプリケーションは、Web クライアントの次のセキュリティプロセスを強化できます。

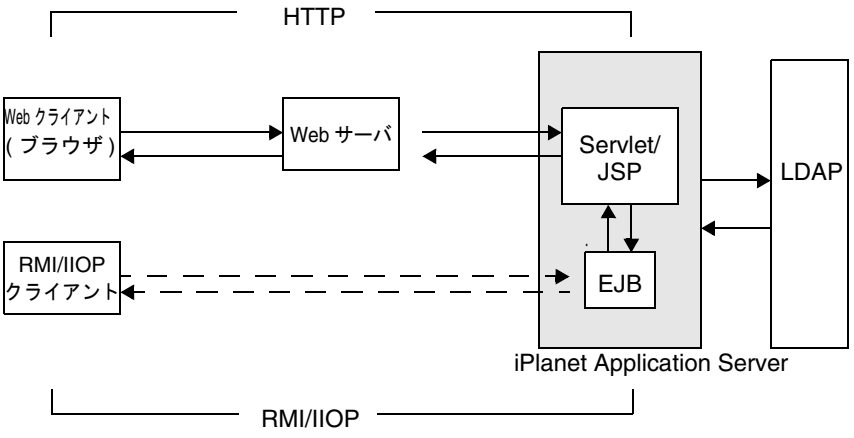
- 呼び出し側を認証する
- 呼び出し側に URL へのアクセスを認可する
- 呼び出し側に EJB ビジネスメソッドへのアクセスを認可する

RMI/IIOP クライアントは RMI/IIOP を使ってブリッジを介した通信を行い、iPlanet Application Server 上で実行中の EJB に直接アクセスします。RMI/IIOP クライアントは Bean メソッドを直接起動します。

安全な EJB コンテナを使ったアプリケーションは、RMI/IIOP クライアントの次のセキュリティプロセスを強化できます。

- 呼び出し側に EJB ビジネスメソッドへのアクセスを認可する

次の図に iPlanet Application Server セキュリティモデルを示します。



Web クライアントと URL の認可

安全な Web コンテナは認証および認可のプロパティを持つことができます。コンテナは基本、証明書、およびフォームベースの 3 つのタイプの認証をサポートします。Web クライアントがメインアプリケーションの URL を要求したときは、Web サーバが Web クライアントからのユーザ認証情報 (たとえば、ユーザ名とパスワード) の収集と iPlanet Application Server への転送を受け持ちます。

iPlanet Application Server は Web リソースに関連付けられたセキュリティポリシー (配置記述子 (DD) から取得される) を調べ、リソースアクセスの許可に使うセキュリティロールを調べます。Web コンテナは各ロールに対してユーザの証明書をテストし、ロールにユーザを割り当てることができるかどうかを判断します。ユーザ、グループ、およびロールについての情報を管理する企業規模のディレクトリサービスである LDAP サーバによって、ユーザの証明書が取得されます。

Web クライアントによる Enterprise Bean メソッドの呼び出し

Web クライアントが Web コンテナによって認証および認可され、JSP が EJB のリモートメソッド呼び出しを実行すると、認証プロセスで収集されたユーザの証明書を使って JSP と Bean の安全な関連付けが確立されます。安全な EJB コンテナには、Bean メソッドのアクセス制御の強化に使う認可プロパティを持つ DD が含まれています。EJB コンテナは LDAP サーバから受信したロール情報を使って、呼び出し側をロールに割り当てることができるかどうかと、Bean メソッドへのアクセスを許可するかどうかを判断します。

RMI/IIOP クライアントによる Enterprise JavaBeans メソッドの呼び出し

RMI/IIOP クライアントの場合、安全な EJB コンテナはセキュリティポリシーを調べて、呼び出し側が Bean メソッドへのアクセス権限を持っているかどうかを判断します。このプロセスは、RMI/IIOP クライアントおよび Web クライアントで同じです。

セキュリティの責任の概要

J2EE プラットフォームの主な目標はセキュリティメカニズムから開発者を解放し、さまざまな環境に安全かつ容易にアプリケーションを配置できるようにすることです。この目標を達成するには、アプリケーションセキュリティの仕様要件のメカニズムをアプリケーションの外側に明確に設定する必要があります。

アプリケーション開発者

アプリケーションの開発者は、次のプログラムセキュリティを指定します。

- セキュリティレベルの指定
- 保護されたオペレーションにアクセスされた場合、セキュリティパーミッションレベルの確認

アプリケーション編成者

アプリケーション編成者またはアプリケーションコンポーネントプロバイダは、コンポーネントに組み込まれた次のようなセキュリティ関連事項をすべて確認する必要があります。

- コンポーネントが `isCallerInRole` または `isUserInRole` を呼び出すときに使うすべてのロール名
- コンポーネントがアクセスするすべての外部リソースへの参照
- コンポーネントが行うすべての内部コンポーネント呼び出しへの参照
- 編成者が各コンポーネントの機能のパラメータのメソッド呼び出しをすべて指定すること、および機密性や整合性のために戻り値を保護することをお勧めします。配置記述子 (DD) はこの目的に使います。

アプリケーション配置者

iPlanet Application Server 配置ツールは、編成者が提供したビューを運用環境固有のポリシーとメカニズムに割り当てるために使います。アプリケーション配置者が設定したセキュリティメカニズムは、コンテナ内で管理されるコンポーネントのためにコンテナによって実装されます。

アプリケーション配置者は編成者が提供したすべてのコンポーネントのセキュリティビューを受け取り、それを使って次のようなアプリケーションにおける特定の企業環境を保護します。

- ユーザグループをセキュリティレベルに割り当てる
- コンポーネントメソッドへのアクセス権限、および呼び出し側が指定するセキュリティ属性とコンテナ権限の対応を定義する権限の改訂

セキュリティの一般的な用語

もっとも一般的なセキュリティプロセスは、認証、認可、およびロールマッピングです。次の節でそれらの用語を定義します。

認証

認証ではユーザを確認します。たとえば、ユーザが Web ブラウザ内でユーザ名とパスワードを入力し、その証明書が LDAP サーバ内に保存されているパーマネントプロファイルと一致したとき、ユーザは認証されます。ユーザは、それ以降のセッションで使われるセキュリティ ID に関連付けられます。

認可

認証された後、認可によってユーザは希望する操作を実行することができます。たとえば、人事管理アプリケーションでは、管理者には社員全員の個人情報を見ることを認可し、社員には自身の個人情報だけを見ることを認可します。

ロールマッピング

クライアントはセキュリティロールによって定義できます。たとえば、会社が社員のデータベースを使って、社内電話帳アプリケーションと支払給与情報の両方を生成します。電話番号と電子メールアドレスには、すべての社員がアクセスできますが、給与情報にアクセスできるのは一部の社員に限られます。給与を表示あるいは変更する権限を持つ社員は、特別なセキュリティロールを持つように定義できます。

ロールはアプリケーション内での職能を定義するのに対し、グループはある方法で関連付けられているユーザの集まりに過ぎません。この点で、ロールとユーザグループは異なります。たとえば、*astronauts*、*scientists*、および場合によっては *politicians* というグループのメンバーはすべて、*SpaceShuttlePassenger* のロールに該当します。

EJB セキュリティモデルは、アプリケーション開発者の記述どおりに、特定のドメインとは関係なくロール (ユーザグループとは区別された) を記述します。グループは配置ドメインに固有です。アプリケーション配置者のロールは 1 つまたは複数のグループにロールを割り当てることです。

iPlanet Application Server では、ロールは Directory Server に設定されているユーザグループに対応しています。LDAP グループにはユーザとほかのグループの両方を含めることができます。

コンテナセキュリティ

コンポーネントコンテナは、J2EE アプリケーションのセキュリティを確保する役目を果たします。コンテナによって確保されるセキュリティ形式には、次の 2 つがあります。

- プログラムによるセキュリティ
- 宣言によるセキュリティ

プログラムによるセキュリティ

プログラムによるセキュリティは、EJB または Servlet が J2EE セキュリティモデルによって指定されたセキュリティ API へのメソッド呼び出しを使って、呼び出し側、またはリモートユーザのセキュリティロールに基づいてビジネスロジックの決定を行う場合です。プログラムによるセキュリティは、宣言によるセキュリティ単独ではアプリケーションのセキュリティモデルの要求を十分に満たすことができない場合に限って使う必要があります。

J2EE 仕様書バージョン 1.2 では、EJB の `EJBContext` インタフェースの 2 つのメソッドおよび Servlet の `HttpServletRequest` インタフェースの 2 つのメソッドで構成されるものとして、プログラムによるセキュリティを定義しています。iPlanet Application Server は、この仕様書で規定されているようにこれらのインタフェースをサポートします。プログラムによるセキュリティの詳細については、J2EE 仕様書バージョン 1.2 の第 3.3.6 節「Programmatic Security」および 365 ページの「プログラムによるログイン」を参照してください。

宣言によるセキュリティ

宣言によるセキュリティは、アプリケーションのセキュリティメカニズムが宣言され、アプリケーションの外部で処理されるときのものです。DD は セキュリティロール、アクセス制御、および認証の要件を持つ J2EE アプリケーションのセキュリティ構造を記述するために iPlanet Application Server によって使われます。

セキュリティを認識するアプリケーションの DD、すなわち web-app コンテナおよび EJB コンテナはセキュリティ要素として XML タグを持ち、アプリケーションのセキュリティの特性を表現します。セキュリティの特性には認証および認可があります。

iPlanet Application Server は、J2EE v1.2 が指定する DTD をサポートし、さらに別のセキュリティ要素が DD に含まれています。

宣言によるセキュリティはアプリケーション配置者に責任があります。XML DD は iPlanet Application Server 配置ツールによって生成されます。詳細については、iPlanet Application Server 配置ツールのオンラインヘルプおよび『管理者ガイド』を参照してください。

アプリケーションレベルのセキュリティ

アプリケーションの XML DD には、アプリケーションの Servlet と EJB にアクセスするときの、すべてのユーザロールの認可記述子が含まれています。アプリケーションレベルでは、アプリケーションのコンテナが使うすべてのロールがこのファイル内に一覧表示される必要があります。これらのロールは、アプリケーションの XML DD ファイル内の `role-name` 要素によって記述されます。ロール名は、EJB XML DD (`ejb-jar` ファイル) および Servlet の XML DD (`web-war` ファイル) の適用対象となります。

Servlet レベルのセキュリティ

安全な Web コンテナはユーザを認証し、Servlet へのアクセスを認可します。ユーザが認証され認可されると、Servlet はユーザの証明書を EJB に転送して Bean との安全な関連付けを確立します。

EJB レベルのセキュリティ

EJB コンテナは EJB XML DD 内に展開されているセキュリティポリシーを使って、Bean メソッドへのアクセスを認可する役割を果たします。

Servlet によるユーザ認証

J2EE 仕様書バージョン 1.2 で要求される 3 つの Web ベースのログインメカニズムは iPlanet Application Server でサポートされています。3 つのメカニズムは次のとおりです。

- HTTP 基本認証
- SSL (Secure Socket Layer) 相互認証
- フォームベースログイン
- プログラムによるログイン

Web アプリケーション DD の `login-config` 要素は、使用される認証メソッド、HTTP 基本認証が使うアプリケーションの範囲名、およびフォームログインメカニズムの属性を記述します。

`login-config` 要素のシンタックスは次のとおりです。

```
<!ELEMENT login-config
(auth-method?,realm-name?,from-login-config?)>
```

Web アプリケーションの DD の要素の詳細については、Java Servlet 仕様書バージョン 2.2 の第 13 章「Deployment Descriptor」を参照してください。

HTTP 基本認証

HTTP 基本認証 (RFC2068) は iPlanet Application Server にサポートされています。HTTP 基本認証プロトコルはアクセスが協定される HTTP 領域を示します。パスワードは base64 エンコード方式で送信されるので、この認証のタイプはそれほど安全ではありません。

SSL (Secure Socket Layer) 相互認証

Secure Socket Layer (SSL) 3.0、およびクライアントとサーバの相互の証明書ベースの認証を実行する方法は、J2EE 仕様書 v1.2 の要件です。このセキュリティメカニズムによって、HTTPS (SSL 上の HTTP) を使ってユーザ認証が提供されます。

iPlanet Application Server の SSL 相互認証メカニズム (HTTPS 認証も同義) は次の一連の暗号をサポートします。

```
SSL_RSA_EXPORT_WITH_RC4_40_MD5
SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA
SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
```

フォームベースログイン

ログイン画面の見た目と使いやすさは、HTTP ブラウザの組み込みメカニズムでは制御できません。J2EE では、ログインするための標準 HTML または Servlet/JSP ベースフォームをパッケージングする機能を説明しています。ログインフォームは Web 保護ドメイン (HTTP 領域) に関連付けられ、まだ認証されていないユーザを認証するために使われます。

認証を適切に進めるために、ログインフォームのアクションは常に `j_security_check` である必要があります。

HTML ページ内でフォームをプログラムする方法を示す HTML サンプルは次のとおりです。

```
<form method="POST" action="j_security_check">
<input type="text" name="j_username">
<input type="password" name="j_password">
</form>
```

プログラムによるログイン

プログラムによるログインを使うと、ユーザはプログラムによって Web コンテナおよび EJB コンテナにログインできます。プログラムによるログインが有益な理由を次に示します。

- ユーザ認証の柔軟性を提供する
- ログアウトのための API を提供する
- シンプルで拡張可能である
- フォームベースなど、中間の Servlet を使うほかの認証タイプと比べて必要なメソッド呼び出しが少ない
- Web および EJB コンテナ全体に一般的なインタフェースを提供する

フォームベースログインとプログラムによるログイン

フォームベース認証を使ってセキュリティ制約付きで Web リソースが配置されると想定します。これらのリソースにアクセスするには、Web コネクタは、ユーザがすでにログインしているかどうかを確認する `FormAuthServlet` を呼び出す必要があります。ユーザがログインしていない場合は、認証を有効にするためにログインページが表示されます。

プログラムによるログインでは、セキュリティ制約なしで Web リソースが配置されます。ユーザが Web リソースにアクセスした場合、`FormAuthServlet` は呼び出されません。その代わり、`IProgrammaticLogin.login` メソッドが呼び出され、明示的にそのユーザを認証します。このメソッドが失敗した場合は、`AuthenticationException` がスローされます。それ以外では、ユーザはログインしています。

IProgrammaticLogin インタフェース

`com.iplanet.ias.security.IProgrammaticLogin` インタフェースを使うと、Web または EJB コンテナ内のユーザがプログラムによってログインできます。このインタフェースには次のメソッドがあります。

- `login`

- logout
- isLoggedIn
- loggedUserName

このインタフェースは、2 つの Java クラスによって実装されます。

- WebProgrammaticLogin クラス
- EjbProgrammaticLogin クラス

IProgrammaticLogin を実装する独自のクラスを作成できますが、お勧めしません。提供されたクラスを使うと、ログイン API ディレクトリを処理する必要がありません。

WebProgrammaticLogin クラス

com.iplanet.ias.security.WebProgrammaticLogin クラスによって、Web コンテナを使ったプログラムによるログインのデータメンバーが初期化されます。このクラスを現状のまま使ったり、サブクラスを作成したりできます。そのシグネチャは次のとおりです。

```
public class WebProgrammaticLogin extends java.lang.Object
implements IProgrammaticLogin
```

その 1 つのコンストラクタは次のとおりです。

```
public WebProgrammaticLogin(
   javax.servlet.ServletContext p_ServletContext、
   javax.servlet.http.HttpServletRequest p_HttpServletRequest、
   javax.servlet.http.HttpServletResponse p_HttpServletResponse)
throws NullValueException
```

必要な WebProgrammaticLogin 入力パラメータに NULL がある場合、com.iplanet.ias.security.NullValueException がスローされます。そのシグネチャは次のとおりです。

```
public class NullValueException extends java.lang.Exception
```

その 1 つのコンストラクタは次のとおりです。

```
public NullValueException(java.lang.String Msg)
```

WebProgrammaticLogin メソッドは次の節で説明します。

login メソッド

login メソッドを使って、ユーザはプログラムによってログインできます。そのシグネチャは次のとおりです。

```
public void login(java.lang.String UserName, java.lang.String
Password) throws ProgAuthenticationException, NullValueException
```

login メソッドは次のとおりです。

- ユーザ名およびパスワードが有効であることを確認する
- 別のユーザがログインしているかどうかを確認する
- ServletContext、HttpRequest、または HttpResponse が NULL かどうかを確認する
- 認証を実行する

必要な login 入力パラメータに NULL がある場合、
com.iplanet.ias.security.NullValueException がスローされます。

認証が失敗した場合は、
com.iplanet.ias.security.ProgAuthenticationException がスローされます。
そのシグネチャは次のとおりです。

```
public class ProgAuthenticationException extends
com.netscape.server.servlet.servletrunner.AuthenticationException
```

その 1 つのコンストラクタは次のとおりです。

```
public ProgAuthenticationException(java.lang.String Msg)
```

logout メソッド

logout メソッドを使って、ユーザはログアウトできます。そのシグネチャは次のとおりです。

```
public void logout(boolean flag)
```

実行される logout は flag の設定によって異なります。

- flag が false の場合は、セッションから主要属性を削除する (ソフトログアウト)
- flag が true の場合は、セッションを無効にする (ディープログアウト)

isLoggedIn メソッド

ユーザがすでにログインしている場合は、isLoggedIn メソッドが true を返します。
そのシグネチャは次のとおりです。

```
public boolean isLoggedIn()
```

loggedUserName メソッド

loggedUserName メソッドでは、ログインしているユーザの主要名を返すか、またはどのユーザもログインしていない場合は、NULL を返します。そのシグネチャは次のとおりです。

```
public java.lang.String loggedUserName()
```

EjbProgrammaticLogin クラス

com.iplanet.ias.security.EjbProgrammaticLogin クラスによって、EJB コンテナを使ったプログラムによるログインのデータメンバーが初期化されます。このクラスを現状のまま使ったり、サブクラスを作成したりできます。そのシグネチャは次のとおりです。

```
public class EjbProgrammaticLogin extends java.lang.Object
implements IProgrammaticLogin
```

その 1 つのコンストラクタは次のとおりです。

```
public EjbProgrammaticLogin() throws NullValueException
```

SecurityContext メンバー変数が NULL で、EjbProgrammaticLogin のインスタンスを作成しようとした場合は、com.iplanet.ias.security.NullValueException がスローされます。そのシグネチャは次のとおりです。

```
public class NullValueException extends java.lang.Exception
```

