Oracle® Containers for J2EE

サービス・ガイド

10*g*（10.1.3.1.0）

部品番号 : B31858-01

2006 年 12 月

ORACLE®

Oracle Containers for J2EE サービス・ガイド, 10*g*（10.1.3.1.0）

部品番号 : B31858-01

原本名 : Oracle Containers for J2EE Services Guide, 10*g* (10.1.3.1.0)

原本部品番号 : B28958-01

原本著者 : Joseph Ruzzi

原本協力者 : Alfred Franci, Bonnie Vaughan, Brian Wright, Dan Hynes, Frances Zhao, Jerry Bortvedt, Kirk Bittler, Tom Collier, Qiang Liu, Vivekananda Maganty, Bob Nettleton, Shengsong Ni, Debabrata Panda, Paul Parkinson, J. J. Snyder, John Speidel, Jerry Steidl, John Lang, Rajkumar Irudayaraj, Lixin Zheng, Peter Wu, Valarie Moore, Anthony Lai, Ashok Banerjee, Cheuk Chau, Editor Ellen Siegal, Erik Bergenholtz, Gary Gilchrist, Irene Zhang, Jon Currey, Jyotsna Laxminarayanan, Krishna Kunchithapadam, Kuassi Mensah, Lars Ewe, Lelia Yin, Mike Lehmann, Mike Sanko, Min-Hank Ho, Nickolas Kavantzas, Olga Peschansky, Rachel Chan, Raymond Ng, Sastry Malladi, Stella Li, Sunil Kunisetty, Thomas Van Raalte.

目次

3 OC4J トランザクション・サポート

4 Oracle Enterprise Messaging Service の使用

5 データソース

6 Remote Method Invocation の使用方法

7 Java Object Cache

8 Oracle XML Query Service

9 アプリケーション・クライアント・コンテナの使用方法

A サード・パーティ・ライセンス

索引

はじめに

Oracle Application Server には、Oracle Containers for J2EE（OC4J）と呼ばれる J2EE 環境が組み込まれています。このマニュアルでは、OC4J によって提供されるサービスについて説明します。

対象読者

このマニュアルは、J2EE アーキテクチャに関する知識があり、Oracle J2EE サービスを理解する必要がある開発者を対象としています。

ドキュメントのアクセシビリティについて

オラクル社は、障害のあるお客様にもオラクル社の製品、サービスおよびサポート・ドキュメントを簡単にご利用いただけることを目標としています。オラクル社のドキュメントには、ユーザーが障害支援技術を使用して情報を利用できる機能が組み込まれています。HTML 形式のドキュメントで用意されており、障害のあるお客様が簡単にアクセスできるようにマークアップされています。標準規格は改善されつつあります。オラクル社はドキュメントをすべてのお客様がご利用できるように、市場をリードする他の技術ベンダーと積極的に連携して技術的な問題に対応しています。オラクル社のアクセシビリティについての詳細情報は、Oracle Accessibility Program の Web サイト `http://www.oracle.com/accessibility/` を参照してください。

ドキュメント内のサンプル・コードのアクセシビリティについて

スクリーン・リーダーは、ドキュメント内のサンプル・コードを正確に読めない場合があります。コード表記規則では閉じ括弧だけを行に記述する必要があります。しかし JAWS は括弧だけの行を読まない場合があります。

外部 Web サイトのドキュメントのアクセシビリティについて

このドキュメントにはオラクル社およびその関連会社が所有または管理しない Web サイトへのリンクが含まれている場合があります。オラクル社およびその関連会社は、それらの Web サイトのアクセシビリティに関しての評価や言及は行っておりません。

Oracle サポート・サービスへの TTY アクセス

アメリカ国内では、Oracle サポート・サービスへ 24 時間年中無休でテキスト電話（TTY）アクセスが提供されています。TTY サポートについては、(800)446-2398 にお電話ください。

関連ドキュメント

詳細は、次の Oracle リソースを参照してください。

他の OC4J ドキュメント:

- 『Oracle Containers for J2EE 開発者ガイド』

  このマニュアルでは、OC4J で稼働するアプリケーションを記述する開発者にとって一般的な内容、つまり、サーブレット、EJB または JSP コンテナなどの特定のコンテナに限定されない問題について説明しています。（一例として、クラス・ロードがあげられます。）

- 『Oracle Containers for J2EE デプロイメント・ガイド』

  このマニュアルには、アプリケーションを OC4J 環境にデプロイするための情報および手順が含まれます。また、Oracle Enterprise Manager 10*g* に付属するデプロイ・プラン・エディタの説明も含まれます。

- 『Oracle Containers for J2EE 構成および管理ガイド』

  このマニュアルでは、Oracle Enterprise Manager 10*g* Application Server Control コンソールの使用、OC4J で提供される標準準拠 MBean の使用、（適切な状況における）OC4J 固有の XML 構成ファイルの直接使用など、OC4J 用のアプリケーションを構成および管理する方法について説明しています。

- 『Oracle Containers for J2EE サーブレット開発者ガイド』

  このマニュアルには、基本的なサーブレット開発や JDBC と EJB の使用など、OC4J でのサーブレットおよびサーブレット・コンテナの使用に関するサーブレット開発者向けの情報が含まれます。

- 『Oracle Containers for J2EE JavaServer Pages 開発者ガイド』

  このマニュアルには、JavaServer Pages 開発と、OC4J での JSP 実装および JSP コンテナに関する情報が含まれます。また、コマンドライン・トランスレータや OC4J 固有の構成パラメータなどの Oracle 機能に関する説明も含まれます。

- 『Oracle Containers for J2EE JSP タグ・ライブラリおよびユーティリティ・リファレンス』

  このマニュアルには、概念的な情報に加え、タグ・ライブラリ、JavaBeans、および OC4J で提供される他の Java ユーティリティの詳細な構文と使用方法に関する情報が含まれます。また、他の Oracle 製品グループのタグ・ライブラリの概要も含まれます。

- 『Oracle Containers for J2EE サービス・ガイド』

  このマニュアルには、JTA、JNDI、JMS、JAAS、Oracle Application Server Java Object Cache など、OC4J で提供される標準ベースの Java サービスに関する情報が含まれます。

- 『Oracle Containers for J2EE セキュリティ・ガイド』

  このマニュアル（『Oracle Application Server セキュリティ・ガイド』とは異なる）では、OC4J に固有のセキュリティ機能とその実装方法について説明しています。このマニュアルには、Java Authentication and Authorization Service（JAAS）の使用方法およびその他の Java セキュリティ・テクノロジに関する情報が含まれます。

- 『Oracle Containers for J2EE Enterprise JavaBeans 開発者ガイド』

  このマニュアルには、Enterprise JavaBeans 開発と、OC4J での EJB 実装および EJB コンテナに関する情報が含まれます。

- 『Oracle Containers for J2EE リソース・アダプタ管理者ガイド』

  このマニュアルでは、J2EE Connector Architecture 機能の概要と、OC4J でのリソース・アダプタの構成方法および監視方法について説明しています。

- 『Oracle Application Server Web Services 開発者ガイド』

  このマニュアルでは、OC4J での Web サービスの開発方法および構成方法について説明しています。

- 『Oracle Application Server Web Services アドバンスト開発者ガイド』

  このマニュアルでは、Web サービス構築に関する高度な内容について説明しています。たとえば、一般的な相互運用性に関する問題を診断する方法、Web サービス管理機能（信頼性、監査、ロギングなど）を有効化する方法、および Java 値タイプのカスタム・シリアライズを使用する方法が含まれます。

  このマニュアルでは、Web Service Invocation Framework（WSIF）、Web サービス・プロバイダ API、メッセージ添付および管理機能（信頼性、監査、ロギングなど）を使用する方法についても説明しています。また、代替 Web サービス計画（JMS をトランスポート・メカニズムとして使用する方法など）の説明も含まれます。

Oracle TopLink ドキュメント :

- 『Oracle TopLink スタート・ガイド』
- 『Oracle Application Server TopLink Mapping Workbench ユーザーズ・ガイド』
- 『Oracle TopLink 開発者ガイド』

Oracle Database の Java 関連ドキュメント :

- 『Oracle Database Java 開発者ガイド』
- 『Oracle Database JDBC 開発者ガイドおよびリファレンス』
- 『Oracle Database JPublisher ユーザーズ・ガイド』

他の Oracle Application Server ドキュメント :

- 『Oracle Application Server 管理者ガイド』
- 『Oracle Application Server セキュリティ・ガイド』
- 『Oracle Application Server Certificate Authority 管理者ガイド』
- 『Oracle Application Server パフォーマンス・ガイド』
- 『Oracle HTTP Server 管理者ガイド』
- 『Oracle Process Manager and Notification Server 管理者ガイド』
- 『Oracle Application Server グローバリゼーション・ガイド』
- 『Oracle Application Server Web Cache 管理者ガイド』
- Oracle Application Server のアップグレード・ガイド

Oracle JDeveloper ドキュメント :

- Oracle JDeveloper オンライン・ヘルプ

他の Oracle Database ドキュメント :

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle Database アプリケーション開発者ガイド - 基礎編』
- 『Oracle Database PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』
- 『Oracle Database PL/SQL ユーザーズ・ガイドおよびリファレンス』
- 『Oracle Database SQL リファレンス』
- 『Oracle Database Net Services 管理者ガイド』
- 『Oracle Database Advanced Security 管理者ガイド』
- 『Oracle Database リファレンス』

Java サーブレットと JavaServer Pages については、次の Web サイトも利用できます。

`http://www.oracle.com/technology/tech/java/servlets/index.html`

サーブレットの詳細は、次の場所にある『Java Servlet Specification』を参照してください。

`http://java.sun.com/products/servlet/download.html#specs`

Sun 社のリソース :

- Java サーブレット・テクノロジの Web サイト :

  `http://java.sun.com/products/servlet/index.html`

- JavaServer Pages テクノロジの Web サイト :

  `http://java.sun.com/products/jsp/index.html`

- J2EE 1.4 Javadoc（サーブレット・パッケージ `javax.servlet` および `javax.servlet.http` を含む）:

  `http://java.sun.com/j2ee/1.4/docs/api/index.html`

表記規則

このマニュアルでは次の表記規則を使用します。

規則	意味
太字	太字は、操作に関連する Graphical User Interface 要素、または本文中で定義されている用語および用語集に記載されている用語を示します。
イタリック	イタリックは、ユーザーが特定の値を指定するプレースホルダ変数を示します。
固定幅フォント	固定幅フォントは、段落内のコマンド、URL、サンプル内のコード、画面に表示されるテキスト、または入力するテキストを示します。

サポートおよびサービス

次の各項に、各サービスに接続するための URL を記載します。

Oracle サポート・サービス

オラクル製品サポートの購入方法、および Oracle サポート・サービスへの連絡方法の詳細は、次の URL を参照してください。

```
http://www.oracle.co.jp/support/
```

製品マニュアル

製品のマニュアルは、次の URL にあります。

```
http://otn.oracle.co.jp/document/
```

研修およびトレーニング

研修に関する情報とスケジュールは、次の URL で入手できます。

```
http://www.oracle.co.jp/education/
```

その他の情報

オラクル製品やサービスに関するその他の情報については、次の URL から参照してください。

```
http://www.oracle.co.jp
http://otn.oracle.co.jp
```

注意： ドキュメント内に記載されている URL や参照ドキュメントには、Oracle Corporation が提供する英語の情報も含まれています。日本語版の情報については、前述の URL を参照してください。

1

OC4J サービスの概要

Oracle Containers for J2EE（OC4J）は、次のテクノロジをサポートします。このマニュアルには各テクノロジに関する章が含まれています。

- Java Naming and Directory Interface（JNDI）
- Java Message Service（JMS）
- OC4J での Remote Method Invocation の使用方法
- データソース
- OC4J トランザクション・サポート
- Java Object Cache
- XML Query Service
- アプリケーション・クライアント・コンテナの使用方法

この章には、これらの各テクノロジの概要と、関連する章へのリンクが含まれます。

注意： これらのテクノロジの他に、OC4J は JavaMail API、JavaBeans Activation Framework（JAF）および Java API for XML Processing（JAXP）をサポートします。これらのテクノロジの詳細は、Sun 社の J2EE ドキュメントを参照してください。

Java Naming and Directory Interface（JNDI）

OC4J により実装される Java Naming and Directory Interface（JNDI）サービスは、Java アプリケーションにネーミングおよびディレクトリ機能を提供します。JNDI は、特定のネーミングまたはディレクトリ・サービス実装とは関係なく定義されます。このため、JNDI を使用すると、Java アプリケーションは単一の API を使用して異なる（場合によっては複数の）ネーミングおよびディレクトリ・サービスにアクセスできます。この共通 API の背後にネーミングとディレクトリの異なるサービス・プロバイダ・インタフェース（SPI）をプラグインすると、様々なネーミング・サービスを処理できます。

詳細は、第 2 章「JNDI の使用」を参照してください。

Java Message Service（JMS）

Java Message Service（JMS）は、Java プログラムに、エンタープライズ・メッセージ製品にアクセスする共通の方法を提供します。JMS は、JMS クライアントがエンタープライズ・メッセージ製品の機能にアクセスする方法を定義するインタフェースと関連セマンティックの集合です。

旧リリースでは、メモリー内、ファイル・ベースおよびデータベースの各永続性オプションを説明する際に、OracleAS JMS および OJMS という用語を使用していました。OracleAS JMS はメモリー内およびファイル・ベースのオプションを示し、OJMS は Streams Advanced Queuing（AQ）の JMS インタフェースを示していました。JMS に関する混乱を避けるため、OracleAS JMS および OJMS という用語のかわりに、OEMS JMS という用語を使用します。これは、オラクル社が、メッセージの永続性に関する 3 つのオプションに対して単一の JMS インタフェースを提供しているという事実を反映しています。3 つのサービス品質オプションの間でメッセージの永続性を変更するのであれば、JMS アプリケーション・コードを変更する必要はありません。

詳細は、第 4 章「Oracle Enterprise Messaging Service の使用」を参照してください。

データソース

データソースは、`javax.sql.DataSource` インタフェースを実装するオブジェクトがインスタンス化されたものです。データソースにより、データベース・サーバーへの接続を取得できます。

詳細は、第 5 章「データソース」を参照してください。

OC4J トランザクション・サポート

EJB では、トランザクションの管理に Java Transaction API（JTA）1.0.1 が使用されます。これらのトランザクションには、1 フェーズ・コミットと 2 フェーズ・コミットが関連します。

詳細は、第 3 章「OC4J トランザクション・サポート」を参照してください。

OC4J での Remote Method Invocation の使用方法

Remote Method Invocation（RMI）は、リモート・プロシージャ・コール・パラダイムの Java 実装の 1 つです。この実装では、分散アプリケーションは、プロシージャ・コールを起動し、戻り値を解析して通信を行います。

OC4J は、Oracle Remote Method Invocation（ORMI）プロトコルを介した RMI と、Internet Inter-ORB Protocol（IIOP）を介した RMI の両方をサポートします。

OC4J は、デフォルトで RMI/ORMI を使用します。RMI/ORMI は、RMI/IIOP によるメリットに加えて、HTTP を介して RMI/ORMI を起動する「RMI トンネリング」と呼ばれる技術などの機能も提供します。

バージョン 2.0 の Enterprise JavaBeans（EJB）の仕様では、Internet Inter-ORB Protocol（IIOP）を介して RMI を使用して、EJB ベースのアプリケーションが、異なるコンテナ間で別のアプリケーションを簡単に起動できるようにします。既存の EJB を、コード行を変更せずに、Bean のプロパティを編集して再デプロイするのみで相互運用可能にできます。J2EE は RMI を使用して、異なるコンテナで実行されている EJB 間の相互運用性を提供します。

RMI/ORMI および相互運用性（RMI/IIOP）の詳細は、第 6 章「Remote Method Invocation の使用方法」を参照してください。

Java Object Cache

Java Object Cache（以前の OCS4J）は、プロセス内、プロセス間およびローカル・ディスク上で Java オブジェクトを管理する Java クラスの集合です。Java Object Cache の主な目的は、取得や作成にコストがかかるオブジェクトのローカル・コピーを管理することによってサーバーのパフォーマンスを大幅に向上させる、強力で柔軟性のある使いやすいサービスを提供することです。キャッシュできるオブジェクトの型やオブジェクトの元のソースに制限はありません。キャッシュ内の各オブジェクトの管理は容易にカスタマイズできます。各オブジェクトには一連の属性が関連付けられており、キャッシュへのロード方法、格納場所（メモリーまたはディスク、あるいはその両方）、無効化の方法（時間ベースまたは明示的なリクエスト）、無効化されたときの通知先などが制御されます。オブジェクトは、グループ単位または個別に無効化できます。詳細は、第 7 章「Java Object Cache」を参照してください。

XML Query Service

XML Query Service（XQS）はエンタープライズ・データの取得、分析、統合および変換に役立つユーザー・モデルを提供します。このサービスは、XQuery（XML 問合せ言語）で作成されています。通常、XQS などのサービスがなければ、XQuery は XML 文書にしかアクセスできません。XQS を使用することで、XML 以外のドキュメント、リレーショナル・データベース、および XML 以外のその他のエンタープライズ情報システムからもデータを取得できます。詳細は、第 8 章「Oracle XML Query Service」を参照してください。

アプリケーション・クライアント・コンテナの使用方法

アプリケーション・クライアント・コンテナには、リモートの J2EE サーバーにデプロイされている J2EE リソースを必要とするアプリケーション・クライアントを実行するための実行環境が用意されています。アプリケーション・クライアント・コンテナには、J2EE サーバーで使用できるのと同じサービスの多くが実装されています。特に、クライアントは、EJB などのリモート・リソースの検索および実行にコンテナの JNDI および RMI サービスを利用できます。詳細は、第 9 章「アプリケーション・クライアント・コンテナの使用方法」を参照してください。

サード・パーティ・ライセンス

この付録では、Oracle Application Server に含まれるサード・パーティ製品に付属のライセンス契約をリストします。詳細は、付録 A「サード・パーティ・ライセンス」を参照してください。

2

JNDI の使用

この章では、Oracle Containers for J2EE（OC4J）によって実装される Java Naming and Directory Interface（JNDI）サービスについて説明します。この章では、JNDI の初期コンテキストの設定方法と、JNDI ルックアップの実行方法に重点を置いて説明します。

この章には、次の項目が含まれます。

- JNDI の概要
- 初期コンテキストの作成と使用
- JNDI コンテキストの参照
- J2EE アプリケーション・コンポーネントからのオブジェクトのルックアップ
- J2EE アプリケーション・クライアントからのオブジェクトのルックアップ
- JNDI 状態レプリケーション

この章の情報を使用するには、JNDI と JNDI API に関する基本的な知識が必要です。チュートリアルや API ドキュメントなど、JNDI に関する基本情報は、Sun 社の次の Web サイトを参照してください。

```
http://java.sun.com/products/jndi/index.html
```

他の JNDI クラスおよびメソッドの詳細は、次の Javadoc を参照してください。

```
http://java.sun.com/products/jndi/1.2/javadoc/index.html
```

JNDI タスク

この章では、次の一般的な JNDI タスクについて説明します。

- 初期コンテキストの作成と使用
- JNDI 状態レプリケーションの有効化
- JNDI コンテキストの参照
- JNDI コンテキストの構成
- JNDI バインディングの作成と JNDI を使用したバインディングのルックアップ（次の項を参照）：
  - J2EE アプリケーション・コンポーネントからのオブジェクトのルックアップ
  - J2EE アプリケーション・クライアントからのオブジェクトのルックアップ

新機能

次の OC4J JNDI の機能および動作は、今回のリリースの新機能です。

- **JNDI ツリー・ブラウザ :** Application Server Control コンソールには、JNDI ツリー・ブラウザが付属しています（「JNDI コンテキストの参照」を参照）。
- **グローバル JNDI ルックアップ :** デフォルトの OC4J 構成の場合、アプリケーション内のルックアップは、現在のアプリケーションのネームスペースにバインドされます。今回のリリースでは、JNDI を構成してアプリケーション間でルックアップを実行できます。2-4 ページの「グローバル JNDI ルックアップの有効化」を参照してください。
- **相対的 Java コンテキスト・ルックアップ :** 今回のリリースでは、他のすべてのルックアップに失敗した場合、相対的ルックアップが試行されます。たとえば、クライアントが（`"java:comp/env/ejb/EJBName"`）ではなく `context.lookup("ejb/EJBName")` を使用してそのリソースをルックアップしても、ルックアップに成功します。このルックアップの試みは、常に `java:comp/env` ネームスペースを相対位置とするため、バインディング名に `java:comp/env` 接頭辞を追加して `context.lookup()` をコールした場合と同じ効果があります。この機能を使用するのに特別な構成は必要ありません。OC4J のこの機能は、デフォルトで使用できます。
- **参照可能なインタフェース :** このリリースでは、`javax.naming.Referenceable` インタフェースを実装するオブジェクトのバインドを完全にサポートしています。
- **プリンシパルとスレッドの関連付け :** 認証やアクセス制御を必要とする作業では、その作業を実行するスレッドにプリンシパル（ユーザー）を関連付けることができます。詳細は、2-8 ページの「JNDI コンテキストおよびスレッド」を参照してください。
- **JNDI での MBean サポート :** 現在、次の JNDI 関連の MBean が OC4J に登録されており、Oracle Enterprise Manager 10*g* Application Server Control コンソール内で使用できます。
  - `JNDIResource`: この JSR-77 MBean は、特定のアプリケーションの JNDI バインディングに対する問合せに対応しています。JNDIResource MBean は、特定の OC4J インスタンスにデプロイされたアプリケーションごとに登録されます。アプリケーションがアンデプロイされると、関連する JNDIResource MBean も登録解除されます。この MBean には、特定のアプリケーションの JNDI バインディングにアクセスする方法として、次の 2 つが用意されています。
    - `getBindingsAsXMLString()`: XML 文書としてバインディング・ツリーを戻します。
    - `getBindingsAsString()`: 主にデバッグに使用します。
  - `JNDINamespace`: この MBean により、クライアントは、特定の OC4J インスタンスに登録されたすべての JNDIResource MBean を反復できます。つまり、このネームスペース Bean を使用して、アプリケーション名で分割された特定のアプリケーション・サーバー・インスタンスのすべての JNDI バインディングを参照できます。`getAllBindingsAsXMLString()` メソッドは、バインディング・ツリー全体を表す XML 文書を戻します。

  どちらの MBean も、OC4J インスタンスにデプロイされたアプリケーションの現在のバインディングを特定するために解析できる XML 文書を戻します。この XML 文書では、検証用として `oc4j-jndi-bindings-10_0.xsd` スキーマを使用します。

  「Application Server Control」ページから JNDI 関連の MBean へのパス

  「OC4J: ホーム」→「管理」タブ→「タスク名」→「システム MBean ブラウザ」→「タスクに移動」→「J2EEDomain:oc4j」、「J2EEServer:standalone」にドリルダウン→ JNDI ネームスペースおよび JNDI リソース
- **廃止予定**

  次の項目は、今回のリリースで廃止予定です。
  - OC4J で提供されていたコンテキスト・ファクトリの以前のパッケージ構造は、廃止され、より一貫性のあるネーミング構造に置き換えられます。2-6 ページの表 2-1「InitialContext のプロパティ」の下にある「注意」を参照してください。

- **サポート中止**

  次の項目は、今回のリリースではサポートされません。

  - `dedicated.connection` 環境プロパティ。
  - `dedicated.rmicontext` 環境プロパティ。2-13 ページの表 2-2「JNDI 関連の環境プロパティ」を参照してください。

追加ドキュメント

次のサイトにある How-To ドキュメントでは、OC4J の機能に関する情報（各機能の概要や機能に関連するコードの抜粋など）を提供しています。

`http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html`

JNDI の概要

JNDI は、J2EE 仕様の核となる部分です。JNDI により、J2EE アプリケーションと各コンポーネントにネーミングおよびディレクトリ機能が提供されます。JNDI は、特定のネーミングまたはディレクトリ・サービス実装とは関係なく定義されます。これにより、J2EE アプリケーションと各コンポーネントは、単一の API を使用して異なるネーミングおよびディレクトリ・サービスにアクセスできます。この共通 API の背後にネーミングとディレクトリの異なるサービス・プロバイダ・インタフェース（SPI）をプラグインすると、様々なネーミング・サービスを処理できます。

J2EE アプリケーション開発者は、JNDI を使用してネーミング・コンテキストを取得します。ネーミング・コンテキストにより、アプリケーションは、データソース、ローカルおよびリモート Enterprise JavaBeans（EJB）、JMS トピックやキューなどの J2EE リソースを使用できます。

初期コンテキストの作成と使用

最も頻繁に行われる JNDI 操作は、次の 2 つです。

- 新規 `javax.naming.InitialContext` インスタンスの作成
- リソースをルックアップするための `InitialContext` の使用

OC4J は、起動時に各アプリケーションのデプロイメント・ディスクリプタのリソース参照を読み取ることによって、各アプリケーションの JNDI 初期コンテキストを構成します。

- EJB 内からのルックアップの場合、リソース参照は、`ejb-jar.xml` に指定されます。`ejb-jar.xml` に指定されたリソース参照は、次に `orion-ejb-jar.xml` 内の実際の JNDI ロケーションにマップされます。
- サーブレットで初期コンテキストを作成する場合、JNDI 実装により、`web.xml` に指定されたリソース参照へのバインディングが、`orion-web.xml` に指定された実際の JNDI ロケーションにマップされます。
- アプリケーション・クライアントの `application-client.xml` ディスクリプタの参照は、リモート・アプリケーション・クライアントからコンテキストが作成されるときにバインドされます。これらの参照は、`orion-application-client.xml` で指定された実際の JNDI ロケーションにバインドされます。

永続性

初期構成後の各アプリケーションの JNDI ネームスペースは、完全にメモリー・ベースです。コンテキストに対して実行時に追加された内容は、永続化されません。

OC4J が停止すると、プログラム的に（たとえば、`Context.bind` のコールにより）作成されたバインディングは、使用できなくなります。

J2EE デプロイメント・ディスクリプタを通じて宣言的に指定されたバインディングは、アプリケーション・サーバーの停止後も維持されます。

JNDI コンテキストの構成

OC4J は、サーバーにデプロイされた各アプリケーション用の JNDI コンテキストを構成します。OC4J サーバーには、少なくとも 1 つのアプリケーション（グローバル・アプリケーション）が常に存在します。このアプリケーションは、サーバー・インスタンス内の各アプリケーションに対するデフォルトの親です。ユーザー・アプリケーションは、グローバル・アプリケーションからプロパティとバインディングを継承します。ユーザー・アプリケーションでは、グローバル・アプリケーションで定義されたプロパティ値のオーバーライド、プロパティに対する新しい値の指定、および必要に応じた新しいプロパティの定義が可能です。ユーザー・アプリケーションのコンテキストのルックアップは、次の順序で実行されます。

- 最初に、ローカル・アプリケーションのネームスペースが調べられます。
- バインディングがローカルで見つからない場合、親アプリケーションのバインディングが調べられます。
- バインディングがローカルまたは親アプリケーションで見つからない場合、次のいずれかの処理に分岐します。
  - グローバル・ルックアップが有効の場合、OC4J インスタンスに現在デプロイされている既知のすべてのアプリケーション・コンテキストを対象にルックアップの解決が試みられます。
  - グローバル・ルックアップが無効の場合（デフォルトの動作）、`NameNotFoundException` がスローされます。

注意： JNDI ネームスペースへのアクセスを保護する方法の詳細は、『Oracle Application Server セキュリティ・ガイド』を参照してください。

グローバル JNDI ルックアップの有効化

グローバル JNDI 機能を使用すると、OC4J インスタンスの既知のすべてのアプリケーションを対象にルックアップが試行されます。現在のアプリケーション内でのルックアップに失敗した場合、デプロイされた各アプリケーションのコンテキストを対象にルックアップが試行されます。特定の名前で最初に成功したルックアップ結果が戻されます。この機能は、デフォルトで無効になっています。

グローバル JNDI 機能を有効化するには、次のようにします。

1. テキスト・エディタを使用して、`J2EE_HOME/config/server.xml` を開きます。
2. `<application-server>` 要素内に `global-jndi-lookup-enabled` 属性を追加して、`true` に設定します。

```
<application-server
   global-jndi-lookup-enabled="true">
   ...
</application-server>
```

3. `server.xml` を保存して閉じます。
4. アプリケーション・サーバーを再起動します。

注意： 各リソースのバインディング名が、OC4J インスタンスにデプロイされたすべてのアプリケーションにおいて一意である必要があります。また、すべてのアプリケーションに対するルックアップが実行される場合、アプリケーションの順序は保証されません。同じインスタンス内の 2 つのアプリケーションに異なるオブジェクトを指し示す同じ名前のバインディングが存在すると、ルックアップによって予期しないオブジェクトが戻される可能性があります。

クラスパスの構成

ターゲット・アプリケーションのクラスは、JNDI ルックアップを実行するアプリケーションのクラスパスに存在する必要があります。クラスパスの構成時には、次のオプションがサポートされています。

- OC4J 共有ライブラリの一部としてターゲット・ライブラリを含めます。これにより、OC4J は、すべてのデプロイ済アプリケーションによってインポートされるシステム・ライブラリとして JAR をロードします。より厳密にシステム・ライブラリを制御する場合は、名前付きシステム・ライブラリとしてターゲット・クラスをデプロイし、これらのライブラリを明示的に各アプリケーションにインポートすることも可能です。

  シリアライズできない Java オブジェクト、またはリモートで使用できない Java オブジェクト（ローカル・オブジェクト）では、共有ライブラリを使用する必要があります。また、コール元およびターゲット・アプリケーションの両方にインポートする必要があります。

  注意： 共有ライブラリ作成の詳細は、『Oracle Containers for J2EE 開発者ガイド』を参照してください。

- ターゲット JAR ファイルをコール元の EAR ファイルに含めます。コール元の EAR ファイルにターゲット JAR をコピーすると、その JAR を次のいずれかの方法で適切なクラスパスにロードできるようになります。

  - サーブレットからグローバル・ルックアップが実行される場合は、EAR 内にデプロイされている Web アプリケーション・ファイル（WAR）の `WEB-INF/lib` ディレクトリにターゲット JAR を配置します。
  - EJB ビジネス・メソッドからグローバル・ルックアップが実行される場合は、コール元アプリケーションのトップレベルの EAR ファイルにターゲット JAR を配置します。コール元の JAR のマニフェストに `Class-Path` タグを追加して、クラスパスにターゲット JAR を追加します。

    次に、（コール元の JAR ファイルの `META-INF` ディレクトリに存在する）必要なマニフェスト・ファイルの例を示します。

    ```
    "Manifest-Version: 1.0
    Class-Path: jndi_globalLookup-ejb2.jar"
    ```

    この例の `Class-Path` 行には、ターゲット JAR（この例では `jndi_globalLookup-ejb2.jar`）がコール元 EJB のクラスパスにロードされるよう指定されています。

環境およびコンストラクタ

OC4J が JNDI 初期コンテキストの構成に使用する環境は、次の 3 つの場所にあります。

- `java.util.Hashtable` インスタンスで明示的に指定された環境。JNDI 初期コンテキスト・コンストラクタに渡されます（このコンストラクタのコード例は、2-14 ページの「例 : EJB をルックアップするアプリケーション・クライアント」を参照してください）。
- システム・プロパティ値。OC4J サーバーまたはアプリケーション・コンテナによって設定されます。
- `jndi.properties` ファイル。（アプリケーション・クライアントの JAR ファイルの一部として）アプリケーションの EAR ファイルに含まれます。

JNDI の `InitialContext` には、次の 2 つのコンストラクタがあります。どちらのコンストラクタを使用しても、初期コンテキストを作成できます。

```
InitialContext()
InitialContext(Hashtable env)
```

InitialContext()

`InitialContext()` コンストラクタでは、デフォルトのコンテキスト環境を使用して `Context` オブジェクトが作成されます。OC4J のサーバー・サイド・アプリケーションでこのコンストラクタを使用すると、起動時に OC4J によって構成されたアプリケーション・コンテキストが戻されます。このコンストラクタは、通常、JSP、EJB およびサーブレットなど、サーバー・サイドで実行されるコードで使用されます。

InitialContext(Hashtable env)

`InitialContext(Hashtable env)` コンストラクタでは、環境パラメータが使用されます。この形式の `InitialContext` コンストラクタは、通常、JNDI 環境を指定する必要があるリモート・クライアント・アプリケーションで使用されます。このコンストラクタの `env` パラメータは `java.util.Hashtable` で、JNDI に必要なプロパティが含まれます。表 2-1 に、`javax.naming.Context` インタフェースで定義されるこれらのプロパティを示します。

表 2-1 InitialContext のプロパティ

プロパティ	値	意味
`java.naming.factory.initial`	`INITIAL_CONTEXT_FACTORY`	新規の初期コンテキスト・オブジェクトの作成時にどの初期コンテキスト・ファクトリを使用するかを指定します。有効な設定は次のとおりです。 ■ `oracle.j2ee.rmi.RMIInitialContextFactory` ■ `oracle.j2ee.naming.ApplicationClientInitialContextFactory` ■ `oracle.j2ee.iiop.IIOPInitialContextFactory` 廃止予定のコンテキスト・ファクトリの詳細は、この表の下にある「注意」を参照してください。
`java.naming.provider.url`	`PROVIDER_URL`	サーバー上のオブジェクトをルックアップするためにクライアント・サイド JNDI コードで使用される URL を指定します。詳細は、2-13 ページの表 2-2「JNDI 関連の環境プロパティ」を参照してください。
`java.naming.security.principal`	`SECURITY_PRINCIPAL`	現在のセキュリティ資格証明のユーザー名を指定します。アプリケーション・クライアント・コードでクライアントを認証するために必要です。サーバー・サイドのコードでは、認証はすでに実行されているため必要ありません。
`java.naming.security.credential`	`SECURITY_CREDENTIALS`	現在のセキュリティ資格証明のパスワードを指定します。アプリケーション・クライアント・コードでクライアントを認証するために必要です。サーバー・サイドのコードでは、認証はすでに実行されているため必要ありません。

これらのプロパティを設定し、新規 JNDI 初期コンテキストを取得するコード例は、2-14 ページの「例 : EJB をルックアップするアプリケーション・クライアント」を参照してください。

注意 : 次のコンテキスト・ファクトリは、廃止予定です。

- `com.evermind.server.rmi.RMIInitialContextFactory`
- `com.evermind.server.ApplicationClientInitialContextFactory`
- `com.oracle.iiop.server.IIOPInitialContextFactory`

廃止予定のコンテキスト・ファクトリ名と置き換えられる新しいファクトリ名の詳細は、表 2-1「InitialContext のプロパティ」の `java.naming.factory.initial` 項目を参照してください。

例 : EJB のルックアップ

次の例は、一般的な Web または EJB アプリケーションにおいて、サーバー・サイドで使用されるコードです。

```
Context ctx = new InitialContext();
HelloHome myhome = (HelloHome) ctx.lookup("java:comp/env/ejb/myEJB");
```

最初の文は、デフォルト環境を使用して新規の初期コンテキスト・オブジェクトを作成します。2 番目の文は、アプリケーションの JNDI ツリーで EJB ホーム・インタフェース参照をルックアップします。

この場合、`myEJB` は、`web.xml`（このコードがサーブレットで実行される場合）または `ejb-jar.xml`（このコードが EJB ビジネス・メソッドで実行される場合）で宣言されているセッション Bean への参照の名前です。この参照は、`<ejb-ref>` タグで定義されます。次に、EJB 参照は、このコードを実行するコール元に応じて、`orion-web.xml` または `orion-ejb-jar.xml` の実際の JNDI ロケーションにマップされます。

次に例を示します。

```
<ejb-ref>
  <ejb-ref-name>ejb/myEJB</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>myEjb.HelloHome</home>
  <remote>myEjb.HelloRemote</remote>
</ejb-ref>
```

JNDI コンテキストおよびスレッド

デフォルトでは、ユーザー名とパスワードを使用して JNDI コンテキストを作成すると、OC4J での初期コンテキストの構成時に、プリンシパルまたはユーザーがコンテキスト・インスタンスにバインドされます。これにより、OC4J ネームスペースでの名前付きオブジェクトのルックアップ、リモート・オブジェクトへの参照の取得、およびリモート・オブジェクトに対する操作の起動を実行する場合、認証およびアクセス制御において指定のプリンシパルまたはユーザーが使用されます。この使用例では、1 つのアプリケーション・クライアント内の複数のスレッドで、コンテキストが共有される可能性があります。

プリンシパルまたはユーザーは、作業を実行するスレッドと関連付けることも可能です。これを行うには、適切なプリンシパルおよび資格証明のプロパティ値を使用してコンテキストを作成します。スレッドとプリンシパルの関連付けを解除するには、コンテキストをクローズします。

スレッドがプリンシパルと関連付けられると、そのプリンシパルは、関連付けられたスレッドのデフォルトになります。その後、プリンシパルまたは資格証明プロパティなしでコンテキストが作成されても、スレッドに関連付けられたプリンシパルは変化しません。

単一のコンテキストを複数のスレッドで共有することは、技術的には可能ですが、別のスレッドにコンテキストを渡しても、そのコンテキストの作成に使用されたプリンシパルは新規スレッドに関連付けられません。プリンシパルを新規スレッドに関連付ける唯一の方法は、そのスレッド内で新規コンテキストを作成することです。また、コンテキストは、それが作成されたスレッド内でのみクローズできます。これらの理由のため、特定のコンテキストで実行されるすべての作業は、同じスレッド内で処理することをお薦めします。

1 つのスレッドは、任意の時点において、ただ 1 つのプリンシパルと関連付けることができます。どのコンテキストもクローズせずに、単一のスレッド内で複数のコンテキストを作成した場合、そのスレッドは、最後に作成されたコンテキストで使用されているプリンシパルと関連付けられます。プリンシパル情報は、スレッドとともにスタックに格納されます。最後のコンテキストがクローズされると、スレッドは、その前のコンテキストの作成に使用されたプリンシパルと関連付けられ、その後も同様に処理されます。

この機能を有効化するには、コマンドラインでシステム・プロパティの `-DAssociateUserToThread=true` を設定します。デフォルトでは、この機能は無効です（`false` に設定されています）。

JNDI コンテキストの参照

JNDI ブラウザを使用すると、JNDI ネームスペース全体を参照して、オブジェクトの特定のセットがアプリケーションで実際にバインドされていることを確認できます。

JNDI ブラウザは、Oracle Enterprise Manager 10*g* Application Server Control コンソール内で次のようにアクセスできます。

JNDI ブラウザへのパス

「OC4J: ホーム」→「管理」タブ→「タスク名 : サービス」→「JNDI ブラウザ」→「タスクに移動」アイコンをクリック→「すべてを開く」

JNDI バインディングを文字列として取得する手順

JNDI コンテキストのバインディングは、文字列表現としても取得できます。これは、デバッグに役立つ場合があります。

「OC4J: ホーム」→「管理」タブ→「タスク名」→「システム MBean ブラウザ」→「タスクに移動」→「J2EEDomain:oc4j」、「J2EEServer:standalone」、「JNDIResource」にドリルダウン→「アプリケーションの選択」→「操作」タブ→ `getBindingsAsString` または `getBindingsAsXMLString` →「起動」

J2EE アプリケーション・コンポーネントからのオブジェクトのルックアップ

この項では、JNDI を使用して、サーブレット、JSP ページ、EJB などの J2EE アプリケーション・コンポーネントから、バインドされたオブジェクトをルックアップする方法について説明します。

OC4J で初期コンテキスト・ファクトリを使用して、J2EE アプリケーション・コンポーネントから次のようにオブジェクトにアクセスできます。

- 同じアプリケーション内のオブジェクトのルックアップ
- 別のアプリケーション内のオブジェクトのルックアップ

同じアプリケーション内のオブジェクトのルックアップ

サーバーで実行されているコードは、アプリケーションの一部として定義されます。コードはアプリケーションの一部であるため、JNDI が使用するプロパティのデフォルトは、OC4J により設定されます。JNDI の `InitialContext` オブジェクトの構成時には、アプリケーション・コード（サーブレットや EJB ビジネス・メソッドなど）でプロパティ値を指定する必要はありません。

注意： アプリケーションでリモート参照（同じ JVM 内の別の J2EE アプリケーションのリソースや J2EE アプリケーションの外部のリソースなど）をルックアップする必要がある場合は、`RMIInitialContextFactory` または `IIOPInitialContextFactory` を使用する必要があります。2-10 ページの「別のアプリケーション内のオブジェクトのルックアップ」を参照してください。

例 : データソースをルックアップするサーブレット

この例では、データベース上で JDBC 操作を実行するために、サーブレットでデータソースを取得します。

Application Server Control コンソールを使用して、データソースの場所を指定します。「データ・ソースの作成」ページの「JNDI ロケーション」フィールドに場所を指定します。第 5 章「データソース」を参照してください。

サーブレットの `web.xml` ファイルで、次のリソースを定義します。

```
<resource-ref>
   <description>
      A data source for the database in which
      the EmployeeService enterprise bean will
      record a log of all transactions.
   </description>
   <res-ref-name>jdbc/EmployeeAppDB</res-ref-name>
   <res-type>javax.sql.DataSource</res-type>
   <res-auth>Container</res-auth>
   <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

対応する `orion-web.xml` のマッピングは次のとおりです。

```
<resource-ref-mapping name="jdbc/EmployeeAppDB" location="jdbc/pool/OracleCache" />
```

`name` 値は、`web.xml` の `<res-ref-name>` 要素で指定されている値と同じです。

`location` 値は、`data-sources.xml` の `<data-source>` 要素の `jndi-name` 属性に対応しています。

この場合、サーブレットの次のコードによって、データソース・オブジェクトへの正しい参照が戻されます。

```
...
try {
  InitialContext ic = new InitialContext();
  ds = (DataSource) ic.lookup("java:comp/env/jdbc/EmployeeAppDB");
  ...
}
catch (NamingException ne) {
  throw new ServletException(ne);
}
...
```

同じアプリケーション内でオブジェクトをルックアップする場合、初期コンテキスト・ファクトリを指定する必要はありません。アプリケーションの起動時に、OC4J によって適切なデフォルトが設定されるためです。通常、同じアプリケーション内のルックアップでは、InitialContext を作成する `javax.naming.InitialContext` の `no-args` コンストラクタのみが必要です。

この場合、同じアプリケーション内または `java:comp/` の下に含まれているオブジェクトのルックアップに URL が必要ないため、プロバイダ URL を指定する必要もありません。

注意： 一部のプラットフォームでは、JDK のバージョンによって、システム・プロパティ `java.naming.factory.url.pkgs` が自動設定され、`com.sun.java.*` が組み込まれることがあります。

このプロパティをチェックし、`com.sun.java.*` が存在している場合は削除してください。

別のアプリケーション内のオブジェクトのルックアップ

別のアプリケーション内のオブジェクトにアクセスする場合、またはスタンドアロンの Java クライアントから J2EE リソースにアクセスする場合、次のいずれかのコンテキスト・ファクトリを使用します。

- `oracle.j2ee.rmi.RMIInitialContextFactory`
- `oracle.j2ee.iiop.IIOPInitialContextFactory`
- `oracle.j2ee.naming.ApplicationClientInitialContextFactory`

RMIInitialContextFactory

RMIInitialContexFactory は、分散ルックアップのために Oracle Remote Method Invocation（RMI）プロトコルを使用する JNDI コンテキストの実装を提供します。このコンテキスト・ファクトリは、リモート・クライアントと、他の OC4J インスタンスにデプロイされたバインディングをルックアップするアプリケーションによって使用されます。

`RMIInitialContextFactory` で使用される JNDI 環境プロパティの詳細は、2-13 ページの表 2-2「JNDI 関連の環境プロパティ」を参照してください。RMI の詳細は、第 6 章「Remote Method Invocation の使用方法」を参照してください。

例 : RMI を使用してリモートから EJB をルックアップするサーブレット

この例では、サーブレットで、異なるマシンの別の OC4J インスタンスで稼働する EJB にアクセスします。この例の EJB は、2-14 ページの「例 : EJB をルックアップするアプリケーション・クライアント」で使用されている EmployeeBean です。

次のコードは、サーブレット・コードから抜粋したものです。

```
Hashtable env = new Hashtable();
env.put("java.naming.factory.initial",
"oracle.j2ee.rmi.RMIInitialContextFactory");
env.put("java.naming.provider.url","ormi://remotehost/bmpapp");
env.put("java.naming.security.principal","SCOTT");
env.put("java.naming.security.credentials","TIGER");
Context context = new InitialContext(env);
Object homeObject =
context.lookup("EmployeeBean");
```

例 : 複数インスタンス環境でリモートから EJB をルックアップするサーブレット

*i*AS 内で OC4J を実行する場合、リモート JNDI サービスにより、リクエストで opmn:ormi プロトコルを使用するよう指定できます。これにより、クライアントでは、ORMI ポート情報をハードコードせずにルックアップを試行できます。OC4J JNDI コードは、この *i*AS インストールに適切な ORMI ポートを決定するために、opmn プロセスと通信します。ルックアップは前述の ORMI の例と似ていますが、java.naming.provider.url プロパティは、opmn:ormi で始まる URL に設定されます。次に、opmn:ormi を使用した同じルックアップ・コードの例を示します。

次のコードは、サーブレット・コードから抜粋したものです。

```
Hashtable env = new Hashtable();
env.put("java.naming.factory.initial",
"oracle.j2ee.rmi.RMIInitialContextFactory");
env.put("java.naming.provider.url","opmn:ormi://remotehost/bmpapp");
env.put("java.naming.security.principal","SCOTT");
env.put("java.naming.security.credentials","TIGER");
Context context = new InitialContext(env);
Object homeObject =
context.lookup("EmployeeBean");
```

opmn URL の参照先となるホーム・インスタンスを指定する方法もあります。opmn の使用方法の詳細は、「第 6 章「Remote Method Invocation の使用方法」」を参照してください。

IIOPInitialContextFactory

IIOPInitialContextFactory は、分散ルックアップのために Internet Inter-ORB Protocol（IIOP）を使用する JNDI コンテキストの実装を提供します。RMI の詳細は、第 6 章「Remote Method Invocation の使用方法」を参照してください。

IIOPInitialContextFactory を使用する状況は、リモート・プロトコルが ORMI ではなく IIOP であることを除けば、RMIInitialContextFactory のときと同様です。

注意 : IIOPInitialContextFactory は、EJB のルックアップにのみ使用できます。この条件は、ApplicationClientInitialContextFactory で corbaname URL を使用する場合にも適用されます。

例 : IIOP を使用してリモートから EJB をルックアップするサーブレット

次に例を示します。

```
Hashtable env = new Hashtable();
env.put("java.naming.factory.initial",
"oracle.j2ee.iiop.IIOPInitialContextFactory");
env.put("java.naming.provider.url","corbaname::remotehost:PORT_NUMBER#APPLICATION_NAME"
);
env.put("java.naming.security.principal","SCOTT");
env.put("java.naming.security.credentials","TIGER");
Context context = new InitialContext(env);
Object homeObject =
context.lookup("EmployeeBean");
```

この例では、corbaname URL は、EJB の場所を指定するのに使用されます。corbaname 文字列で、PORT_NUMBER は OC4J で使用するよう構成されている IIOP のポート番号、remotehost は OC4J が稼働するサーバーの名前、APPLICATION_NAME は EJB を含むアプリケーションの名前です。

この例で使用されるアプリケーションは、ルックアップが正常に機能するためには、IIOP が有効の状態でデプロイされている必要があります。また、リモート・クライアントには、OC4J が生成した IIOP クライアント jar が含まれる必要があります。詳細は、6-20 ページの「ORMI から IIOP トランスポートへの切替え」にある OC4J IIOP に関する説明を参照してください。

J2EE アプリケーション・クライアントからのオブジェクトのルックアップ

この項では、アプリケーション・クライアントを構成して、OC4J インスタンスの内部で稼働するオブジェクトにアクセスする方法について説明します。OC4J アプリケーション・クライアント・コンテナの使用方法の詳細は、第 9 章「アプリケーション・クライアント・コンテナの使用方法」を参照してください。

リモートの J2EE サーバー・アプリケーションで使用可能なリソースをアプリケーション・クライアントでルックアップする場合、クライアントは初期コンテキストの構成に oracle.j2ee.naming パッケージの ApplicationClientInitialContextFactory を使用します。

注意 : アプリケーションが J2EE アプリケーション・クライアントの場合（すなわち、application-client.xml ファイルが存在する場合）は、クライアント・アプリケーションで使用されるプロトコル（ORMI または IIOP）に関係なく、常に ApplicationClientInitialContextFactory を使用する必要があります。プロトコル自体は、JNDI プロパティ java.naming.provider.url で指定します。詳細は、2-13 ページの表 2-2「JNDI 関連の環境プロパティ」を参照してください。

コンポーネントのリモート・アクセスとローカル・アクセスは、リモート・クライアントの観点からすると本質的に同じです。クライアントでは、プロバイダ URL に応じて ORMI または IIOP を使用できます。

OC4J サーバーに接続する Java コードで構成されるアプリケーション・クライアントについて考えてみます。たとえば、ワークステーションで実行されるクライアント・コードで、EJB などのサーバー・オブジェクトに接続して、一部のアプリケーション・タスクを実行するとします。この場合、リモート・クライアントでは、プロパティ java.naming.factory.initial の値を oracle.j2ee.naming.ApplicationClientInitialContextFactory に設定する必要があります。この値は、クライアント・コードで指定できます。または、アプリケーション・クライアントの JAR ファイルの一部である jndi.properties ファイルで指定することもできます。

リモート・オブジェクトにアクセスするために、ApplicationClientInitialContextFactory は、注釈を使用してリソース・インジェクションを利用するか、META-INF/application-client.xml ファイルおよび META-INF/orion-application-client.xml ファイルを読み取ります。どちらのファイルも、アプリケーション・クライアントの JAR ファイルにあります。

環境プロパティ

ORMI プロトコルが使用されている場合、ApplicationClientInitialContextFactory は表 2-2 に示すプロパティを環境から読み取ります。

表 2-2 JNDI 関連の環境プロパティ

プロパティ	意味
dedicated.rmicontext	このプロパティは、OC4J で使用されなくなりました。このプロパティは、次の 2 つの理由により使用されていました。 ■ ロード・バランシングを有効にするため ■ 既知の ORMI および JNDI の不具合を回避するため クライアント・サイドの ORMI ロード・バランシングを有効化するには、「ロード・バランシング」に記載されているプロパティを使用してください。 2-14 ページの「ロード・バランシング」に記載されているプロパティは、このフラグをまだ必要とするいくつかのケースで使用できます。
java.naming.provider.url	ローカル・オブジェクトまたはリモート・オブジェクトの検索時に使用する URL を指定します。書式は次のいずれかです。 [opmn:|http:|https:]ormi://hostname/appname または corbaname:hostname:port corbaname URL の詳細は、6-22 ページの「corbaname の URL の指定」を参照してください。 ORMI プロトコルを使用する場合、カンマ区切りのリストで（フェイルオーバー用の）複数のホストを指定できます。
java.naming.factory.url.pkgs	一部のプラットフォームでは、JDK のバージョンによって、システム・プロパティ java.naming.factory.url.pkgs が自動設定され、com.sun.java.* が組み込まれることがあります。 このプロパティをチェックし、com.sun.java.* が存在している場合は削除してください。
Context.SECURITY_PRINCIPAL	ユーザー名を指定します。このプロパティは、クライアント・サイドのコードでクライアントを認証するために必要です。サーバー・サイドのコードでは、認証はすでに実行されているため必要ありません。このプロパティ名は、java.naming.security.principal としても定義されます。
Context.SECURITY_CREDENTIAL	パスワードを指定します。このプロパティは、クライアント・サイドのコードでクライアントを認証するために必要です。サーバー・サイドのコードでは、認証はすでに実行されているため必要ありません。このプロパティ名は、java.naming.security.credentials としても定義されます。

ロード・バランシング

大量のリクエストが予想される場合、OC4J インスタンス間でリクエストのロード・バランシングを行うと有益である場合があります。ロード・バランシングは、`oracle.j2ee.rmi.loadBalance` プロパティを使用して構成できます。このプロパティは、クライアントの `jndi.properties` ファイル、またはクライアント・コードのハッシュテーブルで設定します。次の表に、`oracle.j2ee.rmi.loadBalance` プロパティの値を示します。

表 2-3 oracle.j2ee.rmi.loadBalance プロパティの値

値	説明
`client`	指定した場合、クライアントは、通信全体に対する最初のルックアップで初期選択された OC4J プロセスと対話します。
`context`	クラスタ化された OC4J 環境の EJB にアクセスする Web クライアント（サーブレットまたは JSP）で使用されます。 指定した場合、`InitialContext()` が起動するたびに、ランダムに選択された OC4J インスタンスの新規 `Context` オブジェクトが戻されます。
`lookup`	クラスタ化された OC4J 環境の EJB にアクセスするスタンドアロン・クライアントで使用されます。 指定した場合、クライアントが `Context.lookup()` をコールするたびに、ランダムに選択された OC4J インスタンスの新規 `Context` オブジェクトが作成されます。

ロード・バランシングの詳細は、6-12 ページの「ORMI リクエストのロード・バランシングの構成」を参照してください。

例 : EJB をルックアップするアプリケーション・クライアント

この項では、アプリケーション・クライアントを構成して、OC4J インスタンスの内部で稼働する EJB にアクセスする方法について説明します。

最初に、EJB を `EmployeeBean` という名前で OC4J にデプロイします。この名前は、`ejb-jar.xml` で次のように定義されます。

```
<ejb-jar>
  <display-name>bmpapp</display-name>
  <description>
     An EJB app containing only one Bean Managed Persistence Entity Bean
  </description>
  <enterprise-beans>
     <entity>
        <description>no description</description>
        <display-name>EmployeeBean</display-name>
        <ejb-name>EmployeeBean</ejb-name>
        <home>bmpapp.EmployeeHome</home>
        <remote>bmpapp.Employee</remote>
        <ejb-class>bmpapp.EmployeeBean</ejb-class>
        <persistence-type>Bean</persistence-type>
        ...
     </entity>
  </enterprise-beans>
..
</ejb-jar>
```

EJB EmployeeBean は、orion-ejb-jar.xml 内の JNDI ロケーション bmpapp/EmployeeBean に次のようにバインドされます。

```
<orion-ejb-jar>
   <enterprise-beans>
       <entity-deployment name="EmployeeBean"
           location="bmpapp/EmployeeBean" table="EMP">
                       ...
       </entity-deployment>
                       ...
   </enterprise-beans>
           ...
</orion-ejb-jar>
```

アプリケーション・クライアント・プログラムは、EmployeeBean EJB を使用し、それを EmployeeBean として参照します。

リソース・インジェクションをリクエストするためにアプリケーション・クライアントのメイン・クラスで注釈を使用する方法と JNDI API の使用方法を示す、アプリケーション・クライアント・プログラムからの抜粋を示します。

注釈の使用方法

```
public class Client {
   @EJB
      private static EmployeeHome home;
      public  static void main(String[] args) {


       // Create a new record.
       Employee rec = home.create(empNo, empName, salary);

       // call method on the EJB
       rec.doSomething();
       ...
   }
```

JNDI API の使用方法

```
public static void main (String args[])
{
 ...
 Context context = new InitialContext();
 /**
  * Look up the EmployeeHome object. The reference is retrieved from the
  * application environment naming context (java:comp/env). The ejb reference is
  * specified in the assembly descriptor (META-INF/application-client.xml).
  */
 Object homeObject =
     context.lookup("java:comp/env/EmployeeBean");
 // Narrow the reference to an EmployeeHome.
 EmployeeHome home =
     (EmployeeHome) PortableRemoteObject.narrow(homeObject,
                                                EmployeeHome.class);
 // Create a new record.
 Employee rec = home.create(empNo, empName, salary);
 // call method on the EJB
 rec.doSomething();
 ...
}
```

次の行でコンテキストを作成するときに、ハッシュテーブルを渡していないことに注意してください。

```
Context context = new InitialContext();
```

これは、コンテキストが jndi.properties ファイルから読み取られる値を使用して作成されるためです。この例では、次の内容が含まれています。

```
java.naming.factory.initial=
   oracle.j2ee.naming.ApplicationClientInitialContextFactory
java.naming.provider.url=ormi://localhost/bmpapp
java.naming.security.principal=SCOTT
java.naming.security.credentials=TIGER
```

または、jndi.properties ファイルを提供するかわりに、InitialContext のコンストラクタにハッシュテーブルを渡すこともできます。その場合、コードは次のようになります。

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
   "oracle.j2ee.naming.ApplicationClientInitialContextFactory");
env.put("java.naming.provider.url","ormi://localhost/bmpapp");
env.put("java.naming.security.principal","SCOTT");
env.put("java.naming.security.credentials","TIGER");
Context initial = new InitialContext(env);
```

アプリケーション・クライアントのコードは EmployeeBean EJB を参照するため、この EJB を application-client.xml ファイルの <ejb-ref> 要素で宣言する必要があります。

```
<application-client>
   <display-name>EmployeeBean</display-name>
   <ejb-ref>
       <ejb-ref-name>EmployeeBean</ejb-ref-name>
       <ejb-ref-type>Entity</ejb-ref-type>
       <home>bmpapp.EmployeeHome</home>
       <remote>bmpapp.Employee</remote>
   </ejb-ref>
</application-client>
```

EmployeeBean EJB が orion-ejb-jar.xml ファイル内で構成されたとおり JNDI ロケーション bmpapp/EmployeeBean にバインドされることに注意してください。アプリケーション・クライアント・プログラムで使用する EJB 参照名は、EJB が実際に orion-ejb-jar.xml でバインドされている JNDI ロケーションにマップする必要があります。これは、orion-application-client.xml ファイルで次のように指定します。

```
orion-application-client.xml file:
<orion-application-client>
   <ejb-ref-mapping name="EmployeeBean" location="bmpapp/EmployeeBean" />
</orion-application-client>
```

application-client.xml ファイルおよび orion-application-client.xml ファイルの詳細は、『Oracle Containers for J2EE 開発者ガイド』の付録 A「OC4J 固有のデプロイメント・ディスクリプタ」を参照してください。

例 : IIOP を使用して EJB をルックアップするアプリケーション・クライアント

前の例では、アプリケーション・クライアントが、リソースをリモート・ルックアップするための基底プロトコルとして ORMI を使用しました。かわりに、リモート・プロトコルとして IIOP を使用するようにアプリケーション・クライアントを構成できます。EmployeeBean をルックアップするコードは、前の例と同様ですが、次の部分が異なります。

jndi.properties ファイルは、IIOP がリモート・プロトコルとして使用されるという事実を反映するように変更する必要があります。これは、jndi.properties の java.naming.provider.url システム・プロパティで構成します。次に、IIOP を使用するよう構成された同じ例の jndi.properties ファイルを示します。

```
java.naming.factory.initial=
   oracle.j2ee.naming.ApplicationClientInitialContextFactory
java.naming.provider.url=corbaname::REMOTE_HOST:IIOP_PORT#bmpapp
java.naming.security.principal=SCOTT
java.naming.security.credentials=TIGER
```

または、IIOP を基底プロトコルとして使用するため、jndi.properties ファイルを指定するかわりに InitialContext のコンストラクタにハッシュテーブルを渡すこともできます。その場合、コードは次のようになります。

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"oracle.j2ee.naming.ApplicationClientInitialContextFactory");
env.put("java.naming.provider.url","corbaname::REMOTE_HOST:IIOP_PORT#bmpapp");
env.put("java.naming.security.principal","SCOTT");
env.put("java.naming.security.credentials","TIGER");
Context initial = new InitialContext(env);
```

この例のようにアプリケーション・クライアントでハッシュテーブルを使用して JNDI プロパティを設定する場合、セキュリティ資格証明をシステム・プロパティとして設定する必要があります。IIOP インターセプターは、システム・プロパティを通じてクライアントの資格証明にアクセスする必要があります。このことは、jndi.properties ファイルの使用時には自動的に行われます。JNDI フレームワークにより、これらのプロパティが自動的にシステム・プロパティとして設定されるためです。

プロパティをコード自体で設定する場合、次の行をクライアント・コードに追加する必要があります。

```
System.setProperty("java.naming.security.principal","SCOTT");
System.setProperty("java.naming.security.credentials ","TIGER");
```

jndi.properties ファイルを使用する場合、またはプログラム的に処理する場合のいずれの例でも、次の要件を満たす必要があります。

- REMOTE_HOST は、OC4J サーバーの稼働するサーバーの名前に設定する必要があります。
- IIOP_PORT は、IIOP リクエストを処理するために OC4J で使用されるポート番号に設定する必要があります。

アプリケーション・クライアントで IIOP を使用する場合、アプリケーション・クライアント・コードやデプロイメント・ディスクリプタを変更する必要はありません。クライアントは、jndi.properties ファイルの使用方法などについて、IIOP が適切に動作するよう構成する必要があります。

デプロイされている各 EJB の IIOP スタブにアクセスするため、リモート・アプリケーション・クライアントでは、デプロイ済アプリケーションの IIOP クライアント jar をクラスパスに含める必要があります。

詳細は、6-20 ページの「ORMI から IIOP トランスポートへの切替え」を参照してください。

JNDI 状態レプリケーション

この項では、JNDI 状態レプリケーションについて説明します。

注意： JNDI 状態レプリケーションでは、複数 OC4J 環境でのクラスタリングがサポートされます。

開発中のアプリケーションを複数 OC4J 環境で使用するための環境としてスタンドアロン・モードの Oracle Application Server を使用している場合、この項の情報は、その計画およびコーディング・プロセスに役立ちます。

この項には、次の項目が含まれます。

JNDI 状態レプリケーションとは

JNDI 状態レプリケーションにより、ある OC4J クラスタの 1 つの OC4J インスタンスにおけるコンテキストの変更が、そのクラスタに存在する他のすべての OC4J インスタンスのネームスペースにレプリケートされます。

JNDI 状態レプリケーションが有効化されている場合は、1 つのサーバー上でシリアライズ可能な値をアプリケーション・コンテキストに（リモート・クライアント、EJB またはサーブレットを使用して）バインドし、それを別のサーバー上で読み取ることができます。

JNDI 状態レプリケーションの有効化

JNDI 状態レプリケーションは、アプリケーションごとに有効化されている必要があります。アプリケーションをデプロイするときに、アプリケーションの `META-INF` ディレクトリにある `orion-application.xml` に `<cluster>` タグを追加できます。この種の構成により、各アプリケーションのクラスタリング・ステータスを制御できます。この `<cluster>` タグを追加することで、HTTP/EJB セッション・レプリケーションと JNDI 状態レプリケーションが可能になります。

`<cluster>` 要素は、クラスタリング構成のための単一のメカニズムとして機能します。`<cluster>` 要素は、次のいずれかのファイルに追加できます。

- `ORACLE_HOME/j2ee/home/config/application.xml` ファイル（グローバル・レベルでクラスタリングを構成する場合）
- アプリケーション固有の `orion-application.xml` ファイル（各アプリケーションにクラスタリングを構成する場合）

`<cluster>` 要素の構成で中心となるのは、次のサブ要素の指定です。

- `<replication-policy>` サブ要素では、レプリケーションの発生するタイミングと、レプリケートするデータを制御します。
- `<protocol>` サブ要素では、`multicast`（デフォルト）、`peer-to-peer` または `database` のうち、レプリケーションに使用するメカニズムを定義します。

次に、OC4J クラスタリングおよび JNDI 状態レプリケーション構成の 2 つの例を示します。

例 1:

この例では、マルチキャスト・ネットワークを使用して通信するようアプリケーションを構成します。クラスタ内のすべてのノードで同じマルチキャスト・ポートを使用することが重要です。

```
<cluster>
    <protocol>
        <multicast ip="230.230.0.30" port="45678" />
    </protocol>
</cluster>
```

例 2:

この例では、JNDI 状態レプリケーションに peer-to-peer プロトコルを使用します。この例の場合、クラスタで稼働するすべてのノードが、管理対象 *i*AS インスタンスの一部である必要があります。この Oracle Application Server インスタンスは、OPMN によって制御されます。

```
<cluster>
    <replication-policy trigger="onRequestEnd" scope="modifiedAttributes" />
    <protocol>
        <peer>
            <opmn-discovery/>
        </peer>
    </protocol>
</cluster>
```

OC4J クラスタリングの詳細は、『Oracle Containers for J2EE 構成および管理ガイド』を参照してください。

JNDI 状態レプリケーションの制限事項

JNDI 状態レプリケーションに依存する場合は、次の制限事項を考慮してください。

- クラスタ全体への変更の伝播
- リモート・オブジェクトのバインド

クラスタ全体への変更の伝播

シリアライズできない値へのバインドは、クラスタ全体に伝播しません。

リモート・オブジェクトのバインド

アプリケーション・コンテキスト内でリモート・オブジェクト（通常は `home` または `EJB` オブジェクト）をバインドすると、その JNDI オブジェクトはクラスタ全体で共有されますが、バインドされている最初のサーバーにシングル・ポイント障害となる部分が発生します。

3

OC4J トランザクション・サポート

この章では、Oracle Containers for J2EE（OC4J）のトランザクション・サポートについて説明します。この章には、次の項目が含まれます。

- 新機能
- OC4J トランザクション・サポートの概要
- プログラミング・モデル : コンテナ管理および Bean 管理のトランザクション
- トランザクション・コーディネータの構成
- OC4J トランザクション・マネージャの管理
- ORMI を通じた OC4J プロセス間でのトランザクション伝播

トランザクション管理タスク

この章では、次のトランザクション管理タスクについて説明します。

- トランザクション・コーディネータの構成
- データベース内トランザクション・コーディネータの考慮事項
- トランザクションの境界設定の管理（「トランザクションの境界設定」を参照）
- 手動によるコミットおよびロールバック操作
- OC4J トランザクション・マネージャの監視

新機能

次の OC4J トランザクション・サポートの機能および動作は、今回のリリースの新機能です。

- 中間層 2 フェーズ・コミット（2PC）コーディネータ : このコーディネータにより、Oracle リソースだけでなく、すべての XA 互換リソースがサポートされます。この機能は、異機種間中間層コーディネータと呼ばれます。
- 新規構成
- トランザクション管理
- ORMI を通じた OC4J プロセス間でのトランザクション伝播
- J2CA 1.5 のトランザクション・インフローのサポート
- **廃止予定**

  次の項目は、今回のリリースで廃止予定であり、将来的にはサポートされなくなります。

  – データベース内トランザクション・コーディネータの使用は、OC4J では推奨されていません。今後は、中間層トランザクション・コーディネータを使用することをお薦めします。詳細は、3-15 ページの「データベース内トランザクション・コーディネータの考慮事項」を参照してください。

- **サポート中止**

  次の項目は、今回のリリースではサポートされません。

  - `server.xml` での `transaction-timeout` の使用。`transaction-manager.xml` に移動しました。

本番環境で OC4J を使用する前に

本番環境で OC4J を使用する前に、次のタスクを実行することをお薦めします。

- 中間層コーディネータを構成して、永続ストアを使用します。永続ストア（およびロギング）は、デフォルトでは無効化されています。3-10 ページの「トランザクション・コーディネータの構成」を参照してください。
- デフォルトのトランザクション・リカバリ・パスワードを変更します。3-27 ページの「リカバリ」を参照してください。

追加ドキュメント

次のサイトにある How-To ドキュメントでは、OC4J の機能に関する情報（各機能の概要や機能に関連するコードの抜粋など）を提供しています。

`http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html`

OC4J トランザクション・サポートの概要

Java Transaction API（JTA）は、J2EE 環境でグローバル（分散）トランザクションをサポートするために Sun 社が開発した仕様です。グローバル・トランザクションでは、データベースやメッセージ・キューなどの複数のエンタープライズ・システムが単一の作業ユニットに統合されます。

JTA により、Open Group の分散トランザクション処理モデルに基づく仕様が Java 環境にマップされます。Open Group の XA 仕様は、トランザクション・マネージャとリソース・マネージャ間のインタフェースを定義しています。JTA 仕様は、アプリケーションとトランザクション・マネージャ間のインタフェースを定義しています。

JTA 仕様は、`http://java.sun.com/products/jta` で入手できます。

トランザクションとは

トランザクションは、状態の変化するシステムにおいて正確な出力結果を保証するためのメカニズムです。プログラマは、トランザクションを使用することで、システム内の多くの状態変化を単一の作業ユニットとして把握できます。JTA トランザクションは、リソースの登録およびトランザクション境界設定で構成されます。トランザクション・スコープ内で発生した変更は、コミットまたはロールバックできます。

一般的に、トランザクションは、アプリケーションによって開始されます。アプリケーションは、共有リソース（1 つ以上のデータベース・システムなど）に対してなんらかの作業を実行し、そのトランザクションをコミットします。コンテナ管理のトランザクションの場合、トランザクションはアプリケーション・サーバーによって開始されます。

J2EE でのトランザクション処理の詳細は、次の書籍を参照してください。

Little、Maron、Pavlik 著『Java Transaction Processing: Design and Implementation』（Prentice Hall、2004）

ACID

J2EE および多くの主要な情報管理システムによってサポートされるトランザクション・モデルは、次に示す ACID 特性を保持するよう設計されているのが普通です。

- **原子性 :** トランザクション・スコープ内の変更は、すべてコミットされるかすべてロールバックされるかのいずれかです。
- **一貫性 :** システムは、整合性のある状態から整合性のある状態に移行します。
- **独立性 :** トランザクションの結果は、そのトランザクションがコミットされるまで外部から参照できません。
- **永続性 :** トランザクション・スコープ内の状態変更は、永続化されます。

システムのインフラストラクチャ自体では、これらすべての特性を保証できないことに注意してください。一貫性の特性を確保するには、整合性のある変更をシステムに加えるアプリケーション・ロジックが必要です。たとえば、人事管理データベースに人ではなく惑星を表す識別子を挿入するトランザクション・ロジックがあるとすると、一貫性のない結果が出力されます (もっとも、アプリケーション・サーバーとデータベースで惑星名が意味のない値であると判別する方法はまずありませんが)。

ACID を確保することで、インフラストラクチャとアプリケーションに負荷がかかることにも注意してください。ACID のオーバーヘッドを抑える必要がある場合は、できるかぎり単一のローカル・リソースのみを使用し、2 フェーズ・コミットを使用しないように作業を設計します。

中間層 2 フェーズ・コミット（2PC）コーディネータ

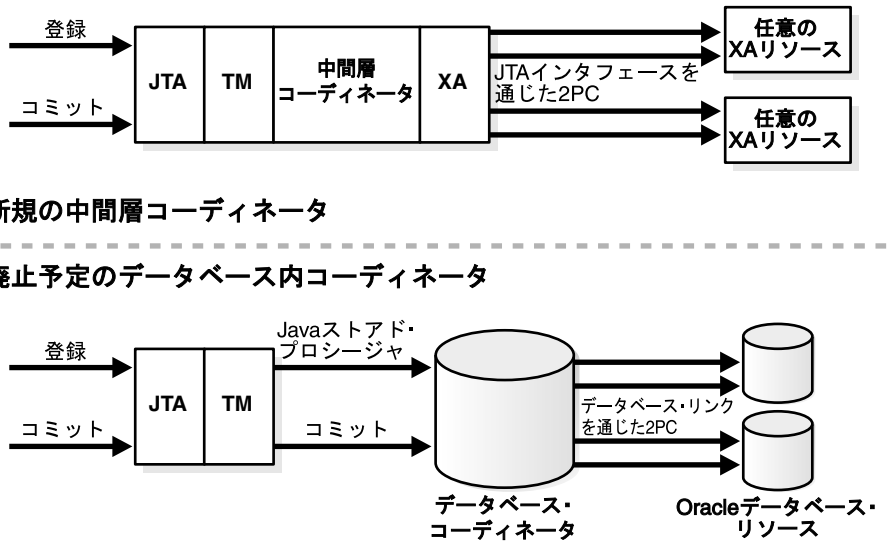
今回のリリースでは、トランザクション調整機能は OC4J に組み込まれています。置き換えられたデータベース内の調整機能は、廃止予定となっています。また、中間層コーディネータは、異機種間環境対応となり、Oracle リソースだけでなくすべての XA 互換リソースをサポートします。

中間層コーディネータの機能は、次のとおりです。

- 任意の XA 準拠リソースのサポート
- インターポジショニングとトランザクション・インフローのサポート
- 最終リソース・コミットによる最適化
- リカバリ・ロギング

図 3-1「新規の中間層コーディネータと廃止予定のデータベース内コーディネータの比較」に、新規の中間層コーディネータと廃止予定のデータベース内コーディネータの違いを示します。

図 3-1 新規の中間層コーディネータと廃止予定のデータベース内コーディネータの比較



ローカルおよびグローバル・トランザクション

トランザクションの複雑さは、アプリケーションがトランザクション内に登録するリソースの数によって決まります。

ローカル・トランザクションには、1 つのリソースのみが含まれます。トランザクション・アクティビティのスコープ設定と調整は、リソース自体に対してローカルに実行されます。ローカル・トランザクションでは、1 フェーズ・コミット（1pc）プロトコルが使用されます。

グローバル・トランザクション（分散トランザクション）では、トランザクション内に複数のリソースが登録されます。たとえば、グローバル・トランザクションは、2 つのデータベースを作業範囲とする場合に使用できます。グローバル・トランザクションをサポートするトランザクション処理システムには、通常、複数の異なる論理コンポーネント（トランザクション・ファクトリ、リソース・マネージャ、コーディネータ、アプリケーションなど）が含まれます。グローバル・トランザクションでは、結果の原子性を保証するため、2 フェーズ・コミット（2pc）プロトコルという終了プロトコルが使用されます。

2 フェーズ・コミット・プロトコル

2 フェーズ・コミット（2pc）プロトコルは、グローバル・トランザクションの複数の参加者が同意に至るためのメカニズムです。このプロトコルは、その名前が示すとおり、2 つのフェーズに分けられます。最初のフェーズは、通常、投票フェーズと呼ばれます。各参加者は、トランザクション作業をコミットする準備ができているかどうかを尋ねられます。作業をコミットできる参加者は、コミット可能であるという投票メッセージをコーディネータに送信します。

投票の結果、トランザクションのコミット準備ができていない参加者が 1 つでも存在する場合は、すべての参加者がロールバックの指示を受けます。

ACID 特性を保証するため、一部のトランザクション・システムでは、2 フェーズ・コミット・プロトコルと 2 フェーズ・ロック・プロトコルを組み合せています。具体的には、準備フェーズが開始すると、トランザクション内で作業を実行できなくなります。

2 フェーズ・コミット・プロトコルでは、システム障害時に結果の原子性を保証するため、トランザクション・マネージャでトランザクションの進捗状況を記録する必要があります。

複数のリソースの使用方法

各リソースは、トランザクションのスコープ内で使用されるときに、アプリケーション・サーバーによって自動的に登録されます。ただし、2 フェーズ・コミット・トランザクションで ACID 特性を保証するには、参加するすべてのリソースが XA に準拠している必要があります（最終リソース・コミットによる最適化の場合を除く）。

最終リソース・コミット

最終リソース・コミットでは、XA に準拠しない単一のリソースが XA トランザクションに参加できます。XA に準拠しない複数のリソースをトランザクションに登録すると、登録試行時に例外がスローされます。

準備フェーズでは、すべての XA 準拠リソースが準備を行います。すべての XA リソースが準備 OK の通知を戻すと、コーディネータは、XA に準拠しないリソースに制御を移します。XA に準拠しないリソースがコミット終了の通知を戻した場合、コーディネータはコミット決定をログに記録し、そうでない場合はロールバック決定を記録します。次に、コーディネータは、すべての XA リソースにその決定を通知します。

注意： 最終リソース・コミットによる最適化は、ほとんどの場合に適切に動作しますが、この最適化を使用する場合には若干のリスクが伴います。XA に準拠しないリソースへの制御の移行中に障害が発生した場合、コーディネータには、そのリソースがコミットされたかロールバックされたかを認識する方法がありません。これにより、結果の原子性が失われ、調整が非常に困難となる可能性があります。

最終リソース・コミットによる最適化では、単一の 1 フェーズ・コミット・リソースのみが特定のトランザクションへの登録を許可されるため、OC4J では、1 フェーズ・コミット・リソースの登録をルート OC4J プロセスに制限しています。1 フェース・コミット・リソースを OC4J の下位プロセスで登録しようとすると、サーバーがリソースを登録できなかったという例外がスローされます。この制限が必要とされる理由は、最終リソース・コミットによる最適化を使用する場合、1 フェーズ・コミット・リソースの登録を下位プロセスに許可すると、グローバル・トランザクション・ツリーに登録されている 1 フェーズ・コミット・リソースが 1 つのみであることを保証する処理でオーバーヘッドが発生するためです。

注意：

- トランザクション・ロギングを none に設定すると、XA に準拠しない複数のリソースをグローバル・トランザクションに登録できますが、この場合 ACID 特性やリカバリ可能性は保証されません。トランザクション・ロギングの設定方法は、3-10 ページの「トランザクション・コーディネータの構成」を参照してください。
- トランザクション・マネージャで永続ストアを使用しない場合、下位ノードでの登録が許可されます。

最終リソース・コミットを使用するには、トランザクション・マネージャで永続ストアを構成してトランザクション・ロギングを有効化します。永続ストア（およびロギング）は、デフォルトでは無効化されています。3-10 ページの「トランザクション・コーディネータの構成」を参照してください。

単一の非 XA リソースによるグローバル・トランザクションへの参加が可能になると同時に、最終リソース・コミットは、最適化の手段としても使用できます。XA 準拠リソースを非 XA リソースとして登録し、最終リソース・コミットを使用すると、パフォーマンスが向上します。これは、リソースでロギングを実行する必要がなくなり、コーディネータのリソースに対するコールが 2 つではなく 1 つのみとなってネットワーク・コールが減少するためです。また、リソースが不明（インダウト）状態とならないため、リソースのロックを防ぐことができます。ただし、最終リソース・コミットは、パフォーマンスの最適化に使用できますが、かわりに正確さの保証は失われます。

OC4J の最終リソース・コミット機能の詳細は、『Oracle Containers for J2EE リソース・アダプタ管理者ガイド』の「トランザクション管理」の章を参照してください。

リソース・マネージャ

リソース・マネージャは、共有リソースの管理を担当します。リソース・マネージャの一般的な例は、リレーショナル・データベース・システムです。

トランザクション・マネージャ

トランザクション・マネージャは、トランザクション・ファクトリとコーディネータの両方の役割を担います。アプリケーションは、トランザクション・マネージャと通信して、トランザクションの開始と終了およびリソースの登録を行います。アプリケーションがトランザクションのコミットをリクエストすると、トランザクション・マネージャは、2 フェーズ・コミット・プロトコルを調整します。

ヒューリスティック

全体の一致を必要とするため、2 フェーズ・コミットは、ブロック・プロトコルです。つまり、最終フェーズのメッセージを配信する前にコーディネータで障害が発生した場合、各参加者は、ブロックされた状態のままリソースを保持する必要があります。最新のトランザクション・システムでは、このような参加者がコミットまたはロールバックの実行を一方的に決定できるように、2 フェーズ・コミットにヒューリスティック（経験則）機能が追加されています。ある参加者の決定が、最終的に別の参加者の決定と一致しなかった場合、原子性に欠ける動作が発生します。通常、これらの決定は、管理的な操作により実行されます。詳細は、3-18 ページの「手動によるコミットおよびロールバック操作」を参照してください。

Synchronization

Synchronization は、トランザクションに登録され、2 フェーズ・コミット・プロトコルの実行前後に通知されるオブジェクトです。たとえば、アプリケーションは、独自の状態を保持しており、必要に応じて（またはトランザクション完了前に）キャッシュをリソースに永続化して、完了後はリソースを解放するのが普通です。

Synchronization は、`UserTransaction` インタフェースからは使用できないため、アプリケーションでは Synchronization を直接登録できません。この操作は、アプリケーション・サーバーでのみ実行できます。CMP の場合は、永続性レイヤーがこの操作を処理します。

EJB の詳細は、『Oracle Containers for J2EE Enterprise JavaBeans 開発者ガイド』を参照してください。

プログラミング・モデル : コンテナ管理および Bean 管理のトランザクション

Enterprise JavaBeans では、コンテナ管理のトランザクションまたは Bean 管理のトランザクションを通じたトランザクション管理に、JTA 1.0.1B を使用します。

注意 : コンテナ管理のトランザクションは、宣言型のトランザクションまたは CMT とも呼ばれます。

Bean 管理のトランザクションは、プログラム的なトランザクションまたは BMT とも呼ばれます。

コンテナ管理のトランザクション

コンテナ管理のトランザクションは、コンテナによって制御されます。つまり、コンテナは、デプロイメント・ディスクリプタ内の定義に従って、既存のトランザクションと結合するか、アプリケーション用に新しいトランザクションを起動します。新規に作成されたトランザクションは、Bean メソッドが完了すると終了します。トランザクションを管理するために、実装でコードを提供する必要はありません。

Bean 管理のトランザクション

Bean 管理のトランザクションは、Bean 実装内でプログラムによって境界設定されます。トランザクション境界は、アプリケーションによって完全に制御されます。

トランザクションの境界設定

トランザクションの境界設定とは、トランザクションの開始と終了の区切りを意味します。

トランザクション境界をユーザー自身が設定するには、Bean を Bean 管理のトランザクションとして指定します。トランザクション境界の設定をコンテナに任せるには、EJB デプロイメント・ディスクリプタにトランザクション属性を設定して、Bean をコンテナ管理のトランザクションとして指定します。

コンテナ管理のトランザクションは、すべての EJB で使用できます。ただし、Bean 管理のトランザクションは、セッション Bean とメッセージドリブン Bean（MDB）でのみ使用できます。つまり、データ・アクセス用に設計されているエンティティ Bean では、コンテナ管理のトランザクションによる境界設定を使用する必要があります。セッション Bean では、どちらのモデルも使用できます。

注意 : トランザクション境界は、アプリケーション・クライアント・コンテナからは設定できません。

境界設定のタイプは、Bean のデプロイメント・ディスクリプタに指定します。次の例に、`<transaction-type>` 要素に `Container` と指定して、セッション Bean をコンテナ管理のトランザクションとして宣言する方法を示します。Bean を構成して Bean 管理のトランザクション境界を設定するには、`<transaction-type>` 要素に `Bean` と指定します。

例 : コンテナ管理のトランザクションとして宣言されるセッション Bean