その 1 つのコンストラクタは次のとおりです。

```
public NullValueException(java.lang.String Msg)
```

EjbProgrammaticLogin メソッドは次の節で説明しています。

login メソッド

login メソッドを使って、ユーザはプログラムによってログインできます。そのシグネチャは次のとおりです。

```
public void login(java.lang.String userName, java.lang.String
password) throws ProgAuthenticationException, NullValueException
```

login メソッドは次のとおりです。

- ユーザ名およびパスワードが有効であることを確認する
- 別のユーザがログインしているかどうかを確認する
- SecurityContext が NULL かどうかを確認する
- 認証を実行する

必要な login 入力パラメータに NULL がある場合、com.iplanet.ias.security.NullValueException がスローされます。

認証が失敗した場合は、com.iplanet.ias.security.ProgAuthenticationException がスローされます。そのシグネチャは次のとおりです。

```
public class ProgAuthenticationException extends
com.netscape.server.servlet.servletrunner.AuthenticationException
```

その 1 つのコンストラクタは次のとおりです。

```
public ProgAuthenticationException(java.lang.String Msg)
```

logout メソッド

logout メソッドを使って、ユーザはログアウトできます。そのシグネチャは次のとおりです。

```
public void logout(boolean flag)
```

EJB コンテナのこのメソッドは flag 値にかかわらず、SecurityContext からログインしているユーザの主要名を削除します。

isLoggedIn メソッド

ユーザがすでにログインしている場合は、isLoggedIn メソッドが true を返します。そのシグネチャは次のとおりです。

```
public boolean isLoggedIn()
```

loggedUserName メソッド

loggedUserName メソッドでは、ログインしているユーザの主要名を返すか、またはどのユーザもログインしていない場合は、NULL を返します。そのシグネチャは次のとおりです。

```
public java.lang.String loggedUserName()
```

Servlet によるユーザ認可

適切な認可レベルを持つユーザのアクセスだけを許可するように Servlet を設定できます。これには iPlanet Application Server 配置ツールを使い、アプリケーションの .ear ファイルおよび Servlet の .war ファイルの DD を生成します。

ロールの定義

アプリケーション全体のすべてのロール名がアプリケーションの XML DD 内で宣言されます。アプリケーションの XML DD 内の security-role および role-name 要素によって、アプリケーションで許可されるすべてのロール名が宣言されます。これらのセキュリティロールは J2EE Web アプリケーションの DD の適用対象となります。

アプリケーションの XML DD 内の security-role 要素は application 要素のサブ要素です。security-role 要素のシンタックスは次のとおりです。

```
<!--
security-role 要素は、アプリケーションに対してグローバルなセキュリティロールを
定義します。2 つのサブ要素があります。一つはセキュリティロールの記述で、もう一つ
はセキュリティロールの名前です。

<!ELEMENT security-role (description?, role-name)>
role-name 要素にはロールの名前が入ります。
<!ELEMENT role-name (#PCDATA)>
```

セキュリティロールの参照

各 Servlet では、Web アプリケーションの DD はアクセスを認可されたすべてのロールを宣言します。web-app XML DD 内の `security-rol-ref` および `role-link` 要素は、認可されたロールをアプリケーションレベルのロール名にリンクします。

アプリケーション編成者は、`security-role-ref` 要素内で宣言されたセキュリティロールのすべての参照を、`security-role` 要素で定義されたセキュリティロールにリンクする必要があります。

アプリケーション編成者は、`role-link` 要素を使って各セキュリティロールの参照をセキュリティロールにリンクします。`role-link` 要素の値は `security-role` 要素で定義されたセキュリティロールの名前の一つである必要があります。

次の DD の例は、セキュリティロール参照をセキュリティロールにリンクする方法を示します。

```
<!ELEMENT security-role-ref (description?, role-name, role-link)>
<!ELEMENT role-link (#PCDATA)>
```

メソッドのパーミッションの定義

Servlet レベルでは、web-app XML DD の `auth-constraint` 要素を使ってメソッドのパーミッションを定義します。

リソースコレクション上の `auth-constraint` 要素を使って、リソースコレクションが許可されるユーザロールを指定する必要があります。ここで使うロールは `security-role-ref` 要素内に存在する必要があります。

```
<!ELEMENT auth-constraint (description?, role-name*)>
```

Web アプリケーション DD のサンプル

Web アプリケーションの DD のサンプルのセキュリティセクションは次のようになります。

```
<web-app>

   <display-name>A Secure Application</display-name>
   <security-role>
       <role-name>manager</role-name>
   </security-role>

   <Servlet>
       <servlet-name>catalog</servlet-name>
       <servlet-class>com.mycorp.CatalogServlet</servlet-class>

       <init-param>
          <param-name>catalog</param-name>
          <param-value>Spring</param-value>
       </init-param>

   <security-role-ref>
       <role-name>MGR</role-name><!-- コードで使われるロール名 -->
       <role-link>manager</role-link>
       </security-role-ref>
   </servlet>

   <servlet-mapping>
       <servlet-name>catalog</servlet-name>
       <url-pattern>/catalog/*</url-pattern>
   </servlet-mapping>

   <web-resource-collection>
       <web-resource-name>SalesInfo</web-resource-name>
       <urlpattern>/salesinfo/*</urlpattern>
       <http-method>GET</http-method>
       <http-method>POST</http-method>

       <user-data-constraint>
       <transport-guarantee>SECURE</transport-guarantee>
       </user-data-constraint>

       <auth-constraint>
          <role-name>manager</role-name>
       </auth-constraint>
   </web-resource-collection>
</web-app>
```

EJB によるユーザ認可

適切な認可レベルを持つユーザのアクセスだけを許可するように EJB を設定できます。これには iPlanet Application Server 配置ツールを使い、アプリケーションの `.ear` ファイルおよび EJB の `.jar` ファイルの DD を生成します。

EJB では、Servlet と同じようにプログラムによるログインを使います。詳細については、365 ページの「プログラムによるログイン」を参照してください。

ロールの定義

アプリケーション配置者は、動作環境で定義したユーザグループおよびユーザアカウントを、アプリケーション編成者が定義したセキュリティロールに割り当てます。

アプリケーション編成者は DD 内に 1 つまたは複数のロールを定義します。アプリケーション編成者は、Enterprise JavaBeans のホームおよびリモートインタフェースのメソッドグループをセキュリティロールに割り当て、アプリケーションのセキュリティビューを定義します。

アプリケーション編成者は次の項目を定義する必要があります。

- `security-role` 要素を使って各セキュリティロールを定義する
- `role-name` 要素を使ってセキュリティロール名を定義する
- `description` 要素を使ってセキュリティロールの説明を提供する (オプション)

`security-role` 要素によって定義したセキュリティロールは `ejb-jar` ファイルレベルの適用対象となり、`ejb-jar` ファイル内のすべての Enterprise JavaBeans に適用されます (J2EE 仕様書はグローバルロール、つまりコンテナに対して包括的なグローバルロールの定義方法を示していません)。

次に DD 内のセキュリティロール定義の例を示します。

```
...
<assembly-descriptor>
   <security-role>
       <description>
          このロールには、自分の目的に使うアプリケーションに
          アクセスできる社員が含まれています。
          このロールは、その社員の情報への
          アクセスだけが許可されています。
       </desciption>
       <role-name>employee</role-name>
       </security-role>
       <security-role>
          <description>
```

```
                このロールは、自分の目的に使う
                アプリケーションの管理業務を実行する権限を持つ
                担当者に割り当てる必要があります。このロールでは
                機密情報である人事および給与情報に
                直接アクセスすることはありません。
            </desciption>
        <role-name>admin</role-name>
        <security-role>
... <assembly-descriptor>
```

メソッドのパーミッションの定義

アプリケーション編成者は、次のように method permission 要素を使って、DD 内でメソッドのパーミッションの関係を定義します。

各 `method-permission` 要素には、1 つまたは複数のセキュリティロールのリストと 1 つまたは複数のメソッドのリストが含まれています。一覧表示されたセキュリティロールは一覧表示されたすべてのメソッドを起動できます。リスト内の各セキュリティロールは `role-name` 要素によって識別され、各メソッド (または下記の一連のメソッド) は method 要素によって識別されます。description 要素を使って オプションの説明を `method-permission` 要素に関連付けることができます。

メソッドパーミッションの関係は、個々の method permission 要素に定義したすべてのメソッドパーミッションの結合として定義されます。

セキュリティロールまたはメソッドは複数の `method-permission` 要素内に存在することがあります。

次の例は、DD 内でセキュリティロールがメソッドパーミッションに割り当てられる方法を示します。

```
...
<method-permission>
   <role-name>employee</role-name>
       <method>
       <ejb-name>EmployeeService</ejb-name>
       <method-name>*</method-name>
   </method>
</method-permission>


<method-permission>
   <role-name>employee</role-name>
   <method>
       <ejb-name>AardvarkPayroll</ejb-name>
       <method-name>findByPrimaryKey</method-name>
   </method>
```

```
    <method>
        <ejb-name>AardvarkPayroll</ejb-name>
        <method-name>getEmployeeInfo</method-name>
    </method>
    <method>
        <ejb-name>AardvarkPayroll</ejb-name>
        <method-name>updateEmployeeInfo</method-name>
    </method
</method-permission>
...
```

ここでの相互作用はありません。配置ツールはこれらをセキュリティ要素に変換します。

「セキュリティロール参照」

Bean の提供者は、DD の `security-rol-ref` 要素内に、Enterprise JavaBeans で使うすべてのセキュリティロール名を宣言する必要があります。

アプリケーション編成者は、`security-role-ref` 要素内で宣言されたセキュリティロールのすべての参照を、`security-role` 要素で定義されたセキュリティロールにリンクする必要があります。アプリケーション編成者は、`role-link` 要素を使って各セキュリティロールの参照をセキュリティロールにリンクします。`role-link` 要素の値は、`security-role` 要素で定義されたセキュリティロールの名前の一つである必要があります。

次の DD の例は、`payroll` という名前の Sudety のロール参照を `payroll-department` という名前のセキュリティロールにリンクする方法を示します。

```
...
<enterprise-beans>
    ...
    <entity>
        <ejb-name>AardvarkPayroll</ejb-name>
        <ejb-class>com.aardvark.payroll.PayrollBean</ejb-class>
    ...
        <security-role-ref>
        <description>
```

このロールは給与支払い部門の社員に割り当てる必要があります。このロールが割り当てられたメンバーは全員の給与記録にアクセスできます。ロールは `payroll-department` ロールにリンクされています。

```
                </description>
                </security-role-ref>
            ....
        </entity>
    ...
</enterprise-bean>
```

シングルサインオンでのユーザ認証

iPlanet Application Server 上でのアプリケーション全体のシングルサインオンは iPlanet Application Server の Servlet および JSP によってサポートされます。この機能によって、ユーザを個別のアプリケーションに対して別々にサインオンさせずに、同一のサインオン情報が必要な複数のアプリケーションでこの情報を共有できます。これらのアプリケーションは一回でユーザを認証できるように作成され、この認証情報は必要に応じてほかの関連するアプリケーションに伝えられます。

シングルサインオンのシナリオを使うアプリケーションの例としては、すべての航空会社を検索し、各航空会社の Web サイトへのリンクを提供する、統合航空券予約サービスがあります。ユーザが統合予約サービスにサインオンすると、そのユーザ情報を各航空会社のサイトで使用できるので、別のサインオンを要求する必要はありません。

シングルサインオンの設定方法

Web コンテナの iPlanet Application Server 固有の DD には `session-info` という名前の要素があり、コンテナ内の Servlet および JSP の認証を指定するフィールドがあります。DD は配置ツールによって作成されます。この節では、DD 内の `session-info` 要素のセキュリティフィールドが連動してシングルサインオン認証を実行する方法を中心に説明します。iPlanet Application Server 固有の Web コンテナの DD の作成方法の詳細については、iPlanet Application Server 配置ツールのオンラインヘルプおよび『管理者ガイド』を参照してください。すべての `session-info` フィールドの詳細な説明については、第 11 章「配置のためのパッケージ化」を参照してください。

表 13-1 は、認証プロセスで使う `session-info` 要素フィールドを示しています。

表 13-1　シングルサインオンのためのセキュリティフィールド

フィールド	説明
domain	このフィールドはブラウザから cookie を送り返すドメインを指定します。デフォルト (ユーザがドメインを指定しない場合) では、cookie を設定する URL のドメインが cookie を送り返すドメインと見なされます。domain には cookie を送る任意のドメインを指定できます。domain には少なくとも 2 つのピリオドが必要で、3 つの場合もあります (.acme.com、.acme.co.in など)。
path	このフィールドはセッションの cookie へのパスを指定します。これはブラウザから cookie を送り返すために URL に必要な最低限のパスです。たとえば、パスを /phoenix に設定すると、次の URL のどれかがアクセスされたときに cookie が送り返されます。 http://my.Who.com/phoenix/birds.html または http://my.Who.com/phoenix/bees.html パスは "/" で始まる必要があります。パスが設定されていない場合、デフォルトのパスが cookie を設定している URL と見なされます。
scope	このフィールドは同じユーザセッションを共有するアプリケーションを「関連付ける」グループの名前を指定します。すなわち、1 つのアプリケーションにサインオンすると、ユーザはほかのアプリケーションにサインオンせずに自動的にアクセスできます。グループ化されたアプリケーションは、iPlanet Application Server 固有のそれぞれの Web XML DD ファイル内に、同じ scope フィールド値を持つ必要があります。

シングルサインオンの例

AirlineSearch および AirlineBooking という名前の iPlanet Application Server 上で管理される 2 つのアプリケーションを考えます。両方とも myairlines.com ドメインの一部で、この 2 つのアプリケーション内のリソースへのアクセスの認証をユーザに要求します。AirlineSearch では、ユーザは利用可能なさまざまな航空会社を検索できます。また、AirlineBooking では、座席、メニュー、出発時刻などのユーザの特別な希望に基づいて予約できます。

AirlineSearch および AirlineBooking の ias-web.xml には次の記述が含まれています。

```
<session-info>
   <path>/iASApp</path>
   <scope>AirlineSignon</scope>
</session-info>
```

ここで、次の URL を使って、AirlineSearch アプリケーションが提供するサービスにまずアクセスします。

```
http://www.myairlines.com/iASApp/AirlineService/showFlights
```

showFlights は、ユーザが要求した時刻のすべてのフライトを表示する Servlet です。ここで、ユーザはログインする必要があります。ユーザはすべてのフライトを参照し、チケットの予約を決めたら、アクセスします。

```
http://www.myairlines.com/iASApp/AirlineService/bookFlights
```

このサイトでは、ユーザの希望に基づいてフライトを予約するサービスを提供します。このサービスは、前のアクセスおよび前の AirlineService アプリケーションに提供されたサインオン情報によって利用可能になります。

両方のアプリケーションは同一のドメイン内にあるので、この例では domain フィールドは設定されていません。ただし、複数ドメイン間でサインオン情報を共有するように拡張できます。

RMI/IIOP クライアントのユーザ認証

RMI/IIOP クライアントパスでのセキュリティは、iPlanet Application Server のセキュリティインフラストラクチャに統合されています。CXS は iPlanet Application Server のセキュリティマネージャを使って、LDAP に保存されているユーザ情報によってクライアントを認証します。クライアントの証明書はブリッジを介してクライアントから EJB に渡されます。クライアントサイドのコールバックによって、ユーザ名とパスワードでのクライアントのログインが開始されます。この情報を得るためにインスタンス化されるオブジェクトタイプは、クライアント上の環境設定によって指定されます。認証に失敗した場合、クライアント側はログインプロセスを再試行するようにセットアップされます。現在、再試行の数は 3 回にハードコードされています。

RMI/IIOP クライアントの DD 内の要素の詳細については、333 ページの「RMI/IIOP クライアント XML DTD」を参照してください。

セキュリティ情報のガイド

次の種類の各情報を、短い説明、情報の保存場所、情報の作成方法、情報へのアクセス方法、および詳しい情報を参照できる場所とともに示します。

- ユーザ情報
- セキュリティロール

ユーザ情報

ユーザ名、パスワードなど

場所 :
Directory Server

作成方法
Mission Console を使って作成するか、LDAP SDK を使ってプログラムで作成します。詳細については、iPlanet Application Server 配置ツールのオンラインヘルプおよび『管理者ガイド』を参照してください。

セキュリティロール

アプリケーションの機能を定義するロール。多数のユーザやグループから構成されています。LDAP グループは、iPlanet Application Server 内のロールとして機能します。

場所 :
Directory Server

作成方法
iPlanet Application Server 配置ツールを使います。

アクセス方法
`isCallerInRole()` を使ってユーザのロールメンバーシップをテストします。

Web サーバからアプリケーションサーバのコンポーネントのセキュリティ

iPlanet Application Server 6.0 SP2 以降では、開発者はコンポーネントごとに Web サーバと KXS の間のトラフィックを選択して暗号化できます。暗号化は、128 ビットキー と RSA Bsafe3.0 ライブラリを使って行います。クレジットカード情報収集 Servlet、ログイン Servlet などの高度なセキュリティが必要なコンポーネント (Servlet や JSP) では、開発者が暗号化を有効にすることをお勧めします。

これらのコンポーネント間でトラフィックの暗号化を有効にするには、暗号化をサポートするアプリケーションサーバ自体を有効にする必要があります。必要な手順は次のとおりです。

1. `CCS0¥¥SECURITY¥¥EnableEncryption=D` を設定します (内部 128 ビット、データタイプは `string`)。
2. KXS ログ内のログメッセージの暗号化を確認する場合、`CCS0¥¥SECURITY¥¥LogEncryption=1` のエントリまたは値を作成します (データタイプは `integer`)。
3. `CCS0¥¥EXTENSIONS¥¥CRYPTEXT¥¥CRYPTSVC¥¥ENGINES¥¥0` キーを作成します。
4. Web サーバと iPlanet Application Server を再起動します。

暗号化を有効にする必要があるすべてのコンポーネントでは、次の手順を行います。

1. `j2eeappreg`、`webappreg`、または `iasdeploy` (推奨) を使ってアプリケーションを登録します。
2. 暗号化するコンポーネント (Servlet や JSP) の `ias-web.xml` ファイル内の `<encrypt>true</encrypt>` を設定します。

暗号化が有効で正しく動作することを確認するには、KXS ログを開き、次のようなメッセージを検索します。

```
[11/Jan/2001 19:58:43:0] info:CRYPT-003:Encrypting 2309 bytes,
keysize = 128 bits

[11/Jan/2001 19:58:43:5] info:NSAPICLI-012:plugin reqstart, tickct:
1903570535

[11/Jan/2001 19:58:43:5] info:NSAPICLI-009:plugin
reqexit:0s+.12995s. (198114 0537)

[11/Jan/2001 19:58:52:2] info:CRYPT-004:Decrypting 1897 bytes,
keysize = 128 bits
```

第 14 章

iPlanet Application Server の機能の活用

この章では、iPlanet Application Server の機能をアプリケーションに実装する方法について説明します。iPlanet Application Server には、iPlanet Application Server 環境で使う Servlet を強化する多くの追加機能があります。これらの機能は正規の Servlet 仕様書には規定されていませんが、第 13 章「安全なアプリケーションの作成」で説明されている Servlet のセキュリティの枠組みのように、一部の機能は Sun Microsystems® の新しい標準をベースにしており、これらの新しい標準に準拠しています。

この章には次の節があります。

- Servlet の結果のキャッシュ
- startup クラス の使用法

Servlet の結果のキャッシュ

iPlanet Application Server には、その後の同じ Servlet への呼び出しをすばやく実行するため、Servlet の結果をキャッシュする機能があります。iPlanet Application Server は一定の期間、リクエストの結果 (たとえば、Servlet の実行) をキャッシュします。この方法では、別のデータ呼び出しを行うと、再びオペレーションを実行する代わりに iPlanet Application Server によってキャッシュされているデータを返すことができます。たとえば、5 分ごとに更新される株式相場を Servlet が返す場合、キャッシュが 300 秒後に期限切れになるように設定します。

結果をキャッシュするかどうか、およびキャッシュする方法は、結果に含まれているデータのタイプによって異なります。たとえば、クイズの投稿の結果は、Servlet への入力が毎回異なるため、キャッシュしても意味がありません。ただし、クイズの結果から収集した人口統計データを示す高度なレポートをキャッシュし、1 時間に 1 回更新することは可能です。

Servlet コンフィグレーションファイル内で特定のフィールドを編集することによって、iPlanet Application Server の Servlet がメモリキャッシュを処理する方法を定義できます。このように、有効な iPlanet Application Server 機能を利用した標準 Servlet をプログラムによって作成できます。

表 14-1 は、Servlet コンフィグレーションファイル内のキャッシュ設定を示しています。

表 14-1 Servlet キャッシュ設定

「Name」	「Type」	「Value」
`cache-timeout`	整数	オプション。この Servlet のメモリキャッシュが解放されるまでの経過時間 (単位は秒)
`cache-size`	整数	オプション。Servlet のメモリキャッシュのサイズ (単位は KB)
`cache-criteria`	文字列	オプション。カンマで区切られた記述子を持つ基準式の文字列。各記述子はどれかの Servlet 入力パラメータとの一致を定義します。
`cache-option`	文字列	オプション。`TIMEOUT_CREATE` と `TIMEOUT_LASTACCESS` のどちらかにキャッシュのタイムアウトオプションを設定します。

これらの設定の詳細については、319 ページの「Servlet のキャッシュを指定する要素」を参照してください。

`cache-criteria` フィールドは、Servlet 結果をキャッシュするかどうかを決める基準を設定します。このフィールドは、リクエスト内の 1 つまたは複数のフィールドをテストします。その結果、値や 1 つまたは複数のフィールドの有無に従って条件付きで結果をキャッシュできます。テストに合格すると、Servlet の結果がキャッシュされます。

表 14-2 は `cache-criteria` フィールドシンタックスを示します。

表 14-2 `CacheCriteria` フィールド

シンタックス	説明
arg	arg の値が入力パラメータリストにあるかどうかのテスト。たとえば、フィールドが `"EmployeeCode"` に設定されている場合、リクエストに `"EmployeeCode"` フィールドが含まれていると、その結果がキャッシュされます。

表 14-2　CacheCriteria フィールド (続き)

シンタックス	説明
arg=v	*arg* が *v* (文字列または数式) と一致するかどうかのテスト。たとえば、フィールドが "stock=NSCP" に設定されている場合に、リクエストに値 NSCP を持つ stock フィールドが含まれていると、その結果がキャッシュされます。別の値を使って Servlet を実行する場合に新しい結果セットをキャッシュするには、引数にアスタリスク (*) を割り当てます。たとえば、基準が "EmployeeCode=*" に設定されていると、リクエストオブジェクトに "EmployeeCode" と呼ばれるフィールドが含まれており、その値が現在キャッシュされている値とは異なる場合は、結果がキャッシュされます。
arg=v1|*v2*	*arg* がリストの値 (*v1*、*v2* など) と一致するかどうかのテスト。次のようにします。"dept=sales|marketing|support".
arg=n1-n2	*arg* の数値が指定範囲内にあるかどうかのテスト。次のようにします。"salary=40000-60000".

startup クラスの使用法

startup クラスは、iPlanet Application Server の起動時にメモリに自動的に読み込まれる、ユーザ定義のクラスオブジェクトです。アプリケーションサーバの環境内で初期化タスクを実行します。StartupClass オブジェクトの特性は次のとおりです。

- そのクラスを実行したサーバが存在するかぎり保持される
- サーバが終了するときに通知を受ける
- kjs プロセスの JVM 内で実行するので、各 kjs プロセスには StartupClass オブジェクトのインスタンスが 1 つある

starup クラスは次の必要条件を満たす必要があります。

- パッケージ com.iplanet.ias.startup 内にあること
- 名前が StartupClass であること
- インタフェース com.iplanet.ias.startup.IStartupClass を実装すること

注　StartupClass オブジェクトは、アプリケーションサーバのインスタンスに 1 つのみ配置できます。

次の節では、starup クラスの作成および使用法について説明します。

- IStartupClass インタフェース
- Startup クラス のビルド

- Startup クラスの配置
- kjs による StartupClass オブジェクトの処理方法

IStartupClass インタフェース

StartupClass クラスは IStartupClass インタフェースを実装する必要があります。IStartupClass インタフェースは 2 つのメソッドを定義しています。

- `public void startUp() throws StartupClassException`

  kjs を起動すると (kjs がこのメソッドを呼び出す StartupClass デフォルトコンストラクタを呼び出すと)、このメソッドはアクティビティを実行するために呼び出されます。このメソッドは任意のアクションを実行できます。メソッドは、kjs エンジンがすべての関連コンテキストを設定したあとに呼び出されるので、EJB および JDBC リソースにアクセスできます。

  例外が発生した場合、このメソッドは com.iplanet.ias.startup.StartupClassException をスローします。

- `public void shutDown()`

  このメソッドは、開始時に割り当てられたすべてのリソースの割り当てを解除する必要があります。

startUp メソッドが失敗した場合、com.iplanet.ias.startup.StartupClassException をスローします。そのシグネチャは次のとおりです。

```
public class StartupClassException extends java.lang.Exception
```

その 1 つのコンストラクタは次のとおりです。

```
public StartupClassException(java.lang.String msg)
```

Startup クラスのビルド

クラスのビルドは Ant を介してサポートされます (Ant の使用は必須ではない)。ビルドに必要なファイルは Install_dir/startup ディレクトリにあるので、このディレクトリで StartupClass ファイルおよび独立した Java ファイルをビルドすることをお勧めします。まず、次の操作を実行します。

- Shell の PATH 環境変数に *install_dir*/bin を入れます。
- Shell の PATH 環境変数に JDK へのパスを入れます。

- *install_dir*/startup ディレクトリで StartupClass ファイルをビルドしない場合は、そのディレクトリからビルドに使うディレクトリに StartupClass.java、startup.properties、および build.xml ファイルをコピーします。

ビルドオプションは次のとおりです。

build compile	*install_dir*/startup ディレクトリ内のすべての Java ファイルをコンパイルし、classes サブディレクトリ内にクラスファイルを配置します。
build jar	build compile を実行し startup.jar ファイルにクラスファイルを保存し、classes サブディレクトリ内にこのファイルを配置します。
build clean_jar	startup.jar ファイルを削除します。
build clean	classes サブディレクトリとそのサブディレクトリを削除します。
build deploy	*install_dir*/STARTUPCLASS に startup.jar ファイルを配置します。
build	デフォルトビルド。build clean、build compile、build jar、および build deploy を順番に実行します。

注　　.jar ファイル名は startup.jar である必要があります。

Startup クラスの配置

配置は iasdeploy ツールを介してサポートされます。配置には次の 2 種類があります。

- ローカル配置

  iasdeploy deploystartup *path*/startup.jar

  次のようにします。

  iasdeploy deploystartup /iasroot/ias/startup/classes/startup.jar

- リモート配置

  iasdeploy deploystartup -host *server* -port *port* -user *userName* -password *password* *path*/startup.jar

次のようにします。

```
iasdeploy deploystartup -host myserver -port 80 -user jjones
-password secret /iasroot/ias/startup/classes/startup.jar
```

path は次のとおりです。

- `iasdeploy` が実行されるディレクトリから `startup.jar` ファイルへの相関パス
- *install_dir*`/startup/classes` などの絶対パス

複数マシンへの同時配置は、`iasdeploy` ツールではサポートされていません。

`startup.jar` ファイルは *install_dir*`/STARTUPCLASS` ディレクトリに配置されます。

注	*install_dir*`/STARTUPCLASS` 以外のディレクトリに starup クラスを配置した場合、アプリケーションサーバが起動すると、メッセージが `kjs` ログで生成されます。

注	配置ツールは、startup クラスモジュールのアセンブリをサポートしません。

kjs による StartupClass オブジェクトの処理方法

独自の JVM で実行する各 `kjs` プロセスには、`StartupClass` オブジェクトのインスタンスが 1 つあります。

`com.kivasoft.engine.Engine.java` の `run` メソッド内では、環境が設定されてからほかのメソッドが実行されるまでの間に、システムクラスローダによって `StartupClass` オブジェクトが作成されます。`StartupClass` オブジェクトが作成されると、`startUp` メソッドが実行されます。

`startUp` メソッドが正しく実行された場合、`kjs` はメソッドが正常に終了するまで実行します。`iascontrol stop` を使った正規の終了では、`kjs` が `StartupClass` オブジェクトの `shutDown` メソッドを呼び出します。

`startUp` メソッドが `com.iplanet.ias.StartupClassException` をスローした場合は、`kjs` が `shutDown` メソッドを呼び出し、`StartupClass` オブジェクトはすぐにガベージを収集します。最後に `kjs` が終了します。

注	各 `kjs` プロセスには `StartupClass` オブジェクトの独自のコピーがあるので、注意して startup クラスを設計する必要があります。共有リソースの同期の問題を処理することをお勧めします。

付録 A

Java Message Service の使用法

この付録では、Java Message Service (JMS) API の使用方法を説明します。iPlanet Application Server では、サードパーティの JMS プロバイダによる Java 環境への統合が可能であり、コネクションプーリングとユーザ ID マッピングという 2 つの付加価値機能を備えています。

この付録には次の節があります。

- JMS API について
- JMS の有効化とプロバイダの統合
- アプリケーションでの JMS の使用法
- JMS 管理
- サンプルアプリケーション

JMS API について

JMS は、エンタープライズメッセージングシステムへの Java 言語インタフェースの標準セットを備える J2EE API で、しばしば「メッセージ指向のミドルウェア」と呼ばれます。このインタフェースは、JMS プロバイダによって実装されます。IBM MQ シリーズでは、iPlanet Application Server は iPlanet Message Queue および JMS プロバイダをサポートしています。iPlanet Message Queue の詳細については、次のマニュアルを参照してください。

`http://docs.iplanet.com/docs/manuals/javamq.html`

`http://java.sun.com/products/jms/index.html` の JMS Web ページでは JMS の目的を次のように説明しています。

エンタープライズメッセージングによって、企業内で重要なビジネスデータおよびイベントを非同期で交換するための信頼性の高い、柔軟性のあるサービスが提供されます。これに加えて、JMS API によって共通 API とプロバイダフレームワークが追加されます。このフレームワークによって、Java プログラミング言語で書かれ、移植可能な、メッセージベースのアプリケーションの開発が可能になります。

また、iPlanet Application Server には、JMS コネクションプーリングとユーザ ID マッピングも含まれています。これらは管理フレームワームを通じて提供され、iPlanet Application Server 固有のコードは必要ありません。アプリケーションでは、これらの機能を透過的に使用でき、コンポーネントの移植性を維持できます。

JMS メッセージングスタイル

JMS では、2 つのメッセージングスタイルがサポートされています。

- Point-to-point：このスタイルを使うと、`Queue` という名前の `Destination` を通してメッセージを送受信することによって、2 つのプログラム間で通信できます。
- Publish/subscribe：このスタイルを使うと、`Topic` という名前の `Destination` を通して複数のメッセージングプログラム間で通信できます。メッセージは、`Topic` をパブリッシュすることによって送信されます。メッセージは、加入者によって受信されます。

メッセージングスタイルにかかわらず、アプリケーションと JMS プロバイダ間のリンクは、コネクションオブジェクトです。アプリケーションは、コネクションオブジェクトをコネクションファクトリから取得します。

JMS プロバイダ間でアプリケーションの移植性を最大に高めるために、プロバイダ固有のメッセージング機能が、管理されたオブジェクトの中にカプセル化されています。JMS 管理オブジェクトによって、次の 4 つの JMS インタフェースを実装できます。そのうちの 2 つは、メッセージングスタイル用です。

- デスティネーション
  - `Queue`
  - `Topic`
- `ConnectionFactory`
  - `QueueConnectionFactory`
  - `TopicConnectionFactory`

JMS プロバイダによって、これらのインタフェースを実装するサプライクラスが提供されます。配置ツールを使って、管理オブジェクトクラスのインスタンスを作成および設定し、それらを配置の要件に合わせて設定します。管理者は、配置ツールを使って、プロバイダ固有のパラメータを設定します。

このプログラムモデルを使うと、プロバイダから完全に独立した JMS プログラムを記述できます。アプリケーションでは、JNDI を使って、名前で管理オブジェクトを検索します。

次のサンプルでは、コネクションファクトリとデスティネーションを検索し、簡単なテキストメッセージをキューに送ります (わかりやすくするために、例外処理は省略)。

```
// JNDI を使ってコネクションファクトリとデスティネーションを検索します。
Context ctx = new InitialContext();

QueueConnectionFactory factory;

factory = (QueueConnectionFactory) ctx.lookup
("java:comp/env/jms/theFactory");Queue queue = (Queue)
ctx.lookup("java:comp/env/jms/theQueue");

// コネクション、セッション、送信者、およびメッセージを作成します。
QueueConnection conn;
conn = factory.createQueueConnection("myUserName", "myPassword");
QueueSession session = connection.createQueueSession (false,
Session.AUTO_ACKNOWLEDGE);
QueueSender sender = session.createSender(queue);
TextMessage msg = session.createTextMessage();
msg.setText("Hello from a simple Java Message Service Application");

// コネクションを確立し、メッセージを送信します。
connection.start();
sender.send(msg);
connection.start();

// すべてのリソースを閉じ、確実にネイティブリソースを解放します。
sender.close();
session.close();
connection.close();
```

アプリケーションではリソースの名前をハードコードしませんでしたが、アプリケーションの配置に関する節で説明したように、代わりに J2EE リソース参照を使っていることに注目してください。iPlanet Application Server 配置マネージャは JMS リソース参照をサポートしないため、アプリケーションでは、JMS サブコンテキストのオブジェクトを直接参照する必要があります。

JMS の有効化とプロバイダの統合

iPlanet Application Server には、JMS プロバイダを統合するソフトウェアがありますが、そのソフトウェアを有効にする必要があります。iPlanet Application Server に JMS プロバイダを統合する方法については、次のマニュアルを参照してください。

install_dir/ias/ias-samples/jms/docs/index.html

アプリケーションでの JMS の使用法

JMS の iPlanet Application Server に対するサポートは、全面的に標準 J2EE API をベースにしています。付加価値機能を使ったアプリケーションコンポーネントは、ほかの J2EE 環境に移植可能です。この節では、iPlanet Application Server に配置されたアプリケーションで JMS を使うときに考慮する必要があるいくつかの問題について説明します。

JNDI とアプリケーションコンポーネントの配置

JMS オブジェクトは、配置ツールによって、iPlanet Application Server ルートの JNDI ネームスペースの JMS サブコンテキスト内に保存されます。JMS サブコンテキストでは、そのサブコンテキスト自体の作成はサポートされません。コンポーネントアプリケーションコンテキストへのリンクは、アプリケーション配置時に確立されます。

デフォルトのパラメータを持つ `InitialContext` が作成されると、JMS オブジェクトは、`jms/` で始まる名前で参照できます。J2EE リソース参照を使うと柔軟性が高まります。これは、389 ページに示すサンプルで実証されています。ここでは、ファクトリについて検索される名前が `java:comp/env/jms/theFactory` でした。iPlanet Application Server JMS では、JMS リソース参照がサポートされていません。JMS オブジェクトは、直接参照される必要があります。

コネクションファクトリプロクシ

iPlanet Application Server では、JMS コネクションプーリングとユーザ ID マッピングがサポートされます。`ConnectionFactoryProxy` クラスは、アプリケーションと JMS プロバイダのコネクションファクトリ間で機能します。2 つのプロクシクラスがあり、メッセージングスタイルごとに 1 つあります。

- `QueueConnectionFactoryProxy`
- `TopicConnectionFactoryProxy`

プロクシクラスによって表された API は標準 JMS API で、`QueueConnectionFactory` と `TopicConnectionFactory` です。管理者だけがプロクシを管理する必要があり、プロクシはアプリケーションに対して透過的に使われます。

簡単な管理プログラムを使って `ConnectionFactoryProxies` の設定を行います。プロクシは、コネクションプーリングとユーザ ID マッピングを処理します。JMS オペレーションは、管理者によって指定されたプロバイダファクトリから、プロクシによって取得されたコネクションに転送されます。

コネクションプーリング

JMS コネクションのセットアップはネットワーク集約型で、そのため費用がかかります。コネクションプーリングを使うと、JMS コネクションの再利用が容易になります。プーリングが有効のままアプリケーションがコネクションを閉じると、プロクシはプロバイダコネクションを閉じるのではなく、そのコネクションをプールに返します。後続のアプリケーションで、同じユーザ名とパスワードを使ってコネクションを作成しようとすると、プロクシはそのコネクションを再利用します。