```
</session>
  <description>no description</description>
  <ejb-name>myEmployee</ejb-name>
  <home>cmtxn.ejb.EmployeeHome</home>
  <remote>cmtxn.ejb.Employee</remote>
  <ejb-class>cmtxn.ejb.EmployeeBean</ejb-class>
  <session-type>Stateful</session-type>
  <transaction-type>Container</transaction-type>
  <resource-ref>
   <res-ref-name>jdbc/OracleMappedDS</res-ref-name>
```

```
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Application</res-auth>
   </resource-ref>
</session>
```

コンテナ管理のトランザクションの境界設定

コンテナ管理のトランザクション（CMT）を使用するように Bean を定義する場合、後述の例に示すとおり、コンテナによる Bean の JTA トランザクションの管理方法を、Bean のデプロイメント・ディスクリプタの <trans-attribute> 要素に指定する必要があります。次の表に、トランザクション属性の設定とその説明を示します。

表 3-1 <trans-attribute> 要素の設定

設定	説明
NotSupported	Bean はトランザクションに関連しません。 Bean の実行者が、トランザクションに関連しているときに Bean をコールすると、実行者のトランザクションが一時停止され、Bean が実行されます。Bean が戻ると、実行者のトランザクションが再開されます。 メッセージドリブン Bean（MDB）の場合、この設定がデフォルトです。
Required	Bean はトランザクションに関連している必要があります。 実行者がトランザクションに関連している場合は、実行者のトランザクションが使用されます。 実行者がトランザクションに関連していない場合は、コンテナによってその Bean 用の新規トランザクションが開始されます。これはデフォルト設定です。 CMP 2.0 エンティティ Bean の場合、この設定がデフォルトです。
Supports	実行者が関連しているトランザクションの状態にかかわらず、そのトランザクションが Bean に使用されます。 実行者がトランザクションを開始している場合は、実行者のトランザクション・コンテキストが使用されます。 実行者がトランザクションに関連していない場合は、Bean も関連しません。 これは、CMP 2.0 エンティティ Bean と MDB を除くすべてのエンティティ Bean のデフォルト設定です。
RequiresNew	実行者がトランザクションに関連しているかどうかに関係なく、その Bean に対してのみ存在する新規トランザクションが開始されます。 実行者が、トランザクションに関連しているときに Bean をコールすると、実行者のトランザクションは Bean が完了するまで一時停止されます。
Mandatory	この Bean を起動する前に、実行者がトランザクションに関連している必要があります。Bean には、実行者のトランザクション・コンテキストが使用されます。
Never	Bean はトランザクションに関連しません。さらに、Bean コール中、実行者がトランザクションに関連することはありません。 実行者がトランザクションに関連している場合は、RemoteException がスローされます。

注意 : 各タイプのエンティティ Bean に対応するデフォルトの <trans-attribute> 設定は、次のとおりです。

- CMP 2.0 のエンティティ Bean の場合、デフォルトは Required です。
- MDB の場合、デフォルトは NotSupported です。
- 他のすべてのエンティティ Bean の場合、デフォルトは Supports です。

次の例に、EJB デプロイメント・ディスクリプタの <container-transaction> 部分を示します。この例では、myEmployee EJB のすべての（*）メソッドに対して RequiresNew トランザクション属性を指定しています。

例 : デプロイメント・ディスクリプタの <container-transaction>

```
<assembly-descriptor>
   <container-transaction>
      <description>no description</description>
      <method>
         <ejb-name>myEmployee</ejb-name>
         <method-name>*</method-name>
      </method>
     <trans-attribute>RequiresNew</trans-attribute>
   </container-transaction>
</assembly-descriptor>
```

トランザクションの開始、コミットまたはロールバックに Bean 実装は必要ありません。これらの操作は、デプロイメント・ディスクリプタに指定したトランザクション属性に基づいて、コンテナによって処理されます。

Bean 管理のトランザクションの境界設定

Web コンポーネント（JSP およびサーブレット）と、ステートレスおよびステートフル・セッション Bean では、プログラムによるトランザクション境界設定を使用できます。エンティティ Bean では、この方法は使用できないため、コンテナ管理によるトランザクション境界設定を使用する必要があります。

注意 : クライアント・サイドのトランザクション境界設定は、アプリケーション・クライアント・コンテナではサポートされません。OC4J では、クライアント・サイドのトランザクション境界設定はサポートされません。この形式のトランザクション境界設定は、J2EE 仕様の要件となっておらず、パフォーマンスと待機時間にも影響があるため、お薦めしません。

デプロイメント・ディスクリプタの <transaction-type> 要素内で Bean を Bean 管理のトランザクション（BMT）として宣言する場合、グローバル・トランザクションの開始、コミットまたはロールバックの境界は、Bean 実装によって設定する必要があります。

Bean 管理の（プログラムによる）トランザクション境界設定を使用する場合、Bean 開発者は、UserTransaction インタフェースを使用してグローバル・トランザクション境界を設定するか、RM 固有のメソッドを使用して RM のローカル・トランザクション境界を設定することが可能です。

Web コンポーネントまたは Bean の開発者は、次の例に示すとおり、UserTransaction インタフェースの begin()、commit() および rollback() メソッドを明示的に発行する必要があります。

```
Context initCtx = new Initial Context();
ut = (UserTransaction)  initCtx.lookup("java:comp/UserTransaction");
```

```
ut.begin();
// Do work.
…
try {
    // Commit the transaction.
    ut.commit();
// Handle exceptions.  Should really catch specific exceptions, not Exception
} catch (Exception ex) { … }
```

初期コンテキストを作成するかわりに、@Resource 注釈を使用してリソース・インジェクションをリクエストすることも可能です。そのような場合、ルックアップは実行時に自動的に行われます。次に例を示します。

```
@Resource

begin();
// Do work.
…
try {
    // Commit the transaction.

   commit();

// Handle exceptions. Should really catch specific exceptions, not Exception
} catch (Exception ex) { … }
```

トランザクション・コーディネータの構成

この項では、2PC トランザクション・コーディネータの構成方法を説明します。この項には、次の項目が含まれます。

- サポート・リソースの構成
- データベース内トランザクション・コーディネータの考慮事項

トランザクション・コーディネータは、Oracle Enterprise Manager 10*g* Application Server Control コンソールまたはトランザクション・マネージャの構成ファイル（J2EE_HOME/config/transaction-manager.xml）を使用して構成されます。この項では、コンソールを使用する方法を主に説明します。

トランザクション・コーディネータの定義

トランザクション・コーディネータは、トランザクション・マネージャ・タスクまたは JTAResource MBean を使用して定義できます。どちらも Application Server Control コンソールの「管理」タブからアクセスできます。

トランザクション・コーディネータの構成時には、変更内容を反映するためにサーバーを再起動する必要があります。また、停止前にすべてのトランクションをパージする必要があります。パージされていないトランザクションがある場合、サーバーは、それが管理者により解決されるまで待機します。トランザクション中にサーバーが強制的に停止され、インダウト状態が続いている場合、構成の変更は保存されません。

トランザクション・マネージャ・タスク

トランザクション・マネージャ・タスクを使用してトランザクション・コーディネータを構成するには、次のようにします。

1. Application Server Control コンソールにログインし、**「OC4J: ホーム」→「管理」→「タスク名 : サービス」→「トランザクション・マネージャ (JTA)」**→「タスクに移動」に移動します。
2. **「管理」**をクリックします。
3. 表 3-2 の説明を参照してコーディネータのフィールドを入力します。
4. **「適用」**をクリックします。
5. OC4J を再起動します。

JTAResource MBean

JTAResource MBean を使用してトランザクション・コーディネータを構成するには、次のようにします。

1. Application Server Control コンソールにログインし、**「OC4J: ホーム」→「管理」→「タスク名 : JMX」→「システム MBean ブラウザ : タスクに移動」→「JTAResource」→「oc4j-tm」**に移動します。
2. **「操作」**をクリックします。
3. 操作のリストから、**「configureCoordinator」**をクリックします。「操作」画面が表示されます。
4. 表 3-2 の説明を参照してフィールドに入力します。
5. **「起動操作」**をクリックします。
6. OC4J を再起動します。

表 3-2 トランザクション・コーディネータのパラメータ

パラメータ	説明
commitCoordinator	2 フェーズ・コミット・コーディネータのタイプ（database または middle-tier）。 データベース内コーディネータの使用は、OC4J では推奨されていません。今後は、中間層コーディネータを使用することをお薦めします。詳細は、3-15 ページの「データベース内トランザクション・コーディネータの考慮事項」を参照してください。
logType	ログ・タイプ（none、file または database）。この設定は、commitCoordinator が middle-tier の場合にのみ適用されます。
logLocation	ログの場所。ファイルの場合はディレクトリ・パスを、データベース・ロギングの場合は JNDI ロケーションを指定します。
userName	データベース・ユーザー名。
password	データベース・パスワード。
retryCount	コーディネータがリソース・マネージャに対して同期的にコマンドを再試行する回数（有効な場合）。再試行は、たとえばリソース・マネージャが XA_RETRY を戻した場合に発生する可能性があります。

コーディネータ属性の設定

トランザクション・コーディネータには、実行時のパフォーマンスの制御に使用される属性が複数含まれています。属性への変更はすぐに反映されるため、サーバーの再起動は不要です。

汎用属性

トランザクション・コーディネータには、トランザクション・マネージャ・タスクまたは JTAResource MBean を使用して定義できる属性が 2 つあります。表 3-3 で、汎用属性を説明します。

表 3-3 汎用属性

属性	説明
transactionTimeout	デフォルトのトランザクション・タイムアウト（秒単位）。
maxConcurrentTransactions	システム例外をスローせずに実行できるサーバー内の同時トランザクションの最大数。（-1 は無制限であることを示します。）

ファイル・ストア・ログの属性

ファイル・ストア・ログの属性は、トランザクション・コーディネータのファイル・ストア・ロギングを使用している際に、ファイル・ストアのパフォーマンスの制御に使用されます。属性を変更するには、JTAResource MBean を使用するか、transaction-manager.xml ファイルを手動で編集する必要があります。表 3-4 で、ファイル・ストア・ログの属性を説明します。

表 3-4 ファイル・ストア・ロギングの属性

属性	説明
minPoolSize	起動時にプールに割り当てるファイルの数。デフォルトは 40 です。最適値は、同時リクエストの最大数に十分対応できる数です。
maxOpenFiles	オープン状態またはアクティブ状態にできるファイル・チャネルの最大数。この数を超えると、XID が再度リクエストされるまで、最も古いチャネルから順に解放されます。デフォルトは 256 です。最適値は、同時リクエストの最大数に十分対応できる数です。
oldFileReleaseSize	maxOpenFiles を超えたときにクローズする最も古いファイル・ハンドルの数。デフォルトは 20 です。最適値は、maxOpenFiles を超える確率に応じて変化します。この最大数を超えることはできるかぎり避け、超えたとしてもその数を最小限に維持する必要があります。

データベース・ストアの属性

データベース・ストア・ログの属性は、トランザクション・コーディネータのデータベース・ストア・ロギングを使用している際に、データベース・ストアのパフォーマンスの制御に使用されます。属性を変更するには、JTAResource MBean を使用するか、transaction-manager.xml ファイルを手動で編集する必要があります。表 3-4 で、ファイル・ストア・ログの属性を説明します。

中間層コーディネータのデータベース・ロギングに関する注意 :

- データベースは Oracle である必要があります。
- 次の場所にあるスキーマをデータベースにインストールする必要があります。

  ORACLE_HOME/j2ee/home/database/j2ee/jta/oracle/2pc_jdbcstore.sql

- ロギングに使用するデータソースは、マネージド・データソースではなくネイティブ・データソースである必要があります。
- ロギングに使用するデータソースは、デフォルト・アプリケーションにデプロイする必要があります。

表 3-5 データベース・ストア・ロギングの属性

属性	説明
batchCreateInterval	トランザクションのバッチ書込みを実行する間隔（ミリ秒単位）。デフォルトは 10 です。最適値は小さい値ですが、データベース・コール（ネットワーク待機時間）のコストや JVM のヒープ・サイズに影響します。
batchStateInterval	トランザクションの状態変更のバッチ書込みを実行する間隔（ミリ秒単位）。デフォルトは 10 です。最適値は小さい値ですが、データベース・コール（ネットワーク待機時間）のコストや JVM のヒープ・サイズに影響します。
batchPurgeInterval	完了済トランザクションのバッチ・パージを実行する間隔（ミリ秒単位）。デフォルトは 100 です。最適値は大きい値ですが、データベース・コール（ネットワーク待機時間）のコストや JVM のヒープ・サイズに影響します。

サポート・リソースの構成

トランザクション・コーディネータでは、実行時の設定に応じてリソースに関する追加の構成が必要な場合があります。

サーバー

トランザクション・マネージャを構成するには、J2EE_HOME/config/server.xml ファイルに次の要素を含める必要があります。

```
<transaction-manager-config path="./transaction-manager.xml"/>
```

> **注意：** transaction-timeout 属性などのすべてのトランザクション・マネージャ構成が、server.xml ファイルではなく transaction-manager.xml ファイルに含まれます。

server.xml のリファレンス・ドキュメントは、『Oracle Containers for J2EE 構成および管理ガイド』の付録 B「OC4J で使用される構成ファイル」の「OC4J サーバー構成ファイル（server.xml）の概要」を参照してください。

データソース

実行時ユーザーに XAResource.recover 権限を付与しない場合、J2EE_HOME/config/data-sources.xml ファイルを使用して、データソースが 2pc トランザクションに参加し、リカバリするために必要なリカバリ情報を指定します。

```
//
<connection-pool
 name="cmt Connection Pool">
 min-connections="10"
 max-connections="30"
 inactivity-timeout="30"
 <connection-factory   factory-class="oracle.jdbc.xa.client.OracleXADataSource"
   user="scott"
   password="tiger"
   url="jdbc:oracle:thin:@localhost:1521:ORCL">

 <xa-recovery-config>
   <password-credential>
   <username>system</username>
   <password>manager</password>
   </password-credential>
 </xa-recovery-config>
 </connection-factory>
 </connection-pool>

<managed-data-source connection-pool-name="cmt Connection Pool"
           jndi-name="jdbc/cmt"
           name="cmt"/>
```

> **注意：** OC4J グローバル・トランザクションに参加するためには、データソースをマネージド・データソースとして指定する必要があります。

リソース・アダプタ

oc4j-ra.xml ファイルでは、JMS コネクタや他の任意のコネクタが 2pc トランザクションに参加し、リカバリするために必要な XA 情報とリカバリ情報を指定します。

2 フェーズ・コミット処理に参加するためには、XAConnectionFactory を使用する必要があります。また、必要な XAResource.recover コールを実行するパーミッションを実行時ユーザーに付与しない場合は、connection-factory で xa-recovery-config 要素を定義する必要があります。

詳細は、『Oracle Containers for J2EE リソース・アダプタ管理者ガイド』の「トランザクション管理」の章と付録の「OC4J リソース・アダプタ構成ファイル」を参照してください。

xa-recovery-config 要素の形式は、次のとおりです。

```
 <connection-factory>
...
 <xa-recovery-config>
   <password-credential>
   <username>adapter_admin_user</username>
   <password>adapter_admin_pw</password>
   </password-credential>
 </xa-recovery-config>
...
 </connection-factory>
```

次のリストに、ベンダー固有のパーミッションの例を示します。RDBMS のリソース・マネージャに関連するパーミッションは、data-sources.xml に指定します。

- Oracle: DBA_PENDING_TRANSACTIONS に対する select 権限と sys.dbms システムに対する execute 権限（リリース 9.2 以上の RDBMS）
- DB2: sysadmin ロール
- MSSQL: XA ストアド・プロシージャに対する execute 権限
- MQSeries: sysadmin ロール

データベース内トランザクション・コーディネータの考慮事項

この項では、データベース内の 2 フェーズ・コミット・コーディネータを使用するための要件について説明します。

注意： データベース内 2 フェーズ・コミット・コーディネータの使用は、OC4J では推奨されていません。今後は、中間層コーディネータを使用することをお薦めします。

データベース内の 2 フェーズ・コミット・エンジンは、次の項目で構成されます。

- 調整を担当するデータベースから、トランザクションに関連する各データベースへと向かう完全修飾データベース・リンク。トランザクションの終了時に、2 フェーズ・コミット・エンジンは、完全修飾データベース・リンクを介して関連するデータベースと通信します。
- 関連する各データベースに対するセッションの作成と、コミットまたはロールバックを実行する責任が与えられたユーザー。通信を実行するユーザーは、関連するすべてのデータベース上に作成され、適切な権限を与えられる必要があります。

注意： データベース内の 2 フェーズ・コミット（2pc）機能には、Oracle Database リリース 9.2.0.4 以上を使用してください。データベース内の 2 フェーズ・コミット（2pc）機能は、Oracle Database リリース 9.2 以下では使用できません。

データベース内コーディネータ構成に関する注意 :

- データベースは Oracle を使用する必要があります。
- コーディネータとして指定するデータソースは、マネージド・データソースではなくネイティブ・データソースである必要があります。
- データベース内コーディネータは、伝播されるトランザクション内では（ルート・コーディネータとしても介在コーディネータとしても）使用できません。また、最終リソース・コミットによる最適化も提供されません。
- コーディネータとして指定するデータソースは、デフォルト・アプリケーションにデプロイする必要があります。

適切な Oracle データベースを 2 フェーズ・コミット・エンジンとして指定および構成する手順は、次のとおりです。

1. Application Server Control コンソールの「JDBC リソース」ページまたは `data-sources.xml` ファイルでマネージド・データソースを定義します。

   次の例は、`data-sources.xml` ファイルでの定義です。

```
 <connection-pool
      name="OracleCommitDS">
      min-connections="10"
      max-connections="30"
      inactivity-timeout="30"
      <connection-factory
          factory-class="oracle.jdbc.xa.client.OracleXADataSource"
          user="scott"
          password="tiger"
          url="jdbc:oracle:thin:@localhost:1521:ORCL">
      </connection-factory>
  </connection-pool>
<managed-data-source connection-pool-name="Example Connection Pool"
                        jndi-name="jdbc/OracleCommitDS"
                        name="OracleCommitDS"/>
```

2. `transaction-manager.xml` で次のように 2 フェーズ・コミット・エンジンのデータソースを参照します。

```
       <database location="jdbc/OracleCommitDS">
           <identity user-name="COORDUSR" password="COORDPW"/>
       </database>
```

   `identity` 要素は、マネージド・データソースで指定した `user` と `password` を上書きする必要がある場合にのみ使用します。

3. ユーザーを作成し、トランザクションに関連するすべてのデータベースに対する適切なパーミッションを付与します。
   - トランザクションを調整する 2 フェーズ・コミット・エンジンにユーザーを作成します。
   - このユーザーは、2 フェーズ・コミット・エンジンから関連する各データベースへのセッションをオープンします。
   - 各データベースに接続するため、このユーザーに CONNECT、RESOURCE および CREATE SESSION 権限を付与します。FORCE ANY TRANSACTION 権限を付与すると、ユーザーによるトランザクションのコミットまたはロールバックが可能になります。

たとえば、トランザクションを調整するユーザーが COORDUSR の場合、2 フェーズ・コミット・エンジンとトランザクションに関連する各データベースに対する操作は、次のようになります。

```
CONNECT SYSTEM/MANAGER;
CREATE USER COORDUSR IDENTIFIED BY COORDUSR;
GRANT CONNECT, RESOURCE, CREATE SESSION TO COORDUSR;
GRANT FORCE ANY TRANSACTION TO COORDUSR;
```

4. CREATE PUBLIC DATABASE LINK コマンドを使用して、2 フェーズ・コミット・エンジンからグローバル・トランザクションに関連する各データベースへと向かう完全修飾のパブリック・データベース・リンクを構成します。これらのリンクは、トランザクション終了時に、2 フェーズ・コミット・エンジンが各データベースと通信するために必要です。これらのリンクにより、COORDUSR は、関連するすべてのデータベースに接続できます。

5. トランザクションに参加するマネージド・データソースごとに、2 フェーズ・コミット・エンジンからそのデータベースへと向かう完全修飾データベース・リンクのプロパティを追加します。

   Application Server Control コンソールの「接続プール」ページまたは data-sources.xml ファイルでプロパティを追加します。

   次の例は、data-sources.xml ファイルでの定義です。

```
 <connection-pool
      name="OracleDS1">
      min-connections="10"
      max-connections="30"
      inactivity-timeout="30"
      <connection-factory
          factory-class="oracle.jdbc.xa.client.OracleXADataSource"
          user="scott"
          password="tiger"
          url="jdbc:oracle:thin:@host1:1521:ORCL">
       <property name="dblink"
 value="DBLINK1.REGRESS.RDBMS.DEV.US.OLE.COM"/>
          </connection-factory>
  </connection-pool>
<managed-data-source connection-pool-name="Example Connection Pool"
                        jndi-name="jdbc/OracleDS1"
                        name="OracleDS1"/>
 <connection-pool
      name="OracleDS2">
      min-connections="10"
      max-connections="30"
      inactivity-timeout="30"
      <connection-factory
          factory-class="oracle.jdbc.xa.client.OracleXADataSource"
          user="scott"
          password="tiger"
          url="jdbc:oracle:thin:@host2:1521:ORCL">
       <property name="dblink"
 value="DBLINK2.REGRESS.RDBMS.DEV.US.OLE.COM"/>
      </connection-factory>
 </connection-pool>
<managed-data-source connection-pool-name="Example Connection Pool"
                        jndi-name="jdbc/OracleDS2"
                        name="OracleDS2"/>
```

OC4J トランザクション・マネージャの管理

この項では、実行時に可能となる次の操作について説明します。

- 手動によるコミットおよびロールバック操作
- OC4J トランザクション・マネージャの監視
- OC4J トランザクション・マネージャのリカバリの管理

手動によるコミットおよびロールバック操作

JTAResource MBean へのパス :

「OC4J: ホーム」→「管理」タブ→「タスク名 : JMX」→「システム MBean ブラウザ」→「タスクに移動」

Application Server Control コンソールを通じてアクセス可能な `JTAResource` MBean の次の操作は、現在のトランザクションの結果を強制的に決定する場合に使用できます。

操作	説明
`heuristicCommit`	インダウト・トランザクションに対する自律型のコミット決定
`heuristicRollback`	インダウト・トランザクションに対する自律型のロールバック決定
`setRollbackOnly`	アクティブなトランザクションに対する自律型の setRollbackOnly 決定

OC4J トランザクション・マネージャの監視

この項では、JTAResource MBean を通じて可能となる次の機能について説明します。

- OC4J トランザクション・サポート統計
- イベント通知のサブスクライブ

OC4J トランザクション・サポート統計

次の表にリストされた統計は、Application Server Control コンソールを通じてアクセスできる `JTAResource` MBean により提供されます。

統計	説明
`activeCount`	アクティブなトランザクションの合計数。 高い値が続く場合、サーバーの負荷が高くなっている可能性があります。 クラスタ全体のバランシングを検討してください。
`committedCount`	コミットされたトランザクションの合計数。 この値は、起動以降（または統計がリセットされてから）のサーバーの平均負荷を確定する際に使用できます。`activeCount` と同様、高い平均値が続く場合、サーバーの負荷が高くなっている可能性があります。 ロード・バランシングを検討してください。
`rolledbackCount`	ロールバックされたトランザクションの合計数。 値が高い場合、システムの問題（データベースの停止など）が原因でパフォーマンスが低下している可能性があります。 詳細なロールバックの原因数とロールバックの原因に関するログを調べてください。

統計	説明
rolledbackDueToTimed OutCount	タイムアウトが原因でロールバックされたトランザクションの合計数。 値が高い場合、トランザクション（またはトランザクション範囲内のアクティビティ）がタイムアウトになる原因となる、いくつかの問題がある可能性があります。指定したタイムアウト値の対応度が不十分であることも考えられます。 関連するトランザクション内で時間のかかっているアクティビティ（特定のタイプまたはアプリケーション）を特定するか、transaction-manager.xml 構成ファイルでトランザクションのタイムアウト値を調整してください。
rolledbackDueToApp Count	setRollbackOnly またはロールバックを明示的にコールするアプリケーションが原因でロールバックされたトランザクションの合計数。 値は様々な理由で高くなりますが、ほとんどの場合、アプリケーション内の複数の処理済の例外（データベースの更新時の SQLException など）が原因です。 setRollbackOnly またはロールバックをコールするアプリケーション・コードを調べて、原因を特定してください。
rolledbackDueTo ResourceCount	登録されたリソース内のエラーが原因でロールバックされたトランザクションの合計数。 値が高い場合、1 つ以上のリソース・マネージャ（データベースなど）、または OC4J とこれらのリソースとの間のネットワーク接続に問題が発生している可能性があります。 OC4J およびリソース・マネージャのログを調べてください。
rolledbackDueToAdmin Count	管理処理が原因でロールバックされたトランザクションの合計数。 値が高い場合、システムまたはアプリケーションの自動化が十分ではないこと（全体にシステム管理が多すぎる、またはトランザクション・アーキテクチャの処理が不適切であるなど）を示します。または大規模な管理を必要とする、特定の問題が発生している可能性もあります。 ログで傾向を分析し、管理処理の担当者に問い合せてください。
rollbackExceptionCount	発生した RollbackExceptions の合計数。 値が高い場合、システムの問題（データベースの停止など）が原因でパフォーマンスが低下している可能性があります。これは、直接的な内部システム障害、またはなんらかの理由によって setRollbackOnly をコールするアプリケーションが原因である可能性があります。 詳細なロールバックの原因数とロールバックの原因に関するログを調べ、setRollbackOnly をコールするアプリケーション・コードを検証してください。
heuristicMixed ExceptionCount	発生した HeuristicMixedExceptions の合計数。 値が高い場合、一貫性がないか不適切な管理処理により、非 ACID になりうる結果が多数発生している可能性があります。 ログで傾向を分析し、管理処理の担当者に問い合せてください。

統計	説明
heuristicRollback ExceptionCount	発生したヒューリスティックなロールバック例外の合計数。 値が高い場合、システムまたはアプリケーションの自動化が十分ではないこと（全体にシステム管理が多すぎる、またはトランザクション・アーキテクチャの処理が不適切であるなど）を示します。または大規模な管理を必要とする、問題が発生している可能性もあります。トランザクションがアクティブなときのルート・トランザクション・マネージャ・レベルにおける管理ロールバックを示す rolledbackDueToAdminCount とは異なり、この数値は下位のトランザクション・マネージャまたはリソース・マネージャが準備状態のときに管理ロールバックされていることを示します。 OC4J およびリソース・マネージャのログで傾向を分析し、管理処理の担当者に問い合せてください。
securityExceptionCount	発生した SecurityExceptions の合計数。 値が高い場合、特に値が 0 より大きい場合、これを実行するスレッドで ID に問題がある可能性があります。 OC4J ログを調べてください。
illegalStateException Count	発生した IllegalStateExceptions の合計数。 この値が高くなることはほとんどありません。高い場合は、前の管理処理が原因と考えられます。 OC4J ログで傾向を分析し、管理処理の担当者に問い合せてください。
systemExceptionCount	発生した SystemExceptions の合計数。 この値が高くなることはありません。高い場合は、システムで重大な障害が発生していることを示します。 OC4J およびリソース・マネージャのログを分析してください。
heuristicCommitted Count	ヒューリスティックにコミットされたトランザクションの合計数。 値が高い場合、システムまたはアプリケーションの自動化が十分ではないこと（全体にシステム管理が多すぎる、またはトランザクション・アーキテクチャの処理が不適切であるなど）を示します。または大規模な管理を必要とする、問題が発生している可能性もあります。これは下位のトランザクション・マネージャが原因であり、準備状態のときにリソース・マネージャが管理ロールバックされることは原因ではありません。 OC4J ログで傾向を分析し、管理処理の担当者に問い合せてください。
heuristicRolledback Count	ヒューリスティックにロールバックされたトランザクションの合計数。 値が高い場合、システムまたはアプリケーションの自動化が十分ではないこと（全体にシステム管理が多すぎる、またはトランザクション・アーキテクチャの処理が不適切であるなど）を示します。または大規模な管理を必要とする、問題が発生している可能性もあります。これは下位のトランザクション・マネージャが原因であり、準備状態のときにリソース・マネージャが管理ロールバックされることは原因ではありません。 OC4J ログで傾向を分析し、管理処理の担当者に問い合せてください。
heuristicCount	ヒューリスティックにロールバックおよびコミットされたすべてのトランザクションの合計数。 heuristicCommittedCount および heuristicRolledbackCount の説明を参照してください。

統計	説明
averageCommitTime	すべてのトランザクションの平均コミット時間。 これは、performTransaction 値の平均です。ただし、これは平均値であるため、システムで値が急上昇している可能性もあり、その場合は他の問題も発生しています。 システムで値が急上昇しているかどうか、詳細な統計およびログ・ファイルを分析してください。また、必要に応じてアーキテクチャ全体を分析してください。
performTransaction	トランザクションの開始から終了までの時間。 これは、パフォーマンスの問題に関する高レベルのインジケータとして役に立ちます。ただし、この値はトランザクションの開始から終了までの時間を測定しただけの値であるため、この時間内に発生したすべてのことがこの値に影響している可能性があります。候補として、アプリケーション・ロジック、データベース・アクティビティ、JMS アクティビティ、トランザクション処理などがあげられます。 必要に応じて、詳細な統計およびアーキテクチャ全体を分析してください。
singlePhaseCommit Completion	1 フェーズ・コミットの完了に必要な時間。 1 フェーズ・コミットには単一のリソースのみが含まれ、2PC コスト（ロギングなど）は含まれません。この値が高い場合は通常、コミットされる単一のリソースに問題（データベースに対するネットワーク待機時間など）があること示します。 コミットに含まれるリソースのメトリックを分析してください。
twoPhaseCommit Completion	2 フェーズ・コミットの完了に必要な時間。 値が高い場合、リソース・マネージャの準備およびコミットのコールまたは OC4J 内のトランザクション・レコードのロギングが遅れていることを示します。 transaction-manager.xml ファイルで、関連するリソースのメトリックおよび OC4J におけるトランザクション・レコードのロギングのパフォーマンス設定を分析してください。
transactionSuspended	トランザクションが一時停止している時間。 値が高い場合、トランザクションが一時停止状態であり、（通常 EJB メソッド・コールからの）別のコンテキストまたはトランザクション以外のコンテキストで、あるいはトランザクション・コンテキストの伝播中に、リターン・コールを待機していることを示します。 アプリケーションを分析し、一時停止中に発生しているアクティビティを特定するか、伝播中の場合はネットワーク待機時間があるかどうかを確認してください。
rollbackCompletion	ロールバックの完了に必要な時間。 値が高い場合、ネットワーク待機時間またはリソース・マネージャの問題の結果、リソース・マネージャのロールバック・コールが遅れていることを示します。 ロールバックに含まれるリソースのメトリックを分析してください。

次の表にリストされた属性は、JTAResource MBean により提供されます。

属性	説明
inDoubtXids	インダウト・トランザクション XID の配列
activeXids	アクティブ・トランザクション XID の配列
heuristicCommittedXids	ヒューリスティックにコミットされたトランザクション XID の配列
heuristicRolledbackXids	ヒューリスティックにロールバックされたトランザクション XID の配列
currentTransactionDetail	アクティブ、インダウト、リカバリ中を含む、サーバー上のすべての現行トランザクションの詳細

次の JTAResource MBean 操作は、統計に関連しています。

操作	説明
clearStats	activeCount 以外のすべての OC4J トランザクション・サポート統計を 0 にリセットします。
addThresholdEvent	指定された OC4J トランザクション件数統計が特定のしきい値を超えたときに起動され、指定された件数の間隔で繰り返されるイベントを追加します。 addThresholdEvent には次のパラメータがあります。 ■ statName: 監視対象とする件数統計の名前 ■ threshold1: イベントが配信されるときの件数 ■ repeatNotificationInterval: 後続のイベントが配信される件数の間隔
removeThresholdEvent	しきい値イベントを削除します。 removeThresholdEvent には次のパラメータがあります。 ■ statName: しきい値イベントが削除される件数統計の名前

イベント通知のサブスクライブ

JTAResource MBean では、MBean の「通知」タブに一連のイベント通知がリストされています。タブは、通知のサブスクライブにも使用できます。

通知	説明
jtaThresholdEvent	指定された JTA 統計の件数が特定のしきい値を超えたときにリスナーに通知し、指定された件数の間隔で繰り返します。 しきい値を設定するには、前の表で説明されている addThresholdEvent 操作を使用します。
jtaAbandonRecoveryEvent	1 つ以上のトランザクションのリカバリが管理処理によって中止された場合に、リスナーに通知します。
jtaSevereMessageEvent	重大なレベルの JTA メッセージが記録された場合に、リスナーに通知します。
jtaProtocolErrorEvent	2 フェーズ・コミットの処理中にプロトコル違反が原因で ProtocolError がスローされた場合にリスナーに通知します。

また、Application Server Control コンソールの「通知サブスクリプション」ページで通知にサブスクライブできます。

「グローバル : 通知サブスクリプション」ページへのパス :
「OC4J: ホーム」→「管理」タブ→「タスク名 : JMX」→「通知サブスクリプション」

通知は、Application Server Control コンソールの「受信済の通知」ページで参照できます。

「グローバル : 受信済の通知」ページへのパス :
「OC4J: ホーム」→「管理」タブ→「タスク名 : JMX」→「受信済の通知」

OC4J トランザクション・マネージャのリカバリの管理

通常、システムが適切に構成されていれば、OC4J または 2PC トランザクションに参加するリソースのいずれかに障害が発生しても、引き続き元の状態に戻り、自動的にリカバリが行われます。

リカバリ管理

Application Server Control コンソールの `JTAResource` MBean を使用すると、リカバリ中止ルールの定義やリカバリ・トランザクションの即時中止を含め、トランザクション・リカバリを管理できます。

`JTAResource` MBean の次の操作を使用して、トランザクション・リカバリを管理できます。

操作	説明
`abandonRecovery`	指定されたトランザクションのリカバリを中止します。
`abandonRecoveryForType`	このタイプ（アプリケーション）のすべてのトランザクションのリカバリを中止します。
`addThresholdEvent`	指定された JTA 統計の件数が特定のしきい値を超えたときに起動され、指定された件数の間隔で繰り返されるイベント通知を追加します。
`clearStats`	すべての JTA 集計数統計を 0 にリセットします。
`configureCoordinator`	2 フェーズ・コミット・コーディネータを構成します（これらの設定を反映するにはサーバーを再起動する必要があります）。
`configureDatabaseLogging Performance`	2 フェーズ・コミット・コーディネータのデータベース・ロギング・パフォーマンスの設定を構成します。
`configureFileLogging Performance`	2 フェーズ・コミット・コーディネータのファイル・ロギング・パフォーマンスの設定を構成します。
`getMBeanInfo`	MBean のローカライズされたメタデータを取得します。
`heuristicCommit`	インダウト・トランザクションに対する自律型のコミットを決定します。
`heuristicRollback`	インダウト・トランザクションに対する自律型のロールバックを決定します。
`removeThresholdEvent`	しきい値イベントを削除します。
`rollbackTransaction`	アクティブなトランザクションに対する自律型のロールバックを決定します。
`setAbandonRecoveryCount ForResourceManager`	指定した参加者（JNDI ロケーション）が、指定された数のリカバリの再試行でリカバリに失敗したトランザクションのリソースのみの場合に、特定タイプ（アプリケーション）のトランザクションの中止を指定するリカバリ中止定義を追加します。

操作	説明
setAbandonRecoveryCount ForType	指定された数のリカバリの再試行でリカバリに失敗した場合に、特定タイプ（アプリケーション）のトランザクションの中止を指定するリカバリ中止定義を追加します。
setAbandonRecoveryTimeFor ResourceManager	指定した参加者（JNDI ロケーション）が、指定された時間（秒）内でのリカバリに失敗したトランザクションのリソースのみの場合に、特定タイプ（アプリケーション）のトランザクションの中止を指定するリカバリ中止定義を追加します。
setAbandonRecoveryTimeFor Type	指定された時間（秒）内でのリカバリに失敗した場合に、特定タイプ（アプリケーション）のトランザクションの中止を指定するリカバリ中止定義を追加します。
setRollbackOnly	アクティブなトランザクションに対する自律型の setRollbackOnly を決定します。

リカバリ再試行間隔

JTAResource MBean にも含まれている recoveryRetryInterval 属性は、トランザクションのリカバリを試行するまでにリカバリ・マネージャが待機する時間（秒単位）の特定に使用されます。再試行は任意の中止定義の対象になります。属性は、デフォルトで 300 秒に設定されます。

ログ

OPMN 環境では、OC4J に障害が発生すると、障害の発生したインスタンスのログを使用する新規インスタンスが OPMN によって起動されます。

スタンドアロン環境では、OC4J インスタンスに障害が発生して、なんらかの理由でそのインスタンスを元の状態に戻すことができない（または戻してはならない）場合、別の OC4J インスタンスにログを移行できます。

スタンドアロンでのログの移行

ログの移行プロセスは、インスタンスがルート・トランザクション・マネージャと介在トランザクション・マネージャのいずれであるか（またはその両方であるか）に応じて変化します。また、ロギング・メカニズムがファイルとデータベースのいずれであるかによっても変化します。

ファイル・ストアを使用するルート・トランザクション・マネージャの場合、新規 OC4J インスタンスのログ位置の構成を障害の発生したインスタンスのログ位置に変更するか、以前のログを手動で新規 OC4J インスタンスのログ位置に移動する必要があります。この作業は、宛先サーバーがオフラインであるか停止しているときに実行する必要があります。

ファイル・ストアを使用する介在トランザクション・マネージャの場合も同じプロセスを使用します。ただし、この介在トランザクション・マネージャの親トランザクション・マネージャは、以前の参照を新しいログ位置に更新する必要があります。この作業は、オフライン管理コマンドの -updateTransactionLogs を使用して実行できます。

データベース・ストアを使用するルート・トランザクション・マネージャの場合、オフライン管理コマンドの -updateTransactionLogs を使用して、oc4j_transaction 表のインスタンス・フィールドを更新することでログを移行します。新規 OC4J インスタンスで完全新規のデータベースを使用する場合は、ログ・ファイルをエクスポートおよびインポートする必要もあります。

データベース・ストアを使用する介在トランザクション・マネージャの場合も同じプロセスを使用します。ただし、この介在トランザクション・マネージャの親トランザクション・マネージャは、以前の参照を新しいログ位置に更新する必要があります。この作業は、オフライン管理コマンドの -updateTransactionLogs を使用して実行できます。

admin.jar では、次の 2 つのオフライン管理コマンドを使用できます。

- -analyzeTransactionLogs
- -updateTransactionLogs

注意： 前述の手順は、スタンドアロン環境のファイルを移行する場合に適用されますが、一定の状況下にある OPMN 環境でも役に立つ可能性があります。

-analyzeTransactionLogs コマンドでは、トランザクション・ログ・ファイルのオフライン分析が可能になります。このユーティリティは、OC4J が実行中の場合には使用しないでください（かわりに Application Server Control コンソールを使用します）。引数は次のとおりです。

引数	説明
-logType ["file" | "database"]	ストア・タイプ。
-location [location]	ストアの場所。 ファイル・ロギング用のディレクトリ。 データベース・ロギング用の接続 URL。
-username [username]	データベース・ロギング専用。
-password [password]	データベース・ロギング専用。

-updateTransactionLogs コマンドでは、トランザクション・ログ・ファイルのオフライン更新が可能になります。このユーティリティは、OC4J が実行中の場合には使用しないでください（かわりに Application Server Control コンソールを使用します）。引数は次のとおりです。

引数	説明
-logType ["file" | "database"]	ストア・タイプ。
-location [location]	ストアの場所。 ファイル・ロギング用のディレクトリ。 データベース・ロギング用の接続 URL。
-username [username]	データベース・ロギング専用。
-password [password]	データベース・ロギング専用。
-instanceId [old instance id] [new instanceid]	このログに関連付ける OC4J インスタンス ID。
-branchLocation [old branch location] [new branch location]	ブランチの場所。一般的には、アプリケーション名を接頭辞とする JNDI ロケーション。
-branchArg [old branch arg] [new branch arg]	ブランチの引数。コンテナ管理のサインオン・ユーザー名など。

ORMI を通じた OC4J プロセス間でのトランザクション伝播

トランザクション・コンテキストの伝播により、複数の OC4J インスタンスが単一のグローバル・トランザクションに参加できます。クライアントのトランザクションにおけるメソッド・サポートのスコープ決定作業が EJB のセマンティクスに含まれるると、OC4J インスタンスが既存のトランザクションのスコープ内にある別の OC4J インスタンスにリモート・コールを行う場合、複数の OC4J インスタンスが同じトランザクションに参加する必要があります。この例の 1 つは、次のような OC4J インスタンスのサーブレットです。

- 最初に、OC4J インスタンス内に存在する EJB への参照が取得されます。
- 次に、トランザクションが開始され、そのトランザクションのスコープ内でリモート EJB に対するメソッド・コールが実行されます。

注意： トランザクション・コンテキストの伝播とサブジェクトの伝播は、以前のリリースの OC4J ではサポートされていません。

複数の OC4J インスタンスが単一のトランザクションに参加する場合、それらの OC4J インスタンスによりグローバル・トランザクションの一部として実行されるすべての作業は、その原子性が保証されます。

Remote Method Invocation が OC4J インスタンス間で使用される場合、リクエスト側の OC4J インスタンスは、現在のトランザクションを示すトランザクション・コンテキストを作成し、ORMI を通じたリモート・コールを使用してそのコンテキストを暗黙的に送信する必要があります。これにより、Remote Method Invocation を受信する OC4J インスタンスは、リクエストの一部として実行する作業を、トランザクション・コンテキストで示されるトランザクションに関連付けることができます。リモート OC4J インスタンスは、リクエスト側の OC4J インスタンスにリソースとして登録されます。これにより、元の OC4J インスタンスがルート・トランザクション・コーディネータとなり、子の OC4J インスタンスがノードとなる OC4J インスタンスのツリーが作成されます。1 つの OC4J インスタンスは、親と子の両方になることが可能です。この場合、そのインスタンスは、親に対してはリソースとして振る舞い、子に対してはコーディネータとして振る舞います。これは、インターポジショニングと呼ばれます。ORMI の詳細は、第 6 章「Remote Method Invocation の使用方法」を参照してください。

ルート・コーディネータは、トランザクションをコミットすると、2 フェーズ・コミット（2pc）プロトコルを実行し、登録されている OC4J インスタンスを含むすべての登録済リソースに完了コールを送信します。ルート・コーディネータは、2 フェーズ・コミット（2pc）プロトコルを使用してトランザクションの完了処理を制御しますが、下位の OC4J インスタンスは、トランザクションの単なるリソースとして機能します。J2CA 1.5 のトランザクション・インフロー規約を使用して OC4J でトランザクションをインポートする場合のように、ルート・コーディネータは、OC4J インスタンスではなく外部のコーディネータでもかまいません。リソースとして登録された OC4J インスタンスは、他の任意の登録済リソースと同じように振る舞い、自身のトランザクション・ブランチの準備とコミットのみを担当します。OC4J インスタンスは、準備を指示するコールを受信すると、自身のすべての登録済リソースの準備を試行し、その結果を戻します。同様に、OC4J インスタンスは、コミットを指示するコールを受信すると、自身のすべての登録済リソースに対して commit をコールし、その結果を親に戻します。

トランザクション・インフロー規約の詳細は、『Oracle Containers for J2EE リソース・アダプタ管理者ガイド』の「インバウンド接続での RA の使用方法」の章を参照してください。

ORMI を通じて OC4J インスタンス間でトランザクションを伝播することにより、トランザクションを複数の OC4J インスタンスに拡張できます。

次のサイトにある How-To ドキュメント「How-To Propagate a transaction context between OC4J instances」と ZIP ファイルには、OC4J インスタンス間でのトランザクション・コンテキストの伝播に関する情報および使用例が含まれます。

```
http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/
how-to-transaction-propagation/doc/how-to-transaction-propagation.html
```

トランザクション伝播の構成

この項では、トランザクション伝播の構成に関する問題について説明します。

有効化と無効化

> **注意：** ほとんどの場合、トランザクション伝播を無効化することはありません。トランザクション伝播の無効化は、トラブルシューティングなどの例外的な状況においてのみ実行します。トランザクション伝播を無効化する場合は、慎重に行ってください。

トランザクション伝播は、OC4J ではデフォルトで有効です。トランザクション伝播を無効化するには、システム・プロパティ `-Drmi.disablePropagation=true` を設定します。このフラグの設定により、トランザクションおよびサブジェクトの伝播を含むすべてのコンテキスト伝播が無効化されます。トランザクション伝播の有効化または無効化は、サーバーにデプロイされているすべてのアプリケーションに影響します。粒度レベルを細かくすることはできません。トランザクション伝播の無効化は、慎重に行う必要があります。この設定については、リクエストに関連するすべての OC4J プロセスで必ず同じ構成にする必要があります。つまり、リクエストに関連するすべての OC4J プロセスで有効化するか、すべての OC4J プロセスで無効化します。これらの構成が混在すると（つまり、リクエストに関連する OC4J プロセスの一部で有効化され、一部で無効化されると）、予期しない動作が発生する可能性があります。また、トランザクション伝播が無効化されると、複数の OC4J プロセスは、単一のグローバル・トランザクションに参加できません。トランザクション伝播の無効化は、その結果をすべて完全に理解した場合にのみ行ってください。トランザクション伝播を無効化すると、問題が発生する可能性があるため、この機能は有効化しておくことをお薦めします。

リカバリ

OC4J プロセスは、トランザクション・リソースとして機能することがあるため、障害時にリカバリ可能である必要があります。トランザクション・リカバリを容易にするため、各 OC4J プロセスでは、トランザクション・リカバリ時にトランザクション・リカバリ・マネージャによって使用されるパスワードを構成する必要があります。OC4J に事前設定されているデフォルトのパスワードは、インストール後に変更する必要があります。

リカバリ・パスワードは、`$J2EE_HOME/config` ディレクトリの構成ファイル `jazn-data.xml` で構成します。トランザクション・リカバリ・パスワードを変更するには、`jazn-data.xml` ファイルの `JtaAdmin` ユーザーの資格証明値を変更します。

```
<user>
    <name>JtaAdmin</name>
    <display-name>JTA Recovery User</display-name>
    <description>Used to recover propagated OC4J transactions</description>
    <credentials>!newJtapassword</credentials>
</user>
```

> **注意：** Oracle Internet Directory などの JAZN 以外のセキュリティ・サービスを使用するよう OC4J が構成されていても、トランザクション・リカバリ・パスワードは `jazn-data.xml` に構成する必要があります。

トランザクション・リカバリの実行時には、リカバリされる OC4J プロセスの実際のセキュリティ・ストアにあるリカバリ・パスワードと、トランザクション処理中に `jazn-data.xml` で構成されたパスワードが一致する必要があります。パスワードが一致しない場合、リカバリ・マネージャは OC4J の下位プロセスと通信できないため、リカバリを完了できません。

ロギング

トランザクション伝播では、追加のトランザクション・ロギング構成を必要としませんが、1 つのトランザクションは複数の OC4J プロセスに拡張されることがあるため、各 OC4J プロセスは、ロギングに関して相互に依存しないよう構成する必要があります。

トランザクション伝播の制限事項

この項では、トランザクション伝播機能に適用される次の制限事項について説明します。

- 下位互換性
- EJB フェイルオーバー

下位互換性

OC4J の旧リリースでは伝播はサポートされていません。トランザクション伝播をサポートする OC4J インスタンスが、トランザクション伝播をサポートしない旧リリースの OC4J にデプロイされた Bean に対して Remote Method Invocation を実行しても、トランザクション・コンテキストは伝播されません。このため、コール元がトランザクションのスコープ内にあっても、リモート・マシンでの作業は、コール元のトランザクションのスコープ内で実行されません。アプリケーション・デプロイヤまたは管理者は、アプリケーションを様々なリリースの OC4J にデプロイする場合、デプロイの前にアプリケーションのトランザクション要件を理解しておく必要があります。

EJB フェイルオーバー

トランザクションが OC4J プロセスに伝播される場合、伝播されたトランザクションのスコープ内で障害が発生し、あるサーバーに対するリクエストがセカンダリ・サーバーにフェイルオーバーされると、そのトランザクションはロールバックされます。

EJB フェイルオーバーの動作の詳細は、『Oracle Containers for J2EE Enterprise JavaBeans 開発者ガイド』を参照してください。

デバッグとトラブルシューティング

デバッグとトラブルシューティングに役立つヒントは、次のとおりです。

- `j2ee-logging.xml` ファイルを使用して、デバッグを有効化できます。トランザクションの問題は、トランザクション参加者に関連することが多いため、J2CA、JDBC、およびその他のロギングを有効化すると、コンテキストを理解するうえで非常に役立ちます。
- トランザクション・マネージャのログ出力名は、`oracle.j2ee.transaction` です。`j2ee-logging.xml` ファイルの詳細は、『Oracle Containers for J2EE 構成および管理ガイド』の「OC4J でのロギング」の章を参照してください。
- Application Server Control コンソールでは、アクティブなトランザクションとリカバリ中のトランザクションすべてに関する情報を参照できます。
- DMS は、すべての統計で有効化する必要があります。
- 未処理の J2CA ローカル・トランザクションに参加中のグローバル・トランザクションにリソースを登録しようとすると、トランザクション・マネージャから例外がスローされます。この動作は、J2CA 1.5 仕様の 7.8.3 項で規定されています。

4

Oracle Enterprise Messaging Service の使用

Oracle Enterprise Messaging Service（OEMS）は、分散アプリケーションを構築および統合するための強力なメッセージ・プラットフォームです。OEMS により、Oracle のメッセージ機能とメッセージ統合ソリューションのためのフレームワークが提供されます。

OEMS を構成する主要機能は、次のとおりです。

- **標準仕様インタフェース**
  - Java Message Service（JMS）および J2EE Connector Architecture（J2CA）
- **メッセージの永続性のサービス品質（Quality of Service）オプション**
  - メモリー内、ファイル・ベースまたは Oracle データベース
- **Oracle 以外のメッセージ・システムとのシームレスな統合**
  - JMS コネクタ、JMS ルーターおよびメッセージ・ゲートウェイ（MGW）

この章では、これらすべての機能について説明します。OEMS 機能のうち、この章で取り上げていないのは、メッセージ・ゲートウェイのみです。メッセージ・ゲートウェイの詳細は、『Oracle Streams アドバンスト・キューイング・ユーザーズ・ガイドおよびリファレンス』を参照してください。

注意：

旧リリースでは、メモリー内、ファイル・ベースおよびデータベースの各永続性オプションを説明する際に、OracleAS JMS および OJMS という用語を使用していました。OracleAS JMS はメモリー内およびファイル・ベースのオプションを示し、OJMS は Streams Advanced Queuing（AQ）の JMS インタフェースを示していました。

JMS に関する混乱を避けるため、OracleAS JMS および OJMS という用語は使用されません。かわりに、OEMS JMS という用語が使用されます。これは、オラクル社が、メッセージの永続性に関する 3 つのオプションに対して単一の JMS インタフェースを提供しているという事実を反映しています。3 つのサービス品質オプションの間でメッセージの永続性を変更するのであれば、JMS アプリケーション・コードを変更する必要はありません。

この章には、次の項目が含まれます。

- JMS コネクタ
- リソース・プロバイダ
- JMS メッセージの送受信
- OEMS JMS での高可用性およびクラスタリングの使用
- JMS ルーター
- OEMS JMS の HTTP トネリングを使用したリモート宛先へのルーティング
- JMS のロギングの構成

JMS タスク

この章では、次の JMS タスクについて説明します。

- 宛先オブジェクトおよびコネクション・ファクトリの構成
- ポートの構成
- JMS メッセージの送受信
- OEMS JMS のデータベース参照の宣言
- Application Server Control コンソールでのファイル・ベースの永続性の有効化
- 論理名を使用したリソースの参照
- サード・パーティ JMS プロバイダの使用
- メッセージドリブン Bean の使用
- OEMS JMS での高可用性およびクラスタリングの使用

JMS の新機能

次の OC4J JMS の機能および動作は、今回のリリースの新機能です。

- **JMS 1.1 および J2EE 1.4 準拠 :** OEMS JMS は、JMS 1.1 および J2EE 1.4 に準拠しています。

  JMS 1.1 仕様は、`http://java.sun.com/products/jms/docs.html` で入手できます。

  J2EE 1.4 仕様は、`http://java.sun.com/j2ee/1.4/docs/index.html` で入手できます。

- **ドメイン統合 :** 同じ JMS コネクションおよびセッションを使用して、OEMS JMS のキューとトピックを操作できます。これは、JMS 1.1 の機能の 1 つです。
- **JMS コネクタ :** JMS コネクタは、OEMS JMS のメモリー内、ファイル・ベースおよびデータベースのリソース・プロバイダと、Oracle 以外の JMS プロバイダを OC4J に統合するための汎用 JMS J2CA リソース・アダプタです。統合可能な Oracle 以外の JMS プロバイダとしてサポートされるのは、WebSphereMQ、Tibco および SonicMQ です。これらすべてのプロバイダは、OC4J と完全に統合されています。JMS コネクタでは、メッセージドリブン Bean（MDB）がサポートされます。その他の主なメリットは、次のとおりです。
  - グローバル・トランザクションのサポート
  - 変化する負荷に対応する MDB
  - JMS 接続プーリング（非管理環境で稼働するアプリケーションには実装されない）
  - パフォーマンス強化機能 : グローバル・トランザクションの遅延登録と JMS 操作の遅延評価。リソースの登録やコネクション・ファクトリおよび宛先のルックアップなどの特定の操作は、必要になるまで実行されません。

  リソース・アダプタの詳細は、『Oracle Containers for J2EE リソース・アダプタ管理者ガイド』を参照してください。
- MDB のサポート : 次の JMS リソース・プロバイダでサポートされます。
  - OEMS JMS のメモリー内、ファイル・ベースおよびデータベース

  - IBM WebSphere MQ ベースの JMS
  - TIBCO Enterprise Message Server
  - SonicMQ
- **ドメイン間でのトランザクション:** キューとトピックを同じトランザクションで使用できます。これは、JMS 1.1 の機能の 1 つです。
- **JMS ルーター:** JMS ルーターは、メッセージをブリッジするルーティング・サービスを提供します。JMS ルーターの詳細は、4-83 ページの「JMS ルーター」を参照してください。
- **Oracle およびサード・パーティ製品のサポート:** 次の JMS メッセージ・プロバイダは、OC4J に付属の JMS コネクタを通じて正式にサポートされます。
  - OEMS JMS のメモリー内、ファイル・ベースおよびデータベース
  - IBM WebSphere MQ for JMS バージョン 6.0 および 5.3 のリソース・プロバイダ
  - TIBCO Enterprise Message Server バージョン 4.3.0
  - SonicMQ JMS 6.0

  サード・パーティ・プロバイダの詳細は、4-14 ページの「リソース・プロバイダ」を参照してください。リソース・アダプタによる JMS プロバイダの構成と使用を確認できるデモは、`http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html` で入手できます。
- **開始順序非依存:** 詳細は、4-51 ページの「JMS コネクタ」を参照してください。
- **デモ・コード:** 様々な JMS 構成機能のデモを、`http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html` で入手できます。

  How-To ドキュメントおよびデモ・セットと、その URL のリストは、4-4 ページの「JMS の How-To ドキュメントおよびデモ・セット」を参照してください。

JMS について

Java クライアントおよび Java 中間層サービスでは、エンタープライズ・メッセージ・システムを使用できる必要があります。Java Message Service（JMS）は、Java プログラムに、これらのシステムにアクセスする共通の方法を提供します。JMS は、アプリケーション・コンポーネント間でデータを渡すための標準のメッセージ API で、異機種間環境およびレガシー環境でのビジネス統合を可能にします。

この章の内容を理解するには、JMS と JMS API に関する基本的な知識が必要です。チュートリアルや API ドキュメントなど、JMS に関する基本情報は、Sun 社の次の Web サイトを参照してください。

`http://java.sun.com/products/jms/index.jsp`

JMS には、次の 2 つのメッセージ・ドメインがあり、それぞれ JMS 宛先タイプおよびドメイン固有の Java インタフェースのセットに関連付けられています。

- Point-to-Point: メッセージは JMS キューを使用してシングル・コンシューマに送信されます。
- パブリッシュ / サブスクライブ : メッセージは、JMS トピックを使用してすべての登録済リスナーに一斉送信されます。

JMS 宛先オブジェクトは、JNDI 環境にバインドされ、J2EE アプリケーションで使用可能になります。

それぞれのメッセージ・ドメインに対応する 2 つのメッセージ・インタフェース・セット以外に、JMS 1.1 以上では、ドメイン非依存のアプリケーション・コードを実装するための共通インタフェース・セットも提供しています。この共通インタフェース・セットでは、2 つのメッセージ・ドメインの動作が個別に管理されますが（この動作は、JMS 宛先タイプとの関連付けに応じて、使用されるメッセージ・ドメインによって制御されます）、両方のメッセージ・ドメ

インに対して共通のプログラミング・インタフェースが提供されます。この共通インタフェース・セットに属するインタフェースと、ドメイン固有のインタフェースとの関連は、JMS 1.1 仕様の表 2-1 を参照してください。

下位互換性

新しい JMS アプリケーションをデプロイする場合は、J2CA 1.5 仕様に基づいており、J2EE 1.4 標準に規定されている JMS コネクタを使用することをお薦めします。このマニュアルでは、OracleAS で導入された新機能について説明します。ただし、オラクル社では、OracleAS の旧リリースでサポートされていた独自仕様の OC4J リソース・プロバイダを通じてデプロイされた JMS アプリケーションのサポートも継続しています。

Oracle JMS コネクタの詳細は、4-51 ページの「JMS コネクタ」を参照してください。

JMS の How-To ドキュメントおよびデモ・セット

How-To ドキュメントおよび使用例のセット（コメント付き構成ファイルを含む）は、How-To Web サイト（`http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html`）で入手できます。

JMS ドキュメントとデモ・セットは、「Messaging（JMS）」という見出しの下にリストされています。デモ・セットのドキュメントおよびデプロイメント・ディスクリプタ・ファイルは、リソース・プロバイダごとに編成されており、サポート対象の様々なリソース・プロバイダに必要とされる各種の構成が含まれます。関連するリソース・プロバイダに適用されるファイルを解凍して使用してください。

JMS 構成の概要

この項では、次の JMS 構成の概要について説明します。

- JMS 構成の順序
- JMS 構成ファイルの構造

この OEMS ドキュメントと、関連デモおよび How-To ドキュメントでは、サポートされる様々なリソース・プロバイダと組み合せて JMS コネクタを使用する方法について、特に OEMS JMS のメモリー内およびファイル・ベースの永続性オプションに重点を置いて説明します。

JMS 構成の順序

この項では、JMS 操作のために次のコンポーネントを準備する方法について説明します。

- アプリケーションの開発と構成
- リソース・プロバイダの構成
- JMS コネクタの構成

リソース・プロバイダのセットと、一致する JMS コネクタのセットという形式で、コネクション・ファクトリおよび宛先オブジェクトの一致セットを作成できます（必須ではありません）。または、JMS コネクタの宛先の一致セットを作成せずに、自動宛先ラッピング機能を使用することもできます。

アプリケーションの開発と構成

JMS メッセージ機能を使用するアプリケーションを開発および構成するためのタスクは、次のとおりです。

- メッセージを送受信するコードの記述
- JMS リソースの論理名の宣言
- JMS リソースの論理名の使用
- MDB クラスの作成と宣言
- メッセージ宛先の宣言
- メッセージ宛先へのリンク
- onMessage トランザクション属性の定義
- アプリケーション・モジュールの一覧表示

詳細は、デモ・セットに含まれる How-To ドキュメントを参照してください。How-To ドキュメントおよびデモ・セットと、その URL のリストは、4-4 ページの「JMS の How-To ドキュメントおよびデモ・セット」を参照してください。

リソース・プロバイダの構成

アプリケーション・コンポーネント開発者とアプリケーション・アセンブラは、複数のコネクション・ファクトリおよび宛先を開発に使用するため、リソース・プロバイダの構成は、複数の作業に分割されるのが普通です。通常、開発時のコネクション・ファクトリと宛先は、デプロイ時のものとは異なります。開発サーバーと本番サーバーは、別個のマシンであることが普通であり、異なる組織方針に基づいて構成されるためです（場合によっては、異なるリソース・プロバイダが使用されることもあります）。このドキュメントでは、本番のデプロイに重点を置いて説明します。

リソース・プロバイダを構成する場合、次のことを決定する必要があります。

- アプリケーションを満足させるのに必要なリソース・プロバイダのコネクション・ファクトリの数とタイプ
- アプリケーションを満足させるのに必要なリソース・プロバイダの宛先の数とタイプ
- リソース・プロバイダのコネクション・ファクトリの作成
- リソース・プロバイダの宛先の作成
- リソース・プロバイダ参照の宣言

詳細は、デモ・セットの How-To ドキュメントを参照してください。How-To ドキュメントおよびデモ・セットと、その URL のリストは、4-4 ページの「JMS の How-To ドキュメントおよびデモ・セット」を参照してください。

JMS コネクタの構成

OEMS への JMS コネクタ機能の導入により、様々なリソース・プロバイダの仕様からの独立が部分的に実現しています。J2CA 1.5 仕様に基づく JMS コネクタは、リソース・プロバイダに対して、互換レイヤーおよび機能付加型のラッパーとして動作します。

JMS コネクタを構成するためのタスクは、次のとおりです。

- `ra.xml` の設定
- JMS コネクタ・インスタンスの作成
- JMS コネクタのコネクション・ファクトリの作成
- JMS コネクタの宛先の作成

詳細は、デモ・セットに含まれる How-To ドキュメントを参照してください。How-To ドキュメントおよびデモ・セットと、その URL のリストは、4-4 ページの「JMS の How-To ドキュメントおよびデモ・セット」を参照してください。

追加情報および使用例

oc4j-connectors.xml、oc4j-ra.xml および ra.xml ファイルの具体的な使用例は、http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html を参照してください。

JMS コネクタの XML ファイルのリファレンス情報は、『Oracle Containers for J2EE リソース・アダプタ管理者ガイド』の付録 A「OC4J リソース・アダプタ構成ファイル」を参照してください。

次に、「How to Configure and Use Oracle's Generic JMS Resource Adapter with OracleAS JMS」と、対応するデモ・ファイルのセットを含む ZIP ファイルへのリンクを示します。これらのドキュメントに記載されている汎用 JMS リソース・アダプタ（Generic JMS Resource Adapter）という用語は、JMS コネクタのことです。

- 「How to Configure and Use Oracle's Generic JMS Resource Adapter with OracleAS JMS」は、次の URL で参照できます。

  http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/how-to-gjra-with-oracleasjms/doc/how-to-gjra-with-oracleasjms.html

- 対応するデモ・ファイルのセットを含む ZIP ファイルは、次の URL で入手できます。

  http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/how-to-gjra-with-oracleasjms/how-to-gjra-with-oracleasjms.zip

JMS 構成ファイルの構造

この項では、コネクション・ファクトリおよび宛先について、JMS 構成ファイル内の参照と定義の間に存在する必要のある整合性について説明します。図 4-1「JMS 構成」に、相互に一致する必要のある参照と定義を示します。

図 4-1 JMS 構成

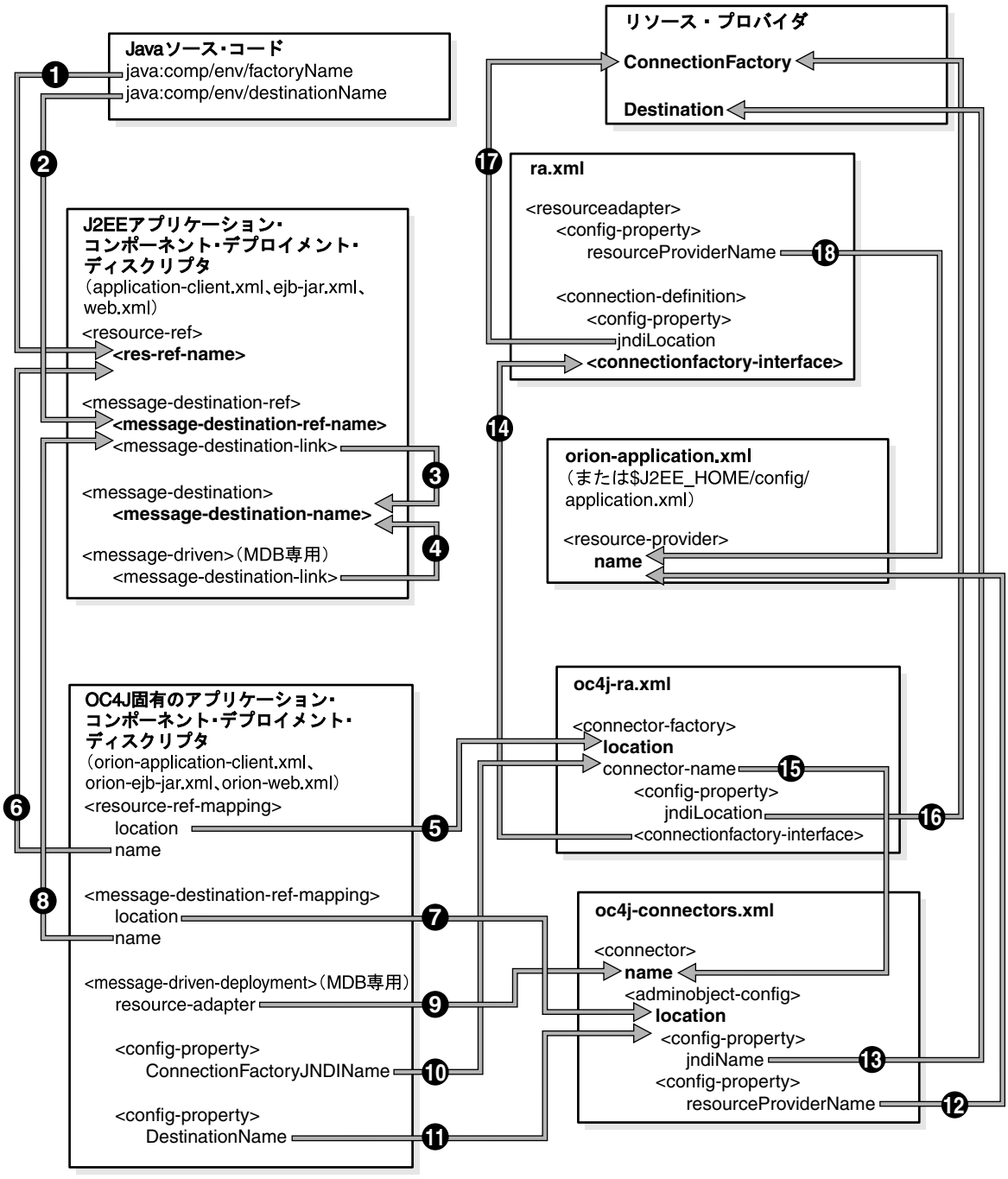


図 4-1 は、Java ソース・コード、アプリケーション・デプロイメント・ディスクリプタ、リソース・アダプタ・デプロイメント・ディスクリプタおよびリソース・プロバイダ間の様々なリンクを示しています。各矢印の元にはリンク参照があり、各矢印の先には参照されるアイテムのリンク・キー（名前、JNDI ロケーションまたは Java インタフェース）があります。任意の矢印の先にあるリンク・キーと、矢印の元にあるリンク参照のテキスト表現は、特に指定のないかぎり常に同じになります。

構成ファイルは次のとおりです。

- `ra.xml`
- `oc4j-ra.xml`
- `application.xml`
- `orion-application.xml`
- `oc4j-connectors.xml`
- `ejb-jar.xml`
- `orion-ejb-jar.xml`
- `application-client.xml`
- `orion-application-client.xml`
- `web.xml`
- `orion-web.xml`

デモ・セットには、図 4-1「JMS 構成」に示されている関係の詳細な説明が含まれます。How-To ドキュメントおよびデモ・セットと、その URL のリストは、4-4 ページの「JMS の How-To ドキュメントおよびデモ・セット」を参照してください。

関連する How-To-gjra-with-*xxx*.zip ファイル（*xxx* は関連するリソース・プロバイダの名前）をダウンロードして解凍します。

関連する how-to-gjra-with-*xxx*.html ドキュメントを開きます。この項の説明は、アイテム間の関係について記載した How-To ドキュメントの項に対応します。

デモ・セットにも、Java コードおよびデプロイメント・ディスクリプタの XML ファイルの使用例が含まれます。

Java ソース・コード

J2EE アプリケーションの Java ソース・コードでは、通常、JMS リソースを参照するために論理名を使用します。論理名（`<resource-ref>` および `<message-destination-ref>` 要素で宣言される参照）は、環境エントリのタイプであり、すべての環境エントリは、JNDI サブコンテキストの `java:comp/env/` に配置されます。詳細は、関連する How-To ドキュメントの「Application Component Provider Task #3: Use Logical Names for JMS Resources」を参照してください。

図 4-1 には、Java ソース・コードから J2EE アプリケーション・コンポーネント・デプロイメント・ディスクリプタに向かう次の 2 つのタイプのリンクがあります。次にリストする各説明の番号は、図のリンク番号に対応しています。

- **1:** 典型的なアウトバウンド・コネクション・ファクトリ・チェーンの 1 番目のリンクは、Java ソース・コードから J2EE アプリケーション・コンポーネント・デプロイメント・ディスクリプタの `<resource-ref>` 要素に向かいます。リンク参照は、JMS コネクション・ファクトリを取得するために JNDI ルックアップで使用されるロケーションです。リンク参照には、`java:comp/env` 接頭辞を含める必要があります。`<resource-ref>` 要素のリンク・キーは、その `<res-ref-name>` サブ要素の値です。リンク・キーには、java:comp/env 接頭辞を含めないでください。`java:comp/env` 接頭辞を除けば、リンク参照とリンク・キーは同一です。
- **2:** 典型的なアウトバウンド宛先チェーンの 1 番目のリンクは、Java ソース・コードから J2EE アプリケーション・コンポーネント・デプロイメント・ディスクリプタの `<message-destination-ref>` 要素に向かいます。リンク参照は、JMS 宛先を取得する

ために JNDI ルックアップで使用されるロケーションです。リンク参照には、java:comp/env 接頭辞を含める必要があります。<message-destination-ref> 要素のリンク・キーは、その <message-destination-ref-name> サブ要素の値です。リンク・キーには、java:comp/env 接頭辞を含めないでください。java:comp/env 接頭辞を除けば、リンク参照とリンク・キーは同一です。

J2EE アプリケーション・コンポーネント・デプロイメント・ディスクリプタ（application-client.xml、ejb-jar.xml、web.xml）

アプリケーション・コンポーネント・プロバイダによって使用される論理名は、物理宛先と多対 1 の関係にあります。アプリケーション・アセンブラは、物理宛先と 1 対 1 の関係にある論理宛先を作成します。次に、アプリケーション・アセンブラは、自身が作成したメッセージ宛先に、アプリケーション・コンポーネント・プロバイダが作成したメッセージ宛先参照および MDB をリンクする必要があります。これを行うには、適切なメッセージ宛先を名付ける <message-destination-link> 要素を追加します。これらのリンクは、宛先チェーンの一部ではありませんが、これにより宛先チェーンを完成するために必要な情報がデプロイヤに提供されます。詳細は、関連する how-to-gjra-with-xxx.html ドキュメント（xxx は関連するリソース・プロバイダの名前）の「Application Assembler Task #2: Link to Message Destinations」を参照してください。

図 4-1 には、J2EE アプリケーション・コンポーネント・デプロイメント・ディスクリプタ内で完結する次の 2 つのタイプのリンクがあります。

- **3:** アウトバウンド・メッセージの場合、この情報専用リンクは、<message-destination-ref> 要素から <message-destination> 要素に向かいます。どちらの要素も、J2EE アプリケーション・コンポーネント・デプロイメント・ディスクリプタに含まれます（必ずしも同じである必要はありません）。リンク参照は、<message-destination-link> サブ要素の値です。<message-destination> 要素のリンク・キーは、その <message-destination-name> サブ要素の値です。リンク参照には、接頭辞として、リンク・キーを含むファイルの名前とそれに続く # 文字が付く場合があります。（この接頭辞は、異なるファイルの様々なリンク・キーがまったく同じ値を保持する場合にのみ必要です。）このオプションの接頭辞を除けば、リンク参照とリンク・キーは同一です。
- **4:** MDB およびインバウンド・メッセージの場合、この情報専用リンクは、<message-driven> 要素から <message-destination> 要素に向かいます。どちらの要素も、J2EE アプリケーション・コンポーネント・デプロイメント・ディスクリプタに含まれます（必ずしも同じである必要はありません）。リンク参照は、<message-destination-link> サブ要素の値です。前述のとおり、<message-destination> 要素のリンク・キーは、その <message-destination-name> サブ要素の値です。リンク参照には、接頭辞として、リンク・キーを含むファイルの名前とそれに続く # 文字が付く場合があります。（この接頭辞は、異なるファイルの様々なリンク・キーがまったく同じ値を保持する場合にのみ必要です。）このオプションの接頭辞を除けば、リンク参照とリンク・キーは同一です。

OC4J 固有のアプリケーション・コンポーネント・デプロイメント・ディスクリプタ（orion-application-client.xml、orion-ejb-jar.xml、orion-web.xml）

図 4-1 には、OC4J 固有のアプリケーション・コンポーネント・デプロイメント・ディスクリプタから別のファイルに向かう次の 7 つのタイプの参照があります。

- アプリケーション・コンポーネントでは、J2EE アプリケーション・コンポーネント・デプロイメント・ディスクリプタにコネクション・ファクトリの論理名を宣言します。次に、デプロイヤは、それらの論理名を JMS コネクタのコネクション・ファクトリにマップします。

  詳細および使用例は、関連する How-To ドキュメントの「Deployer Task #1: Map Logical Connection Factories to RA ConnectionFactories」を参照してください。

図には、<resource-ref-mapping> 要素で指定される次の 2 つのタイプのコネクション・ファクトリ・リンクがあります。

- **6:** 典型的なアウトバウンド・コネクション・ファクトリ・チェーンの 2 番目のリンクは、OC4J 固有のアプリケーション・コンポーネント・デプロイメント・ディスクリプタの <resource-ref-mapping> 要素から J2EE アプリケーション・コンポーネント・デプロイメント・ディスクリプタの <resource-ref> 要素に戻ります。リンク参照は、<resource-ref-mapping> 要素の name 属性の値です。前述のとおり、<resource-ref> 要素のリンク・キーは、その <res-ref-name> サブ要素の値です。
- **5:** 典型的なアウトバウンド・コネクション・ファクトリ・チェーンの 3 番目のリンクは、OC4J 固有のアプリケーション・コンポーネント・デプロイメント・ディスクリプタの <resource-ref-mapping> 要素から oc4j-ra.xml ファイルの <connector-factory> 要素に向かいます。リンク参照は、<resource-ref-mapping> 要素の location 属性の値です。<connector-factory> 要素のリンク・キーは、その location 属性の値です（この値は、特定の <connector-factory> 要素によって定義されるリソース・アダプタのコネクション・ファクトリがバインドされる JNDI ロケーションとも一致します）。

■ アプリケーション・コンポーネントでは、J2EE アプリケーション・コンポーネント・デプロイメント・ディスクリプタに宛先の論理名を宣言します。次に、デプロイヤは、アプリケーション・アセンブラによって <message-destination-link> および <message-destination> の形式で提供される任意の情報を使用して、それらの論理名を JMS コネクタの宛先にマップします。

<message-destination> ごとに、デプロイヤは、その <message-destination> にリンクするすべての <message-destination-ref> および MDB を同じ宛先にマップする必要があります。

詳細および使用例は、関連する How-To ドキュメントの「Deployer Task #2: Map Logical Destinations to RA Destinations」を参照してください。

図には、<destination-ref-mapping> 要素で指定される次の 2 つのタイプの宛先リンクがあります。

- **8:** 典型的なアウトバウンド宛先チェーンの 2 番目のリンクは、OC4J 固有のアプリケーション・コンポーネント・デプロイメント・ディスクリプタの <message-destination-ref-mapping> 要素から J2EE アプリケーション・コンポーネント・デプロイメント・ディスクリプタの <message-destination-ref> 要素に戻ります。リンク参照は、<message-destination-ref-mapping> 要素の name 属性の値です。前述のとおり、<message-destination-ref> 要素のリンク・キーは、その <message-destination-ref-name> サブ要素の値です。
- **7:** 典型的なアウトバウンド宛先チェーンの 3 番目のリンクは、OC4J 固有のアプリケーション・コンポーネント・デプロイメント・ディスクリプタの <message-destination-ref-mapping> 要素から oc4j-connectors.xml ファイルの <adminobject-config> 要素に向かいます。リンク参照は、<message-destination-ref-mapping> 要素の location 属性の値です。<adminobject-config> 要素のリンク・キーは、その location 属性の値です（この値は、特定の <adminobject-config> 要素によって定義されるリソース・アダプタの宛先がバインドされる JNDI ロケーションとも一致します）。

■ MDB ごとに、デプロイヤは、MDB のインバウンド・メッセージ要件を満たすために使用される JMS コネクタ・インスタンス、コネクション・ファクトリおよび宛先を指定する必要があります。

詳細および使用例は、関連するドキュメントの「Deployer Task #2: Map Logical Destinations to RA Destinations」および「Deployer Task #3: Configure the MDB」を参照してください。

図 4-1 には、orion-ejb-jar.xml ファイルから oc4j-connectors.xml および oc4j-ra.xml ファイルに向かう次の 3 つのタイプのインバウンド・メッセージ・リンクがあります。

- **9:** コンテナに対し、特定の MDB のインバウンド・メッセージ要件を処理するための JMS コネクタ・インスタンスを指定するリンクは、`orion-ejb-jar.xml` ファイルの MDB の `<message-driven-deployment>` 要素から `oc4j-connectors.xml` ファイルの JMS コネクタ・インスタンスの `<connector>` 要素に向かいます。リンク参照は、`<message-driven-deployment>` 要素の resource-adapter 属性の値です。`<connector>` 要素のリンク・キーは、その name 属性の値です（この値は、特定の `<connector>` 要素によって定義されるリソース・アダプタ・インスタンスがバインドされる JNDI ロケーションとも一致します）。
- **10:** MDB およびインバウンド・メッセージでは、ここまでに説明した 3 つのコネクション・ファクトリ・リンクのかわりに、1 つのリンクが使用されます。このリンクは、`orion-ejb-jar.xml` ファイルの `<message-driven-deployment>` 要素から `oc4j-ra.xml` ファイルの `<connector-factory>` 要素に向かいます。リンク参照は、`<message-driven-deployment>` の `ConnectionFactoryJNDIName` 構成プロパティの値です。前述のとおり、`<connector-factory>` 要素のリンク・キーは、その location 属性の値です（この値は、特定の `<connector-factory>` 要素によって定義されるリソース・アダプタのコネクション・ファクトリがバインドされる JNDI ロケーションとも一致します）。これは、アウトバウンドの 3 番目のリンクと同じ場所にリンクしていることに注意してください。コネクション・ファクトリ・チェーンの残りは、インバウンド・メッセージとアウトバウンド・メッセージの両方で同じです。
- **11:** MDB およびインバウンド・メッセージでは、ここまでに説明した 3 つの宛先リンクのかわりに、1 つのリンクが使用されます。このリンクは、`orion-ejb-jar.xml` ファイルの `<message-driven-deployment>` 要素から `oc4j-connectors.xml` ファイルの `<adminobject-config>` 要素に向かいます。リンク参照は、`<message-driven-deployment>` の `DestinationName` 構成プロパティの値です。前述のとおり、`<adminobject-config>` 要素のリンク・キーは、その location 属性の値です（この値は、特定の `<adminobject-config>` 要素によって定義されるリソース・アダプタの宛先がバインドされる JNDI ロケーションとも一致します）。これは、アウトバウンドの 3 番目のリンクと同じ場所にリンクしていることに注意してください。宛先チェーンの残りは、インバウンド・メッセージとアウトバウンド・メッセージの両方で同じです。

oc4j-connectors.xml

JMS コネクタの宛先は、リソース・プロバイダ（RP）の宛先のラッパーとして機能します。JMS コネクタで RP の宛先をルックアップするには、JMS コネクタが、RP の宛先の JNDI ロケーションを特定できる必要があります。

詳細および使用例は、関連する How-To ドキュメントの「Resource Adapter Task #4: Create RA Destinations」を参照してください。リソース・プロバイダの宛先の作成方法は、関連する How-To ドキュメントの「Configuring the Resource Provider」を参照してください。

図 4-1 には、`oc4j-connectors.xml` に定義された JMS コネクタの宛先を RP の宛先に関連付ける次の 2 つのタイプのリンクがあります。

- **12:** 宛先チェーンの最後の部分は、（JMS コネクタの宛先を定義する）`oc4j-connectors.xml` ファイルの `<adminobject-config>` 要素からリソース・プロバイダの宛先に向かいます。これは、2 つのパラレル・リンクで構成されます。1 番目のリンクは、`<adminobject-config>` 要素から個々のアプリケーションの `orion-application.xml` ファイル（または、デフォルト・アプリケーションの `application.xml` ファイル）の `<resource-provider>` 要素に向かいます。リンク参照は、`<adminobject-config>` の `resourceProviderName` 構成プロパティの値です。`<resource-provider>` 要素のリンク・キーは、その name 属性の値です（この値は、`<resource-provider>` 要素によって定義されるリソース・プロバイダ参照がバインドされる `java:comp/resource` の下の JNDI ロケーション（*providerName*）とも一致します）。
- **13:** 2 番目のリンクにより、`<adminobject-config>` 要素からリソース・プロバイダの宛先に向かう接続が完成します。リンク参照は、`<adminobject-config>` の `jndiName` 構成プロパティの値です。リンク・キーは、リソース・プロバイダ（RP）の JNDI コンテキスト内にある RP の宛先の JNDI ロケーション（*resourceName*）です。同時に、これら 2 つ

のリンクにより、JMS コネクタの宛先に RP の宛先として次の完全な JNDI ロケーションが提供されます。