ユーザ ID マッピング

また、`ConnectionFactoryProxy` には、ユーザ ID マッピング機能が用意されています。JMS プロバイダは、アプリケーションサーバと同じセキュリティインフラストラクチャを使わず、別のユーザネームスペースを持っています。ユーザ ID マッピングによって、管理者は柔軟にセキュリティインフラストラクチャを設計できます。

コネクションファクトリプロクシクラスによって 2 つのマッピング形式が用意されています。

- デフォルトユーザ名
- 明示的なユーザ ID マッピング

コネクションプーリングと同様に、この機能は標準 JMS API 内のプロクシクラスによって実装されます。このユーザ ID マッピングを使うと、メッセージングシステムへのアクセスを管理する iPlanet Application Server ユーザセキュリティメカニズムによって配置が決まります。

デフォルトユーザ名について

デフォルトユーザ名とパスワードを使うと、複数のアプリケーションユーザが、1 つのメッセージンシステムプロバイダのユーザ ID とパスワードを共有できます。

プロクシの作成時に、管理者はデフォルトのプロクシユーザ名とパスワードを定義します。引数なしのコネクションクリエートメソッドを起動するアプリケーションでは、コネクション作成時にプロバイダファクトリにこれらの値を渡します。たとえば、アプリケーションが次の呼び出しを行う場合、

```
connection = proxy.createQueueConnection();
```

デフォルトユーザ名が設定されていると、iPlanet Application Server のプロクシ実装によって、次の JMS コネクションを取得します。

```
connection = providerFactory.createQueueConnection (defaultUserName,
defaultPassword);
```

明示的なユーザ ID マッピングについて

明示的なユーザ ID マッピングも使用できます。マップには、エントリリストがあります。各エントリは、固有のユーザ ID キーによって参照され、各エントリには次の 2 つの値があります。

- `jmsUserName`
- `jmsPassWord`

管理者は、`jmsuadm` ツールを使ってマップを作成します。エントリの値はコネクション作成時に使われます。たとえば、アプリケーションでは、プロクシを使って、次のようにコネクションを作成します。

```
connection = proxy.createQueueConnection(userString,
passWordString);
```

iPlanet Application Server プロクシでは、マップ内の `userString` のエントリを検索します。エントリが見つかると、プロクシは、アプリケーションから提供されたパスワードを無視し、エントリからの `jmsUserName` と `jmsPassWord` 値を JMS プロバイダファクトリに渡します。これにより、プロクシは次のコネクションを実行します。

```
connection = providerFactory.createConnection (entry.jmsUserName,
entry.jmsPassWord);
```

`userString` に一致するエントリがユーザ ID マップ内で見つからない場合は、アプリケーションによって与えられた値が JMS プロバイダファクトリ (`providerFactory`) に渡されます。

ConnectionFactoryProxies とアプリケーションクリエートスレッド

Servlet では Java スレッドを作成できますが、この方法は、お勧めしません。JMS コネクションプーリングインフラストラクチャでは、ユーザクリエートスレッドが認識されません。アプリケーションでは、ユーザクリエートスレッドから、コネクションクリエートまたはコネクションクローズメソッドを呼び出すことができません。これらのメソッドを呼び出そうとすると、次のような結果になります。

```
javax.jms.IllegalStateException
```

これは、JMS ベータ版では実装されていません。ベータ版では、アプリケーションクリエートスレッドから、コネクションクリエートまたはコネクションクローズを実行しようとすると、KJS がクラッシュします。

サポートされていない JMS 機能

iPlanet Application Server は、JMS 仕様書に記載されている JMS `XAConnection` およびサーバセッションプール機能はサポートしていません。

JMS 管理

JMS API は、移植性が管理されているオブジェクトに依存します。プロバイダ固有の配置は、管理されているオブジェクトにカプセル化され、アプリケーションコードが移植可能になります。iPlanet Application Server の環境では、JMS 管理は 4 つのタスクから構成されています。

- JMS プロバイダファクトリおよびデスティネーションの作成
- ユーザ ID マッピングの作成
- `ConnectionFactoryProxies` の作成
- iPlanet Application Server レジストリコネクションプーリングパラメータの変更

JMS オブジェクト配置ツール

各 JMS 製品には管理プログラムが必要です。このツールによって、オブジェクトを作成し、それを iPlanet Application Server JNDI の名前にバインドします。この節では、JMS JNDI コンテキストで動作するようにツールを設定するために必要な Java プロパティとシステムパスについて説明します。特定のツールの設定方法については、プロバイダのマニュアルを参照してください(iPlanet Application Server 用の IBM MQ JMS の配置ツールを起動するスクリプトについては次の節で説明します)。

表 A-1 は、`InitialContext` 作成時に JMS コンテキストにアクセスするために使うプロパティ値を示しています。

表 A-1 Java プロパティの名前と値

Java プロパティの名前	プロパティ値
`Java.naming.factory.initial`	`com.netscape.server.jndi.ExternalContextFactory`
`Java.naming.provider.url`	`/jms`

JMS 配置ツール用の JNDI プロパティ

`JMSContext` にアクセスするために必要な Java クラスの場合は、Java の実行時 `classpath` に次の 3 つの `.jar` ファイルを加えます。

- `GX_ROOTDIR/classes/java/jms.jar`
- `GX_ROOTDIR/classes/java/javax.jar`
- `GX_ROOTDIR/classes/java/kfcjdk11.jar`

ここで、`GX_ROOTDIR` は iPlanet Application Server がインストールされた場所です。次に例を示します。

```
/usr/iPlanet/ias6/ias
```

Solaris では、`LD_LIBRARY_PATH` に次のディレクトリが含まれている必要があります。

```
$GX_ROOTDIR/gxlib
```

IBM MQ の JMS オブジェクト管理

iPlanet Application Server には IBM MQ JMS 管理プログラムを起動する mqjmsadm スクリプトが組み込まれています。このスクリプトは、GX_ROOTDIR/jms/bin 内にあります。この管理プログラムは Java クラスです。mqjmsadm は、管理者または入力ファイルからの入力を受け入れる対話型コマンドラインプログラムです。

操作は、JMS 管理のための MQSeries のマニュアルに説明があります。mqjmsadm は自動的に JNDI 設定を処理するので、-cfg オプションを使う必要はありません。たとえば、コネクションファクトリとキューは、次の mqjmsadm セッションで作成できます。

```
# mqjmsadm
```

そのレスポンスは次のとおりです。

```
5648-C60 (c) Copyright IBM Corp. 1999. All Rights Reserved.

Starting MQSeries Classes for Java(tm) Message Service
Administration

Connected to LDAP server on localhost port 389InitCtx define
q(theQueue) queue(SYSTEM.DEFAULT.LOCAL.QUEUE)
InitCtx> define qcf(theFactory)
InitCtx display> ctx

Contents of InitCtx
a  aQueue             com.ibm.mq.jms.MQQueue
a  theProviderFactorycom.ibm.mq.jms.MQQueueConnectionFactory
2 Object(s)
0 Context(s)
2 Binding(s), 2 Administered
InitCtx> end
```

JMS コンテキストではサブコンテキストがサポートされていません。そのため、JMSAdmin コマンドを使ってサブコンテキストを操作すると、エラーメッセージが生成されます。

コネクションファクトリプロクシ管理

コネクションファクトリプロクシは、jmspadm コマンド (JMS プロクシ管理者) によって作成されます。このコマンド (UNIX 用のシェルスクリプトまたは NT 用の BAT ファイル) は、与えられたパラメータによってコネクションファクトリプロクシを作成し、それを JNDI にバインドする java プログラムを起動します。プロクシパラメータは、コマンドライン引数によって設定されます。

コマンドによって、プロクシ上で 3 つの操作が行われます。

- プロクシの作成
- プロクシの削除
- プロクシパラメータの一覧表示

プロクシの作成

プロクシを作成するには、次のコマンドを入力します。

```
jmspadm proxyName factoryName <-p or +p> <-u user password> <-m
userMapNam>
```

最初の 2 つの引数は必須です。

- 新しいプロクシに与えられる JNDI 名
- プロクシの対象になるコネクションファクトリの JNDI 名

JMS オブジェクトは、JMS サブコンテキストにしかないので、提供された名前が `jms` で始まっていない場合は、その文字列が先頭に付きます。たとえば、次の 2 つのコマンドは、同じ結果になります。

- `jmspadm theFactory theProviderFactory`
- `jmspadm jms/theFactory jms/theProviderFactory`

プロバイダ固有のツールを使って、`jmspadm` を実行する前にファクトリを作成し、ファクトリクラスを使用可能にします。残りの引数はオプションです。これらは、実行時にプロクシの操作を制御するときに使います。デフォルト設定は次のとおりです。

- コネクションプーリングは有効。`-p.` を使ってコネクションプーリングを無効にする
- デフォルトの `userid` と `password` はなし。`-u.` を使って設定する
- ID マッピングはなし。ユーザ ID マッピングの JNDI 名がプロクシによって使われるように設定する方法については次の節で説明する

プロクシの削除

プロクシを削除するシンタックスは次のとおりです。

```
jmspadm -d proxyName
```

プロクシパラメータの一覧表示

JNDI に保存されたすべてのプロクシを一覧表示するには、コマンド jmspadm -l を使います。

ユーザ ID マッピング管理

ユーザ ID マッピングを作成するには、管理者は、XML ファイルを用意する必要があります。このファイルが準備できたら、jmsuadm コマンドを使います。このコマンドには、3 つの種類があります。

- jmsuadm mapName mapFileName は、指定されたファイルを読み取り、ユーザ ID マッピングを作成する
- jmsuadm -d mapName はマップを削除する
- jmsuadm -l はマップ名を一覧表示する

セキュリティ保護のため、マップの内容は一覧表示できません。管理者は、注意して入力ファイルを保護する必要があります。

入力ファイルのフォーマットは XML です。DTD のパブリック名は次のとおりです。

```
-//Sun Microsystems, Inc.//DTD iAS JMS User Identity Map 1.0//EN
```

次の入力ファイルには、2 人の JMS ユーザのマッピング例を示します。

```
<?xml version="1.0" encoding="iso8859-1"?>

<!DOCTYPE jms-user-id-map PUBLIC "-//Sun Microsystems, Inc.//DTD iAS
JMS User Identity Map 1.0//EN" "TODO:fill this in">

<jms-user-id-map>
   <user>
       <name>bob</name>
       <jms-name>jmsuser</jms-name>
       <jms-password>secret</jms-password>
   </user>

   <user>
       <name>nancy</name>
       <jms-name>jmsuser2</jms-name>
       <jms-password>private</jms-password>
   </user>
</jms-user-id-map>
```

各ユーザ要素は、上記の例の次の 3 つの要素をすべて含んでいる必要があります。

- name

- jms-name
- jms-password

次のように、空の値も使用できます。

```
<jms-name></jms-name>
```

コネクションプーリングの設定

JMS コネクションプールの一部のパラメータは iPlanet Application Server レジストリに保存されます。必要があれば、iPlanet Application Server の bin ディレクトリ内の kregedit プログラムを使ってパラメータを調整できます。

パラメータは、登録されたデータソースごとに次のレジストリパス内に格納されます。

```
SOFTWARE¥iPlanet¥ApplicationServer¥6.5¥CCS0¥Datasource¥
```

コネクションプールのパラメータについては、『iPlanet Application Server 管理者ガイド』の第 8 章「データベース接続の管理」を参照してください。

サンプルアプリケーション

JMS サンプルアプリケーションは次のディレクトリにあります。

install_dir/ias/ias-samples/jms

デフォルトの JMS プロバイダ

iPlanet Message Queue (iMQ) for Java 2.0 SP1 は、iPlanet Application Server のデフォルトのメッセージングミドルウェアです。iMQ for Java 2.0 SP1 は、iPlanet Application Server プロダクト CD に収録されています。

実行時の注意事項

この付録には次のトピックがあります。

- 実行時環境
- クラスローダの階層
- ダイナミック再読み込み

実行時環境

コンポーネントをスタンドアロンモジュールとして登録するか、アプリケーションとして登録するかによって、ファイルシステムおよびレジストリの状態が変わります。図 B-1 は、スタンドアロンモジュールの実行時環境です。図 B-2 は、アプリケーションの実行時環境です。

標準モジュールの実行時環境

次の図は、モジュールベースで展開した場合の実行時環境です。ファイルシステムエントリの場合、モジュールは次のように抽出されます。

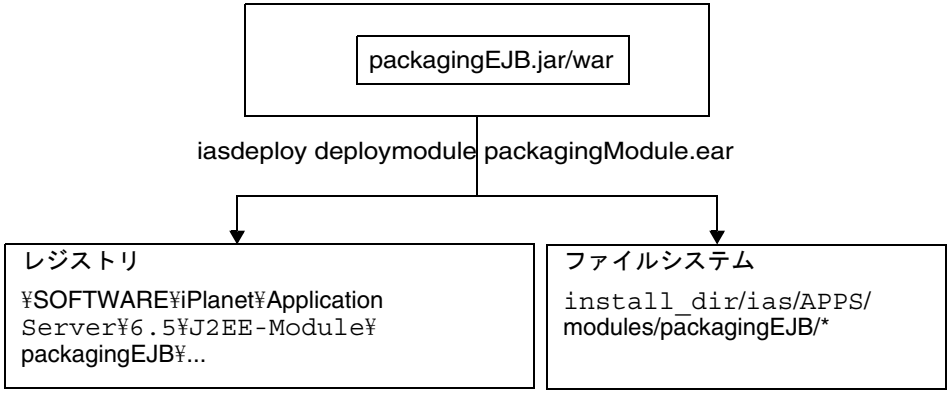
install_dir/ias/APPS/modules/*module_name*/*extracted_class*

レジストリエントリは、次のキーの下に追加されます。

SOFTWARE¥iPlanet¥Application Server¥6.5¥J2EE-Module¥*module_name*

ヒント すべての標準モジュールは、同じディレクトリの同じ LDAP を持つ場所に抽出されます。このため、モジュールの名前は重複しないようにしてください。

図 B-1　スタンドアロンモジュールの実行時環境



アプリケーションの実行時環境

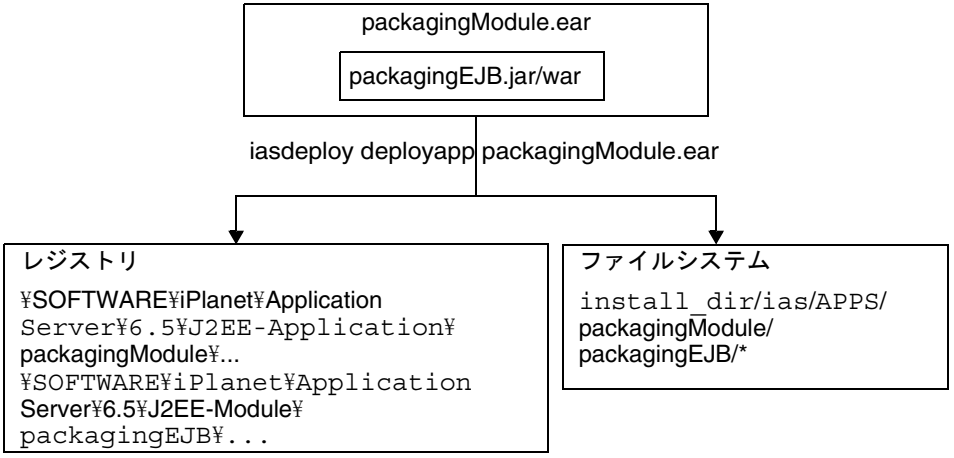
次の図は、アプリケーションベースで展開した場合の実行時環境です。ファイルシステムエントリの場合、アプリケーションは次のように抽出されます。

install_dir/ias/APPS/*app_name*/*module_name*/*extracted_class*

レジストリエントリの場合、アプリケーション内のモジュールは次のキーの下に追加されます。

SOFTWARE¥iPlanet¥Application Server¥6.5¥J2EE-Module¥*module_name*

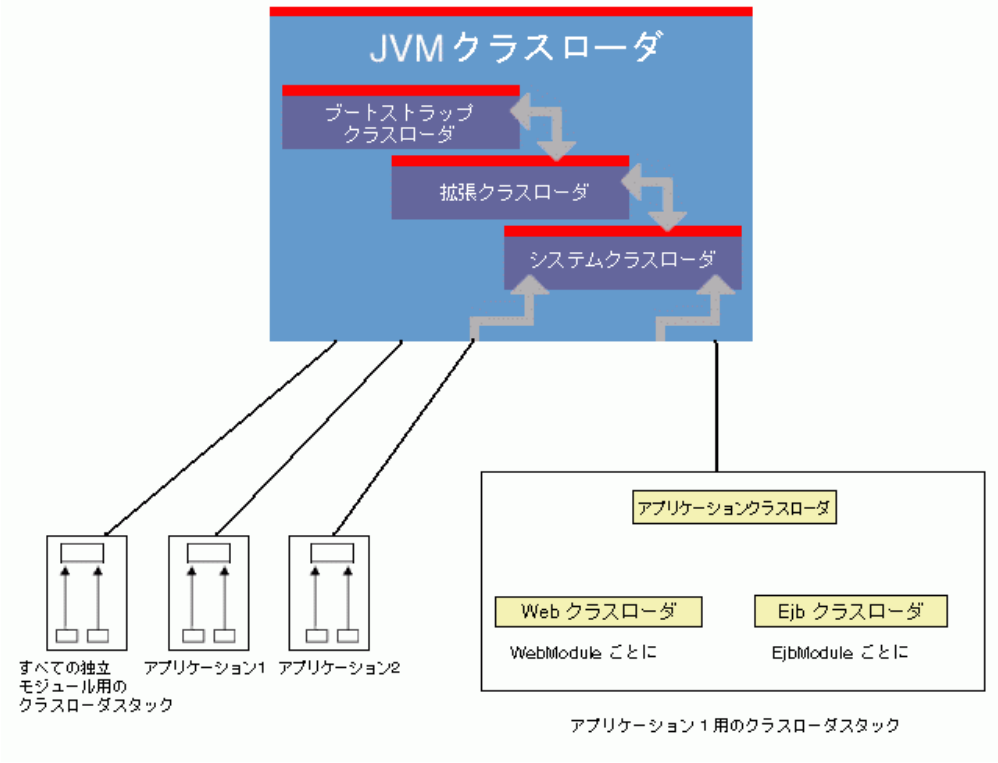
図 B-2　　アプリケーションの実行時環境



クラスローダの階層

Java 仮想マシン (JVM) のクラスローダは、依存関係の解決に必要な Java クラスファイルを動的に読み込みます。たとえば、`java.util.Enumeration` のインスタンスを作成する場合は、クラスローダの 1 つが関連するクラスを実行時環境に読み込みます。iPlanet Application Server 実行時環境のクラスローダは、図 B-3 に示す階層に構成されています。