```
java:comp/resource/providerName/resourceName
```

oc4j-ra.xml

JMS コネクタのコネクション・ファクトリは、リソース・プロバイダ（RP）のコネクション・ファクトリのラッパーとして機能します。JMS コネクタで RP のコネクション・ファクトリをルックアップするには、JMS コネクタが、RP のコネクション・ファクトリの JNDI ロケーションを特定できる必要があります。

詳細および使用例は、関連する How-To ドキュメントの「Resource Adapter Task #3: Create RA Connection Factories」を参照してください。リソース・プロバイダのコネクション・ファクトリの作成方法は、関連する How-To ドキュメントの「Configuring the Resource Provider」を参照してください（ただし、これらのコネクション・ファクトリがすべて自動的に前もって作成される OEMS JMS のデータベースの永続性オプションは除きます）。

図 4-1 には、`oc4j-ra.xml` に指定された JMS コネクタのコネクション・ファクトリの実装を定義し、それらを RP のコネクション・ファクトリに関連付ける次の 3 つのタイプのリンクがあります。

- **15:** 理論上、このリンクは、（JMS コネクタのコネクション・ファクトリを定義する）`oc4j-ra.xml` ファイルの `<connector-factory>` 要素を（JMS コネクタ・インスタンスを定義する）`oc4j-connectors.xml` ファイルの `<connector>` 要素に関連付けます。現時点では、このリンクは実際に使用されない可能性がありますが、将来的な互換性を考慮して次のように設定する必要があります。リンク参照は、`<connector-factory>` 要素の `connector-name` 属性の値です。前述のとおり、`<connector>` 要素のリンク・キーは、その `name` 属性の値です（この値は、特定の `<connector>` 要素によって定義される JMS コネクタ・インスタンスがバインドされる JNDI ロケーションとも一致します）。
- **16:** コネクション・ファクトリ・チェーンの最後の部分は、（JMS コネクタのコネクション・ファクトリを定義する）`oc4j-ra.xml` ファイルの `<connector-factory>` 要素から RP のコネクション・ファクトリに向かいます。これは、2 つのパラレル・リンクで構成されます。1 番目のリンクは、リソース・プロバイダ参照がバインドされる `java:comp/resource` の下の JNDI ロケーション（*providerName*）を指定します。そのリンク参照は、`ra.xml` ファイルに配置されます（矢印番号 18 の説明を参照）。2 番目のリンクにより、`<connector-factory>` 要素からリソース・プロバイダのコネクション・ファクトリに向かう接続が完成します。リンク参照は、`<connector-factory>` の `jndiLocation` 構成プロパティの値です。リンク・キーは、リソース・プロバイダ（RP）の JNDI コンテキスト内にある RP のコネクション・ファクトリの JNDI ロケーション（*resourceName*）です。同時に、これら 2 つのリンクにより、JMS コネクタのコネクション・ファクトリに RP のコネクション・ファクトリとして次の完全な JNDI ロケーションが提供されます。

  ```
  java:comp/resource/providerName/resourceName
  ```

  このリンクは、実際にはオプションの上書き設定です。（矢印番号 17 の説明を参照。）慣例的に、このオプションは常に設定します（上書きする値と同じであっても設定します）。
- **14:** JMS コネクタのコネクション・ファクトリ（`oc4j-ra.xml` ファイルの `<connector-factory>` 要素）の実装の詳細は、`<connector-factory>` 要素を `ra.xml` ファイルの `<connection-definition>` 要素にリンクすることで定義します。リンク参照は、`<connector-factory>` の `<connectionfactory-interface>` サブ要素の値です。`<connection-definition>` 要素のリンク・キーは、その `<connectionfactory-interface>` サブ要素の値です。

ra.xml

JMS コネクタを使用する場合、ra.xml ファイルのほとんどの内容は変更する必要がありません。ra.xml ファイルは、J2EE Connector Architecture 1.5 のスキーマ・ファイル（JMS 以外のリソース・アダプタも含め、多くのタイプのリソース・アダプタと連携するよう設計されている汎用スキーマ）に基づいているためです。

詳細および使用例は、関連する How-To ドキュメントの「Resource Adapter Task #1: Customize the ra.xml File」を参照してください。

図 4-1 には、デフォルトの JMS コネクタのコネクション・ファクトリ設定を ra.xml に定義する次の 2 つのタイプのリンクがあります。

- **18:** これは、コネクション・ファクトリ・チェーンの最後の部分における 1 番目のリンクです。（矢印番号 16 の説明を参照。）このリンクは、ra.xml ファイルの <resourceadapter> 要素から個々のアプリケーションの orion-application.xml ファイル（または、デフォルト・アプリケーションの application.xml ファイル）の <resource-provider> 要素に向かいます。リンク参照は、<resourceadapter> の resourceProviderName 構成プロパティの値です。前述のとおり、<resource-provider> 要素のリンク・キーは、その name 属性の値です（この値は、<resource-provider> 要素によって定義されるリソース・プロバイダ参照がバインドされる java:comp/resource の下の JNDI ロケーションとも一致します）。ra.xml ファイルのリンク参照値は、単なるデフォルトであり、任意の JMS コネクタ・インスタンスに関して、そのインスタンスの oc4j-connectors.xml ファイルにある <connector> の resourceProviderName 構成プロパティを使用して上書きできます。上書きは一般的には不要であるため、この図の矢印では示されていません。
- **17:** このリンクは、<connection-definition> にリンクする <connector-factory>（矢印番号 14 の説明を参照）に jndiLocation 構成プロパティ（矢印番号 16 の説明を参照）が含まれない場合、デフォルトとして機能します。リンク参照は、<connection-definition> の jndiLocation 構成プロパティの値です。リンク・キーは、リソース・プロバイダ（RP）の JNDI コンテキスト内にある RP のコネクション・ファクトリの JNDI ロケーションです。

アプリケーション・クライアントの JMS コネクタの省略

JMS コネクタによって提供されるほとんどの機能は、アプリケーション・クライアントに適用されません。アプリケーション・クライアントをできるかぎり軽量化するため、アプリケーション・クライアントで JMS コネクタを使用しないよう選択できます。JMS コネクタを使用しない場合、JMS コネクタにのみ必要な JAR ファイルをクラスパスに含める必要はありません（JMS コネクタに必要な JAR ファイルのリストは表 4-9 を参照してください）。

JMS コネクタを使用しないアプリケーション・クライアントは、JMS コネクタを使用するアプリケーション・コンポーネントと通信できます（ただし、基礎となるリソース・プロバイダ（RP）の宛先が両方のコンポーネントで同じである必要があります）。JMS コネクタの省略は、次のように orion-application-client.xml ファイルで JMS コネクタのリソースではなくリソース・プロバイダのリソースを参照することで行います。

1. **JMS コネクタのコネクション・ファクトリの省略**

   <resource-ref-mapping> 要素ごとに、その location 属性の値を次のように設定します。

   ```
   java:comp/resource/providerName/resourceName
   ```

   ここで、*providerName* は、図 4-1「JMS 構成」の矢印番号 18 のリンク・キーと同じであり、*resourceName* は、図 3-1 の矢印番号 16 のリンク・キーと同じです。これにより、図 3-1 の矢印番号 6、18、16 が、RP のコネクション・ファクトリへの直接リンクで置き換えられ、JMS コネクタの oc4j-ra.mxl および ra.xml ファイルが省略されます。

2. **JMS コネクタの宛先の省略**

   <message-destination-ref-mapping> 要素ごとに、その location 属性の値を次のように設定します。

   ```
   java:comp/resource/providerName/resourceName
   ```

ここで、*providerName* は、図 4-1「JMS 構成」の矢印番号 12 のリンク・キーと同じであり、*resourceName* は、図 3-1 の矢印番号 13 のリンク・キーと同じです。これにより、図 3-1 の矢印番号 7、12、13 が、RP の宛先への直接リンクで置き換えられ、JMS コネクタの `oc4j-connectors.xml` ファイルが省略されます。

JNDI に直接アクセスする一部のサード・パーティ・ツールまたはライブラリには、`java:comp/resource` 構文や特定のリソース・プロバイダの特殊な名前（`Queues/` 接頭辞や OEMS JMS のデータベース・オプションで使用されるその他の接頭辞など）を許可しない厳密なロケーション制限または検証ルールが存在する場合があります。その場合、JMS コネクタは省略できません。（一般的に、この制限は、OEMS JMS のメモリー内およびファイル・ベースのオプションには適用されません。これらのリソースについては、JNDI ルックアップ単独で `resourceName` を使用できるためです。つまり、

`java:comp/resource/`*`providerName`*`/`

という接頭辞は、OEMS JMS のメモリー内またはファイル・ベースのオプションの使用時には、単なるオプションとなります。）

アプリケーションは、JMS コネクタの宛先を、RP のコネクション・ファクトリから導出された任意のオブジェクトに渡すことはできません。また、RP の宛先を、JMS コネクタのコネクション・ファクトリから導出された任意のオブジェクトに渡すこともできません。JMS コネクタでは、すべての送信、受信および参照メッセージの `JMSDestination` および `JMSReplyTo` ヘッダー・フィールドで、任意の宛先タイプが別の宛先タイプに自動的に変換されます。たとえば、JMS コネクタを使用しないアプリケーション・クライアントが、RP のメッセージの `JMSReplyTo` ヘッダー・フィールドに RP の宛先を設定してメッセージを送信し、JMS コネクタを使用する別のアプリケーション・コンポーネントがそのメッセージを受信して `JMSReplyTo` ヘッダー・フィールドを読み取るとします。このとき、受信側は、元の RP のメッセージおよび宛先をラップする JMS コネクタ互換のメッセージおよび宛先を取得します。これ以外の場合には、自動変換されることはありません。たとえば、この使用例において、メッセージを直接送信するかわりに ObjectMessage のボディとして送信した場合、受信側が ObjectMessage のボディを抽出したときに、JMS コネクタのメッセージではなく RP のメッセージが取得されます。また、この RP メッセージの `JMSReplyTo` ヘッダー・フィールドには、JMS コネクタの宛先ではなく RP の宛先が含まれます。

リソース・プロバイダ

アプリケーション・クライアントが JMS メッセージの送受信に使用する、基礎となるコネクション・ファクトリおよび宛先は、リソース・プロバイダ・オブジェクトです。OC4J では、JMS コネクタの使用により、OEMS JMS（メモリー内、ファイル・ベース、データベース）、IBM MQ、TIBCO および Sonic の各リソース・プロバイダをプラグインします。

最終的には、リソース・プロバイダ内で宛先とコネクション・ファクトリを作成する必要があります。

リソース・プロバイダを構成する一般的な手順は、次のとおりです。

- リソース・プロバイダ参照の宣言
- リソース・プロバイダのコネクション・ファクトリの作成
- リソース・プロバイダの宛先の作成

JMS プロバイダごとに、プロバイダを構成し、コネクション・ファクトリおよび宛先オブジェクトを作成するための独自の手順があります。OEMS JMS 以外のリソース・プロバイダの詳細は、各プロバイダのドキュメントを参照してください。

- OEMS JMS のメモリー内およびファイル・ベースのリソース・プロバイダのコネクション・ファクトリと宛先は、作成後に `jms.xml` ファイルの JNDI にバインドされます。OEMS JMS の永続性オプションの詳細は、4-17 ページの「OEMS JMS のメモリー内およびファイル・ベースの永続性」を参照してください。
- OEMS JMS のデータベースの永続性オプションのコネクション・ファクトリと宛先は、SQL プロシージャを使用して作成します。OEMS JMS のデータベースの永続性オプションの詳細は、4-41 ページの「OEMS JMS のデータベースの永続性」を参照してください。

リソース・プロバイダ参照の宣言

クライアントは、希望する統合機能およびサービス品質（QOS）機能に応じて、それぞれ独自のリソース・アダプタを持つ 1 つ以上の異なる JMS リソース・プロバイダを使用できます。

リソース・プロバイダへの参照は、`orion-application.xml` ファイルおよび `application.xml` ファイルの 1 つ以上の `<resource-provider>` 要素で宣言します。

- OEMS JMS のメモリー内およびファイル・ベースの永続性 : これら 2 つの OEMS JMS の永続性オプションは、OC4J とともにインストールされます。
- OEMS JMS のデータベースの永続性 : OEMS JMS のデータベースの永続性オプションは、Oracle データベースの機能の 1 つであり、Streams Advanced Queuing（AQ）メッセージ・システムに基づきます。

  OEMS JMS のデータベースの永続性オプションのメリットは、次のとおりです。

  - Oracle データベースが基盤となります。
  - OEMS JMS のデータベースの永続性と、他の Oracle データベースのトランザクションを 1 フェーズ・コミット・トランザクション内で一緒に使用できます。
  - AQ により提供される拡張機能（PL/SQL および OCI との相互運用性など）にアクセスできます。
- サード・パーティ JMS プロバイダの使用 : 次のサード・パーティ JMS プロバイダと統合できます。
  - WebSphere MQ for JMS バージョン 6.0 および 5.3 のリソース・プロバイダ
  - TIBCO Enterprise Message Server バージョン 4.3.0
  - SonicMQ 6.0

次のいずれかのファイルを使用して、リソース・プロバイダ参照を宣言します。

- リソース・プロバイダ参照をすべてのアプリケーションに認識させるには（グローバル）、グローバル `application.xml` ファイルを使用します。
- リソース・プロバイダ参照を単一のアプリケーションにのみ認識させるには（ローカル）、そのアプリケーションに固有の `orion-application.xml` ファイルを使用します。

次のコードを適切な XML ファイルに追加します。

```
<resource-provider class="providerClassName" name="providerName">
     <description>description </description>
     <property name="name" value="value" />
</resource-provider>
```

`<resource-provider>` 属性は、次のように構成します。

- `class`: リソース・プロバイダ・クラスの名前。
  - OEMS JMS のメモリー内およびファイル・ベースのオプションでは、`com.evermind.server.jms.Oc4jResourceProvider` を使用します。
  - OEMS JMS のデータベース・オプションでは、`oracle.jms.OjmsContext` を使用します。
  - すべてのサード・パーティ・リソース・プロバイダでは、`com.evermind.server.deployment.ContextScanningResourceProvider` を使用します。
- `name`: リソース・プロバイダの識別名。この名前を使用して、リソース・プロバイダの JNDI コンテキストをアプリケーションの JNDI に次のようにマップします。

  `java:comp/resource/providerName/`

<resource-provider> のサブ要素は、次のように構成します。

- <description> サブ要素 : 特定のリソース・プロバイダの説明。
- <property> サブ要素 : リソース・プロバイダに提供されるパラメータの識別に使用される name および value 属性。name 属性でパラメータ名を指定し、value 属性でその値を指定します。

OC4J で稼働するアプリケーションまたはリソース・アダプタでリソース・プロバイダにアクセスするには、<resource-provider> 要素でリソース・プロバイダ参照を宣言する必要があります。リソース・プロバイダ参照には、OC4J がリソース・プロバイダと対話するための様々なデータが含まれます。また、リソース・プロバイダ参照には、リソース・プロバイダのリソースにアクセスするための JNDI サブコンテキストも含まれます。リソース・プロバイダ参照（および JNDI アクセス）は、orion-application.xml に配置して特定のアプリケーションのみでローカルに使用するか、%ORACLE_HOME%/j2ee/home/config/application.xml に配置してすべてのアプリケーションで使用することができます。

リソース・プロバイダ参照を宣言する場合は、常にリソース・プロバイダ参照に使用する名前と、リソース・プロバイダ・インタフェースを実装する Java クラスを指定する必要があります。

リソース・プロバイダ参照により、リソース・プロバイダの JNDI コンテキスト（リソース・プロバイダのコネクション・ファクトリおよび宛先を含む）が、アプリケーションからアクセス可能な JNDI サブコンテキストにマップされます（この場合、特に JMS コネクタからアクセスできることが重要です）。リソース・プロバイダのリソースに対して、アプリケーションがアクセスできることよりも JMS コネクタがアクセスできることの方が重要なのは、JMS コネクタの使用時には、アプリケーションはリソース・プロバイダの任意のリソースを直接ルックアップまたは使用する必要がなくなるためです（使用してはならないのが普通です）。この JNDI サブコンテキストは、java:comp/resource/providerName であり、providerName はリソース・プロバイダ参照の名前です。

http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html にあるデモ・セットには、リソース・プロバイダ参照の宣言の例が含まれます。How-To ドキュメントおよびデモ・セットと、その URL のリストは、4-4 ページの「JMS の How-To ドキュメントおよびデモ・セット」を参照してください。

How-To-gjra-with-*xxx*.zip ファイル（*xxx* は関連するリソース・プロバイダの名前）をダウンロードして解凍します。

/src/META-INF/orion-application.xml というファイルを見つけてください。

詳細は、関連する How-To ドキュメントの「Configuring the Resource Provider」を参照してください。

デモでは、データソースとリソース・プロバイダは、アプリケーションに対してローカルに宣言されています。データソース定義が $J2EE_HOME/config/data-sources.xml に、リソース・プロバイダ定義が $J2EE_HOME/config/application.xml に配置されていれば、それらの定義はすべてのアプリケーションに認識されます。（データソースを必要とするか、含んでいるのは、OEMS JMS のデータベースの永続性を使用するプロバイダのデモのみです。）

OC4J によってサポートされる各リソース・プロバイダでリソース・プロバイダ参照を宣言する方法の詳細（Java クラスやリソース・プロバイダ参照名の例など）は、次の項を参照してください。

- OEMS JMS のメモリー内およびファイル・ベースの永続性
- OEMS JMS のデータベースの永続性
- 4-44 ページの「OEMS JMS のデータベース参照の宣言」
- 4-49 ページの「IBM WebSphere MQ のリソース・プロバイダ参照の宣言」
- 4-49 ページの「TIBCO Enterprise Message Service のリソース・プロバイダ参照の宣言」
- 4-50 ページの「SonicMQ のリソース・プロバイダ参照の宣言」

OEMS JMS のメモリー内およびファイル・ベースの永続性

OEMS JMS のメモリー内およびファイル・ベースのオプションでは、次の機能が提供されます。

- JMS 1.1 仕様への準拠
- J2EE 1.4 仕様への準拠
- メモリー内またはファイル・ベースのメッセージの永続性における選択肢の提供
- 配信できないメッセージ用の例外キューの提供

注意： Application Server Control コンソール、MBean またはサンプル・コードで OC4J JMS または OracleAS JMS という用語が使用されている場合、それらは OEMS JMS のファイル内およびファイル・ベースの永続性オプションを示します。

この項には、次の項目が含まれます。

宛先オブジェクトおよびコネクション・ファクトリの構成

宛先オブジェクトは、キューまたはトピックです。OEMS JMS のメモリー内およびファイル・ベースのオプションは、OC4J とともにすでにインストールされています。したがって、構成する必要があるのは、アプリケーションで使用するカスタムの宛先オブジェクトとコネクション・ファクトリのみです。

宛先オブジェクトとコネクション・ファクトリを構成するための主要ツールは、Application Server Control コンソールです。XML ファイルを直接編集することも可能です。

デフォルトの宛先オブジェクトおよびコネクション・ファクトリ

デフォルトのメリットは、次のとおりです。

- デフォルトのコネクション・ファクトリは、独自に作成するコネクション・ファクトリのテンプレートとしてコピーして使用できます。また、本番環境でも変更することなく使用できます。
- デフォルトの宛先は、独自に作成する宛先のテンプレートとしてコピーして使用できます。ただし、本番環境では使用できません。

XA（グローバル・トランザクション対応）、非 XA、および様々な JMS インタフェースの異なる組合せに対して、6 つのデフォルト・コネクション・ファクトリが作成されます。アプリケーションでは、Application Server Control コンソールまたは `jms.xml` ファイルで追加することなく、これらのコネクション・ファクトリを使用できます。`connection-factory` 要素の 1 つ以上のオプション属性にデフォルト以外の値を指定する場合にのみ、新規コネクション・ファクトリを定義します。

デフォルトのコネクション・ファクトリ・オブジェクトは、OC4J によって内部的に作成され、JMS コネクションが作成される OC4J サーバー内のデフォルトの JNDI ロケーションにバインドされます。

次のデフォルトのコネクション・ファクトリが作成されます（`jms.xml` ファイルで明示的に定義していない場合でも作成されます）。カスタムのコネクション・ファクトリを作成する場合は、これらのデフォルトを予約済の JNDI ロケーションとして扱い、他の JNDI ロケーションを使用する方が安全です。

	XA	非 XA
デフォルトのキュー・コネクション・ファクトリ	`jms/XAQueueConnectionFactory`	`jms/QueueConnectionFactory`
デフォルトのトピック・コネクション・ファクトリ	`jms/XATopicConnectionFactory`	`jms/TopicConnectionFactory`
デフォルトの統合コネクション・ファクトリ	`jms/XAConnectionFactory`	`jms/ConnectionFactory`

デフォルトの宛先は、次のとおりです。

- デフォルトのキュー : `jms/demoQueue`
- デフォルトのトピック : jms/demoTopic

Application Server Control コンソールでの構成

Application Server Control コンソールは、OEMS JMS のメモリー内およびファイル・ベースの永続性オプションのコネクション・ファクトリおよび宛先オブジェクトを構成するための主要ツールです。宛先オブジェクトごとに、その名前、ロケーションおよび宛先タイプ（キューまたはトピック）を指定する必要があります。

Application Server Control コンソールへのパス :

「OC4J: ホーム」→「管理」タブ→「サービス」→「JMS プロバイダ」→「タスクに移動」→「OracleAS JMS の構成」→適切なタブを選択。

表 4-1「構成要素」に、OracleAS JMS のリソース・プロバイダの構成要素とその属性を示します。

構成要素

表 4-1 に、各構成要素と、Application Server Control コンソール、MBean および jms.xml ファイルでの設定場所を示します。

表 4-1 構成要素

コンソールおよび MBean での設定場所	jms.xml の要素	説明および属性
JMSAdministrator MBean により、サーバーのホスト名とポート、および複数の関連属性と操作を指定可能。 **JMSAdministrator MBean へのパス :** 「OC4J: ホーム」→「管理」タブ→「タスク名 : JMX」→「システム MBean ブラウザ」→「タスクに移動」→「J2EEDomain:oc4j」、「J2EEServer:standalone」、「JMSAdministratorResource」、「JMSAdministrator」にドリルダウン	<jms-server>	サーバー構成のルート要素です。 <jms-server> 要素には、次の属性が含まれます。 host: このサーバーがバインドされる String（DNS またはドット表記法によるホスト名）で定義されるホスト名です。デフォルトでは、サーバーは、0.0.0.0（構成ファイルでは [ALL]）にバインドされます。オプション設定です。 port: このサーバーがバインドされる int（有効な TCP/IP ポート番号）として定義されるポートです。デフォルト設定は、9127 です。この設定は、OC4J のスタンドアロン構成にのみ適用されます。Oracle Application Server の構成では、構成ファイル内のポート設定は、OC4J サーバーの起動時に（OPMN などで）使用されるコマンドライン引数で上書きされます。オプション設定です。
リソース・プロバイダの宛先を作成し、その属性を「宛先の追加」ページで指定。 **「宛先の追加」ページへのパス :** 「OC4J: ホーム」→「管理」タブ→「タスク名 : サービス」→「JMS プロバイダ」→「タスクに移動」→「宛先」タブ→「新規作成」	<queue>	この要素では、キューを構成します。キューは、OC4J の起動時に使用可能となり、サーバーが再起動または再構成されるまで使用できます。0（ゼロ）個以上のキューを任意の順序で構成できます。新規に構成したキューは、OC4J を再起動するまで使用できません。 <queue> 要素には、次の属性が含まれます。 name: この必須属性は、キューのプロバイダ固有の名前（String）です。この名前には、空でなく有効な任意の文字列を使用できます（空白や他の特殊文字も使用できますが、お薦めしません）。この属性で指定した名前を Session.createQueue() で使用すると、プロバイダ固有の名前を JMS キューに変換できます。2 つの宛先に同じ名前を指定することはできません。デフォルトはありません。 location: この必須属性では、キューがバインドされる JNDI ロケーション（String）を指定します。この値は、有効な名前に関する JNDI ルールに従う必要があります。 persistence-file: オプションのパスおよびファイル名（String）です。persistence-file 属性のパスは、ファイルの絶対パス、または application.xml に定義されている persistence ディレクトリに対する相対パスです。デフォルト・パスは、Oracle Application Server 環境の場合は *J2EE_HOME*/persistence/<group> で、スタンドアロン環境の場合は *J2EE_HOME*/persistence です。 キューおよびトピックごとに、独自の永続性ファイル名を指定する必要があります。同じ永続性ファイルに書き込む 2 つのオブジェクトを使用することはできません。 persistence-file 属性の詳細は、4-33 ページの「永続性のリカバリ」を参照してください。

表 4-1 構成要素（続き）

コンソールおよび MBean での設定場所	jms.xml の要素	説明および属性
トピックの宛先を作成し、その属性を「宛先の追加」ページで指定。 **「宛先の追加」ページへのパス：** 「OC4J: ホーム」→「管理」タブ→「タスク名 : サービス」→「JMS プロバイダ」→「タスクに移動」→「宛先」タブ→「新規作成」	`<topic>`	この要素では、トピックを構成します。トピックは、OC4J の起動時に使用可能となり、サーバーが再起動または再構成されるまで使用できます。0（ゼロ）個以上のトピックを任意の順序で構成できます。新規に構成したトピックは、OC4J を再起動するまで使用できません。 `<topic>` 要素には、次の属性が含まれます。 `name`: この必須属性は、トピックのプロバイダ固有の名前（`String`）です。この名前には、空でなく有効な任意の文字列を使用できます（空白や他の特殊文字も使用できますが、お薦めしません）。この属性で指定した名前を `Session.createTopic()` で使用すると、プロバイダ固有の名前を JMS トピックに変換できます。2 つの宛先に同じ名前を指定することはできません。デフォルトはありません。 `location`: この必須属性では、トピックがバインドされる JNDI ロケーション（`String`）を指定します。この値は、有効な名前に関する JNDI ルールに従う必要があります。デフォルトはありません。 `persistence-file`: オプションのパスおよびファイル名（`String`）です。`persistence-file` 属性のパスは、ファイルの絶対パス、または `application.xml` に定義されている `persistence` ディレクトリに対する相対パスです。デフォルト・パスは、Oracle Application Server 環境の場合は *`J2EE_HOME`*`/persistence/<group>` で、スタンドアロン環境の場合は *`J2EE_HOME`*`/persistence` です。 キューおよびトピックごとに、独自の永続性ファイル名を指定する必要があります。同じ永続性ファイルに書き込む 2 つのオブジェクトを使用することはできません。 `persistence-file` 属性の詳細は、4-33 ページの「永続性のリカバリ」を参照してください。
「説明」フィールドは、説明の適用されるトピックまたはキューを作成する「宛先の追加」ページに存在。	`<description>`	`<queue>` または `<topic>` のサブ要素です。キューまたはトピックの用途をユーザーに知らせるユーザー定義文字列です。オプション設定です。

表 4-1 構成要素（続き）

コンソールおよび MBean での設定場所	jms.xml の要素	説明および属性

コネクション・ファクトリを作成し、その属性を「コネクション・ファクトリの追加」ページで指定。
 	
「コネクション・ファクトリの追加」または「コネクション・ファクトリの編集」ページへのパス：
 	
「OC4J: ホーム」→「管理」タブ→「タスク名 : サービス」→「JMS プロバイダ」→「タスクに移動」→**「コネクション・ファクトリ」タブ**→「新規作成」または「プロパティの編集」
	`<connection-factory>`
	
	または
	
	`<queue-connection-factory>`
	
	または
	
	`<topic-connection-factory>`
		コネクション・ファクトリの構成です。コネクション・ファクトリ要素には、次の属性が含まれます。
		
		■ `location`: 必須。コネクション・ファクトリのバインド先となる JNDI ロケーション。この値は、有効な名前に関する JNDI ルールに準拠する必要があります。
		
		■ `host`: オプション。このコネクション・ファクトリの接続先となる固定の OC4J ホスト。デフォルトでは、コネクション・ファクトリは、`jms-server` 要素に構成されているものと同じホストを使用します。デフォルト以外の値を使用すると、ローカルで使用可能な OC4J サーバーや他の Oracle Application Server またはクラスタ構成を省略し、すべての JMS 操作を特定の OC4J JVM に送ることができます。オプションの文字列で、DNS またはドット表記法によるホスト名です。デフォルトは、`ALL` です。
		
		■ `port`: オプション。このコネクション・ファクトリの接続先となる固定のポート。デフォルトでは、コネクション・ファクトリは、`jms-server` 要素に構成されているものと同じポート（あるいは、コマンドラインにより Oracle Application Server またはクラスタ構成に対して指定したポートの値）を使用します。デフォルト以外の値を使用すると、ローカルで使用可能なサーバーや他の Oracle Application Server またはクラスタ構成を省略し、すべての JMS 操作を特定の OC4J JVM に送ることができます。オプションの int で、有効な TCP/IP のポート番号です。デフォルトは `9127` です。
		
		■ `username`: オプション。このコネクション・ファクトリから作成される JMS デフォルト接続の認証に使用するユーザー名。アプリケーションで接続を作成し、アプリケーションと `oc4j-ra.xml` ファイルのいずれでも `username/password` を指定しない場合、この要素の `username` および `password` 属性が使用されます。ユーザー名自体は、他の OC4J 機能を使用して正しく作成および構成する必要があります。オプションの文字列です。デフォルトは、空の文字列です。
		
		■ `password`: オプション。このコネクション・ファクトリから作成される JMS デフォルト接続の認証に使用するパスワード。パスワード自体は、他の OC4J 機能を使用して正しく作成および構成する必要があります。プロパティの `password` 属性は、パスワードの間接化をサポートします。詳細は、『Oracle Containers for J2EE セキュリティ・ガイド』を参照してください。オプションの文字列です。デフォルトは、空の文字列です。
		
		■ `clientID`: オプション。このコネクション・ファクトリで作成される接続用として、管理上構成されている固定の JMS `clientID`。`clientID` を指定しない場合、デフォルトは空の文字列です。この値は、JMS 仕様に基づいて、クライアント・プログラムで上書きすることもできます。`clientID` が使用されるのは、トピックの永続サブスクリプションの場合のみで、この値はキューと非永続トピックの操作には関係しません。オプションの文字列です。デフォルトは、空の文字列です。

表 4-1 構成要素（続き）

コンソールおよび MBean での設定場所	jms.xml の要素	説明および属性

XA 対応のコネクション・ファクトリを作成し、その属性を「コネクション・ファクトリの追加」ページで指定。
「コネクション・ファクトリの追加」または「コネクション・ファクトリの編集」ページへのパス：
「OC4J: ホーム」→「管理」タブ→「タスク名 : サービス」→「JMS プロバイダ」→「タスクに移動」→**「コネクション・ファクトリ」タブ**→「新規作成」または「プロパティの編集」
	`<xa-connection-factory>`
	または
	`<xa-queue-connection-factory>`
	または
	`<xa-topic-connection-factory>`
		コネクション・ファクトリ構成の XA バリアントです。
		XA 対応のコネクション・ファクトリ要素には、前述の XA 非対応のコネクション・ファクトリ要素と同じ属性が含まれます。

	`<log>`	ファイルまたは ODL 形式による JMS アクティビティのロギングを有効化します。ロギングの詳細は、『Oracle Containers for J2EE 構成および管理ガイド』の OC4J ロギングの有効化に関する項を参照してください。
`JMSAdministratorResource MBean` でシステム・プロパティ設定を編集。 **JMSAdministratorResource MBean のシステム・プロパティ設定へのパス：**「OC4J: ホーム」→「管理」タブ→「タスク名 : JMX」→「システム MBean ブラウザ」→「タスクに移動」→「J2EEDomain:oc4j」、「J2EEServer:standalone」、「JMSAdministratorResource」、「JMSAdministrator」→「操作」タブ→ setConfigProperty にドリルダウン	`<config-properties>`	システム・プロパティを設定します。設定は、`jms.xml` ファイルに永続化されます。 `<config-property>`: `<config-properties>` のサブ要素です。 これらの設定の詳細は、4-38 ページの「JMS 構成プロパティ」を参照してください。

jms.xml を使用した構成

OEMS JMS のメモリー内およびファイル・ベースの構成設定は、jms.xml ファイルに永続化されます。jms.xml の設定は、次のとおりです。

- コネクション・ファクトリ
- 宛先
- JMS ルーター・ジョブ
- グローバル構成

> **注意：** XML ファイルで直接変更した構成を有効化するには、OC4J インスタンスを再起動する必要があります。

後述の例は、jms.xml ファイル内の <jms-server> 以降の要素の構造を示しています。この例では、次のように宛先とコネクション・ファクトリを構成します。

- キュー MyQueue を JNDI ロケーション jms/MyQueue に指定
- トピック MyTopic を JNDI ロケーション jms/MyTopic に指定
- コネクション・ファクトリ（統合）を JNDI ロケーション jms/Cf に指定
- キュー・コネクション・ファクトリを JNDI ロケーション jms/Qcf に指定
- XA トピックのコネクション・ファクトリを JNDI ロケーション jms/xaTcf に指定