図 B-3　　クラスローダの実行時階層



実行時にクラスおよびリソースを読み込むために、委譲方式が使用されています。この委譲方式では、クラスローダの各インスタンスが親のクラスローダと関連付けられます。親のクラスローダは、システムクラスローダであっても、別のカスタムクラスローダであってもかまいません。

クラスローダのインスタンスがクラスまたはリソースを検索するために呼び出されると、自らがクラスまたはリソースを検出しようとする前に、クラスやリソースの検索を親のクラスローダに委譲します。親クラスローダがクラスを読み込めない場合、カスタムクラスローダで、`findClass()` と呼ばれるメソッドが呼び出されます。

つまり、カスタムクラスローダは、親が読み込めないクラスだけを読み込みます。このようなクラスは、指定されたファイルシステムやネットワークなど、新しいタイプのクラスリポジトリに存在する可能性があります。クラスローダが参照するクラスリポジトリは、それぞれ異なります。クラスローダおよびクラスローダが参照するファイルについては、表 B-1 で説明します。

注

- iPlanet Application Server 内でバージョン指定が有効な場合、WebClassLoader/EjbClassLoader によって J2EE のコンポーネントクラス (Servlet/JSP および EJB 実装クラス) が読み込まれます。一方、その他のすべてのクラス （ヘルパークラス) は ApplicationClassLoader によって読み込まれます。

  同じパッケージ内に入っていた 2 つのクラスが別々のクラスローダで読み込まれ、しかも実行時のパッケージが異なる場合、それらは別のパッケージのクラスであると JVM にみなされます。

- したがって、servlet/JSP/EJB 実装クラスからのヘルパークラスにある protected メソッドにアクセスすると、`IllegalAccerssError` がスローされます。

- ダイナミックなクラス読み込みを利用したい場合は、protected メソッドを public にしてください。

表 B-1 iPlanet Application Server のクラスローダ

クラスローダ	説明
ブートストラップクラスローダ	ブートストラップクラスローダは、`rt.jar` の実行時クラスおよび `i18n.jar` の国際化クラスを参照する ブートストラップクラスローダは、拡張クラスローダおよびシステムクラスローダの親ローダである
拡張クラスローダ	インストール済み拡張クラスローダは、JRE の `lib/ext` ディレクトリにある JAR ファイルのクラスを参照する
システムクラスローダ	システムクラスパスクラスローダは、システムプロパティ `java.class.path` に指定されたパスにある JAR ファイルのクラスを参照する。システムクラスローダがクラスを読み込むには、関連するディレクトリをクラスパスに指定する必要がある。つまり、`iasenv.ksh` (UNIX の場合)、実行時環境 (UNIX または NT の場合)、または `¥Software¥iPlanet¥Application Server¥6.5¥Java¥Classpath` レジストリエントリ (NT の場合) を指定しなければならない
モジュールクラスローダ	iPlanet Application Server モジュールクラスローダは、*install_dir*`/ias/APPS/modules/*` の下にあるすべてのディレクトリにあるクラスを参照する。すべてのモジュールがこのクラスローダを共有する モジュールクラスローダは、アプリケーションクラスローダの別のインスタンスである

表 B-1　iPlanet Application Server のクラスローダ (続き)

クラスローダ	説明
アプリケーションクラスローダ	登録済み J2EE アプリケーションは、それぞれ対応するクラスローダによって読み込まれる。アプリケーションクラスローダは、*install_dir*/ias/APPS/*app_name* とそのすべてのサブディレクトリの下にあるクラスを参照する このクラスローダは、アプリケーション内の Web/Ejb クラスローダの親である。同様に、すべての登録済みスタンドアロンモジュールのクラスを読み込むためのモジュールクラスローダがある。モジュールクラスローダはアプリケーションクラスローダの別のインスタンスにほかならなく、すべてのスタンドアロンモジュールにおいて 1 つのアプリケーションに関して同じ階層構造を持つ 実際、すべてのスタンドアロンモジュールはデフォルトアプリケーションの一部とみなされる
Web クラスローダ	J2EE アプリケーション内の各 Web モジュール (WAR) には、1 つの Web クラスローダが割り当てられる。Web モジュール内のすべての Web コンポーネント、つまり Servlet クラスと JSP クラス (直接または間接的に javax.servlet.Servlet インタフェースを実装している) は、Web クラスローダによって読み込まれる。スタンドアロンの Web モジュールの場合は、Web クラスローダが作成され、モジュールクラスローダがその親である **注** Web コンポーネントクラスは、ダイナミックな再読み込みが有効な場合にだけ、Web クラスローダに読み込まれる。ダイナミックな再読み込みが無効な場合、アプリケーション内のすべてのクラスがアプリケーションクラスローダ (スタンドアロンの Web モジュールに対してはモジュールクラスローダ) で読み込まれる

表 B-1　iPlanet Application Server のクラスローダ (続き)

クラスローダ	説明
Ejb クラスローダ	J2EE アプリケーション内の各 Ejb モジュール (JAR) には、1 つの Ejb クラスローダが割り当てられる。Ejb モジュール内のすべての EJB コンポーネント (EJB 実装クラス) は、このクラスローダによって読み込まれる。このクラスローダには、アプリケーションクラスローダという共通の親クラスローダがある スタンドアロンの Ejb モジュールの場合は、Ejb クラスローダが作成され、モジュールクラスローダがその親である **注** Ejb コンポーネントクラスは、ダイナミックな再読み込みが有効な場合にだけ、Ejb クラスローダに読み込まれる。ダイナミックな再読み込みが無効な場合、アプリケーション内のすべてのクラスがアプリケーションクラスローダ (スタンドアロンの EJB モジュールに対してはモジュールクラスローダ) で読み込まれる

クラスローダ階層ごとの制限事項と対応策について説明します。

- すべての独立モジュール用の iPlanet Application Server モジュールクラスローダと iPlanet Application Server アプリケーションクラスローダとの間で、対話は行われません。このため、J2EE アプリケーションから J2EE スタンドアロンモジュールクラスを読み込んだり、J2EE モジュールから J2EE アプリケーションを読みこむことはできません。この問題に対応するには、たとえば、関連するパスをシステムクラスパス内の必要なクラスに指定してください。パスを指定したクラスが、システムクラスローダによって読み込まれます。次の例を参照してください。

```
http://developer.iplanet.com/appserver/samples/pkging/docs/sampleC.html
```

  ただし、システムクラスパス内にクラスへの相対パスを含めると、Application Server の制御下に配置されたすべてのアプリケーションに対してクラスが公開されるため、アプリケーションのセキュリティ要件を危険にさらすことになります。また、システムクラスパス内のクラスはシステムクラスローダによって読み込まれるため、再読み込みできなくなります。

- J2EE の仕様ではアプリケーションが別のアプリケーション内のコンポーネントにアクセスできなければなりませんが、iPlanet Application Server の各 J2EE アプリケーションはそれぞれのクラスローダによって読み込まれるため、アプリケーションとして登録された 2 つの EAR ファイルは、相手ファイルのクラスを読み込めません。つまり、2 つのアプリケーションのクラスは個別に読み込まれるため、複数のアプリケーションから名前が似ている 2 つのクラスを読み込んでも、クラスローダ内で互いに上書きされることがありません。

- 1 つの iPlanet Application Server アプリケーションクラスローダだけが、すべての J2EE のスタンドアロンモジュールを読み込むために割り当てられます。これにより、2 つのスタンドアロンモジュール間で対話することができます。このとき、スタンドアロンモジュール内のクラスの名前が重複していてはなりません。たとえば、`ejb1.jar` が `com.samples.company.DBConnector` を読み込もうとし、`war1.war` が `com.samples.company.DBConnector` を読み込もうとすると、一方が他方を上書きします。

ヒント すべてのスタンドアロンモジュールの読み込みに使用されるクラスローダは 1 つだけなので、あるスタンドアロンモジュールのクラスに対して、他のすべてのスタンドアロンモジュールからアクセスできるようにすると、セキュリティ上の潜在的な危険性が高くなります。このため、スタンドアロンモジュールは、すべてのユーザがアクセスできる再利用可能なコンポーネントだけで構成するようにしてください。

注 Servlet、JSP、または EJB がアクセスするファイルなどのリソースは、クラスローダのクラスパスが指定するディレクトリにある必要があります。たとえば、Web クラスローダのクラスパスには次のようなディレクトリがあります。

module_name`/WEB-INF/classes`
module_name`/WEB-INF/compiled_jsp`
module_name`/WEB-INF/lib`

Servlet がリソースにアクセスする場合、これらのディレクトリにリソースがないと読み込まれません。

ダイナミック再読み込み

Servlet、JSP、および EJB 実装クラスは、サーバの動作中にダイナミックに再読み込みされます。このため、モジュール、アプリケーションコード、および記述子を変更しても、サーバを再起動する必要がありません。この機能は、変更したコードをすぐにテストできるため、開発環境で役に立ちます。

ダイナミック再読み込みは、運用環境にはお勧めしません。パフォーマンスが低下することがあります。また、再読み込みを行うと、送信時のセッションが無効になります。クライアントはセッションを再起動する必要があります。

ダイナミック再読み込みが無効になっている場合、モジュールクラスローダ 1 つとアプリケーションクラスローダ 1 つだけが読み込まれます。

ダイナミック再読み込みの有効化

すべてのクラスに対するダイナミック再読み込みは、次の方法でオンまたはオフにできます。

- Administration Tool の使用
- レジストリの変更

Administration Tool の使用

1. Administration Tool の左側のペインで、アプリケーションサーバのインスタンスを選択します。
2. 「ダイナミッククラスの再読み込みを有効にする」チェックボックスをクリックします。
3. 「適用」をクリックします。

   ダイナミックなクラスの再読み込みを有効にするために、Administration Tool によってレジストリ内で適切な変更が行われます。

レジストリの変更

```
SYSTEM_JAVA¥Versioning¥Disable
```

デフォルトでは `1` に設定されており、ダイナミック再読み込みが無効であることを示しています。`0` の値は、ダイナミック再読み込みを有効にします。

レジストリは、`kregedit` ツールを使って編集できます。詳細については、『管理者ガイド』を参照してください。

Servlet および JSP のダイナミック再読み込み

ダイナミック再読み込みを有効にすると、Servlet および JSP 用のサーバに組み込まれます。iPlanet Application Server の動作中に行った変更は、その Servlet および JSP に次回のリクエストが着信したときに有効になります。

EJB のダイナミック再読み込み

ダイナミック再読み込みを有効にすると、EJB のサーバに組み込まれます。iPlanet Application Server の動作中に行った変更は、その EJB に次回の作成リクエストが着信したときに有効になります。

ただし、EJB のインタフェースとヘルパークラスの再読み込みは動的に行われないので、それらを変更した場合は、サーバを再起動する必要があります。

EJB がセッション時に変更されると、EJB コンテナによりセッションに関連した EJB インスタンスのステートが直列化され、インスタンスプールを作成し直した後に直列化が解除されます。

ダイナミック再読み込みの制限

J2EE コンポーネント (Servlet または Ejb 実装クラス) がほかのクラスから直接参照されている場合、参照されている J2EE コンポーネントが属するモジュールのダイナミック再読み込みは機能しません。たとえば、ヘルパークラスから Servlet や EJB 実装クラスの新しいインスタンスを作成する場合です。これは、J2EE ではお勧めしません。

ある J2EE コンポーネントがヘルパークラスのパッケージアクセス番号にアクセスしている場合 (番号が保護されていても、J2EE コンポーネントとヘルパークラスが同じパッケージ内にあるとこの状況が発生)、しかもダイナミック再読み込みが有効になっている場合は、IleegalAccesError が発生します。このような状況では、ダイナミック再読み込みを無効にする必要があります。

注　iPlanet Application Server 6.0 SPx を 6.5 に移行した場合は、EJB のスタブを再度生成する必要があります。生成しない場合、EJB 実装クラスのダイナミック再読み込みは機能しません。

付録 C

サンプル配置ファイル

この付録には、アプリケーションおよびコンポーネントの配置に使われる iPlanet Application Server 配置記述子 (DD) のサンプルファイルがあります。

この付録には、次の DD XML のサンプルファイルがあります。

- アプリケーション DD XML ファイル
- Web アプリケーション DD XML ファイル
- EJB-JAR DD XML ファイル
- RMI/IIOP Client DD XML ファイル
- リソース DD XML ファイル

アプリケーション DD XML ファイル

アプリケーション DD によって、すべてのアプリケーションの内容をトップレベルから見ることができます。2 つのタイプのアプリケーション DD があります。一つは J2EE アプリケーション DD で、もう一つは iPlanet Application Server アプリケーション DD です。これらの記述子は、DTD によって指定された XML ファイルです。

J2EE アプリケーション DD については、J2EE 仕様書バージョン 2.1 の第 8.4 節の「J2EE: application XML DTD」に説明があります。iPlanet Application Server アプリケーション DD については、第 11 章「配置のためのパッケージ化」の Web アプリケーション DTD を参照してください。

サンプルアプリケーション DD XML ファイル

この節では、J2EE アプリケーション DD XML ファイルの例を示します。この後に示す J2EE アプリケーション DD のファイル名は application.xml です。

```
<?xml version="1.0" ?>

<!DOCTYPE application PUBLIC '-//Sun Microsystems, Inc.//DTD J2EE
Application 1.2//EN'
'http://java.sun.com/j2ee/dtds/application_1_2.dtd'>

<Application>
   <description>Application description</description>
   <display-name>estore</display-name>
   <module>
       <ejb>estoreEjb.jar</ejb>
   </module>
   <module>
       <web>
       <web-uri>estore.war</web-uri>
       <context-root>estore</context-root>
   </web>
   </module>
   <security-role>
       <description>the customer role</description>
       <role-name>customer</role-name>
   </security-role>
</application>
```

Web アプリケーション DD XML ファイル

Web アプリケーション DD は、開発者、編成者、および配置者間で、Web アプリケーションの要素および設定情報を伝えます。これらの記述子は、DTD によって指定された XML ファイルです。

Web アプリケーション ARchive (.war) ファイルには、J2EE Web アプリケーション DD と iPlanet Application Server Web アプリケーション DD があります。J2EE Web アプリケーション DD については、Java Servlet 仕様書バージョン 2.2 の 第 13 章「Deployment Descriptors」に説明があります。iPlanet Application Server アプリケーション DD については、第 11 章「配置のためのパッケージ化」の Web アプリケーション DTD を参照してください。

サンプル Web アプリケーション DD XML ファイル

この節では、J2EE Web アプリケーション DD XML ファイルの例を示します。この後に示す Web アプリケーション DD のファイル名は web.xml です。

```
<?xml version="1.0" ?>
<!DOCTYPE web-app>

<web-app>
   <description>no description</description>
   <display-name>DukesPetStoreWebTier</display-name>
   <Servlet>
       <description>no description</description>
       <display-name>centralJsp</display-name>
       <servlet-name>webTierEntryPoint</servlet-name>
       <jsp-file>Main.jsp</jsp-file>
       <load-on-startup>-1</load-on-startup>
   </servlet>
   <servlet-mapping>
       <servlet-name>webTierEntryPoint</servlet-name>
       <url-pattern>/control/*</url-pattern>
   </servlet-mapping>
   <session-config>
       <session-timeout>54</session-timeout>
   </session-config>
   <welcome-file-list>
       <welcome-file>/index.html</welcome-file>
   </welcome-file-list>
   <error-page>
       <exception-type>java.lang.Exception</exception-type>
       <location>/errorpage.jsp</location>
   </error-page>
   <security-constraint>
       <web-resource-collection>
          <web-resource-name>MySecureBit0</web-resource-name>
          <description>no description</description>
          <url-pattern>/control/placeorder</url-pattern>
          <http-method>POST</http-method>
          <http-method>GET</http-method>
       </web-resource-collection>
       <auth-constraint>
          <description>no description</description>
          <role-name>customer</role-name>
       </auth-constraint>
       <user-data-constraint>
```

```
            <description>no description</description>
            <transport-guarantee>NONE</transport-guarantee>
        </user-data-constraint>
    </security-constraint>
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>MySecureBit1</web-resource-name>


            <description>no description</description>
            <url-pattern>/Main.jsp/signin</url-pattern>
            <http-method>POST</http-method>
            <http-method>GET</http-method>
        </web-resource-collection>
        <auth-constraint>
            <description>no description</description>
            <role-name>customer</role-name>
        </auth-constraint>
        <user-data-constraint>
            <description>no description</description>
            <transport-guarantee>NONE</transport-guarantee>
        </user-data-constraint>
    </security-constraint>
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>MySecureBit1</web-resource-name>
            <description>no description</description>
            <url-pattern>/control/signin</url-pattern>
            <http-method>POST</http-method>
            <http-method>GET</http-method>
        </web-resource-collection>
        <auth-constraint>
            <description>no description</description>
            <role-name>customer</role-name>
        </auth-constraint>
        <user-data-constraint>
            <description>no description</description>
            <transport-guarantee>NONE</transport-guarantee>
        </user-data-constraint>
    </security-constraint>
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>MySecureBit0</web-resource-name>
            <description>no description</description>
            <url-pattern>/Main.jsp/placeorder</url-pattern>
            <http-method>POST</http-method>
            <http-method>GET</http-method>
        </web-resource-collection>
```

```
    <auth-constraint>
       <description>no description</description>
       <role-name>customer</role-name>
    </auth-constraint>
    <user-data-constraint>
       <description>no description</description>
       <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>


</security-constraint>

<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>default</realm-name>
    <form-login-config>
       <form-login-page>/estore/login.jsp</form-login-page>
       <form-error-page>/estore/error.html</form-error-page>
    </form-login-config>
</login-config>

<security-role>
    <description>the customer role</description>
    <role-name>customer</role-name>
</security-role>

<ejb-ref>
    <description>no description</description>
    <ejb-ref-name>account</ejb-ref-name>
    <ejb-ref-type>Entity</ejb-ref-type>
    <home>com.sun.estore.account.ejb.AccountHome</home>
    <remote>com.sun.estore.account.ejb.Account</remote>
</ejb-ref>
<ejb-ref>
    <description>no description</description>
    <ejb-ref-name>order</ejb-ref-name>
    <ejb-ref-type>Entity</ejb-ref-type>
    <home>com.sun.estore.order.ejb.OrderHome</home>
    <remote>com.sun.estore.order.ejb.Order</remote>
</ejb-ref>
<ejb-ref>
    <description>no description</description>
    <ejb-ref-name>mailer</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.sun.estore.mail.ejb.MailerHome</home>
    <remote>com.sun.estore.mail.ejb.Mailer</remote>
</ejb-ref>
<ejb-ref>
```

```
        <description>no description</description>
        <ejb-ref-name>estorekeeper</ejb-ref-name>
        <ejb-ref-type>Session</ejb-ref-type>
        <home>com.sun.estore.control.ejb.EStorekeeperHome</home>
        <remote>com.sun.estore.control.ejb.EStorekeeper</remote>
    </ejb-ref>
    <ejb-ref>
        <description>no description</description>
        <ejb-ref-name>catalog</ejb-ref-name>

        <ejb-ref-type>Session</ejb-ref-type>
        <home>com.sun.estore.catalog.ejb.CatalogHome</home>
        <remote>com.sun.estore.catalog.ejb.Catalog</remote>
    </ejb-ref>
    <ejb-ref>
        <description>no description</description>
        <ejb-ref-name>cart</ejb-ref-name>
        <ejb-ref-type>Session</ejb-ref-type>
        <home>com.sun.estore.cart.ejb.ShoppingCartHome</home>
        <remote>com.sun.estore.cart.ejb.ShoppingCart</remote>
    </ejb-ref>
    <ejb-ref>
        <description>no description</description>
        <ejb-ref-name>inventory</ejb-ref-name>
        <ejb-ref-type>Session</ejb-ref-type>
        <home>com.sun.estore.inventory.ejb.InventoryHome</home>
        <remote>com.sun.estore.inventory.ejb.Inventory</remote>
    </ejb-ref>
</web-app>
```

サンプル iPlanet Application Server Web アプリケーション DD XML ファイル

この節では、iPlanet Application Server Web アプリケーション DD XML ファイルの例を示します。この後に示す iPlanet Application Server Web アプリケーション DD のファイル名は `ias-web.xml` です。

```
<?xml version="1.0" ?>
<!DOCTYPE web-app>

<ias-web-app>
    <Servlet>
        <servlet-name>webTierEntryPoint</servlet-name>
        <guid>{Deadbeef-AB3F-11D2-98C5-000000000000}</guid>
```

```
    </servlet>

    <ejb-ref>
        <ejb-ref-name>account</ejb-ref-name>
        <jndi-name>ejb/estoreWar/account</jndi-name>
    </ejb-ref>
    <ejb-ref>
        <ejb-ref-name>order</ejb-ref-name>
        <jndi-name>ejb/estoreWar/order</jndi-name>
    </ejb-ref>
    <ejb-ref>
        <ejb-ref-name>mailer</ejb-ref-name>
        <jndi-name>ejb/estoreWar/mailer</jndi-name>
    </ejb-ref>
    <ejb-ref>
        <ejb-ref-name>estorekeeper</ejb-ref-name>
        <jndi-name>ejb/estoreWar/estorekeeper</jndi-name>
    </ejb-ref>
    <ejb-ref>
        <ejb-ref-name>catalog</ejb-ref-name>
        <jndi-name>ejb/estoreWar/catalog</jndi-name>
    </ejb-ref>
    <ejb-ref>
        <ejb-ref-name>cart</ejb-ref-name>
        <jndi-name>ejb/estoreWar/cart</jndi-name>
    </ejb-ref>
    <ejb-ref>
        <ejb-ref-name>inventory</ejb-ref-name>
        <jndi-name>ejb/estoreWar/inventory</jndi-name>
    </ejb-ref>
</ias-web-app>
```

EJB-JAR DD XML ファイル

EJB-JAR ファイルには、Enterprise JavaBeans 仕様書バージョン 1.1 に定義されている形式の DD と、第 11 章「配置のためのパッケージ化」で定義されている形式の iPlanet Application Server EJB DD が含まれています。

サンプル J2EE EJB-JAR DD XML ファイル

この節では、J2EE EJB DD XML ファイルの例を示します。この後に示す EJB-JAR DD のファイル名は `ejb-jar.xml` です。

```
<?xml version="1.0" ?>

<ejb-jar>
   <description>no description</description>
   <display-name>Ejb1</display-name>
   <enterprise-beans>
       <session>
          <description>no description</description>
          <display-name>TheMailer</display-name>
          <ejb-name>TheMailer</ejb-name>
          <home>com.sun.estore.mail.ejb.MailerHome</home>
          <remote>com.sun.estore.mail.ejb.Mailer</remote>
          <ejb-class>com.sun.estore.mail.ejb.MailerEJB</ejb-class>
          <session-type>Stateless</session-type>
          <transaction-type>Container</transaction-type>
          <ejb-ref>
             <ejb-ref-name>account</ejb-ref-name>
             <ejb-ref-type>Entity</ejb-ref-type>
             <home>com.sun.estore.account.ejb.AccountHome</home>
             <remote>com.sun.estore.account.ejb.Account</remote>
             <ejb-link>TheAccount</ejb-link>
          </ejb-ref>
          <ejb-ref>
             <ejb-ref-name>order</ejb-ref-name>
             <ejb-ref-type>Entity</ejb-ref-type>
             <home>com.sun.estore.order.ejb.OrderHome</home>
             <remote>com.sun.estore.order.ejb.Order</remote>
             <ejb-link>TheOrder</ejb-link>
          </ejb-ref>
          <resource-ref>
             <description>description</description>
             <res-ref-name>MailSession</res-ref-name>
             <res-type>javax.mail.Session</res-type>
             <res-auth>Application</res-auth>
          </resource-ref>
       </session>
       <session>
          <description>no description</description>
          <display-name>TheEstorekeeper</display-name>
          <ejb-name>TheEstorekeeper</ejb-name>
          <home>com.sun.estore.control.ejb.EStorekeeperHome</home>
          <remote>com.sun.estore.control.ejb.EStorekeeper</remote>
          <ejb-class>com.sun.estore.control.ejb.EStorekeeperEJB
                    </ejb-class>
          <session-type>Stateful</session-type>
          <transaction-type>Container</transaction-type>
          <env-entry>
```

```
    <env-entry-name>sendConfirmationMail</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>false</env-entry-value>
</env-entry>
<ejb-ref>
    <ejb-ref-name>account</ejb-ref-name>
    <ejb-ref-type>Entity</ejb-ref-type>
    <home>com.sun.estore.account.ejb.AccountHome</home>
    <remote>com.sun.estore.account.ejb.Account</remote>
    <ejb-link>TheAccount</ejb-link>
</ejb-ref>


<ejb-ref>
    <ejb-ref-name>order</ejb-ref-name>
    <ejb-ref-type>Entity</ejb-ref-type>
    <home>com.sun.estore.order.ejb.OrderHome</home>
    <remote>com.sun.estore.order.ejb.Order</remote>
    <ejb-link>TheOrder</ejb-link>
</ejb-ref>
<ejb-ref>
    <ejb-ref-name>mailer</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.sun.estore.mail.ejb.MailerHome</home>
    <remote>com.sun.estore.mail.ejb.Mailer</remote>
    <ejb-link>TheMailer</ejb-link>
</ejb-ref>
<ejb-ref>
    <ejb-ref-name>catalog</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.sun.estore.catalog.ejb.CatalogHome</home>
    <remote>com.sun.estore.catalog.ejb.Catalog</remote>
    <ejb-link>TheCatalog</ejb-link>
</ejb-ref>
<ejb-ref>
    <ejb-ref-name>cart</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.sun.estore.cart.ejb.ShoppingCartHome</home>
    <remote>com.sun.estore.cart.ejb.ShoppingCart</remote>
    <ejb-link>TheCart</ejb-link>
</ejb-ref>
<ejb-ref>
    <ejb-ref-name>inventory</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.sun.estore.inventory.ejb.InventoryHome
              </home>
    <remote>com.sun.estore.inventory.ejb.Inventory
              </remote>
```

```
            <ejb-link>TheInventory</ejb-link>
        </ejb-ref>
    </session>

    <entity>
        <description>no description</description>
        <display-name>TheOrder</display-name>
        <ejb-name>TheOrder</ejb-name>
        <home>com.sun.estore.order.ejb.OrderHome</home>
        <remote>com.sun.estore.order.ejb.Order</remote>
        <ejb-class>com.sun.estore.order.ejb.OrderEJB</ejb-class>
        <persistence-type>Bean</persistence-type>


        <prim-key-class>java.lang.Integer</prim-key-class>
        <reentrant>False</reentrant>
        <resource-ref>
            <description>description</description>
            <res-ref-name>EstoreDataSource</res-ref-name>
            <res-type>javax.sql.DataSource</res-type>
            <res-auth>Application</res-auth>
        </resource-ref>
    </entity>
    <entity>
        <description>no description</description>
        <display-name>TheAccount</display-name>
        <ejb-name>TheAccount</ejb-name>
        <home>com.sun.estore.account.ejb.AccountHome</home>
        <remote>com.sun.estore.account.ejb.Account</remote>
        <ejb-class>com.sun.estore.account.ejb.AccountEJB
                    </ejb-class>
        <persistence-type>Bean</persistence-type>
        <prim-key-class>java.lang.String</prim-key-class>
        <reentrant>False</reentrant>
        <resource-ref>
            <description>description</description>
            <res-ref-name>EstoreDataSource</res-ref-name>
            <res-type>javax.sql.DataSource</res-type>
            <res-auth>Application</res-auth>
        </resource-ref>
    </entity>

    <session>
        <description>no description</description>
        <display-name>TheCart</display-name>
        <ejb-name>TheCart</ejb-name>
        <home>com.sun.estore.cart.ejb.ShoppingCartHome</home>
        <remote>com.sun.estore.cart.ejb.ShoppingCart</remote>
```

```
            <ejb-class>com.sun.estore.cart.e
            <transaction-type>Container</transaction-type>
        </session>
        <session>
            <description>no description</description>
            <display-name>TheInventory</display-name>
            <ejb-name>TheInventory</ejb-name>
            <home>com.sun.estore.inventory.ejb.InventoryHome</home>
            <remote>com.sun.estore.inventory.ejb.Inventory</remote>
            <ejb-class>com.sun.estore.inventory.ejb.InventoryEJB
                      </ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>


            <resource-ref>
                <description>description</description>
                <res-ref-name>InventoryDataSource</res-ref-name>
                <res-type>javax.sql.DataSource</res-type>
                <res-auth>Application</res-auth>
            </resource-ref>
        </session>
        <session>
            <description>no description</description>
            <display-name>TheCatalog</display-name>
            <ejb-name>TheCatalog</ejb-name>
            <home>com.sun.estore.catalog.ejb.CatalogHome</home>
            <remote>com.sun.estore.catalog.ejb.Catalog</remote>
            <ejb-class>com.sun.estore.catalog.ejb.CatalogEJB
                      </ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
            <resource-ref>
                <description>description</description>
                <res-ref-name>InventoryDataSource</res-ref-name>
                <res-type>javax.sql.DataSource</res-type>
                <res-auth>Application</res-auth>
            </resource-ref>
        </session>
    </enterprise-beans>
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>TheMailer</ejb-name>
                <method-intf>Remote</method-intf>
                <method-name>sendOrderConfirmationMail</method-name>
                <method-param>int</method-param>
            </method>
```

```
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheMailer</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>getPrimaryKey</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheMailer</ejb-name>
         <method-intf>Remote</method-intf>


         <method-name>getEJBHome</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheMailer</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>getHandle</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheMailer</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>isIdentical</method-name>
         <method-param>javax.ejb.EJBObject</method-param>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheEstorekeeper</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>getPrimaryKey</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheEstorekeeper</ejb-name>
```

```
      <method-intf>Remote</method-intf>
      <method-name>handleEvent</method-name>
      <method-param>com.sun.estore.control.event.EStoreEvent
                </method-param>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>TheEstorekeeper</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>getShoppingCart</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>


<container-transaction>
   <method>
      <ejb-name>TheEstorekeeper</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>getAccount</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>TheEstorekeeper</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>getOrder</method-name>
      <method-param>int</method-param>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>TheEstorekeeper</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>getEJBHome</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>TheEstorekeeper</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>getHandle</method-name>
   </method>
```

```
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheEstorekeeper</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>getOrders</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheEstorekeeper</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>isIdentical</method-name>
         <method-param>javax.ejb.EJBObject</method-param>


      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheEstorekeeper</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>getCatalog</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheOrder</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>getPrimaryKey</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheOrder</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>getOrderDetails</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheOrder</ejb-name>
```

```
      <method-intf>Remote</method-intf>
      <method-name>getEJBHome</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>TheOrder</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>getHandle</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>TheOrder</ejb-name>
      <method-intf>Remote</method-intf>


      <method-name>remove</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>TheOrder</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>isIdentical</method-name>
      <method-param>javax.ejb.EJBObject</method-param>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>TheAccount</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>getPrimaryKey</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>TheAccount</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>changeContactInformation</method-name>
      <method-param>com.sun.estore.util.ContactInformation
                </method-param>
   </method>
```

```
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheAccount</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>getEJBHome</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheAccount</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>getHandle</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>


   <container-transaction>
      <method>
         <ejb-name>TheAccount</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>remove</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheAccount</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>getAccountDetails</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheAccount</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>isIdentical</method-name>
         <method-param>javax.ejb.EJBObject</method-param>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheCart</ejb-name>
```

```
      <method-intf>Remote</method-intf>
      <method-name>updateItemQty</method-name>
      <method-param>java.lang.String</method-param>
      <method-param>int</method-param>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>TheCart</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>deleteItem</method-name>
      <method-param>java.lang.String</method-param>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>TheCart</ejb-name>

      <method-intf>Remote</method-intf>
      <method-name>getPrimaryKey</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>TheCart</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>empty</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>TheCart</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>getEJBHome</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>TheCart</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>getHandle</method-name>
   </method>
```

```
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheCart</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>addItem</method-name>
         <method-param>java.lang.String</method-param>
         <method-param>int</method-param>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheCart</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>getItems</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>


   <container-transaction>
      <method>
         <ejb-name>TheCart</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>addItem</method-name>
         <method-param>java.lang.String</method-param>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheCart</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>isIdentical</method-name>
         <method-param>javax.ejb.EJBObject</method-param>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheInventory</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>getPrimaryKey</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
```

```
<container-transaction>
   <method>
      <ejb-name>TheInventory</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>getEJBHome</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>TheInventory</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>getHandle</method-name>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>TheInventory</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>updateInventory</method-name>

      <method-param>com.sun.estore.inventory.ejb.
                InventoryDetails</method-param>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>TheInventory</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>updateQuantity</method-name>
      <method-param>java.lang.String</method-param>
      <method-param>int</method-param>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
   <method>
      <ejb-name>TheInventory</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>isIdentical</method-name>
      <method-param>javax.ejb.EJBObject</method-param>
   </method>
   <trans-attribute>Required</trans-attribute>
</container-transaction>
<container-transaction>
```

```
      <method>
         <ejb-name>TheInventory</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>getInventory</method-name>
         <method-param>java.lang.String</method-param>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheInventory</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>getQuantity</method-name>
         <method-param>java.lang.String</method-param>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheCatalog</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>getPrimaryKey</method-name>


      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheCatalog</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>getEJBHome</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheCatalog</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>getHandle</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   <container-transaction>
      <method>
         <ejb-name>TheCatalog</ejb-name>
         <method-intf>Remote</method-intf>
         <method-name>searchProducts</method-name>
```

```
            <method-param>java.util.Vector</method-param>
         </method>
         <trans-attribute>Required</trans-attribute>
      </container-transaction>
      <container-transaction>
         <method>
            <ejb-name>TheCatalog</ejb-name>
            <method-intf>Remote</method-intf>
            <method-name>findProducts</method-name>
            <method-param>com.sun.estore.catalog.ejb.Category
                  </method-param>
         </method>
         <trans-attribute>Required</trans-attribute>
      </container-transaction>
      <container-transaction>
         <method>
            <ejb-name>TheCatalog</ejb-name>
            <method-intf>Remote</method-intf>
            <method-name>isIdentical</method-name>
            <method-param>javax.ejb.EJBObject</method-param>
         </method>
         <trans-attribute>Required</trans-attribute>
      </container-transaction>


      <container-transaction>
         <method>
            <ejb-name>TheCatalog</ejb-name>
            <method-intf>Remote</method-intf>
            <method-name>getAllCategories</method-name>
         </method>
         <trans-attribute>Required</trans-attribute>
      </container-transaction>
   </assembly-descriptor>
</ejb-jar>
```