```
<jms>
  <jms-server>

    <queue  name="MyQueue" location="jms/MyQueue" persistence-file="/tmp/MyQueue">
       <description>The demo queue. </description>
    </queue>

    <topic name="MyTopic" location="jms/MyTopic" persistence-file="/tmp/MyTopic">
      <description>The demo topic. </description>
    </topic>

    <connection-factory location="jms/Cf">
    </connection-factory>

    <queue-connection-factory location="jms/Qcf">
    </queue-connection-factory>

    <xa-topic-connection-factory location="jms/xaTcf"
        username="foo" password="bar" clientID="baz">
    </xa-topic-connection-factory>


    <log>
      <file path="../log/jms.log" />
    </log>

    <config-properties>
      <config-property name="oc4j.jms.debug" value="true">
      </config-property>
    </config-properties>

  </jms-server>

   <jms-router>
      <!-- JMS router configuration is shown in the
      "JMS Router Configuration in jms.xml" section.
      -->
```

```
   </jms-router>

</jms>
```

<jms-router> 以降の要素の詳細な構造例は、4-90 ページの「jms.xml での JMS ルーターの構成」を参照してください。

ポートの構成

スタンドアロン OC4J インスタンスでは、JMSAdministrator MBean でポート範囲を設定できます。変更を有効にするには、OC4J インスタンスを再起動する必要があります。再起動を必要とするのは、ポート設定の特殊な場合です。

管理されている完全な Oracle Application Server 環境では、Application Server Control コンソールを使用してポート範囲を構成します。

Application Server Control コンソールでポート範囲を構成する場合のパス :

「OC4J: ホーム」→「管理」タブ→「タスク名 : JVM プロパティ」→「JMS ポート」

JMS メッセージの送受信

JMS メッセージを送受信するためのコードは、関連する JMS コネクタまたはリソース・プロバイダに依存しません。

この例は、
http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html
にあるデモ・セットの MyChannel.java ファイルからの抜粋です。

/src/common/MyChannel.java というファイルを見つけてください。

デモにおいては、MyChannel.java が JMS メッセージを送受信する唯一のクラスです。他のすべてのクラスでは、MyChannel をコールして送受信を行います。MyChannel は、様々なリソース・プロバイダのすべてに対応します。実際、コネクション・ファクトリと宛先のルックアップに使用できる（論理名に基づかない）代替 JNDI ロケーションを説明する Player.java のいくつかのコメントを除けば、すべてのリソース・プロバイダの .java ソースは同じものです。

How-To ドキュメントおよびデモ・セットと、その URL のリストは、4-4 ページの「JMS の How-To ドキュメントおよびデモ・セット」を参照してください。

```
   public MyChannel(String connectionFactoryName, String destinationName) throws
Exception {

        Context ctx = new InitialContext();

        // Get the destination.
        Destination destination = (Destination) ctx.lookup(destinationName);

        // Get the connection factory.
        ConnectionFactory cf = (ConnectionFactory)
 ctx.lookup(connectionFactoryName);

        // Use the connection factory to create a connection.
        connection = cf.createConnection();

        // Start the connection.
        connection.start();

        // Use the connection to create a session.
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

        // Use the session and destination to create a message producer and a message
consumer.
        producer = session.createProducer(destination);
        consumer = session.createConsumer(destination);
```

```
    }

    /**
     * Send message.
     * prerequisite: channel is open
     * @param obj object to be sent
     */
    public void send(Serializable obj) throws JMSException {

        // Use the session to create a new message.
        ObjectMessage msg = session.createObjectMessage(obj);

        // Use the message producer to send the message.
        producer.send(msg);
        System.out.println("Sent message: " + obj);
    }

    /**
     * Receive message (wait forever).
     * prerequisite: channel is open
     * @return object which was received in message, or <code>null</code> if no message
was received
     */
    public Serializable receive() throws JMSException {

        // Use the message consumer to receive a message.
        ObjectMessage msg = (ObjectMessage) consumer.receive();

        System.out.println("Got message: " + msg.getObject());
        return msg.getObject();
    }

    /**
     * Receive message (wait a while).
     * prerequisite: channel is open
     * @param timeout maximum time (in milliseconds) to wait for a message to arrive
     * @return object which was received in message,
     * or <code>null</code> if no message was received
     */
    public Serializable receive(long timeout) throws JMSException {

        // Use the message consumer to receive a message (if one comes in time).
        ObjectMessage msg = (ObjectMessage) consumer.receive(timeout);

        if (msg == null) return null;
        System.out.println("Got message: " + msg.getObject());
        return msg.getObject();
    }

    /**
     * Close channel.
     * prerequisite: channel is open
     * Once a MyChannel object is closed, it may no longer be used to send or receive
     * messages.
     */
    public void close() throws JMSException {

        // Close the connection (and all of its sessions, producers and consumers).
        connection.close();
    }

    private Connection connection;
    private Session session;
```

```
    private MessageProducer producer;
    private MessageConsumer consumer;
}
```

JMS ユーティリティの使用

OEMS JMS 実装は、OEMS との通信に使用されるコマンドライン・ユーティリティに含まれており、JMS 宛先のリスト作成や参照などのタスクを実行します。ユーティリティ・クラス（com.evermind.server.jms.JMSServerUtils）は、oc4j-internal.jar の OC4J に付属しています。このユーティリティによって提供されるタスクの多くは、Oracle Enterprise Manager 10*g* コンソールを使用してアクセス可能な一連の MBean からも使用できます。MBean の使用方法の詳細は、4-29 ページの「JMS MBean の使用」を参照してください。

ユーティリティでは、実行時に次の JAR ファイルが必要です。これらの JAR ファイルは、CLASSPATH 環境変数に追加できます。

- *J2EE_HOME*¥oc4j.jar
- *J2EE_HOME*¥oc4j-api.jar
- *J2EE_HOME*¥oc4jclient.jar
- *J2EE_HOME*¥rmic.jar
- *J2EE_HOME*¥lib¥adminclient.jar
- *J2EE_HOME*¥lib¥connector.jar
- *J2EE_HOME*¥lib¥javax77.jar
- *J2EE_HOME*¥lib¥jmxri.jar
- *J2EE_HOME*¥lib¥oc4j-internal.jar

ユーティリティの構文は次のとおりです。

```
java com.evermind.server.jms.JMSServerUtils [generic-options] [command]
[command-options] [arguments]
```

使用方法の詳細を参照するには、help コマンドを使用します。

```
java com.evermind.server.jms.JMSServerUtils help
```

すべてのオプションおよびコマンドは、表 4-2、表 4-3 および表 4-4 に説明されています。汎用オプションは、OracleAS JMS サーバーへの接続に使用されます。

> **注意：** JMSServerUtils を使用してサーバーに接続するには、OEMS JMS サーバーが実行されている必要があります。また、JMSServerUtils で、管理者ロール内のユーザーとして接続できるのは JMS サーバーのみです。ユーザーは、セキュリティ・ユーザー・マネージャ内でロールに追加されます。セキュリティ・ロール内でのユーザーの定義の詳細は、『Oracle Containers for J2EE セキュリティ・ガイド』を参照してください。

表 4-2 JMS ユーティリティの汎用オプション

オプション	説明
-host <hostname>	OracleAS JMS サーバーがインストールされている（リモート）ホスト。クライアントが OracleAS JMS サーバーと同じノードに存在している場合、このオプションは不要です。
-port <port>	OracleAS JMS サーバーへのアクセスに使用される（リモート）ポート。デフォルトの JMS ポート番号は 9127 です。
-username <username>	JMS 接続を作成する際に OracleAS JMS サーバーにアクセスするためのユーザー名。このユーザーは、管理ロール内のユーザー・マネージャ・セキュリティ構成に定義されています。

表 4-2 JMS ユーティリティの汎用オプション（続き）

オプション	説明
`-password <password>`	JMS 接続を作成する際に OracleAS JMS サーバーにアクセスするためのパスワード。このパスワードは、管理ロール内のユーザー・マネージャ・セキュリティ構成に定義されています。
`-clientID <ID>`	すべての JMS 接続にこの識別子を使用します。トピックの永続サブスクリプションを特定する場合にのみ必要です。

コマンドは、実行される処理の内容を表します。表 4-3 で各コマンドを説明します。一部のコマンドには、実行する処理をさらに特定するための独自のオプション（`command-options`）があります。

表 4-3 JMS ユーティリティ・コマンド

コマンド	説明
`help`	すべてのユーティリティ・コマンドの詳細なヘルプを出力します。
`check [<other-selector>]`	`-selector` コマンド・オプションを指定して、JMS メッセージ・セレクタの妥当性をチェックします。オプションで、指定された 2 つのセレクタが等価として処理されているかどうかをチェックします（永続サブスクリプションの再アクティブ化に便利です）。2 つ目のセレクタは、オプションの `<other-selector>` で指定します。
`knobs`	使用可能なすべてのシステム・プロパティ（表 4-6 を参照）と OC4J JMS サーバーの現在の設定を表示します。
`stats`	OC4J JMS サーバーの使用可能なすべての DMS 統計を表示します（JMS 以外の統計も含まれます）。（DMS の詳細は、『Oracle Application Server パフォーマンス・ガイド』を参照してください。）
`destinations`	OC4J JMS が認識しているすべての永続宛先オブジェクトのリストを出力します。
`durables`	OC4J JMS が認識しているすべての永続サブスクリプションのリストを出力します。
`subscribe <topic>`	<topic> に永続サブスクリプションを作成します。名前、メッセージ・セレクタ、ローカルであるかどうか、および永続サブスクリプションのクライアント識別子を指定します。これにより、既存のアクティブでない永続サブスクリプションが置き換えられます。名前は、`-name` コマンド・オプションを使用して特定します。メッセージ・セレクタは、`-selector` コマンド・オプションを使用して特定します。永続サブスクリプションがローカルであるかどうかは、`-noLocal` コマンド・オプションを使用して特定します。クライアント識別子は、`-clientID` 汎用オプションを使用して定義します。
`unsubscribe`	既存のアクティブでない永続サブスクリプションを削除します。永続サブスクリプションは、名前（`-name` コマンド・オプション）およびクライアント識別子（`-clientID` 汎用オプション）で特定します。
`browse <destination>`	指定された宛先（`jms.xml` に定義されているキューまたはトピックの永続サブスクリプション）のメッセージを参照します。
`drain <destination>`	指定された宛先（キューまたはトピックの永続サブスクリプション）のメッセージをデキューします。
`copy <from-destination> <to-destination>`	ある宛先（キューまたはトピックの永続サブスクリプション）から別の宛先にメッセージをコピーします。発信側と受信側の宛先が同一の場合はコマンドは実行されず、かわりにエラーが生成されます。

表 4-3 JMS ユーティリティ・コマンド（続き）

コマンド	説明
move <from-destination> <to-destination>	ある宛先（キューまたはトピックの永続サブスクリプション）から別の宛先にメッセージを移動します。発信側と受信側の宛先が同一の場合はコマンドは実行されず、かわりにエラーが生成されます。

表 4-4 JMS ユーティリティ・コマンドのオプション

コマンド・オプション	説明
-selector <selector>	指定された JMS メッセージ・セレクタを使用して、キュー・レシーバおよび永続サブスクライバを作成します。
-noLocal [true|false]	true に設定されている場合、サブスクライバは同じ接続でパブリッシュされたメッセージを参照できません。永続サブスクライバの作成時に使用します。デフォルト値は、false です。
-name <name>	トピックで実行中の永続サブスクリプションの名前を定義します。このオプションは、トピックの読取りを行うコマンドでは必須で、キューの読取りでは無視されます。
-silent	処理中にはメッセージを出力しません。標準エラーと出力された、処理済のメッセージの合計件数を保持します。
-count <count>	現行の操作中には、指定された数を超えるメッセージは処理しません。数値が負の値またはゼロの場合は、選択されたすべてのメッセージが処理されます。

次の例では、クライアント・ユーティリティと同じコンピュータに配置されている OracleAS JMS サーバーに接続して例外キューを参照し、このキューで処理されたメッセージの数を戻します。

```
java com.evermind.server.jms.JMSServerUtils -username <username> -password
 <password> browse jms/Oc4jJmsExceptionQueue
```

トピックのリスニング

トピックをリスニングするには、まず（JMSUtils を使用して）永続サブスクリプションを設定し、（所有する任意のパブリッシャを使用して）いくつかのメッセージをパブリッシュし、（JMSServerUtils で参照して）これらのメッセージを取得します。

トピックをリスニングするには、次のようにします。

1. 永続サブスクリプションを設定します。

   ```
   java com.evermind.server.jms.JMSServerUtils -username oc4jadmin -password welcome1
   -port 9127 -clientID demedclient subscribe -name demedjmsutils "Demo Topic"
   ```

2. そのトピックでいくつかのメッセージをパブリッシュします。

3. 次のようにしてメッセージを参照します。

   ```
   java com.evermind.server.jms.JMSServerUtils -username oc4jadmin -password welcome1
   -port 9127 -clientID demedclient browse -name demedjmsutils "Demo Topic"
   ```

サブスクリプションと参照の両方で、同じ clientID および name を使用します。これらは JMS の基本的なオプションで、clientID はコネクション・ファクトリを、name はそのコネクション・ファクトリの特定のサブスクリプションを指定します。

JMS MBean の使用

OEMS JMS 実装は、OEMS との通信に使用される一連の JMX MBean に含まれており、JMS 宛先のリスト作成や参照などのタスクを実行します。MBean には、Oracle Enterprise Manager 10*g* コンソールを使用してアクセスします。MBean を介して使用可能な属性や操作の多くは、コマンドライン・ユーティリティからも使用できます。コマンドライン・ユーティリティの使用方法の詳細は、4-26 ページの「JMS ユーティリティの使用」を参照してください。

次の MBean には、Oracle Enterprise Manager 10*g* コンソールを使用してアクセスします。

- `JMSAdministrator` MBean へのパス：

  「OC4J: ホーム」→「管理」タブ→「タスク名 : JMX」→「システム MBean ブラウザ」→「タスクに移動」→「J2EEDomain:oc4j」、「J2EEServer:standalone」、「JMSAdministratorResource」、「JMSAdministrator」にドリルダウン

- `JMS` MBean へのパス：

  「OC4J: ホーム」→「管理」タブ→「タスク名 : JMX」→「システム MBean ブラウザ」→「タスクに移動」→「J2EEDomain:oc4j」、「J2EEServer:standalone」、「JMSResource」、「JMS」にドリルダウン

- 様々な JMSDestinationResource MBean へのパス：

  「OC4J: ホーム」→「管理」タブ→「タスク名 : JMX」→「システム MBean ブラウザ」→「タスクに移動」→「J2EEDomain:oc4j」、「J2EEServer:standalone」、「JMSResource」、「JMS」、「JMSDestinationResource」にドリルダウン→希望の宛先を示す MBean を選択

表 4-5 JMS MBean 参照

MBean 実装	コマンドラインの等価	説明
`JMSAdministrator` MBean の `configProperties` 属性	`knobs`	使用可能なすべてのシステム・プロパティ（表 4-6 を参照）と現在の設定を表示します。
`JMS` MBean の「統計」タブ	`stats`	`JMS` MBean を通じて、次の OEMS JMS のメモリー内およびファイル・ベースの統計を使用できます。 ■ `activeHandlers` ■ `activeConnections` ■ `pendingMessageCount` ■ `messageDequeued` ■ `messageExpired` ■ `messageCommitted` ■ `messageRolledBack` ■ `messageEnqueued` ■ `messageRecovered` ■ `messageDiscarded` ■ `messagePagedIn` ■ `messageCount`
`JMSAdministrator` Mbean の `validateSelector` 操作	`check <selector>`	指定された JMS メッセージ・セレクタの妥当性をチェックします。この操作には、引数 `selector` が含まれます。
`JMSAdministrator` Mbean の `areSelectorsEqual` 操作	`check <sel1> <sel2>`	指定された 2 つのセレクタを等価に扱えるかどうかをチェックします。これは、永続サブスクリプションを再アクティブ化する際に役立ちます。 この操作には、引数 `sel1` および `sel2` が含まれます。

表 4-5 JMS MBean 参照（続き）

MBean 実装	コマンドラインの等価	説明
ドメインが topic であるすべての JMSDestinationResource MBean の subscribe 操作	subscribe	宛先に永続サブスクリプションを作成します。これにより、既存のアクティブでない永続サブスクリプションが置き換えられます。 この操作には、次の引数が含まれます。 ■ name: 永続サブスクライバの名前です。 ■ noLocal: true の場合、サブスクライバは、独自の接続によってパブリッシュされるメッセージの配信を抑止できます。 ■ xact: true の場合、セッションがトランザクション処理されます。 ■ clientId: クライアント ID です。 ■ selector: メッセージ・セレクタです。
ドメインが topic であるすべての JMSDestinationResource MBean の unsubscribe 操作	unsubscribe	永続サブスクリプションを削除します。 この操作には、次の引数が含まれます。 ■ name: 永続サブスクライバの名前です。 ■ xact: true の場合、セッションがトランザクション処理されます。 ■ clientId: クライアント ID です。
すべての JMSDestinationResource Mbean の browse 操作	browse	この宛先を参照します。 この操作には、次の引数が含まれます。 ■ sub: 永続サブスクライバの名前であり、ドメインが topic である MBean でのみ使用可能です。 ■ xact: true の場合、セッションがトランザクション処理されます。 ■ clientId: クライアント ID です（オプション）。 ■ selector: メッセージ・セレクタです（オプション）。 ■ count: 処理するメッセージの最大数です（すべてを処理する場合は 0）。
すべての JMSDestinationResource Mbean の copy 操作	copy	メッセージをこの宛先から指定された宛先にコピーします。 この操作には、次の引数が含まれます。 ■ sub: 永続サブスクライバの名前であり、ドメインが topic である MBean でのみ使用可能です。 ■ toDestination: メッセージの移動先となる宛先です。 ■ xact: true の場合、セッションがトランザクション処理されます。 ■ clientID: クライアント ID です（オプション）。 ■ selector: メッセージ・セレクタです（オプション）。 ■ count: 処理するメッセージの最大数です（すべてを処理する場合は 0）。

表 4-5 JMS MBean 参照（続き）

MBean 実装	コマンドラインの等価	説明
すべての JMSDestinationResource Mbean の drain 操作	drain	この宛先からメッセージを排出します。 この操作には、次の引数が含まれます。 ■ sub: 永続サブスクライバの名前であり、ドメインが topic である MBean でのみ使用可能です。 ■ xact: true の場合、セッションがトランザクション処理されます。 ■ clientId: クライアント ID です（オプション）。 ■ selector: メッセージ・セレクタです（オプション）。 ■ count: 処理するメッセージの最大数です（すべてを処理する場合は 0）。
すべての JMSDestinationResource Mbean の move 操作	move	メッセージをこの宛先から指定された宛先に移動します。 この操作には、次の引数が含まれます。 ■ sub: 永続サブスクライバの名前であり、ドメインが topic である MBean でのみ使用可能です。 ■ toDestination: メッセージの移動先となる宛先です。 ■ xact: true の場合、セッションがトランザクション処理されます。 ■ clientID: クライアント ID です（オプション）。 ■ selector: メッセージ・セレクタです（オプション）。 ■ count: 処理するメッセージの最大数です（すべてを処理する場合は 0）。

ファイル・ベースの永続性の構成

次の項では、ファイル・ベースの永続性について説明します。

- Application Server Control コンソールでのファイル・ベースの永続性の有効化
- jms.xml ファイルでのファイル・ベースの永続性の有効化
- 永続性のリカバリ

ファイル・ベースの永続性が有効の場合、OC4J では、次の操作が自動的に実行されます。

- 永続性ファイルが存在しない場合、OC4J により、自動的にファイルが作成されて適切なデータで初期化されます。
- 永続性ファイルが存在しているが空の場合、OC4J により、適切なデータで初期化されます。

注意： OC4J サーバーがアクティブなときに、永続性ファイルのコピー、削除または名前変更を実行しないでください。このような操作を実行すると、データが破損してメッセージが消失する可能性があります。

OC4J がアクティブではないときに、永続性ファイルを削除すると、その永続性ファイルに関連付けられている宛先からすべてのメッセージおよび永続サブスクリプションが削除されます。ファイルは、OC4J の再起動時に、JMS サーバーにより通常の方法で再度初期化されます。

詳細は、4-34 ページの「永続性ファイルの管理」を参照してください。

永続性が有効化されていても、ファイルに残るのは特定のメッセージのみです。メッセージが永続化されるためには、次の条件がすべて満たされている必要があります。

- Application Server Control コンソールで永続性ファイルを指定するか、jms.xml ファイルで宛先の persistence-file 属性を設定することにより、宛先オブジェクトの永続化を定義していること。
- メッセージに PERSISTENT 配信モードが設定されていること。これは、デフォルト・モードです。

  非永続配信モード（DeliveryMode.NON_PERSISTENT として定義される）が設定されたメッセージが永続的な宛先に送信されても、それらのメッセージは永続化されません。
- 宛先がキューであること。または、宛先がトピックであり、コンシューマが永続サブスクライバであること。

DeliveryMode を PERSISTENT または NON_PERSISTENT に設定する方法は、JMS 仕様を参照してください。

メッセージ・プロデューサにデフォルトの DeliveryMode を設定する方法は、http://java.sun.com/j2ee/1.4/docs/api/javax/jms/MessageProducer.html#setDeliveryMode(int) を参照してください。

デフォルト設定を上書きしてメッセージごとに DeliveryMode を設定する方法は、http://java.sun.com/j2ee/1.4/docs/api/javax/jms/MessageProducer.html#send(javax.jms.Destination,%20javax.jms.Message,%20int,%20int,%20long) および http://java.sun.com/j2ee/1.4/docs/api/javax/jms/MessageProducer.html#send(javax.jms.Message,%20int,%20int,%20long) を参照してください。

ファイル・ベースの永続性を有効化する際の注意

前述の条件が満たされる場合、ファイル・ベースの永続性オプションにより、メッセージ用にリカバリ可能な永続記憶域が提供されます。各宛先は、宛先オブジェクトに送信されたメッセージを格納するためのファイルを指し示す相対または絶対パス名に関連付けることができます。ファイルは、ファイル・システムの任意の場所に配置できます（必ずしも *J2EE_HOME* ディレクトリ内である必要はありません）。同じディレクトリに複数の永続性ファイルを配置できます。永続性ファイルは、リモート・ネットワークのファイル・システムに配置するか、ローカル・ファイル・システムの一部とすることが可能です。

Application Server Control コンソールでのファイル・ベースの永続性の有効化 Application Server Control コンソールは、宛先オブジェクト用にファイル・ベースの永続性を有効化するための主要ツールです。次のパスを使用して、Application Server Control コンソールで永続性ファイルのパラメータを指定します。

Application Server Control コンソールで宛先の永続性ファイルを指定する場合のパス：

「OC4J: ホーム」→「管理」タブ→「タスク名 : サービス」→「JMS プロバイダ」→「タスクに移動」→「宛先」→「新規作成」→「永続性ファイル」

永続性ファイルは、新規の宛先を作成するときに Application Server Control コンソールで指定できます。既存の宛先の永続性ファイルの指定は、コンソールでは変更できません。永続性の指定は、jms.xml ファイルで変更できます。4-33 ページの「jms.xml ファイルでのファイル・ベースの永続性の有効化」を参照してください。

jms.xml ファイルでのファイル・ベースの永続性の有効化 宛先オブジェクトのファイル・ベースの永続性は、jms.xml ファイルの persistence-file 属性を指定することで有効化できます。

次の XML 構成の例に、persistence-file 属性を使用してファイル名を pers と定義する方法を示します。

```
<queue name="foo" location="jms/persist" persistence-file="pers">
</queue>
```

persistence-file 属性のパスは、ファイルの絶対パス、または application.xml に定義されている persistence ディレクトリに対する相対パスです。

OC4J サーバーは、永続性ファイル用のディレクトリを一切作成しません。そのため、永続性ファイルを jms.xml に定義する場合、次のように既存の絶対ディレクトリを指定する必要があります。

```
persistence-file="/this/dir/exists/PersistenceFile"
```

または、次のようにファイル名のみを指定します。

```
persistence-file="PersistenceFile"
```

ファイル名のみを指定する場合、永続性ファイルは、デフォルトで $J2EE_HOME/persistence（スタンドアロン・インスタンスの場合）または $J2EE_HOME/persistence/<group_name>（完全な Oracle Application Server 環境の場合）に作成されます。

persistence-file 属性の詳細は、4-19 ページの表 4-1「構成要素」を参照してください。

Oracle Application Server では、複数の OC4J インスタンスが同じファイル・ディレクトリに書き込む場合があり、その際に同じ永続性ファイル名を指定する場合もあります。この属性を設定するとファイル・ベースの永続性が有効化されますが、永続性ファイルが他の OC4J インスタンスにより上書きされる可能性もあります。

永続性のリカバリ 次の項では、永続性のリカバリの様々な側面について説明します。

- リカバリ可能性の有効範囲
- 永続性ファイルの管理
- JMS クライアントへのエラーのレポート
- リカバリ手順

リカバリ可能性の有効範囲

OEMS JMS のファイル・ベースの永続性オプションでは、発生する可能性のある一部の障害からのリカバリが可能ですが、すべての障害からリカバリできるわけではありません。次のいずれかの障害が発生した場合、永続性ファイルのリカバリ可能性は保証されません。

- メディアの破損 : 永続性ファイルを保持するディスク・システムが異常終了したか破損した場合。
- 外部的な破損 : 永続性ファイルが、削除、編集、変更されるか、（ソフトウェアによって）他の方法で破壊された場合。永続性ファイルへの書込みは、JMS サーバーにのみ許可する必要があります。
- エラーを戻さない失敗または破損 : JDK の I/O メソッドが、エラーを戻さずに失敗したか、読取りまたは書込み中のデータを破壊してエラーを戻さない場合。
- java.io.FileDescriptor.sync() の失敗 : sync() コールが、指定のディスクリプタに関連付けられているファイル・バッファすべてを、基礎となるファイル・システムに正常かつ完全にフラッシュしない場合。

永続性ファイルの管理

JMS サーバーの稼働中は、現在使用中の永続性ファイルのコピー、削除または名前変更を実行しないでください。使用中の永続性ファイルに対してこれらのアクションを実行すると、リカバリ不可能なエラーが発生します。

ただし、OEMS サーバーで永続性ファイルが使用されていない場合は、これらのファイルに対して次の管理およびメンテナンス操作を実行できます。

- 削除 : 永続性ファイルを削除すると、すべてのメッセージが削除され、トピックの場合はすべての永続サブスクリプションが削除されます。起動時に、OEMS JMS により新しい空の永続性ファイルが初期化されます。
- コピー : 既存の永続性ファイルをアーカイブまたはバックアップのためにコピーできます。既存の永続性ファイルが破損した場合に、(OEMS JMS の宛先名とファイル間の関連付けが維持されていれば) 適切なパス名によって示される以前のバージョンを使用して、JMS 宛先の以前の内容に戻すことができます。

永続性ファイルの連結、分割、並替え、マージはできません。このような操作を実行すると、永続性ファイル内のデータが破損し、リカバリ不可能になります。

OEMS JMS のメモリー内およびファイル・ベースのオプションでは、内部構成およびトランザクション状態管理のために、ユーザー指定の永続性ファイルとロック・ファイルに加えて特殊ファイル `jms.state` が使用されます。OEMS JMS サーバーは、通常の操作中にこのファイルとその内容をクリーン・アップします。アーカイブを目的とする場合でも、このファイルは削除、移動、コピーまたは変更しないでください。`jms.state` ファイルを操作すると、メッセージとトランザクションが消失する可能性があります。

注意 : `jms.state` ファイルの位置は、OC4J の運用モード (スタンドアロンまたは Oracle Application Server) に応じて次のように異なります。

- スタンドアロン : *`J2EE_HOME`*`/persistence` ディレクトリ
- Oracle Application Server: *`J2EE_HOME`*`/persistence/<group_name>` ディレクトリ

persistence ディレクトリの位置は、`application.xml` ファイルで定義されます。

JMS クライアントへのエラーのレポート

JMS クライアントがメッセージをエンキューまたはデキューしたり、トランザクションをコミットまたはロールバックするときの操作順序は、次のとおりです。

1. クライアントがファンクション・コールを実行します。
2. 事前永続性操作が行われます。
3. 永続性が発生します。
4. 事後永続性操作が行われます。
5. クライアントのファンクション・コールが戻ります。

事前永続性フェーズまたは永続性フェーズで障害が発生すると、クライアントは、`JMSException` や他のなんらかのタイプのエラーを受け取りますが、永続性ファイルは変更されません。

事後永続性フェーズで障害が発生すると、クライアントは、`JMSException` や他のなんらかのタイプのエラーを受け取ります。ただし、永続性ファイルはそのまま更新され、OEMS JMS は操作が成功した場合と同様にリカバリします。

異常終了

OC4J が正常終了すると、ロック・ファイルが自動的にクリーン・アップされます。ただし、（`kill -9` コマンドなどにより）OC4J が異常終了すると、ロック・ファイルはファイル・システムに残ります。通常、残されたロック・ファイルは、OC4J に認識されます。認識されない場合は、異常終了してから OC4J を再起動する前に、ロック・ファイルを手動で削除する必要があります。

ロック・ファイルのデフォルトの位置は、persistence ディレクトリです（*`J2EE_HOME`*`/persistence`）。persistence ディレクトリは、`application.xml` ファイルで定義されます。その他の位置は、宛先オブジェクトの `persistence-file` 属性内で設定できます。

リカバリ手順 ロック・ファイルは、複数の OC4J プロセスが同じ永続性ファイルに書き込まないようにします。複数の OC4J JVM が、同じ位置の同じ永続性ファイルを指し示すように構成されていると、データが相互に上書きされ、永続化された JMS メッセージが破損または消失する可能性があります。このような共有違反を防止するため、OEMS JMS では、各永続性ファイルがロック・ファイルに関連付けられます。そのため、各永続性ファイル（`/path/to/persistenceFile` など）は、`/path/to/persistenceFile.lock` というロック・ファイルに関連付けられます。永続性ファイルの詳細は、4-31 ページの「ファイル・ベースの永続性の構成」を参照してください。

OC4J には、ロック・ファイルを作成および削除するための適切なパーミッションが必要です。

終了後に再起動した場合、次のいずれかのロック・ファイル操作が適用されます。

- 正常終了 : ロック・ファイルは、自動的にクリーン・アップされます。再起動は通常どおり進行します。
- 異常終了 : 再起動時に、ロック・ファイルが認識されます。再起動は通常どおり進行します。
- 異常終了 : 再起動時に、共有違反を示すクリティカル・メッセージが配信されます。再起動は継続できません。エラー・メッセージに示されたロック・ファイルを削除し、再起動してください。複数のロック・ファイルが関連している場合、それぞれのファイルを削除する必要があります。

JMS 永続性のロック・ファイルには、サーバーおよび persistence ディレクトリの位置情報が含まれています。JMS サーバーの起動時にロック・ファイルが存在し、ロック・ファイルが（同じ IP アドレスを持つ）同じサーバーで、同じ persistence ディレクトリの位置を使用して作成されている場合、JMS サーバーは、そのロック・ファイルの制御を継承して正常に起動します。

これ以降に記載する OEMS JMS のファイル・ベースのリカバリ手順では、関連するすべてのロック・ファイルが削除されていると仮定して説明を続けます。

OEMS JMS は、異常終了時に OEMS JMS で構成されていたすべての永続性ファイルに対してリカバリ操作を実行します。つまり、OC4J が異常終了し、ユーザーが JMS サーバーの構成を変更して OC4J を再起動した場合、JMS サーバーは、元の構成に存在していたすべての永続性ファイルをリカバリしようとします。JMS サーバーは、リカバリの成功後に、指定された新しい構成に移行します。

古い構成のリカバリに失敗すると、JMS サーバーは起動しません。サーバーは停止されるか、またはリカバリに成功するまで繰り返し再起動されます。

JMS サーバーは、`jms.state` ファイルに現行の永続性構成をキャッシュします。このファイルは、トランザクション状態の維持にも使用されます。現行構成を完全にリカバリしない場合は、`jms.state` ファイルとすべてのロック・ファイルを削除し、構成の変更を受け入れることで、サーバーを白紙の状態で起動できます。（この方法はお薦めしません。）`jms.state` ファイルが見つからない場合は、JMS サーバーにより新規作成されます。

なんらかの理由で `jms.state` ファイル自体が破損した場合は、このファイルを削除する必要があります。これに伴い、保留中のすべてのトランザクション（コミットされたが、そのトランザクションに参加する個々の宛先オブジェクトすべてによってコミットされていないトランザクション）は消失します。

異常終了時にメッセージ・アクティビティが進行中だった場合、OEMS JMS は、永続性ファイルのリカバリを試行します。データの（前述したタイプの）破損は、破損データのクリーン・アウトにより処理されますが、これによりメッセージとトランザクションが消失する可能性があります。

永続性ファイルのヘッダーが破損すると、この種の破損ファイルはユーザー構成エラーと区別できないことが多いため、OEMS JMS ではファイルをリカバリできなくなる場合があります。`oc4j.jms.forceRecovery` 管理プロパティ（4-38 ページの表 4-6「システム・プロパティ」を参照）を使用すると、（メッセージの消失やユーザー構成エラーの隠蔽というデメリットはありますが）無効なデータをすべて消去してリカバリを続行するように JMS サーバーに指示できます。

事前定義済の例外キュー

OEMS JMS のメモリー内およびファイル・ベースのオプションには、JMS 仕様の拡張機能として、配信できないメッセージを処理するための事前定義済の例外キューが付属しています。これは、単一の永続的なグローバル例外キューであり、OEMS JMS の宛先から配信できないメッセージが格納されます。例外キューには、次の固定プロパティがあります。

- 固定の名前：`jms/Oc4jJmsExceptionQueue`
- 固定の JNDI ロケーション：`jms/Oc4jJmsExceptionQueue`
- 固定の永続性ファイル：`Oc4jJmsExceptionQueue`

注意： `Oc4jJmsExceptionQueue` 永続性ファイルの位置は、OC4J の運用モード（スタンドアロンまたは Oracle Application Server）に応じて次のように異なります。

- スタンドアロンのディレクトリ：`J2EE_HOME/persistence`
- Oracle Application Server のディレクトリ：`J2EE_HOME/persistence/<group_name>`

`persistence` ディレクトリの位置は、`application.xml` ファイルで定義されます。

例外キューは、JMS サーバーとそのクライアントで常に使用可能です。`jms.xml` 構成ファイルで明示的に定義することはできません。定義しようとすると、エラーが発生します。例外キューの名前、JNDI ロケーション、および persistence ディレクトリへのパス名は、各ネームスペースにおける予約語になります。これらの名前で他のエンティティを定義しようとすると、エラーが発生します。

メッセージは、期限切れまたはリスナー・エラーが原因で配信できなくなることがあります。次の項では、期限切れで配信できないメッセージに対して実行される操作について説明します。

メッセージの期限切れ デフォルトでは、永続的な宛先に送信されたメッセージが期限切れになると、そのメッセージは例外キューに移動されます。期限切れになったメッセージの `JMSXState` は、`EXPIRED` を示す値である `3` に設定されますが、メッセージ・ヘッダー、プロパティおよびボディは特に変更されません。メッセージは `ObjectMessage` にラップされ、ラップされたメッセージが例外キューに送られます。

ラップしている `ObjectMessage` の `DeliveryMode` は、元のメッセージと同じです。

デフォルトでは、非永続的または一時的な宛先オブジェクトで期限切れになったメッセージは、例外キューに移動されません。通常、これらの宛先オブジェクトに送信されたメッセージは、永続化の必要がないと判断され、期限切れバージョンも存在しません。

送信先となる宛先オブジェクトが永続的、非永続的、一時的のいずれであるかにかかわらず、期限切れのメッセージをすべて例外キューに送信するよう指定できます。これには、OC4J サーバーの起動時に、`oc4j.jms.saveAllExpired` 管理プロパティを `true` に設定します（4-38 ページの表 4-6「システム・プロパティ」を参照）。この場合、期限切れのメッセージはすべて例外キューに移動します。これにより、非永続的なメッセージが例外キューに送られた場合でも、その非永続的な性質は変化しません。

メッセージのページング

OEMS JMS のメモリー内およびファイル・ベースのオプションでは、次の場合にメッセージ本文のページング・インおよびページング・アウトがサポートされます。

- メッセージが永続的な配信モードに設定されている場合
- メッセージが永続的な宛先オブジェクトに送信される場合（4-31 ページの「ファイル・ベースの永続性の構成」を参照）
- 宛先がキューである場合。または、宛先がトピックであり、コンシューマが永続サブスクライバである場合
- OC4J サーバーの JVM のメモリー使用量がユーザー定義のしきい値を超えた場合

ページングされるのはメッセージ本文のみです。メッセージ・ヘッダーとプロパティは、ページングされません。ページングしきい値は、システム・プロパティの `oc4j.jms.pagingThreshold` を使用して設定できます（4-38 ページの表 4-6「システム・プロパティ」を参照）。

しきい値の範囲は、`0.0` ～ `1.0` です。JVM メモリーを使用しない Java プログラムを記述するのはまず不可能であり、プログラムは、JVM ヒープがいっぱいになる前にメモリーをすべて消費して終了するのが普通です。

たとえば、ページングしきい値が `0.5` の場合に JVM のメモリー使用率が `0.6` に上昇すると、JMS サーバーは、メモリー使用率がしきい値を下回るまで、またはメッセージ本文をそれ以上ページング・アウトできなくなるまで、可能なかぎり多くのメッセージ本文をページング・アウトしようとします。

ページング・アウトされたメッセージが JMS クライアントからリクエストされると、JMS サーバーは、（JVM のメモリー使用率とは関係なく）そのメッセージ本文を自動的にページング・インし、正しいメッセージ・ヘッダーと本文をクライアントに配信します。クライアントに配信されたメッセージは、サーバー JVM でのメモリー使用率に応じて再びページング・アウトの対象とみなすことができます。

メモリー使用率がページングしきい値を下回ると、JMS サーバーは、メッセージ本文のページング・アウトを停止します。すでにページング・アウトされていたメッセージ本文が自動的にページング・インされることはありません。メッセージ本文のページング・インは、必要な場合（つまり、メッセージがデキューされるかクライアントにより参照される場合）にのみ発生します。

デフォルトでは、ページングしきい値は `1.0` に設定されます。つまり、デフォルトでは、メッセージ本文はページングされません。

ページングしきい値として適切な値は、JMS アプリケーション、送受信するメッセージのサイズ、および実際の使用例における試行結果とメモリー使用率の監視結果に応じて選択する必要があります。

ページングしきい値には、正しい値を指定する必要があります。JMS のセマンティクスは、ページングが有効であるかどうかにかかわらず常に維持されます。ページングしきい値の設定により、JMS サーバーは、ページングを使用しない場合よりも多くのメッセージをメモリー内で処理できます。

JMS 構成プロパティ

OEMS JMS のメモリー内およびファイル・ベースのオプションと JMS クライアントの実行時構成は、JVM システム・プロパティを通じて管理します。これらのプロパティは、JMS の基本機能には影響しません。これらのプロパティは、JMS サーバーに固有の機能、拡張機能およびパフォーマンスの最適化に関係します。knobs コマンドライン・コマンドを使用すると、これらのプロパティを確認できます。

実行時に構成プロパティを編集するための主要ツールは、`JMSAdministrator` MBean です。

別の方法として、スタンドアロン環境では、次のようにコマンドライン引数で構成プロパティを渡すことができます。