サンプル iPlanet Application Server EJB-JAR DD XML ファイル

この節では、iPlanet Application Server EJB-JAR DD XML ファイルの例を示します。この後に示す EJB-JAR DD のファイル名は `ias-ejb-jar.xml` です。

```
<ias-ejb-jar>
   <enterprise-beans>
       <session>
          <ejb-name>TheMailer</ejb-name>
          <guid>{Deadbabe-AB3F-11D2-98C5-0060B0EF0618}</guid>
          <pass-timeout>100</pass-timeout>
          <session-timeout>300</session-timeout>
          <is-thread-safe>false</is-thread-safe>
          <pass-by-value>false</pass-by-value>
          <ejb-ref>
             <ejb-ref-name>account</ejb-ref-name>
             <jndi-name>ejb/estoreEjb/TheAccount</jndi-name>
          </ejb-ref>
          <ejb-ref>
             <ejb-ref-name>order</ejb-ref-name>
             <jndi-name>ejb/estoreEjb/TheOrder</jndi-name>
         /ejb-ref>
       </session>
       <session>
          <ejb-name>TheEstorekeeper</ejb-name>
          <guid>{Deadbabe-AB3F-11D2-98C5-000011112222}</guid>
          <pass-timeout>100</pass-timeout>
          <session-timeout>300</session-timeout>
          <is-thread-safe>false</is-thread-safe>
          <pass-by-value>false</pass-by-value>
          <ejb-ref>
             <ejb-ref-name>account</ejb-ref-name>
             <jndi-name>ejb/estoreEjb/TheAccount</jndi-name>
          </ejb-ref>
          <ejb-ref>
             <ejb-ref-name>order</ejb-ref-name>
             <jndi-name>ejb/estoreEjb/TheOrder</jndi-name>
          </ejb-ref>
          <ejb-ref>
             <ejb-ref-name>mailer</ejb-ref-name>
             <jndi-name>ejb/estoreEjb/TheMailer</jndi-name>
          </ejb-ref>
          <ejb-ref>
             <ejb-ref-name>catalog</ejb-ref-name>
             <jndi-name>ejb/estoreEjb/TheCatalog</jndi-name>
          </ejb-ref>
          <ejb-ref>
             <ejb-ref-name>cart</ejb-ref-name>
             <jndi-name>ejb/estoreEjb/TheCart</jndi-name>
```

```
   </ejb-ref>
   <ejb-ref>
      <ejb-ref-name>inventory</ejb-ref-name>
      <jndi-name>ejb/estoreEjb/TheInventory</jndi-name>
   </ejb-ref>
</session>
<session>
   <ejb-name>TheInventory</ejb-name>
   <guid>{deadbabe-ab3f-11d2-98c5-999999990002}</guid>
   <pass-timeout>100</pass-timeout>
   <is-thread-safe>false</is-thread-safe>
   <pass-by-value>false</pass-by-value>
   <session-timeout>300</session-timeout>
</session>
<session>
   <ejb-name>TheCatalog</ejb-name>
   <guid>{deadbabe-ab3f-11d2-98c5-999999990003}</guid>
   <pass-timeout>100</pass-timeout>
   <is-thread-safe>false</is-thread-safe>
   <pass-by-value>false</pass-by-value>
   <session-timeout>300</session-timeout>
</session>
<session>
   <ejb-name>TheCart</ejb-name>
   <guid>{deadbabe-ab3f-11d2-98c5-999999990001}</guid>
   <pass-timeout>100</pass-timeout>
   <is-thread-safe>false</is-thread-safe>
   <pass-by-value>false</pass-by-value>
   <session-timeout>300</session-timeout>
</session>
<entity>
   <ejb-name>TheAccount</ejb-name>
   <guid>{deadbabe-ab3f-11d2-98c5-999999990000}</guid>
   <pass-timeout>100</pass-timeout>
   <is-thread-safe>false</is-thread-safe>
   <pass-by-value>false</pass-by-value>
   <pool-manager>
      <commit-option>NO_CACHE_READY_INSTANCE</commit-option>
      <Ready-pool-timeout>0</Ready-pool-timeout>
      <Ready-pool-maxsize>0</Ready-pool-maxsize>
   </pool-manager>
</entity>
<entity>
   <ejb-name>TheOrder</ejb-name>
   <guid>{deadbabe-ab3f-11d2-98c5-333344445555}</guid>
   <pass-timeout>100</pass-timeout>
   <is-thread-safe>false</is-thread-safe>
```

```
            <pass-by-value>false</pass-by-value>
            <persistence-manager>
               <persistence-manager-factory-class-name>
                   com.netscape.server.ejb.PersistenceManagerFactory
                   </persistence-manager-factory-class-name>
               <properties-file-location>
               EmployeeRecord_pm1.xml
               </properties-file-location>
               <external-xml-location>
               </external-xml-location>
            </persistence-manager>
            <pool-manager>
               <commit-option>NO_CACHE_READY_INSTANCE</commit-option>
               <Ready-pool-timeout>0</Ready-pool-timeout>
               <Ready-pool-maxsize>0</Ready-pool-maxsize>
            </pool-manager>
         </entity>
   </enterprise-beans>
</ias-ejb-jar>
```

iPlanet Application Server クライアント DD XML ファイル

サンプル iPlanet Application Server DD XML ファイルは次のとおりです。

```
<?xml version="1.0" encoding="UTF-8"?>
<ias-application-client>
   <ejb-ref>
      <ejb-ref-name>External</ejb-ref-name>
      <jndi-name>ejb/com.sun.cts.tests.appclient.deploy.ejb.ejbref.
            Test</jndi-name>
   </ejb-ref>
   <ejb-ref>
      <ejb-ref-name>External1</ejb-ref-name>
      <jndi-name>ejb/com.sun.cts.tests.appclient.deploy.ejb.ejbref.
            Test1</jndi-name>
   </ejb-ref>
</ias-application-client>
```

RMI/IIOP Client DD XML ファイル

サンプル RMI/IIOP クライアント DD XML ファイルは次のとおりです。

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE application-client PUBLIC '-//Sun Microsystems, Inc.//DTD
J2EE Application Client 1.2//EN'
'http://java.sun.com/j2ee/dtds/application-client_1_2.dtd'>

<application-client>
   <display-name>appclient_ejb_depC_ejbref_client</display-name>
   <description>CTS appclient ejbref test</description>
   <ejb-ref>
      <ejb-ref-name>External</ejb-ref-name>
      <ejb-ref-type>Session</ejb-ref-type>
      <home>com.sun.cts.tests.appclient.deploy.ejb.ejbref.
            TestHome</home>
      <remote>com.sun.cts.tests.appclient.deploy.ejb.ejbref.
            Test</remote>
      <ejb-link>Test</ejb-link>
   </ejb-ref>
   <ejb-ref>
      <ejb-ref-name>External1</ejb-ref-name>
      <ejb-ref-type>Session</ejb-ref-type>
      <home>com.sun.cts.tests.appclient.deploy.ejb.ejbref.
            Test1Home</home>
      <remote>com.sun.cts.tests.appclient.deploy.ejb.ejbref.
            Test1</remote>
   </ejb-ref>
</application-client>
```

リソース DD XML ファイル

サンプルリソース DD XML ファイルは次のとおりです。

```
<ias-resource>
   <resource>
      <jndi-name>jdbc/SampleSybaseDS1</jndi-name>
      <jdbc>
          <database>nasqadev</database>
          <datasource>SYBFRED</datasource>
          <username>aparna</username>
          <password>aparnak</password>
```

```
            <driver-type>SYBASE_CTLIB</driver-type>
        </jdbc>
    </resource>
</ias-resource>
```

用語集

この用語集では、iPlanet Application Server の配置および開発環境を説明するために使われる一般的な用語を定義します。標準 J2EE の用語については、次のサイトにある用語集を参照してください。

`http://java.sun.com/j2ee/glossary.html`

ACL　アクセス制御リスト (Access Control List)。ユーザまたはグループと、それらに指定された権限のリスト。「コンポーネント ACL」および「汎用 ACL」を参照

Administration Server　管理タスクを処理する iPlanet Application Server のプロセス

API　Application Programmer Interface。コンピュータプログラムが、API を解釈するために設計されたほかのソフトウェアまたはハードウェアと通信するために使われる命令の集まり

AppLogic　iPlanet Application Server アプリケーション内で明確なモジュール型タスクを実行する機能を果たす iPlanet Application Server 固有のクラス。NAS 2.1 では、AppLogic はフォーム入力の処理、データへのアクセス、HTML テンプレートの入力に使われるデータの作成などのアクションの実行に使われた。iPlanet Application Server では、この機能は Servlet および JSP が果たす

AppPath　アプリケーションファイルが存在するディレクトリの名前が含まれている iPlanet Application Server レジストリエントリ。このエントリは、Web サーバのドキュメントパスと同様に、アプリケーションの論理ディレクトリツリーのトップを定義する。デフォルトでは、`AppPath` には値 *BasePath*`/APPS` が含まれている。*BasePath* は、iPlanet Application Server のベースディレクトリ (`BasePath` は、iPlanet Application Server の変数でもある)

BasePath　iPlanet Application Server のインストールディレクトリを含む iPlanet Application Server レジストリエントリ (iPlanet Application Server サブディレクトリを含む)。その他の iPlanet 製品も `BasePath` にインストールできる。たとえば、UNIX マシンで `/usr/local/iPlanet` にインストールした場合、`BasePath` は `/usr/local/iPlanet/ias` になる。`BasePath` は、`AppPath` のビルディングブロック

Bean 管理トランザクション　「宣言によるトランザクション」を参照

Bean プロパティファイル　EJB 配置情報を含んでいるテキストファイル。情報のタイプは、`javax.ejb.DeploymentDescriptor` で定義されている

C++ サーバ　C++ オブジェクトを実行して管理する iPlanet Application Server のプロセス

classpath　ほかのクラスまたはパッケージから派生しているかどうかという点から、Java クラスまたはパッケージを識別するパス。クラスファイルおよびクラス名を参照次に例を示す

```
com.myDomain.myPackage.myClass.
```

cookie　呼び出し側である Web ブラウザに対して送信され、その後、そのブラウザから呼び出しが行われるたびにブラウザ側に記録される情報の小さなコレクション。サーバは、cookie によって、同じクライアントからの呼び出しであるかどうかを認識できる。cookie はドメイン特有である。cookie は、アプリケーションとサーバ間の、ほかのデータ交換の場合と同じ Web サーバセキュリティ機能を利用できる

CORBA　Common Object Request Broker Architecture。オブジェクト指向の分散コンピューティングに関する標準アーキテクチャ定義

Directory Server　iPlanet Application Server にバンドルされている LDAP サーバ。iPlanet Application Server の各インスタンスは、Directory Server を使ってユーザおよびグループに関する情報などの共有サーバ情報を保存する

Enterprise JavaBeans (EJB)　多層分散アーキテクチャにおけるアプリケーションのビジネスロジックコンポーネント。EJB は、予期されるロールによって Bean を定義する Java EJB 標準仕様に従う。EJB は、操作するデータ構造体やメソッドなどの、1 つまたは複数のアプリケーションタスクあるいはアプリケーションオブジェクトをカプセル化する。通常、EJB はパラメータを受け取って戻り値を返す。EJB は常にコンテナのコンテキスト内で動作する。コンテナは、EJB とそれらを管理するサーバ間のリンクとして機能する。「コンテナ」、「セッション EJB」、および「エンティティ EJB」を参照

Executive Server　ロードバランスやプロセス管理などの実行関数を処理する iPlanet Application Server のプロセス

GUID　グローバルに固有であることを保証された、iPlanet Application Server アプリケーションのコンポーネントを識別するために使われる 128 ビットの 16 進数値

HTML　Hypertext Markup Language の略。Web ブラウザに表示できるドキュメントを記述するためのマークアップ言語。テキストの各ブロックは、テキストの種類を指定したコードで囲む

HTML ページ　HTML でコード化され、Web ブラウザで表示することを目的としたページ

HTTP　インターネットでハイパーテキストドキュメントを送受信するためのプロトコル

HTTP Servlet　`javax.servlet.HttpServlet` を拡張する Servlet。HTTP Servlet には、HTTP プロトコルのサポートが組み込まれている。「汎用 Servlet」と対照的

IDL　インタフェース定義言語 (Interface Definition Language)。リモートプロシージャ呼び出し (RPC) の関数インタフェースを記述する言語。複数のマシンのパラメータを整列化するプロキシおよびスタブコードを、コンパイラが生成するために使用する

IIOP　Internet Inter-ORB Protocol。CORBA をベースにした、RMI クライアントおよびサーバ用の転送プロトコル

iPlanet Application Server RowSet　iPlanet Application Server の拡張機能を実装する `RowSet` オブジェクト。`iASRowSet` クラスは、`ResultSet` のサブクラス

iPlanet Application Server レジストリ　ツリーで構成されるアプリケーションメタデータのコレクション。アクティブメモリ内または即時アクセス可能な Directory Server 上で常に利用できる

J2EE　Java 2 Enterprise Edition。多層 Web ベースエンタープライズアプリケーションを開発し、配置するための環境。J2EE プラットフォームは、一連のサービス、アプリケーションプログラミングインタフェース (API)、およびこれらのアプリケーションを開発する機能を提供するプロトコルから構成されている

jar ファイル規約　Enterprise JavaBeans (EJB) のパッケージ (`.jar` ファイル) に存在しなければならない情報を指定する規約。「Enterprise JavaBeans (EJB)」を参照

Java サーバ　Java オブジェクトを実行および管理する iPlanet Application Server のプロセス

JavaBeans　個々に再利用可能な Java オブジェクト

JavaServer Pages (JSP)　HTML または XML タグ、JSP タグ、および Java コードを組み合わせて記述したテキストページ。JSP はプログラミング言語の能力と標準ブラウザページのレイアウト機能をあわせ持つ

JDBC　Java Database Connectivity API。開発者がデータ認識コンポーネントを作成するときに使う、標準ベースの一連のクラスおよびインタフェース。JDBC は、プラットフォームやベンダとは無関係にデータソースと接続して対話するためのメソッドを実装する

JNDI　Java Naming and Directory Interface。JNDI は、アプリケーションがネットワーク経由でリモートサービスを見つけてアクセスするための、プラットフォームに依存しない一定の方法を提供する。iPlanet Application Server では、データソースと Enterprise JavaBeans (EJB) コンポーネント用の JNDI 検索がサポートされている

JTA　Java Transaction API。アプリケーションおよび J2EE サーバによるトランザクションへのアクセスを可能にする API

kas　「Administration Server」を参照

kcs　「C++ サーバ」を参照

kjs　「Java サーバ」を参照

kxs　「Executive Server」を参照

LDAP　Lightweight Directory Access Protocol。LDAP は、TCP/IP 上で実行するオープンディレクトリアクセスプロトコルである。グローバルなサイズおよび多数のエントリに拡張できます。Directory Server、すなわち iPlanet Application Server にバンドルされている LDAP サーバを使うと、アプリケーションサーバがネットワーク経由でアクセスできる 1 つの一元化されたディレクトリ情報リポジトリに社内情報をすべて保存できる

ResultSet　`java.sql.ResultSet` インタフェースを実装するオブジェクト。`ResultSets` は、データベースまたはほかのソースの表形式データから取得した一連の行のカプセル化に使われる

RMI　Remote Method Invocation (RMI)。オブジェクトをリモートプロセスに渡せるようにリモートインタフェースを記述するための一連の Java 標準 API

RowSet　データベースまたはほかのソースの表形式データから取得した一連の行をカプセル化するオブジェクト。RowSet は、`java.sql.ResultSet` インタフェースを拡張して、`ResultSet` が JavaBeans コンポーネントとして機能できるようにする

Servlet　`Servlet` クラスのインスタンス。Servlet は、サーバで実行する再利用可能なアプリケーションである。iPlanet Application Server では、Servlet は、プレゼンテーションロジックの実行、ビジネスロジックの起動、およびプレゼンテーションレイアウトの起動または実行によって、アプリケーションでの対話ごとにセントラルディスパッチャとしての役割を果たす

Servlet エンジン　すべての Servlet メタファンクションを処理する内部オブジェクト。インスタンス化および実行などのサービスを Servlet に提供する一連のプロセス

Servlet ランナー　リクエストオブジェクトおよびレスポンスオブジェクトを持つ Servlet を起動する Servlet エンジンの一部。「Servlet エンジン」を参照

SQL　Structured Query Language (SQL)。リレーショナルデータベースアプリケーションで一般に使われる言語。SQL2 および SQL3 は、この言語のバージョンを表す

URI　Universal Resource Identifier は、あるドメインの特定のリソースについて記述する。ローカルではベースディレクトリのサブセットとして記述される。たとえば、`/ham/burger` がベースディレクトリで、URI が `toppings/cheese.html` である場合、対応する URL は、`http://domain:port/toppings/cheese.html` となる

URL　Uniform Resource Locator の略。HTML ページまたはほかのリソースを一意に指定するアドレス。Web ブラウザは URL を使って、表示するページを指定する。URL では、転送プロトコル (HTTP、FTP など)、ドメイン (`www.my-domain.com` など)、URI (オプション) などを記述する

Web アプリケーション　WWW (World Wide Web) を利用して接続やユーザインタフェース (UI) を実現するコンピュータプログラム。ユーザは、プラットフォーム上の Web ブラウザを使って Web アプリケーションに接続したり、実行したりする。アプリケーションのユーザインタフェースは、ブラウザによって表示される HTML ページである。アプリケーション自体は、Web サーバやアプリケーションサーバ上で動作する

Web コネクタプラグイン　iPlanet Application Server との通信を可能にする Web サーバの拡張機能

Web サーバ　HTML ページおよび Web アプリケーションを保存して管理するホスト。Web サーバは、Web ブラウザからのユーザリクエストに応答する

Web ブラウザ　HTML または XML で記述した Web ページなどの、WWW 上のリソースを表示するために使われるソフトウェア

XA プロトコル　分散トランザクション対応のデータベース業界標準プロトコル

XML　XML (Extensible Markup Language) は、HTML スタイルタグを使って、ドキュメントをフォーマットするだけでなく、ドキュメントで使われるさまざまな種類の情報を識別する

アプリケーション　ユーザのためにタスクまたはサービスを実行するコンピュータプログラム。「Web アプリケーション」を参照

アプリケーションイベント　iPlanet Application Server レジストリを使って登録する特定のアクション。イベントは、タイマーが期限切れになったときや、実行時にイベントがアプリケーションコードから呼び出される (トリガされる) ときに発生する。イベントの一般的な使用法として、定期的なバックアップ、作業日の最後に行うアカウントの調整、警告メッセージの送信などがある

アプリケーションサーバ　クライアント / サーバ環境でアプリケーションを実行するプログラム。アプリケーションを構成するロジックを実行し、Web ブラウザとデータソースとの間でミドルウェアとしての役割を果たす

アプリケーション層　アプリケーションの概念的な分割

クライアント層 : ユーザインタフェース (UI)。エンドユーザは、クライアントソフトウェア (Web ブラウザ) と対話してアプリケーションを使う

サーバ層 : アプリケーションのコンポーネント内で定義されている、アプリケーションを構成するビジネスロジックおよびプレゼンテーションロジック

データ層 : アプリケーションがデータソースと対話できるようにするデータアクセスロジック

アプレット　Web ブラウザで実行する、Java で書かれた小さなアプリケーション。通常、アプレットは、特別な機能を提供する Web ページに呼び出されたり、埋め込まれたりする。これに対し、*Servlet* は、サーバで実行される小さなアプリケーション

インスタンス　特定のクラスに基づくオブジェクト。クラスの各インスタンスは、独自の変数値およびステートを持つ別個のオブジェクトである。ただし、1 つのクラスのすべてのインスタンスは、そのクラスで指定された変数およびメソッド定義を共有する

インスタンス化　実行時にオブジェクトにメモリを割り当てるプロセス。「インスタンス」を参照

インタフェース　オブジェクトによって提供されるサービスの記述。インタフェースは、関数セット、呼び出されるメソッドを定義するが、実装コードを含まない。クラスと同様に、インタフェースは、特定のタイプのオブジェクト特性を定義する。ただし、クラスと異なりインタフェースは常に抽象的。クラスはオブジェクトを作成するためにインスタンス化されるが、インタフェースは一連のサービスを提供するオブジェクトによって実装される。「クラス」と対照的

エンティティ EJB　エンティティ Enterprise JavaBeans (EJB) は、データベースの行などの物理的なデータに関連している。エンティティ Beans は、パーシスタントデータに結び付けられるので生存期間が長い。エンティティ Beans は、常にトランザクションおよびマルチユーザを認識する。「セッション EJB」を参照

カラム　データベーステーブル内のフィールド

キャッシュされた行セット　`CachedRowSet` オブジェクトを使うと、データソースからデータを取り込み、そのデータを確認したり変更したりしながらデータをデータソースから切り離すことができる。キャッシュされた行セットには、取得した元のデータ、およびアプリケーションによるデータの変更の両方が記録される。アプリケーションが元のデータソースを更新しようとすると、行セットはデータソースに再び接続され、変更された行だけがデータベースにマージされます。

行　テーブル内の各列の値を格納する 1 つのデータレコード

クライアント　リソースを起動するエンティティ

クライアント規約　クライアントと EJB コンテナ間の通信ルールを決め、EJB を使うアプリケーションのために均一な開発モデルを設定し、クライアントとの関係を統一することによって Bean の効率よい再利用を保証する規約。「Enterprise JavaBeans (EJB)」を参照

クラス　特定のタイプのオブジェクトの特性を定義するメソッドおよびメンバー変数の集まり。クラスは、このタイプのオブジェクトに関して可能なデータタイプおよび動作を定義する。「インタフェース」と対照的

クラスタ　互いに連携して同じサーバソフトウェアを実行するホストの集まり

クラスファイル　通常、`.class` 拡張子を持つ、コンパイルされたクラスが含まれているファイル。クラス名および classpath を参照通常、次のようなファイルシステム内の位置によって参照される