```
java -D<propertyname>=<value>
```

これらのプロパティ設定は、`jms.xml` ファイルに永続化されます。

JMSAdministratorResource MBean の JMS 構成プロパティ設定へのパス：

「OC4J: ホーム」→「管理」タブ→「タスク名 : JMX」→「システム MBean ブラウザ」→「タスクに移動」→「J2EEDomain:oc4j」、「J2EEServer:standalone」、「JMSAdministratorResource」、「JMSAdministrator」にドリルダウン→「操作」タブ→ setConfigProperty

表 4-6 に、OEMS JMS リソース・プロバイダのメモリー内およびファイル・ベースのオプションのシステム・プロパティとその説明を示します。

表 4-6 システム・プロパティ

JVM システム・プロパティ	サーバー / クライアント	説明
`oc4j.jms.serverPoll`	JMS クライアント	JMS コネクションが OC4J サーバーを ping して、通信例外を例外リスナーにレポートする間隔（ミリ秒単位）。 型は long、デフォルトは `15000` です。
`oc4j.jms.messagePoll`	JMS クライアント	JMS コンシューマが新規メッセージの有無をチェックするまでに待機する最大間隔（ミリ秒単位）。 型は long、デフォルトは `1000` です。
`oc4j.jms.listenerAttempts`	JMS クライアント	メッセージが配信不可能と宣言されるまでの、リスナーによる配信試行回数。 このプロパティで配信試行回数が制限されるのは、`setMessageListener()` メソッドのいずれかを使用して JMS に登録された MessageListener によってメッセージが受信される場合のみです。受信メソッドのいずれかを使用してメッセージが受信される場合は（以前の MessageListener への配信試行によって、メッセージがすでに配信不可能であると宣言されている場合を除き）、配信試行回数は制限されません。JMS コネクタの場合、受信メソッドを使用して MDB 用にインバウンド・メッセージ機能を実装するため、このプロパティは適用されません。また、J2EE 1.4 におけるメッセージ・リスナーの使用に関する制限のため、このプロパティは、アプリケーション・クライアントにのみ適用されます。MDB でメッセージ配信試行を制限するには、`MaxDeliveryCnt` アクティブ化仕様プロパティを使用する必要があります。`orion-ejb-jar.xml` デモ・ファイルには、`MaxDeliveryCnt` プロパティに関するコメントが含まれます。 型は int、デフォルトは `5` です。
`oc4j.jms.maxOpenFiles`	OC4J サーバー	永続性ファイルのオープン・ファイル記述子の最大数。これは、オペレーティング・システムにより許可されているオープン・ファイル記述子の最大数より多くの永続的な宛先オブジェクトでサーバーを構成する場合に使用します。 型は int、デフォルトは `64` です。

表 4-6 システム・プロパティ（続き）

JVM システム・プロパティ	サーバー / クライアント	説明
oc4j.jms.saveAllExpired	OC4J サーバー	すべての宛先オブジェクト（永続的、非永続的および一時的）に存在する期限切れになったすべてのメッセージを例外キューに保存します。 型は boolean、デフォルトは false です。
oc4j.jms.socketBufsize	JMS クライアント	クライアント / サーバー通信に TCP/IP ソケットを使用する場合は、ソケットの I/O ストリームに指定されたバッファ・サイズを使用します。最小バッファ・サイズの 8KB が規定されます。クライアントとサーバーの間で送信されるメッセージのサイズが大きいほど、バッファ・サイズを大きくすると妥当なパフォーマンスが得られます。 型は int、デフォルトは 64*1024 です。
oc4j.jms.debug	JMS クライアント	true の場合、JMS クライアントと JMS サーバーで NORMAL イベントのトレースが有効化されます。すべてのログ・イベント（NORMAL、WARNING、ERROR および CRITICAL）が、stderr に送信され、（可能であれば）*J2EE_HOME*/log/server.log または *J2EE_HOME*/log/jms.log のいずれかにも送信されます。このプロパティを true に設定すると、通常は大量のトレース情報が生成されます。 型は boolean、デフォルトは false です。
oc4j.jms.noDms	JMS クライアント	true の場合、インスツルメンテーションが無効化されます。 型は boolean、デフォルトは false です。
oc4j.jms.forceRecovery	OC4J サーバー	true の場合、破損した永続性ファイルが強制的にリカバリされます。デフォルトでは、JMS サーバーは、ヘッダーが破損した永続性ファイルのリカバリを実行しません（一般的に、この状態は構成エラーと区別できないためです）。強制リカバリを実行すると、通常、JMS サーバーは正常に起動し、永続性ファイルと宛先オブジェクトが使用可能になります。 型は boolean、デフォルトは false です。
oc4j.jms.pagingThreshold	OC4J サーバー	JMS サーバーは、メモリー使用率がこの値を超えるとメッセージ本文のページングを開始します。この値は、JVM で使用中のメモリー量の見積です。この値の範囲は、0.0（プログラムでメモリーをまったく使用していない状態）～ 1.0（プログラムで使用可能なすべてのメモリーを使用している状態）です。 型は double、デフォルトは 1.0 です。 詳細は、4-37 ページの「メッセージのページング」を参照してください。

表 4-6 システム・プロパティ（続き）

JVM システム・プロパティ	サーバー / クライアント	説明
`oc4j.jms.pseudoTransaction Enlistment`	OC4J サーバー	このプロパティは下位互換性のためにのみ使用します。デフォルトでは無効化されています。`true` に設定すると、自動登録機能が有効化されます。この機能を使用すると、XA JMS 接続だけでなく XA JMS 以外の接続でも、ネイティブ JMS プロバイダによって OC4J グローバル・トランザクションに自動的に登録できます。この機能は、今後のリリースでは使用できなくなります。 XA または XA JMS 以外の接続を OC4J グローバル・トランザクションに登録するには、この構成プロパティは設定しないでください。かわりに次のようにします。 JMS API を使用して、OC4J-JMS の javax.jms.XA* 実装を使用している OC4J グローバル・トランザクションに XA 接続を明示的に登録します。また、XA JMS 以外の接続から作成された、指定の JMS セッションのローカル・トランザクションを明示的にコミットまたはロールバックします。

jms.xml ファイルでの JMS 構成プロパティの設定

次の例は、`jms.xml` ファイルで構成プロパティを設定する方法を示しています。

```
<config-properties>
  <config-property name="oc4j.jms.debug" value="true">
  </config-property>
</config-properties>
```

この例は、4-23 ページの「jms.xml を使用した構成」に記載されている `jms.xml` の一部です。このプロパティは、4-19 ページの表 4-1「構成要素」にも記載されています。

OEMS JMS のメモリー内およびファイル・ベースのリソース名

この項では、変数 *resourceName* を使用して、リソース・プロバイダ（RP）の JNDI コンテキスト内に存在する RP のリソース（コネクション・ファクトリまたは宛先）の JNDI ロケーションを示します。

OEMS JMS リソースの *resourceName* は、4-17 ページの「宛先オブジェクトおよびコネクション・ファクトリの構成」で説明したとおり、コンソールで JNDI ロケーションとして指定します。コネクション・ファクトリの *resourceName* の値は、特定の OEMS JMS コネクション・ファクトリを識別するために使用されます（図 4-1 の矢印番号 16 のリンク・キーを参照）。宛先の *resourceName* の値は、特定の OEMS JMS 宛先を識別するために使用されます（図 4-1 の矢印番号 13 のリンク・キーを参照）。コネクション・ファクトリおよび宛先の *resourceName* の値は、JMS コネクタが省略される場合にも使用されることがあります（4-13 ページの「アプリケーション・クライアントの JMS コネクタの省略」を参照）。

OEMS JMS のメモリー内およびファイル・ベースの直接ルックアップを使用するアプリケーション・クライアントに必要なクラスパス

OEMS JMS のメモリー内およびファイル・ベースのオプションをアプリケーション・クライアントから直接使用する場合、表 4-7「OEMS JMS のメモリー内およびファイル・ベースのルックアップに必要なクライアント・サイドの JAR ファイル」にリストされたクラスパスに各 JAR ファイルが含まれている必要があります。

表 4-7 OEMS JMS のメモリー内およびファイル・ベースのルックアップに必要なクライアント・サイドの JAR ファイル

JAR	ORACLE_HOME パス
`oc4jclient.jar`	`/j2ee/<instance>`
`jta.jar`	`/j2ee/<instance>/lib`
`jms.jar`	`/j2ee/<instance>/lib`
`jndi.jar`	`/j2ee/<instance>/lib`
`javax77.jar`	`/j2ee/<instance>/lib`
`optic.jar`	`/opmn/lib`
（`opmn:ormi` 接頭辞が Oracle Application Server 環境で使用される場合にのみ必須）	

OEMS JMS のデータベースの永続性

OEMS JMS のデータベースの永続性オプションは、Oracle Database Streams Advanced Queuing（AQ）機能への JMS インタフェースです。この項では、AQ を永続的なメッセージ・ストアとして使用する OEMS JMS の構成方法と使用方法について詳細に説明します。

OEMS JMS のメモリー内およびファイル・ベースのオプションと同様に、OEMS JMS のデータベースのオプションを通じて AQ にアクセスする場合は、アプリケーションで JMS コネクタを使用することをお薦めします。

AQ を使用して OEMS JMS を構成する方法の詳細は、『Oracle Streams アドバンスト・キューイング・ユーザーズ・ガイドおよびリファレンス』を参照してください。

この項には、次の項目が含まれます。

- OEMS JMS のデータベース・オプションの使用方法
- Oracle Application Server および Oracle データベースと組み合せた OEMS JMS のデータベース・オプションの使用方法

注意：

以前、Oracle Application Server のドキュメントや関連資料、および『Oracle Streams アドバンスト・キューイング・ユーザーズ・ガイドおよびリファレンス』では、OEMS JMS のデータベースの永続性オプションのことを OJMS と記載していました。OJMS という頭字語は、OEMS JMS のデータベースの永続性オプションを示しています。

OEMS JMS のデータベース・オプションの使用方法

OEMS JMS のデータベース・オプションで、リソース・プロバイダの宛先オブジェクト（キューおよびトピック）を作成してアクセスする手順は、次のとおりです。

- データベースに OEMS JMS をインストールして構成します。4-42 ページの「OEMS JMS のデータベース・オプションのインストールと構成」を参照してください。
- データベースで、RDBMS ユーザーを作成します。JMS アプリケーションでは、RDBMS ユーザーをバックエンド・データベースに接続し、権限を割り当てます。4-43 ページの「ユーザーの作成と権限の割当て」を参照してください。
- JMS 宛先オブジェクトを作成します。4-43 ページの「OEMS JMS のデータベース・オプションの宛先オブジェクトの作成」を参照してください。
- 必要に応じて、データソースまたは LDAP ディレクトリ・エントリを作成します。4-44 ページの「OEMS JMS のデータベース参照の宣言」を参照してください。
- OC4J の XML 構成で、バックエンド・データベースの情報を使用して、`orion-application.xml` ファイルの `<resource-provider>` 要素に OEMS JMS のデータベース・オプションを定義します。
- JNDI ルックアップを介して実装内のリソースにアクセスします。4-24 ページの「JMS メッセージの送受信」を参照してください。

OEMS JMS のデータベース・オプションのインストールと構成　管理者または DBA は、『Oracle Streams アドバンスト・キューイング・ユーザーズ・ガイドおよびリファレンス』および共通のデータベース・マニュアルに従って、OEMS JMS をインストールする必要があります。OEMS JMS のインストールおよび構成後に、追加の構成を適用します。これには、次が含まれます。

1. 管理者または DBA は、JMS クライアントからデータベースへの接続に使用する RDBMS ユーザーを作成する必要があります。このユーザーに、OEMS JMS 操作を実行するための適切なアクセス権限を付与します。OEMS JMS により、データベース・ユーザーは、適切なアクセス権限があれば任意のスキーマのキューにアクセスできます。4-43 ページの「ユーザーの作成と権限の割当て」を参照してください。
2. 管理者または DBA は、JMS 宛先オブジェクトをサポートするための表およびキューを作成する必要があります。4-43 ページの「OEMS JMS のデータベース・オプションの宛先オブジェクトの作成」を参照してください。

注意： 次の各項では、キュー、トピック、表の作成と、権限の割当てを実行する SQL を使用します。

これらの使用例は、`http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html` を参照してください。

ユーザーの作成と権限の割当て JMS クライアントからデータベースへの接続に使用する RDBMS ユーザーを作成します。このユーザーに、OEMS JMS 操作を実行するためのアクセス権限を付与します。必要な権限は、必要な機能に応じて異なります。各種機能に必要な権限の詳細は、『Oracle Streams アドバンスト・キューイング・ユーザーズ・ガイドおよびリファレンス』を参照してください。

次の例では、jmsuser を作成します。このユーザーは、固有のスキーマ内で作成し、OEMS JMS 操作に必要な権限を付与する必要があります。次の文を実行するには、SYS DBA であることが必要です。

```
DROP USER jmsuser CASCADE ;

GRANT connect,resource,AQ_ADMINISTRATOR_ROLE TO jmsuser
   IDENTIFIED BY jmsuser ;
GRANT execute ON sys.dbms_aqadm  TO  jmsuser;
GRANT execute ON sys.dbms_aq     TO  jmsuser;
GRANT execute ON sys.dbms_aqin   TO  jmsuser;
GRANT execute ON sys.dbms_aqjms  TO  jmsuser;

connect jmsuser/jmsuser;
```

ユーザーの必要に応じて、2 フェーズ・コミット権限やシステム管理権限など、他の権限の付与が必要になる場合があります。2 フェーズ・コミット権限の詳細は、第 3 章「OC4J トランザクション・サポート」を参照してください。

OEMS JMS のデータベース・オプションの宛先オブジェクトの作成 DBMS_AQADM パッケージの詳細と、OEMS JMS のデータベース・オプションのメッセージ・タイプは、『Oracle Streams アドバンスト・キューイング・ユーザーズ・ガイドおよびリファレンス』を参照してください。

> **注意：** これらの使用例は、
> http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html を参照してください。

次の例では、OEMS JMS のデータベースにキューとトピックを作成します。

1. OEMS JMS の宛先（キューまたはトピック）を処理する表を作成します。

   トピックとキューの両方でキュー表を使用します。次の SQL の例では、キュー用に 1 つの表（demoTestQTab）を作成します。

   ```
   DBMS_AQADM.CREATE_QUEUE_TABLE(
          Queue_table            => 'demoTestQTab',
          Queue_payload_type     => 'SYS.AQ$_JMS_MESSAGE',
          sort_list => 'PRIORITY,ENQ_TIME',
          multiple_consumers  => false,
          compatible             => '8.1.5');
   ```

   multiple_consumers パラメータは、複数のコンシューマが存在するかどうかを指定します。キューの場合、multiple_consumers は false に設定します。トピックの場合、multiple_consumers は true に設定します。

2. JMS 宛先を作成します。次の SQL の例では、キュー表 demoTestQTab 内に demoQueue というキューを作成し、キューを開始します。

   ```
   DBMS_AQADM.CREATE_QUEUE(
         Queue_name          => 'demoQueue',
         Queue_table         => 'demoTestQTab');

   DBMS_AQADM.START_QUEUE(
         queue_name         => 'demoQueue');
   ```

例：
次の例では、トピック表 demoTestTTab 内に demoTopic というトピックを作成する方法を示します。作成後は、2 つの永続サブスクライバがトピックに追加されます。最後に、トピックを開始します。

注意： OEMS JMS のデータベース・オプションでは、DBMS_AQADM.CREATE_QUEUE メソッドを使用してキューとトピックの両方を作成します。

```
DBMS_AQADM.CREATE_QUEUE_TABLE(
        Queue_table             => 'demoTestTTab',
        Queue_payload_type      => 'SYS.AQ$_JMS_MESSAGE',
        multiple_consumers  => true,
        compatible              => '8.1.5');
DBMS_AQADM.CREATE_QUEUE( 'demoTopic', 'demoTestTTab');
DBMS_AQADM.START_QUEUE('demoTopic');
```

注意：

OEMS JMS のデータベース・オプションでは、DBMS_AQADM.CREATE_QUEUE メソッドに渡される名前（Queue_name）が宛先の JNDI 名に組み込まれます。たとえば、demoQueue という Queue_name を使用して作成されたキューは、Queues/demoQueue という JNDI 名で使用できます。

OEMS JMS の宛先を JMS コネクタの宛先でラップするには、OEMS JMS が提供する宛先の JNDI 名と、oc4j-connectors.xml ファイルに定義された JMS コネクタの宛先に対応する <config-property> の jndiName（$J2EE_HOME/config でのグローバル設定またはアプリケーションの .ear ファイルにオプションで含まれるローカル設定）が一致している必要があります。図 4-1「JMS 構成」の矢印番号 13 を参照してください。

OEMS JMS のデータベース参照の宣言

リソース・プロバイダ参照の宣言の概要は、4-15 ページの「リソース・プロバイダ参照の宣言」を参照してください。

リソース・プロバイダ参照を宣言する場合は、常にリソース・プロバイダ参照に使用する名前と、リソース・プロバイダ・インタフェースを実装する Java クラスを指定する必要があります。

OEMS JMS のデータベース・オプションの場合、クラスは oracle.jms.OjmsContext です。OJMSReference という名前のリソース・プロバイダ参照を宣言するには、次のようにします。

```
<resource-provider class="oracle.jms.OjmsContext" name="OJMSReference">
...
</resource-provider>
```

たとえば、OEMS JMS の参照の名前が OJMSReference で、リソース・プロバイダ・キューの JNDI コンテキスト内の JNDI ロケーションが Queues/MY_QUEUE である場合、アプリケーションとリソース・アダプタは、JNDI ロケーション java:comp/resource/OJMSReference/Queues/MY_QUEUE を使用してこのリソース・プロバイダ・キューにアクセスできます。

OEMS JMS の参照を宣言する一般的な手順は、次のとおりです。

1. 最初に、data-sources.xml ファイルにローカル・データソースを作成します。デモ・セットには、この例が含まれます。/ojms/src/META-INF/data-sources.xml を参照してください。
2. 次に、OC4J に data-sources.xml ファイルの配置場所を指示するため、orion-application.xml に <data-sources> 要素を追加します。

   ```
   <data-sources path="data-sources.xml"/>
   ```

   data-sources のパスは、orion-application.xml ファイルに対する相対パスです。
3. 最後に、リソース・プロバイダ参照にデータソースを設定するため、以前作成した <resource-provider> 要素に <property> サブ要素を追加します。

   ```
   <resource-provider class="oracle.jms.OjmsContext" name="OJMSReference">
       <property name="datasource"
   value="jdbc/xa/MyChannelDemoDataSource"></property>
   </resource-provider>
   ```

   データソースの構成方法の詳細は、このマニュアルの「データソース」の章を参照してください。使用するドライバ（OCI またはシン）を選択する場合は、実際のアプリケーション・パフォーマンスを測定するのが最も効果的です。OCI ドライバは、非 XA 操作では一般的に高速ですが、XA 操作ではシン・ドライバより大幅に遅くなる可能性があります。

How-To ドキュメントおよびデモ・セットと、その URL のリストは、4-4 ページの「JMS の How-To ドキュメントおよびデモ・セット」を参照してください。

OEMS JMS のデータベース・オプションのリソース名

OEMS JMS のデータベース・オプションのリソースの *resourceName* は、次のとおりです。

typeName/instanceName

typeName および *instanceName* の値は、リソースのタイプ（コネクション・ファクトリまたは宛先）に応じて変化します。次に、この詳細を説明します。

OEMS JMS のデータベース・オプションのコネクション・ファクトリの場合 :

typeName は、コネクション・ファクトリ・タイプに応じて次のいずれかになります。

- ConnectionFactories
- QueueConnectionFactories
- TopicConnectionFactories
- XAConnectionFactories
- XAQueueConnectionFactories
- XATopicConnectionFactories

instanceName には、何を指定してもかまいません（OEMS JMS のデータベース・オプションのコネクション・ファクトリはカスタマイズできず、同じコネクション・ファクトリ・タイプの複数のインスタンスは必要とされないため、この名前は無視されます）。

たとえば、OEMS JMS のデータベース・オプションで、非 XA キューのコネクション・ファクトリの *resourceName* は次のようになります（無視される *instanceName* は、任意で myQCF に設定します）。

```
QueueConnectionFactories/myQCF
```

前の手順で作成されたコネクション・ファクトリの *resourceName* の値は、OEMS JMS のデータベース・オプションで特定のコネクション・ファクトリを識別するために使用されます（図 4-1「JMS 構成」の矢印番号 16 のリンク・キーを参照）。これらの値は、リソース・アダプタが省略される場合にも使用されることがあります（4-13 ページの「アプリケーション・クライアントの JMS コネクタの省略」を参照）。

次の表に、実装される javax.jms.* インタフェースを示します。

表 4-8 実装される javax.jms.* インタフェース

typeName	CF	QCF	TCF	XACF	XAQCF	XATCF
ConnectionFactories	○					
QueueConnectionFactories	○	○				
TopicConnectionFactories	○		○			
XAConnectionFactories	○			○		
XAQueueConnectionFactories	○	○		○	○	
XATopicConnectionFactories	○		○	○		○

アプリケーションで（<res-type> 要素の指定どおり）javax.jms.TopicConnectionFactory を必要とする場合、適切なコネクション・ファクトリを戻すタイプ名は、TopicConnectionFactories および XATopicConnectionFactories のみです。

OEMS JMS のデータベース・オプションの宛先の場合 :

typeName は、宛先タイプに応じて次のいずれかになります。

- Queues
- Topics

> **注意 :** 宛先は、DBMS_AQADM.CREATE_QUEUE_TABLE に渡された multiple_consumers パラメータが false の場合、キュー（*typeName* = Queues）になります。それ以外の場合、宛先はトピック（*typeName* = Topics）になります。

instanceName は、宛先の名前（DBMS_AQADM.CREATE_QUEUE に渡される Queue_Name パラメータ）です。宛先が <resource-provider> 要素の username プロパティで指定されたユーザーによって所有されていない場合は、*instanceName* に *owner* 接頭辞（*owner* は宛先の所有者）を付ける必要があります。（付ける必要がない場合でも、*owner* 接頭辞は許可されます。）

たとえば、OEMS JMS のデータベース・オプションで、DBMS_AQADM.CREATE_QUEUE のコール内で使用される demoQueue というキューの *resourceName* は、次のとおりです。

```
Queues/demoQueue
```

owner（someUser など）を指定する必要がある場合、*resourceName* は次のようになります。

```
Queues/someUser.demoQueue
```

前の手順により作成された宛先の *resourceName* の値は、OEMS JMS のデータベース・オプションで特定の宛先を識別するために使用されます（図 4-1 の矢印番号 13 のリンク・キーを参照）。これらの値は、リソース・アダプタが省略される場合にも使用されることがあります（4-13 ページの「アプリケーション・クライアントの JMS コネクタの省略」を参照）。

OEMS JMS のデータベースの永続性を使用したメッセージの送受信 アプリケーションでは、メッセージの送受信に JMS コネクタを使用することをお薦めします。この方法の場合、使用するリソース・プロバイダとは無関係に送受信コードを記述できます。4-24 ページの「JMS メッセージの送受信」に記載されている例は、渡される宛先およびコネクション・ファクトリのロケーションを別にするだけで、OEMS JMS のデータベースを含むすべてのリソース・プロバイダで使用できます。

OEMS JMS のデータベース・オプションの直接ルックアップを使用するアプリケーション・クライアントに必要なクラスパス

OEMS JMS のデータベース・オプションをアプリケーション・クライアントから直接使用する場合、表 4-9「OEMS JMS のデータベース・オプションのルックアップに必要なクライアント・サイドの JAR ファイル」にリストされたクラスパスに各 JAR ファイルが含まれている必要があります。

表 4-9 OEMS JMS のデータベース・オプションのルックアップに必要なクライアント・サイドの JAR ファイル

JAR	ORACLE_HOME パス
oc4jclient.jar	/j2ee/<instance>
ejb.jar	/j2ee/<instance>/lib
jta.jar	/j2ee/<instance>/lib
jms.jar	/j2ee/<instance>/lib
jndi.jar	/j2ee/<instance>/lib
javax77.jar	/j2ee/<instance>/lib
adminclient.jar	/j2ee/<instance>/lib
ojdbc14dms.jar	/j2ee/<instance>/../../oracle/jdbc/lib
dms.jar	/j2ee/<instance>/../../oracle/lib
bcel.jar	/j2ee/<instance>/lib
optic.jar	/opmn/lib
(opmn:ormi 接頭辞が Oracle Application Server 環境で使用される場合にのみ必須)	

Oracle Application Server および Oracle データベースと組み合せた OEMS JMS のデータベース・オプションの使用方法

この項では、OEMS JMS のデータベース・オプションと Oracle Application Server を組み合せて使用するユーザーにとって一般的な問題について説明します。

- aqapi.jar のコピー時のエラー
- メッセージ・セレクタの使用時における順序の保存

aqapi.jar のコピー時のエラー OEMS JMS のデータベース・オプションを Oracle Application Server と組み合せて使用する場合に発生する一般的なエラーは、次のとおりです。

```
PLS-00306 "wrong number or types of arguments"
```

このメッセージが表示される場合は、Oracle Application Server で使用されている aqapi.jar ファイルに、AQ で使用されている Oracle データベースのリリースとの互換性がありません。OEMS JMS のデータベース・オプションの現行リリースでサポートされている旧 Oracle データベースは、9.0.1.4、9.2.0.2 以上、10.1.0 以上および 10.2.0 以上です。

Oracle Application Server および Oracle Database のどちらにも、OEMS JMS クライアント JAR ファイルが付属しています。よくある間違いは、双方の環境に互換性があると誤解して、Oracle データベースのインストール環境と Oracle Application Server のインストール環境の間で aqapi.jar ファイルをコピーすることです。どちらのインストールからもこのファイルをコピーしないでください。

Oracle Application Server のインストール環境の場合、OEMS JMS のデータベース・オプションのクライアント JAR ファイルは、*ORACLE_HOME*/rdbms/jlib/aqapi.jar です。このファイルは、CLASSPATH に含める必要があります。

メッセージ・セレクタの使用時における順序の保存 データベース永続性モードでメッセージ・セレクタを使用する際に順序が重要な場合には、Java システム・プロパティ `oracle.jms.orderWithSelector` を true に設定する必要があります。このプロパティは、デフォルトで `false` に設定されています。

このシステム・プロパティを使用するには、AQ ユーザーに `ALTER SESSION` 権限が必要です。また、`ORDER BY` 句に変換されるため、このプロパティを `true` に設定するとパフォーマンスに影響があります。

サード・パーティ JMS プロバイダの使用

この項では、サポートされるサード・パーティ JMS リソース・プロバイダへの参照を宣言する方法について簡単に説明します。

リソース・プロバイダに XA インタフェースが存在し、JMS コネクタとアプリケーションがそのインタフェースを使用するよう構成されている場合、OC4J では、サポートされるすべてのリソース・プロバイダにおいて 2 フェーズ・コミット（2pc）に対応します。サポートされるすべてのサード・パーティ・プロバイダには、XA インタフェースがあります。

OC4J でサポートされる各サード・パーティ JMS プロバイダのバージョンは、4-48 ページの「サード・パーティ JMS プロバイダの使用」にリストされています。

この項では、次のサード・パーティ JMS プロバイダの参照を宣言する方法について説明します。

- IBM WebSphere MQ のリソース・プロバイダ参照の宣言
- TIBCO Enterprise Message Service のリソース・プロバイダ参照の宣言
- SonicMQ のリソース・プロバイダ参照の宣言

コンテキスト・スキャン・リソース・プロバイダ・クラスは、汎用のリソース・プロバイダ・クラスです。このクラスは、サード・パーティのメッセージ・プロバイダで使用するために OCJ に付属しています。

注意： リソース・プロバイダ参照を宣言するには、次のファイルのいずれかを使用します。

- リソース・プロバイダ参照をすべてのアプリケーションに認識させるには（グローバル）、グローバル `application.xml` ファイルを使用します。
- リソース・プロバイダ参照を単一のアプリケーションに認識させるには（ローカル）、そのアプリケーションに固有の `orion-application.xml` ファイルを使用します。

IBM WebSphere MQ のリソース・プロバイダ参照の宣言

WebSphere MQ は、IBM 社のメッセージ・プロバイダです。WebSphere MQ のリソースは、次のロケーションで使用できます。

`java:comp/resource/`*`providerName`*
ここで、*providerName* は、`<resource-provider>` 要素で使用される名前です。

WebSphere MQ のリソース・プロバイダ参照を宣言する手順は、次のとおりです。

1. システムに WebSphere MQ をインストールして構成します。次に、ベンダーから提供されている例またはツールを実行してインストールが成功していることを確認します。手順の詳細は、WebSphere MQ ソフトウェアに付属のドキュメントを参照してください。
2. WebSphere MQ をカスタム・リソース・プロバイダとして追加します。

   次の例は、`orion-application.xml` ファイルの `<resource-provider>` 要素を使用して、WebSphere MQ のリソース・プロバイダ参照を宣言する方法を示しています。

```
<resource-provider
    class="com.evermind.server.deployment.ContextScanningResourceProvider"
    name="MQSeries">
  <description> MQSeries resource provider </description>
  <property
      name="java.naming.factory.initial"
      value="com.sun.jndi.fscontext.RefFSContextFactory">
  </property>
  <property
      name="java.naming.provider.url"
      value="file:/home/developer/services_guide_demo/mqseries/src/bindings">
  </property>
</resource-provider>
```

   この例は、デモ・セットの環境での構成方法を示しています。この例は、デモ作成者の環境にのみ適用されるため、他の環境で使用する場合は適切に編集する必要があります。
3. IBM 社のドキュメントに従って、WebSphere MQ の JMS クライアントに必要な JAR ファイルを `J2EE_HOME/applib` に追加します。

   詳細は、デモ・セットの `how-to-gjra-with-mqseries.html` ドキュメントにある「Resource Provider Task #3: Declare a Resource Provider Reference」を参照してください。

How-To ドキュメントおよびデモ・セットと、その URL のリストは、4-4 ページの「JMS の How-To ドキュメントおよびデモ・セット」を参照してください。

TIBCO Enterprise Message Service のリソース・プロバイダ参照の宣言

TIBCO Enterprise Message Service は、TIBCO Software 社のメッセージ・プロバイダです。TIBCO のリソースは、次のロケーションで使用できます。

`java:comp/resource/`*`providerName`*

ここで、*providerName* は、`<resource-provider>` 要素で使用される名前です。

TIBCO のリソース・プロバイダ参照を宣言する手順は、次のとおりです。

1. システムに TIBCO Enterprise Message Service をインストールして構成します。次に、ベンダーから提供されている例またはツールを実行してインストールが成功していることを確認します。手順の詳細は、TIBCO ソフトウェアに付属のドキュメントを参照してください。
2. TIBCO をカスタム・リソース・プロバイダとして追加します。次の例は、`orion-application.xml` ファイルの `<resource-provider>` 要素を使用して、TIBCO のリソース・プロバイダ参照を宣言する方法を示しています。

```
<resource-provider
   class="com.evermind.server.deployment.ContextScanningResourceProvider"
   name="TibcoJMSReference">
    <property
```