```
.../com/myDomain/myPackage/myClass.
```

クラス名　Java 仮想マシンにおけるクラスの名前。「クラスファイル」および「classpath」を参照

クラスローダ　特定のルールに従って Java クラスを読み込む機能を果たす Java コンポーネント

グループ　ローカルシステム管理者によって管理される、何らかのルールに従って関連付けられているユーザのグループ。「ユーザ」および「ロール」を参照

グローバルデータベースコネクション　複数のコンポーネントに利用可能なデータベースコネクション。データベースコネクションにはリソースマネージャが必要

グローバルトランザクション　トランザクションマネージャによって管理および調整され、1 つのデータベースおよびプロセスに制限されないトランザクション。トランザクションマネージャは通常、XA プロトコルを使ってデータベースのバックエンドと対話する。「ローカルトランザクション」を参照

継承　サブクラスがスーパークラスのメソッドおよび変数定義を自動的に含むメカニズム。プログラマは、スーパークラスに影響を与えずに、サブクラスの継承特性を変更したり追加したりできる

コミットする　必要なコマンドをデータベースに送信することによって、トランザクションを実行すること。「トランザクション」を参照

コンテキスト、サーバ　オブジェクトによって表される、サーバのステートのプログラムビュー

コンテナ　EJB に関するサービスの実行および提供を行うプロセス

コントロール記述子　Enterprise JavaBeans (EJB) トランザクションおよびセキュリティロパティだけでなく、Bean メソッドの個々のプロパティオーバーライド (オプション) を指定できるようにする一連の EJB 設定エントリ

コンパイル済みコマンド　実行の繰り返しを効率よくするために、SQL で書かれた、あらかじめコンパイルされているデータベースコマンド。コンパイル済みコマンドにはパラメータを入れることができる。コンパイル済みステートメントには、1 つまたは複数のコンパイル済みコマンドが含まれている

コンパイル済みステートメント　データをフェッチするために繰り返して使われるクエリ、更新、または挿入ステートメントがカプセル化されているクラス。コンパイル済みステートメントには、1 つまたは複数のコンパイル済みコマンドが含まれている

コンポーネント　Servlet、Enterprise JavaBeans (EJB)、または JavaServer Page (JSP)

コンポーネント ACL　実行者であるユーザまたはグループを定義する Servlet あるいは EJB コンフィグレーションファイルにおけるプロパティ

コンポーネント規約　Enterprise JavaBeans (EJB) とそのコンテナ間の関係を確立する規約。「Enterprise JavaBeans (EJB)」を参照

サーバ　ほかのコンピュータで実行しているクライアントソフトウェアに、特定のサービスを提供するコンピュータまたはソフトウェアパッケージ。サーバは、特定のタイプのクライアントソフトウェアと通信するように設計されている

再利用可能なコンポーネント　複数の容量、たとえば複数のリソースまたはアプリケーションが使えるように作成されたコンポーネント

システム管理者　iPlanet Application Server ソフトウェアのインストールや管理、および iPlanet Application Server 運用アプリケーションの配置の責任者

状態のあるセッション EJB　特定のクライアントとのセッションを表す Enterprise JavaBeans (EJB)。複数のクライアント起動メソッドのステートを自動的に管理する

状態のないセッション EJB　状態のないサービスを表す EnterpriseJavaBeans (EJB)。状態のないセッション Bean は、完全にトランジェントであり、特定のクライアントが限られた時間必要とするビジネスロジックの一時的な部分がカプセル化される

ステート　1. 指定された時間におけるエンティティの環境または状態 2. iPlanet Application Server 機能インタフェース IState2 を使って、アプリケーションのステートが保存できる分散データ保存メカニズム

スティッキー cookie　常に同じ Executive Server プロセスにクライアントを強制的に接続させるためにクライアントに返される cookie

スティッキーロードバランス　初期クライアントリクエストについてはロードバランスを行うが、その後のリクエストは初期リクエストと同じプロセスにゆだねるロードバランス方法。「ロードバランス」を参照

ストアドプロシージャ　SQL で書かれ、データベースに保存されるステートメントのブロック。ストアドプロシージャを使って、レコードの変更、挿入、または削除などのすべてのタイプのデータベースオペレーションを実行できる。ストアドプロシージャを使うと、ネットワークを介して送信される情報量が減るのでデータベースのパフォーマンスが向上する

ストリーミング　HTTP によるデータの通信方法を管理するための技術。結果がストリーミングされると、そのデータの最初の部分をすぐに利用できる。結果がストリーミングされないと、結果全体が取得されるまで利用できない。ストリーミングを使うと、大量のデータを効率よく返すことができるため、アプリケーションの体感的なパフォーマンスが向上する

スレッド　プロセス内の実行のシーケンス。プロセスで複数のスレッドが同時に実行される場合はマルチスレッド。各スレッドが逐次実行される場合はシングルスレッド

セキュリティ　認証されたクライアントだけがアプリケーションのリソースを使えるように制限すること

セッション cookie　ユーザセッション識別子が含まれているクライアントに返される cookie

セッション EJB　セッション Enterprise JavaBeans (EJB) は、データのリクエストなどの動作のユニットに関連する。セッション Beans は生存期間が短く、クライアントリクエストの生存期間とセッション Beans の生存期間は同じ。セッション Beans は、状態がない、またはあって、トランザクションを認識できる。「状態のあるセッション EJB」、「状態のないセッション EJB」、および「エンティティ EJB」を参照

セッションタイムアウト　ユーザセッションの有効期限。この特定の時間を超えると、iPlanet Application Server によってユーザセッションが無効になる。「ユーザセッション」を参照

設定　コンポーネントのメタデータを提供するプロセス。通常、特定のコンポーネントのコンフィグレーションは、コンポーネント実行時にレジストリにアップロードされるファイルに保持される

宣言によるセキュリティ　セキュリティプロパティをコンポーネントのコンフィグレーションファイル内で宣言し、コンポーネントのコンテナ (例 : Bean のコンテナや Servlet エンジン) にセキュリティを暗黙的に管理させること。このタイプのセキュリティには、プログラムの制御は必要ない。プログラムセキュリティとは反対の概念

宣言によるトランザクション　トランザクションのプロパティを Bean プロパティファイル内で宣言し、Bean のコンテナにトランザクションを暗黙的に管理させること。このタイプのトランザクションには、プログラムの制御は必要ない。プログラムトランザクションとは反対の概念

属性　Servlet によって設定可能な、リクエストオブジェクト内の Name-value ペア。パラメータ」と対照的。一般的には、属性はメタデータの単位

ダイナミック再読み込み　サーバを再起動せずにコンポーネントを更新して再読み込みすること。デフォルトでは、Servlet、JavaServer Page (JSP)、および Enterprise JavaBeans (EJB) コンポーネントをダイナミックに再読み込みできる

直列化可能　オブジェクトを解体および再構築できる場合、そのオブジェクトは直列化可能である。直列化可能なオブジェクトは、複数のサーバに保存したり分散したりできる

データアクセスロジック　データソースとの対話を伴うビジネスロジック

データソース　データベースなどの、データのソースへのハンドル。データソースは、iPlanet Application Server で登録された後に、コネクションを確立してデータソースと対話するために、プログラムによって取得される。データソース定義はデータのソースへの接続方法を指定する

データベース　リレーショナルデータベース管理システム (RDBMS) の一般名。関連する組織化された大量のデータの作成および操作が可能なソフトウェアパッケージ

データベースコネクション　データベースコネクションとは、データベースまたはほかのデータソースとの通信リンクである。コンポーネントは、複数のデータベースコネクションを同時に作成および操作して、データにアクセスできる

テーブル　データベースの行および列内に保存されている関連データの特定のグループ

電子商取引　インターネットを介して行われるビジネスを意味する業界用語

同一場所に置く　関連するコンポーネントと同じメモリ空間にコンポーネントを配置することによってリモートプロシージャコールを避け、パフォーマンスを向上させる

登録　iPlanet Application Server が、Servlet、Enterprise JavaBeans (EJB)、およびほかのアプリケーションリソースへのアクセス権を取得するプロセス。アイテムごとに iPlanet Application Server レジストリにエントリを設定することに由来する

トランザクション　グループとして成功または失敗する一連のデータベースコマンド。トランザクション全体が成功するには、そのトランザクションに関連するすべてのコマンドが成功する必要がある

トランザクション　使われていないときに解放されるリソース。パーシスタントとは反対の概念

トランザクションコンテキスト　ローカルまたはグローバルなトランザクションの範囲。「ローカルトランザクション」および「グローバルトランザクション」を参照

トランザクションマネージャ　通常 XA プロトコルを使ってグローバルトランザクションを制御するオブジェクト。「グローバルトランザクション」を参照

認証　ユーザが入力したユーザ名およびパスワードを確認するプロセス

パーシスタント　アプリケーションが存在している間の Bean の作成および管理を意味する。iPlanet Application Server では、Bean が自身のパーシスタンスを管理する。これを Bean 管理パーシスタンスと呼ぶ。トランザクションとは反対の概念

バージョン付け　「ダイナミック再読み込み」を参照

パーミッション　ユーザまたはグループに与える、または拒否する一連の権限。「ACL」も参照

配置　1 つまたは複数の iPlanet Application Server あるいは 1 つまたは複数の Web サーバがアプリケーションを実行できるように、1 つまたは複数のサーバ上にプロジェクトのすべてのファイルのコピーを作成すること

配置記述子　Enterprise JavaBeans (EJB) をどこに、どのように配置するかを決める属性。「Enterprise JavaBeans (EJB)」を参照

パッケージ　共通ディレクトリ内に保存されている、関連するクラスのコレクション。それらは、頻繁に、Java アーカイブ (`.jar`) ファイルにリテラルにパッケージされる

パラメータ　パラメータは、フォームフィールドデータや HTTP ヘッダー情報など、クライアントから送信される名前 - 値ペアであり、リクエストオブジェクト内にカプセル化されている。「属性」と対照的。一般的には、Java メソッドまたはデータベースコンパイル済みコマンドに渡される引数を指す

汎用 ACL　ユーザまたはグループを 1 つまたは複数の権限に関連付ける、Directory Server 内の特定のリスト。一連の権限を記録するようにこのリストを定義し、自由にアクセスできる

汎用 Servlet　`javax.servlet.GenericServlet` を拡張する Servlet。汎用 Servlet はプロトコルに依存しない。これは、汎用 Servlet は本来、HTTP やその他の転送プロトコルをサポートしていないことを意味する。「HTTP Servlet」と対照的

汎用アプリケーション　設定目的のためにアプリケーション構造に厳密ではなく構成された、グローバルに利用可能なコンポーネントのコレクション

ビジネスメソッド　ビジネスロジックのコースで、データベースのクエリの実行やユーザ認証などの一つのビジネスタスクを実行するメソッド

ビジネスロジック　アプリケーションの要件によって決められる実装ルール

プール　設定済みのリソースを増やしてパフォーマンスを向上させる。リソースがプールされていると、コンポーネントは新しくインスタンス化しなくても、プールから既存のインスタンスを使用できる。iPlanet Application Server では、データベースコネクション、Servlet インスタンス、および Enterprise JavaBeans (EJB) インスタンスをすべてプールできる

ファインダーメソッド　クライアントがグローバルに利用可能なディレクトリで、Bean または Bean のコレクションを調べることができるようにするメソッド。「Enterprise JavaBeans (EJB)」を参照

フェールオーバーリカバリ　Bean がサーバクラッシュに透過的に耐えられるようにするプロセス

フォームアクションハンドラ　フォーム上の特定のボタンに基づいてアクションを実行する、Servlet または AppLogic 内で特別に定義されているメソッド

不活性化　Bean を破棄せずに EJB のリソースを解放するメソッド。これによって、Bean はパーシスタントになり、インスタンス化せずに再び呼び出すことができる。「Enterprise JavaBeans (EJB)」を参照

プライマリキークラス名　Bean のプライマリキーの完全修飾クラス名を指定する変数。JNDI 検索に使われる

プリゼンテーションロジック　アプリケーションでページを作成するアクティビティ。リクエストの処理、レスポンスコンテンツの生成、クライアントに返すページのフォーマット化など

プリンシパル　認証の結果として、エンティティに割り当てられる ID

プレゼンテーションレイアウト　ページコンテンツの作成およびフォーマット化

プログラムセキュリティ　コンポーネントのコンテナ (Bean のコンテナや Servlet エンジンなど) による処理ではなく、コードを記述して明示的にセキュリティを制御すること。宣言によるセキュリティとは反対の概念

プログラムトランザクション　Enterprise JavaBeans (EJB) のコンテナによる処理でなく、コードを記述して明示的にトランザクションを制御すること。宣言によるトランザクションとは反対の概念

プロセス　アクティブプログラムの実行シーケンス。プロセスは、1 つまたは複数のスレッドから構成される

プロパティ　アプリケーションコンポーネントの動作を定義する 1 つの属性

分散可能セッション　クラスタ内のすべてのサーバ間に分散できるユーザセッション

分散トランザクション　別個のサーバ上に配置されている複数の異種データベースに適用可能な 1 つのトランザクション

分離レベル　(JDBC) データソースコネクション時に、`ResultSets` などの呼び出し側オブジェクトからトランザクションの変更が見えるようにレベルを設定する

ホームインタフェース　クライアントによる Enterprise JavaBeans (EJB) の作成や削除を可能にするメソッドを定義するメカニズム。「Enterprise JavaBeans (EJB)」を参照

メタデータ　コンポーネントの名前やその動作の仕様などの、コンポーネントに関する情報

メモリキャッシュ　パフォーマンスを向上させるために、Servlet が特定の時間その結果をキャッシュできるようにする iPlanet Application Server の機能。その時間内の Servlet への後続の呼び出しには、キャッシュに保存された結果が与えられるので、Servlet を実行し直す必要がない

ユーザ　アプリケーションを使う人。プログラム的に言うと、アプリケーションがクライアントを認識する際の手掛かりとなるユーザ名、パスワード、および一連の属性。「グループ」および「ロール」を参照

ユーザインタフェース (UI)　Web アプリケーションにおいて、ユーザが表示する画面を定義し、ユーザが対話に使うページ

ユーザセッション　サーバによって記録される、ユーザとアプリケーション間の一連の対話。セッションでは、ユーザステート、パーシスタントオブジェクト、および ID 認証が管理される

呼び出し可能なステートメント　ストアドプロシージャからのリザルトセットの戻しをサポートしているデータベースのデータベースプロシージャまたは関数呼び出しがカプセル化されているクラス

リクエストオブジェクト　クライアントによって生成されたページおよびセッションデータが含まれているオブジェクトであり、入力パラメータとして Servlet または JavaServer Page (JSP) に渡される

リソースマネージャ　グローバルに利用可能なデータソースを制御するオブジェクト

リモートインタフェース　クライアントによる Enterprise JavaBeans (EJB) のメソッドの呼び出し方法について記述する。「Enterprise JavaBeans (EJB)」を参照

リモートプロシージャコール (RPC)　リモートオブジェクトまたはサービスにアクセスするメカニズム

レスポンスオブジェクト　呼び出しているクライアントを参照して、そのクライアントへの出力を生成するメソッドを提供するオブジェクト

ローカルセッション　1 つのサーバだけに見えるユーザセッション

ローカルデータベースコネクション　ローカルコネクションのトランザクションコンテキストは、複数のプロセスまたはデータソース全体に分散できない。現在のプロセスおよび現在のデータソースに対してローカルである

ローカルトランザクション　1 つのデータベースに固有で、1 つのプロセス内に制限されるトランザクション。ローカルトランザクションは、1 つのバックエンドでのみ動作する。ローカルトランザクションは通常、JDBC API を使って区別されます。「グローバルトランザクション」を参照

ロードバランス　クラスタ内の複数のサーバ間でユーザ負荷を均等に分散する技術。「スティッキーロードバランス」を参照

ロール　アプリケーションにおいてサブジェクトを機能別にグループ分けしたもの。配置環境では 1 つまたは複数のグループによって表される。「ユーザ」および「グループ」を参照

ロールバック　トランザクションをキャンセルする。「トランザクション」を参照

索引

G

H

I

J

L

N

O

P

R

S

T

U

れ

ろ