```
            name="java.naming.factory.initial"
            value="com.tibco.tibjms.naming.TibjmsInitialContextFactory">
        </property>
        <property
            name="java.naming.provider.url"
            value="tibjmsnaming://jleinawe-sun:7222">
        </property>
    </resource-provider>
```

この例は、デモ・セットの環境での構成方法を示しています。この例は、デモ作成者の環境にのみ適用されるため、他の環境で使用する場合は適切に編集する必要があります。

3. TIBCO 社のドキュメントに従って、TIBCO の JMS クライアントに必要な JAR ファイルを `J2EE_HOME/applib` に追加します。

   詳細は、デモ・セットの `how-to-gjra-with-tibco.html` ドキュメントにある「Resource Provider Task #3: Declare a Resource Provider Reference」を参照してください。

How-To ドキュメントおよびデモ・セットと、その URL のリストは、4-4 ページの「JMS の How-To ドキュメントおよびデモ・セット」を参照してください。

SonicMQ のリソース・プロバイダ参照の宣言

SonicMQ は、Sonic Software 社のメッセージ・プロバイダです。SonicMQ のリソースは、次のロケーションで使用できます。

`java:comp/resource/`*`providerName`*

ここで、*providerName* は、`<resource-provider>` 要素で使用される名前です。

SonicMQ のリソース・プロバイダ参照を宣言する手順は、次のとおりです。

1. システムに SonicMQ をインストールして構成します。次に、ベンダーから提供されている例またはツールを実行してインストールが成功していることを確認します。手順の詳細は、Sonic ソフトウェアに付属のドキュメントを参照してください。

2. SonicMQ をカスタム・リソース・プロバイダとして追加します。次の例は、`orion-application.xml` ファイルの `<resource-provider>` 要素を使用して、SonicMQ のリソース・プロバイダ参照を宣言する方法を示しています。

```
<resource-provider
   class="com.evermind.server.deployment.ContextScanningResourceProvider"
   name="SonicJMSReference">
    <property
        name="java.naming.factory.initial"
        value="com.sonicsw.jndi.mfcontext.MFContextFactory">
    </property>
    <property
        name="java.naming.provider.url"
        value="tcp://stadd41:2506">
    </property>
    <property
        name="com.sonicsw.jndi.mfcontext.domain"
        value="Domain1">
    </property>
</resource-provider>
```

この例は、デモ・セットの環境での構成方法を示しています。この例は、デモ作成者の環境にのみ適用されるため、他の環境で使用する場合は適切に編集する必要があります。

3. Sonic 社のドキュメントに従って、SonicMQ の JMS クライアントに必要な JAR ファイルを `J2EE_HOME/applib` に追加します。

   詳細は、デモ・セットの `how-to-gjra-with-sonic.html` ドキュメントにある「Resource Provider Task #3: Declare a Resource Provider Reference」を参照してください。

How-To ドキュメントおよびデモ・セットと、その URL のリストは、4-4 ページの「JMS の How-To ドキュメントおよびデモ・セット」を参照してください。

JMS コネクタ

Oracle Application Server には、JMS コネクタと呼ばれる、J2CA 1.5 に準拠したリソース・アダプタがあります。JMS コネクタは、アプリケーションが JMS 1.1 または 1.02b を実装する任意の JMS プロバイダにアクセスする際に統一されたメカニズムを提供します。OracleASjms は JMS コネクタのインスタンスで、OEMS JMS のメモリー内およびファイル・ベースのオプションで使用できるように構成されています。JMS コネクタでは、Oracle 固有の API は使用されていません。サポートされる JMS 実装には、OEMS JMS に加え、IBM Websphere MQ JMS、TIBCO Enterprise Message Server、SonicMQ JMS などのサード・パーティ製品が含まれます。

JMS コネクタは、OC4J 実装で JMS を使用する場合に推奨されます。JMS コネクタは、J2CA 1.5 標準と、JMS 1.1 および 1.02b 標準に準拠しています。OC4J 用の最小限のカスタマイズが含まれますが、個々の JMS プロバイダ用のカスタマイズは含まれません。JMS コネクタは、すべての標準的な JMS 実装をシームレスに統合するよう設計されています。

注意： JMS プロバイダではカスタムの拡張機能がサポートされることが多いため、一般的に JMS コネクタでは、特定の JMS プロバイダへの最適なアクセスは保証されません。

JMS コネクタには、次の機能が含まれます。

- JNDI マッピング
- MDB 統合（変化するメッセージ負荷に応じた動的調整を含む）
- グローバル・トランザクション・サポート（トランザクション・リカバリの標準ベースのサポートを含む）。トランザクション・サポートの詳細は、第 3 章「OC4J トランザクション・サポート」を参照してください。

注意： OEMS JMS は、JMS コネクタを使用しないグローバル・トランザクションでは使用できません。

- 真に汎用的な JMS 接続プーリング
- 容易なデプロイ（順序非依存など）
- JMS 操作の遅延解決（開始順序非依存、JMS プロバイダの開始と停止などの動的管理の許可、プロバイダ障害時における接続の再試行を含む）
- パフォーマンス
- JSR-77 統計

通常、JMS コネクタは、接続先の EIS が JMS リソース・プロバイダである場合に使用します。ただし、JMS コネクタは、EIS で J2EE アプリケーション・コンポーネントへの通知手段として JMS メッセージ機能を使用する場合にも使用できます。この場合、リソース・アダプタ（および JMS リソース・プロバイダ）は、EIS 固有のリソース・アダプタに存在するインバウンド通信機能のかわりに使用できます。この 2 面アダプタ・ソリューション（EIS 固有のアダプタをアウトバウンド通信に使用し、JMS コネクタをインバウンド通信に使用する方法）により、通常であれば不可能である EIS と J2EE アプリケーション間の双方向通信が可能になります。

リソース・アダプタの詳細は、『Oracle Containers for J2EE リソース・アダプタ管理者ガイド』を参照してください。

JMS コネクタの変更

OC4J に付属の JMS コネクタは、OEMS JMS のメモリー内およびファイル・ベースの永続性オプションをサポートするよう事前に構成されています。OEMS JMS のデータベース・オプション、またはサポートされる Oracle 以外の JMS プロバイダのいずれかを統合する場合は、JMS コネクタ用に別の構成を作成する必要があります。アプリケーションのローカル・アダプタを使用して OEMS JMS のメモリー内およびファイル・ベースのオプションに接続する場合も、別の JMS コネクタを作成することが可能です。

How-To ドキュメントおよびデモ・セットと、その URL のリストは、4-4 ページの「JMS の How-To ドキュメントおよびデモ・セット」を参照してください。

新規 JMS コネクタ・モジュールを作成および構成する手順は、次のとおりです。

1. 接続先のリソース・プロバイダのデモ・セットに移動します。
2. How-To ドキュメントを読みます。
3. 新規 JMS コネクタ・モジュールのテンプレートとして使用する次のファイルをコピーします。
   - `ra.xml`
   - `oc4j-ra.xml`
   - `oc4j-connectors.xml`
4. How-To ドキュメントの「Configuring the Resource Adapter」の指示に従って、`ra.xml`、`oc4j-ra.xml` および `oc4j-connectors.xml` ファイルを変更します。
5. 次の構造を使用して、新規 `.rar` ファイルを作成します（ファイル名は任意です）。

   ```
   META-INF/ra.xml

   META-INF/oc4j-ra.xml
   ```

6. 希望する可視性レベルに応じて、次のいずれかの場所に新規 `oc4j-connectors.xml` を配置します。
   - アプリケーション単位のローカルな可視性を確保するには、アプリケーションの `.ear` ファイルのトップレベルにある `META-INF/` ディレクトリに新規 `oc4j-connectors.xml` ファイルを配置します。

     グローバルな可視性を確保するには、1 つ以上の新規 `<connector>` 要素を `$J2EE_HOME/config/oc4j-connectors.xml` ファイルにコピーします。新規コネクタの名前（`<connector>` の `name` 属性）が、他のコネクタまたは任意の JNDI オブジェクトの名前と競合していないことを確認してください。
7. 次のように、デプロイの準備を行います。
   - ローカルな可視性を確保する場合は、次のようにします。
     - 新規 `.rar` ファイルを `.ear` ファイルに配置します。
     - 次のような `<connectors>` 要素を `.ear` ファイルの `META-INF/orion-application.xml` ファイルに挿入します。

       ```
       <connectors path="oc4j-connectors.xml"/>
       ```

     - 次のような `<module>` 要素を `.ear` ファイルの `META-INF/application.xml` ファイルに挿入します。

       ```
       <module>
           <connector>rarFileName</connector>
       </module>
       ```

   - グローバルな可視性を確保する場合は、『Oracle Containers for J2EE デプロイメント・ガイド』に記載されているコネクタのデプロイ手順に従ってください。

JMS コネクタの構成

Application Server Control コンソールは、JMS コネクタを構成するための主要ツールです。

コネクション・ファクトリおよび宛先

デフォルト・アプリケーションには、デフォルトの JMS コネクタ用に次の宛先オブジェクトが定義されています。

- JNDI ロケーション `OracleASjms/MyQueue1` にバインドされたキュー
- JNDI ロケーション `OracleASjms/MyTopic1` にバインドされたトピック
- JNDI ロケーション `OracleASjms/Queues` にバインドされたキューの JNDI サブコンテキストをラップする自動宛先
- JNDI ロケーション `OracleASjms/Topics` にバインドされたトピックの JNDI サブコンテキストをラップする自動宛先

これらの宛先および JNDI サブコンテキストをラップする自動宛先は、コンソールや `JMSAdministrator` MBean で事前に宛先を定義することなくアプリケーションで使用できます。自動宛先ラッピング機能を使用する場合、JNDI ルックアップの名前には次の形式を使用します。

```
<JNDIsubcontext>/providerName
```

ここで、`<JNDIsubcontext>` は、`oc4j-connectors.xml` ファイルで指定された JNDI サブコンテキストをラップする自動宛先です。たとえば、自動宛先ラッピングを JMS コネクタと組み合せて使用する場合、JNDI 名は次の形式になります。

```
OracleASjms/Queues/jms/<queue_name>
```

または

```
OracleASjms/Topics/jms/<topic_name>
```

JMS コネクタのデフォルトのコネクション・ファクトリは、次のとおりです。

	XA	非 XA
デフォルトのキュー・コネクション・ファクトリ	`OracleASjms/MyXAQCF`	`OracleASjms/MyQCF`
デフォルトのトピック・コネクション・ファクトリ	`OracleASjms/MyXATCF`	`OracleASjms/MyTCF`
デフォルトの統合コネクション・ファクトリ	`OracleASjms/MyXACF`	`OracleASjms/MyCF`

この表のデフォルト・コネクション・ファクトリは、OC4J に付属のデフォルトの `oc4j-ra.xml` ファイルに明示的に宣言されています。

データベースの永続性の宛先

自動宛先ラッピングは、OEMS JMS のデータベース・オプションを使用するために JMS コネクタをカスタマイズする際にも使用できます。前述のデフォルトのコネクタと同様に、JNDI ルックアップの名前には次の形式を使用します。

```
<JNDIsubcontext>/providerName
```

ここで、<JNDIsubcontext> は、oc4j-connectors.xml ファイルで指定された JNDI サブコンテキストをラップする自動宛先で、<providerName> は、接頭辞 queue または topic が付く物理データベース宛先です。

たとえば、自動宛先ラッピングを、OracleDB/wrap の JNDIsubcontext とともにデプロイされた OEMS データベースの JMS コネクタと組み合せて使用する場合、そのデータベースの JNDI 名は次の形式になります。

```
OracleDB/wrap/Queues/<queue_name>
```

または

```
OracleDB/wrap/Topics/<topic_name>
```

JMS コネクタのコネクション・ファクトリおよび宛先の作成

JMS コネクタのコネクション・ファクトリおよび宛先を構成する方法（oc4j-connectors.xml、oc4j-ra.xml および ra.xml ファイルの例を含む）は、http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html のドキュメントを参照してください。

関連する how-to-gjra-with-*xxx*.zip ファイル（*xxx* はリソース・プロバイダの名前）をダウンロードして解凍します。関連する How-To ドキュメントの「Configuring the Resource Adapter」を参照してください。

JMS コネクタの XML ファイルのリファレンス情報は、『Oracle Containers for J2EE リソース・アダプタ管理者ガイド』の付録 A「OC4J リソース・アダプタ構成ファイル」を参照してください。

JMS コネクタの設定

表 4-10 に、JMS コネクタの構成設定とその説明を示します。

Application Server Control コンソールの JMS コネクタ設定へのパス :

「OC4J: ホーム」→「アプリケーション」タブ→「ビュー : スタンドアロン・リソース・アダプタ」→「OracleAS JMS」

表 4-10 JMS コネクタの構成設定

コンソール設定	XML 永続性ファイル	説明
「OC4J: ホーム」→「アプリケーション」タブ→「ビュー : スタンドアロン・リソース・アダプタ」→「OracleAS JMS」→**「コネクション・ファクトリ」タブ**→「作成」ボタン **コネクション・ファクトリ・インタフェース** 図 4-1 の矢印番号 14 のリンク参照。	この設定は、oc4j-ra.xml ファイルの <connectionfactory-interface> 要素で永続化されます。	コネクション・ファクトリ・インタフェース設定では、作成するコネクション・ファクトリのタイプを定義します。

表 4-10 JMS コネクタの構成設定（続き）

コンソール設定	XML 永続性ファイル	説明
JNDI ロケーション 図 4-1 の矢印番号 5 のリンク・キー。	この設定は、oc4j-ra.xml ファイルの <connector-factory> 要素の location 属性で永続化されます。	JNDI ロケーション設定では、リソース・アダプタの新規コネクション・ファクトリをバインドする JNDI ロケーションを指定します。EIS に接続するアプリケーション・コンポーネントがコネクション・ファクトリを特定できるように、有効な JNDI ロケーションを入力します。 この JNDI ロケーションは、jndiLocation とは異なることに注意してください。
接続プーリング	この設定は、oc4j-ra.xml ファイルの <connection-pooling> 要素に永続化されます。	接続プーリングにより、EIS への一連の接続をアプリケーション内で再利用できます。アプリケーションでは、独自の排他的接続プールを作成するか、このリソース・アダプタで使用可能ないずれかの共有接続プールを使用するかを選択できます。 接続プールなし : 接続プーリングを無効化する場合、このオプションを選択します。 プライベート接続プールの使用 : 選択したコネクション・ファクトリで排他的に使用する新規接続プールを作成する場合、このオプションを選択します。 共有接続プールの使用 : 複数のコネクション・ファクトリで利用可能な共有接続プールを使用する場合、このオプションを選択します。 oc4j-ra.xml ファイルには、JMS コネクタの OC4J 固有の構成が含まれます。<connector-factory> のサブ要素には、コネクション・ファクトリの接続プーリングを設定するための <connection-pooling> と、コンテナ管理のサインオンを設定するための <security-config> があります。各 connector-factory で、プライベート接続プールを構成するか、<oc4j-connector-factories> の <connection-pool> サブ要素を通じて設定される共有接続プールを使用することが可能です。
jndiLocation 図 4-1 の矢印番号 16 のリンク参照。	この設定は、oc4j-ra.xml ファイルの <connector-factory> 要素の <config-property> サブ要素で永続化されます。	jndiLocation 設定では、作成されるリソース・アダプタのコネクション・ファクトリでラップするリソース・プロバイダのコネクション・ファクトリを指定します。 この jndiLocation は、JNDI ロケーションとは異なることに注意してください。
< なし >	この設定は、oc4j-ra.xml ファイルの <connector-factory> 要素の <config-property> サブ要素で永続化されます。	clientId 設定では、JMS コネクタ接続で作成される新規リソース・プロバイダ接続に適用するクライアント ID を指定します。
「OC4J: ホーム」→「アプリケーション」タブ→「ビュー : スタンドアロン・リソース・アダプタ」→「OracleAS JMS」→**「管理対象オブジェクト」タブ**→「作成」ボタン **オブジェクト・クラス**	この設定は、oc4j-connectors.xml ファイルの <adminobject-config> 要素の <adminobject-class> サブ要素で永続化されます。	オブジェクト・クラス設定では、作成される管理対象オブジェクト（宛先）のタイプを定義します。ドロップダウン・リストから選択します。
JNDI ロケーション 図 4-1 の矢印番号 7 および 11 のリンク・キー。	この設定は、oc4j-connectors.xml ファイルの <adminobject-config> 要素の location 属性で永続化されます。	JNDI ロケーション設定では、リソース・アダプタの新規管理対象オブジェクト（宛先）をバインドする JNDI ロケーションを指定します。

表 4-10 JMS コネクタの構成設定（続き）

コンソール設定	XML 永続性ファイル	説明
jndiName 図 4-1 の矢印番号 13 のリンク参照。	この設定は、oc4j-connectors.xml ファイルの <adminobject-config> 要素にある <config-property> サブ要素の value 属性で永続化されます。	jndiName 設定は、作成されるリソース・アダプタの管理対象オブジェクト（宛先）でラップするリソース・プロバイダの宛先の JNDI 名です。 **注意：**この jndiName 設定の説明は、個別にバインドされるリソース・プロバイダのキューおよびトピックに適用されます。詳細は、デモ・セットの oc4j-connectors.xml ファイルのコメントを参照してください。
resourceProviderName 図 4-1 の矢印番号 12 のリンク参照。	この設定は、oc4j-connectors.xml ファイルの <adminobject-config> 要素にある <config-property> サブ要素の value 属性で永続化されます。	resourceProviderName 設定では、作成されるリソース・アダプタの管理対象オブジェクト（宛先）でラップされる宛先を所有するリソース・プロバイダを指定します。
「ResourceAdapter」→「管理」→「**enableDMS**」	この設定は、ra.xml のサブ要素である <resourceadapter> の <config-property> サブ要素で永続化されます。 また、この設定は、oc4j-connector.xml のサブ要素である <connector> の <config-property> サブ要素でも永続化されます。 oc4j-connector.xml ファイルは、ra.xml よりも優先されます。	このプロパティは、JMS リソース・アダプタにより DMS パフォーマンス・メトリックが収集されるかどうかを指定します。このプロパティは、デフォルトで true に設定されます。このプロパティを false に設定すると、JMS リソース・アダプタにより DMS パフォーマンス・メトリックの収集が停止されます。
「ResourceAdapter」→「管理」→「**loggerName**」	この設定は、ra.xml のサブ要素である <resourceadapter> の <config-property> サブ要素で永続化されます。 また、この設定は、oc4j-connector.xml のサブ要素である <connector> の <config-propety> サブ要素でも永続化されます。 oc4j-connector.xml ファイルは、ra.xml よりも優先されます。	このプロパティは、JMS リソース・アダプタのロギングを処理するログ出力の名前を指定します。ログ出力は j2ee-logging.xml で定義できます。このプロパティのデフォルト値は null です（null の場合、ログ出力の名前は oracle.j2ee.ra.jms.generic に設定されます）。

XML ファイルでの JMS コネクタの構成

JMS コネクタの構成設定は、次のファイルで永続化されます。

- `oc4j-connectors.xml`: `oc4j-connectors.xml` ファイルは、JMS コネクタ・インスタンスと JMS コネクタの宛先を作成する際に使用されます。
- `oc4j-ra.xml`: `oc4j-ra.xml` ファイルには、JMS コネクタの OC4J 固有の構成が含まれます。Application Server Control コンソールを使用して JMS コネクタのコネクション・ファクトリを作成または編集すると、OC4J によって `oc4j-ra.xml` ファイルが更新されます。
- `ra.xml`: オラクル社が提供する標準の JMS コネクタ・モジュール構成ファイルです。JMS コネクタを構成する場合、`ra.xml` のエントリは、通常、デフォルトとして機能します。

`oc4j-connectors.xml`、`oc4j-ra.xml` および `ra.xml` ファイルの具体的な使用例は、`http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html` を参照してください。

JMS コネクタの XML ファイルのリファレンス情報は、『Oracle Containers for J2EE リソース・アダプタ管理者ガイド』の付録 A「OC4J リソース・アダプタ構成ファイル」を参照してください。

注意： XML ファイルで直接変更した構成を有効化するには、OC4J インスタンスを再起動する必要があります。

メッセージドリブン Bean の使用

OC4J では、JMS コネクタを使用してメッセージ・プロバイダをプラグインする MDB をサポートしています。このプロバイダには、OEMS JMS やサポートされているサード・パーティのメッセージ・プロバイダが含まれます。これにより、JMS 接続プーリングや、変化する負荷に応じてサイズ変更（動的負荷調整）を行う MDB リスナー・スレッド・セットなど、各種の有益な機能が提供されます。

MDB の使用方法の詳細な例は、`http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html` のメッセージングのデモに含まれます。

How-To の .zip ファイルのいずれかをダウンロードして解凍します。MDB の構成例は、`/src/ejb/META-INF/ejb-jar.xml` および `/src/ejb/META-INF/orion-ejb-jar.xml` ファイルで確認できます。

接続プーリング

JMS コネクタでは、J2CA の接続プーリングが使用されます。接続プーリングの詳細は、『Oracle Containers for J2EE リソース・アダプタ管理者ガイド』を参照してください。

MDB エンドポイントの微調整

各 MDB エンドポイントは、一連の構成プロパティ（ActivationSpec プロパティと呼ばれる）と関連付けられています。このプロパティは、JMS コネクタによりメッセージ・エンドポイントがアクティブ化され、メッセージ・エンドポイントがメッセージドリブン Bean と関連付けられる際に使用されます。このプロパティは、特にパフォーマンス、ロギング、例外キュー、メッセージ承認およびサブスクリプションの操作に使用されます。

通常、このプロパティは `orion-ejb-jar.xml` に設定されています。ただし、JMS コネクタがアプリケーションの `.ear` ファイルに含まれる場合には、`ejb-jar.xml` ファイルでもこのプロパティを設定できます。各ファイルでの構文は異なります。

`orion-ejb-jar.xml` を使用する場合、プロパティは次の構文を使用して、各 `<message-driven-deployment>` ノードに追加されます。

```
<config-property>
   <config-property-name>ExamplePropertyName</config-property-name>
   <config-property-value>ExampleValue</config-property-value>
</config-property>
```

ejb-jar.xml ファイルを使用する場合、プロパティは次の構文を使用して、各 <message-driven> ノードに追加されます。

```
<activation-config-property>
   <activation-config-property-name>ExamplePropertyName
   </activation-config-property-name>
   <activation-config-property-value>ExampleValue
   </activation-config-property-value>
</activation-config-property>
```

次の表に、MDB エンドポイントの微調整に使用可能な構成プロパティを説明します。

表 4-11 MDB の構成プロパティ

構成プロパティ	説明
ReceiverThreads	（整数でデフォルトは 1）
	エンドポイントに作成するリスナー・スレッドの最大数を設定します。キューでは、メッセージを使用できる割合を増加する際には、複数のスレッドを使用すると便利です。トピックでは、この値は常に 1 である必要があります。（各リスナー・スレッドにより、独自のセッションと TopicSubscriber が取得されます。永続サブスクライバでは、同じサブスクリプション名のサブスクライバを複数持つことはできません。非永続サブスクライバでは、スレッド数が多いとサブスクライバの数が増え、各メッセージのコピーも増加するため、複数のスレッドを持つことにメリットはありません。）ListenerThreadMinBusyDuration を参照してください。
ListenerThreadMaxPollInterval	（ミリ秒でデフォルトは 5000）
	リスナー・スレッドによるポーリングで、処理を待機しているメッセージの有無が確認されます。ポーリングを頻繁に実行すると、指定したリスナー・スレッドが（平均では）より迅速に新しいメッセージに対応できます。リソース・プロバイダは、ポーリングが行われるたびに受信リクエストを処理する必要があるため、ポーリングを頻繁に実行するとオーバーヘッドが発生します。
	オラクル社の JMS コネクタ実装では、（アクティビティが認識されると）アクティビティの期間中はポーリング間隔が短く（ポーリング率が高く）なり、非アクティブの期間中はポーリング間隔が長く（ポーリング率が低く）なる、調整可能なアルゴリズムが適用されます。ListenerThreadMaxPollInterval プロパティには、この調整可能なアルゴリズムで使用されるポーリング間隔に上限があります。
ListenerThreadMaxIdleDuration	（ミリ秒でデフォルトは 300000）
	メッセージを受信していないリスナー・スレッドが保持される期間を表します。（エンドポイントがアクティブなかぎり、少なくとも 1 つのリスナー・スレッドが残ります。）
ListenerThreadMinBusyDuration	（ミリ秒でデフォルトは 10000）
	リスナー・スレッドがメッセージを受信した際に、（新しいメッセージの到着を待機する必要があったため）ListenerThreadMinBusyDuration で指定された期間中（ミリ秒）に一度もアイドル状態になっておらず、エンドポイントの現行のリスナー数が ReceiverThreads の値より少ない場合、（アプリケーション・サーバーにより）追加のリスナー・スレッドが作成されます。

表 4-11 MDB の構成プロパティ（続き）

構成プロパティ	説明
MaxDeliveryCnt	（整数でデフォルトは 5）
	メッセージに JMSXDeliveryCount プロパティがあり、そのプロパティの値が MaxDeliveryCnt の値よりも大きい場合、メッセージは破棄されます（onMessage には送信されません）。例外キューが有効な場合（UseExceptionQueue を参照）は、そのメッセージのコピーが例外キューに送信されます。MaxDeliveryCnt を 0 に設定すると、メッセージは破棄されません。（MDB が例外をスローすることでメッセージに対応すると、メッセージは配信されたとみなされず、再度配信されることに注意してください。MDB が指定されたメッセージに常に例外をスローすることで対応し、メッセージが破棄されないように MaxDeliveryCnt が 0 に設定されていると、MDB が無限ループに陥り、同じメッセージの処理に何度も失敗します。）
EndpointFailureRetryInterval	（ミリ秒でデフォルトは 60000）
	JMS プロバイダに障害が発生すると、リスナー・スレッドは自動的に終了します。（たとえば、リスナー・スレッドが MessageConsumer.recieve() をコールすると例外が発生します。）エンドポイントの実行中にすべてのリスナー・スレッドが終了した場合は、エンドポイント障害とみなされます。この状態では、MDB はメッセージを受信できません。これは、維持ネットワークの障害または JMS サーバーが起動していないなどの原因で発生します。この場合、JMS コネクタは、成功するかエンドポイントが停止されるまで、EndpointFailureRetryInterval で指定された間隔（ミリ秒）ごとに新しいリスナー・スレッドの開始が試行され（JMS プロバイダに再接続され）ます。リスナー・スレッドの作成に成功すると、リスナー・スレッド管理は通常の動作に戻ります。（ReceiverThreads、ListenerThreadMaxIdleDuration および ListenerThreadMinBusyDuration を参照してください。）これにより、接続障害からのほぼ透過的なリカバリが実現されますが、どんな場合にも完全に透過的であるとはかぎりません。 ■ 新規 JMS セッションの作成が必要ですが、JMS メッセージの順序保証が適用されるのは単一のセッション内で受信されたメッセージのみであるため、再接続後に受信したメッセージは再接続前に受信したメッセージを考慮せずに順序付けされます。これが当てはまるかどうかは、JMS プロバイダの動作によって異なります。（JMS で必要な程度よりも厳密である完全なメッセージ順序付けを提供する JMS プロバイダでは、メッセージの順序付けに関する問題は表示されません。） ■ エンドポイントが非永続サブスクライバの場合、メッセージは失われるか破棄されます。この問題は、キューを使用している場合、または永続サブスクライバとともにトピックを使用している場合には発生しません。実際にメッセージの消失 / 重複が発生するかどうか（また、2 つの問題のどちらが発生するか）は、JMS プロバイダの固有の動作によって異なります。この問題を回避するには、失敗する可能性のある JMS プロバイダ接続とともに、非永続サブスクライバを使用しないでください。
ClientId	（文字列でデフォルトはなし）
	設定すると、リスナー・スレッドにより使用される接続は、このクライアント ID を使用するように設定されます。

表 4-11 MDB の構成プロパティ（続き）

構成プロパティ	説明
ResUser、ResPassword	（文字列でデフォルトは null）
	これらのプロパティを使用すると、ユーザー / パスワードをリソース・プロバイダに渡すことができます。これらのプロパティのどちらも設定されていない場合、MDB のインバウンド・メッセージの処理（および有効化されている場合は例外キューの処理）に使用される接続は、引数のない create*Connection メソッドを使用して作成されます。これらのプロパティのいずれかまたは両方が設定されている場合は、ユーザー / パスワード引数として create*Connection メソッドに渡されます。（設定されていないプロパティが 1 つのみの場合は、create*Connection のその特定の引数には NULL が使用されます。）ResPassword プロパティでは、標準のパスワードの間接化オプションがサポートされています（たとえば、->joeuser で joeuser のパスワードを表すことができます）。
AcknowledgeMode	（文字列でデフォルトは Auto-acknowledge）
	Auto-acknowledge または Dups-ok-acknowledge に設定されている必要があります。メッセージを使用し、MDB の onMessage メソッドをコールするリスナー・スレッドにより提供されるサービス品質を制御します。
MessageSelector	（文字列でデフォルトのメッセージのフィルタ処理はなし）
	MDB の onMessage メソッドに送信されるメッセージのフィルタ処理に使用されるセレクタ式。（たとえば、リスナー・スレッドに作成される JMS MessageConsumers の messageSelector として使用されます。）
SubscriptionDurability	（文字列でデフォルトは NonDurable）
	トピックの場合は、Durable または NonDurable に設定されている必要があります。（キューには設定しないでください。）リスナー・スレッドにより使用されるトピック・コンシューマの永続性を制御します。SubscriptionDurability が Durable（および DestinationType が javax.jms.Topic または javax.jms.Destination）に設定されている場合は、SubscriptionName プロパティが必要です。
SubscriptionName	（文字列でデフォルトはなし）
	このプロパティは、SubscriptionDurability が Durable（および DestinationType が javax.jms.Topic または javax.jms.Destination）の場合に必要です。（それ以外の場合は無視されます。）リスナー・スレッドにより使用される永続性サブスクライバの作成に使用される名前です。指定された JMS サーバーでは、指定されたサブスクリプション名を割り当てる MDB は多くとも 1 つである必要があります（この MDB に所有されているリスナー・スレッドは最高でも 1 つである必要があります）。
TransactionTimeout	（ミリ秒でデフォルトは 300000）
	現行のトランザクションを終了するまでに JMS コネクタがメッセージの到着を待機する期間を制限します。トランザクション・マネージャにより、トランザクションが存在できる期間が制限されます（transaction-manager.xml を参照してください）。TransactionTimeout は、問題が発生しないかぎり、onMessage のルーチン中にはトランザクション・マネージャによるトランザクションのタイムアウトが行われないように設定する必要があります。たとえば、トランザクション・マネージャのタイムアウトが 30 秒に設定されていて、onMessage ルーチンは問題が発生しないかぎり 10 秒より長くかからない場合、このプロパティは 20 秒（20000 ミリ秒）に設定します。

表 4-11 MDB の構成プロパティ（続き）

構成プロパティ	説明
UseExceptionQueue	（ブールでデフォルトは false）
	UseExceptionQueue が true の場合、そうでない場合には破棄されるメッセージは例外キューに送信されます。（これが発生するのは、最大配信数を超えた場合のみです。MaxDeliveryCnt プロパティを参照してください。）元のメッセージを例外キューに直接送信するのではなく、次の手順を実行します。 ■ 同じタイプの新しいメッセージを作成します。 ■ 元のメッセージのプロパティおよびボディを新しいメッセージにコピーします。 ■ ヘッダーがコピーされている場合、そのメッセージを例外キューに送信すると多くの場合は破棄（リソース・プロバイダにより上書き）されます。そのため、元のメッセージのヘッダーはコピーのプロパティに変換し、getJMS{Header} を使用して取得された各ヘッダーをプロパティ GJRA_CopyOfJMS{Header} に割り当てます。javax.jms.Destination は有効なプロパティ・タイプではないため、宛先ヘッダーを説明メッセージに変換します。（現在これと同じサービス、特に JMSXDeliveryCount プロパティは JMSX* プロパティでは提供されていません。） ■ （先行する）コピー・プロセスまたは増加プロセスの一部が失敗しても、中止しないでください。残りの手順の完了を試行してください。（バイト / マップ / ストリーム・メッセージ・タイプでは、ボディの一部はコピーされたがその他の部分がコピーされていないことを意味します。） ■ コピー・プロセスが完全に成功したら、GJRA_CopySuccessful というブール・プロパティに値 true を設定して追加します。 ■ メッセージが配信されなかった理由を示す、GJRA_DeliveryFailureReason という文字列プロパティを追加します。 ■ MDB の onMessage メソッドにより配信失敗の直前に例外が生成された場合には、例外情報が含まれる GJRA_onMessageExceptions という文字列プロパティを追加します。結果のメッセージを例外キューに送信します。 ■ 結果のメッセージを例外キューに送信します。例外キューへのメッセージの送信は、一度しか試行されないことに注意してください。この試行に失敗すると、メッセージは例外キューに追加されずに破棄されます。 前の手順のその他の実行方法は、IncludeBodiesInExceptionQueue プロパティを参照してください。 ExceptionQueueName プロパティは必須です。 第 1 の宛先として使用されるだけでなく、ConnectionFactoryJndiName プロパティで指定されるコネクション・ファクトリは、例外キューにも使用されます。（DestinationName プロパティで指定される）第 1 の宛先がトピックの場合、コネクション・ファクトリはキューおよびトピックの両方でサポートされている必要があります。（たとえば、指定されたコネクション・ファクトリの <connectionfactory-interface>（oc4j-ra.xml を参照）の場合は、javax.jms.ConnectionFactory または javax.jms.XAConnectionFactory である必要があります。）
ExceptionQueueName	（文字列でデフォルトはなし） 例外キューとして使用する javax.jms.Queue オブジェクトの JNDI ロケーションです。（例外キューの使用の詳細は、UseExceptionQueue プロパティを参照してください。）このプロパティは、UseExceptionQueue が true の場合には必須で、UseExceptionQueue が false の場合には無視されます。

表 4-11 MDB の構成プロパティ（続き）

構成プロパティ	説明
IncludeBodiesInExceptionQueue	（ブールでデフォルトは true）
	例外キューに送信されたメッセージにメッセージ本文を含めるかどうかを制御します。（例外キューの使用の詳細は、UseExceptionQueue プロパティを参照してください。）通常の操作中に例外キューに大量のメッセージが送信され、例外キューにはメッセージ本文が不要な場合には、パフォーマンスを向上させるためにこのプロパティを false に設定します。 UseExceptionQueue が false の場合、このプロパティは無視されます。このプロパティを使用できない場合が 2 つあります。 ■ 元のメッセージにメッセージ本文がない場合は、例外キューに送信されるメッセージにも本文は含まれません。 ■ なんらかの理由で元のメッセージのコピーを作成できない場合には、かわりに元のメッセージが例外キューに送信されます。これにより、例外キューにメッセージ本文が送信されます。
LogLevel	（文字列でデフォルトはなし）
	JMS コネクタで記録するメッセージの詳細レベルを制御します。これらのメッセージは主に JMS コネクタ自体のデバッグを目的としていますが、JMS コネクタの使用に関する問題のデバッグにも便利です。本番コードでは、このプロパティは設定しないでください。（デバッグ目的で一時的にのみ設定してください。JMS コネクタの今後のリリースで特定のログ・メッセージおよびログ・レベルが追加 / 削除 / 変更されます。）使用可能な値は次のとおりです。 ■ ConnectionPool ■ ConnectionOps ■ TransactionalOps ■ ListenerThreads ■ INFO ■ CONFIG ■ FINE ■ FINER ■ FINEST ■ SEVERE ■ WARNING ■ OFF

動的負荷調整

リスナー・スレッドの動的負荷調整は、次の <activation-config> プロパティに基づきます。表 4-11「MDB の構成プロパティ」で説明されています。

- ListenerThreadMaxIdleDuration
- ListenerThreadMinBusyDuration
- ReceiverThreads

MDB インスタンスの動的負荷調整を含む MDB 構成の詳細は、『Oracle Containers for J2EE Enterprise JavaBeans 開発者ガイド』を参照してください。

論理名を使用したリソースの参照

この項では、クライアント・アプリケーションで論理名を使用する方法について説明します。これにより、OC4J に固有ではないデプロイメント・ディスクリプタに含まれるインストール環境固有の JMS 構成に依存する設定の数を減らすことができます。この間接化により、クライアント実装を任意の JMS 構成に対して汎用的にすると同時に、特定の JMS リソース・プロバイダから分離することができます。

論理名を使用すると、リソース・プロバイダに依存しないクライアント・アプリケーション・コードを作成できます。論理名を構成および使用する一般的な手順は、次のとおりです。

1. クライアント・アプリケーション・コードで、JMS の宛先およびコネクション・ファクトリの論理名を使用します。
2. J2EE アプリケーション・コンポーネント・デプロイメント・ディスクリプタ（`application-client.xml` や `ejb-jar.xml` など）で論理名を宣言します。
3. OC4J 固有のアプリケーション・コンポーネント・デプロイメント・ディスクリプタ（`orion-application-client.xml` や `orion-ejb-jar.xml` など）の明示的な JNDI ロケーションに論理名をマップします。

論理名を構成して使用する方法は、`http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html` の How-To ドキュメントと .java および .xml ファイルを参照してください。

how-to-gjra-with-*xxx*.zip ファイルのいずれか（*xxx* はリソース・プロバイダの名前）をダウンロードして解凍します。

この項には、次の項目が含まれます。

クライアントでは、JMS の宛先とコネクション・ファクトリを使用してメッセージを送受信します。クライアントで JMS の宛先オブジェクトまたはコネクション・ファクトリを取得する場合は、論理名（環境エントリ）を使用することをお薦めします。一般的に、論理名を適用できない場合を除き、JNDI ロケーションを直接使用することは避けてください。または、アプリケーションの移植性を確保するために、その他のメカニズムを使用します。

論理名の宣言方法

アプリケーション・コードで論理名を使用するには、アプリケーションをデプロイする前に、次のいずれかの XML デプロイメント・ファイルに論理名を宣言する必要があります。

- スタンドアロン Java クライアントの場合 : `application-client.xml` ファイル
- EJB の場合 : `ejb-jar.xml` ファイル
- JSP またはサーブレットの場合 : `web.xml` ファイル

詳細は、`http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html` の How-To ドキュメントを参照してください。「Developing the Application Components」のタスク番号 2「Declare Logical Names for JMS Resources」を参照してください。

コネクション・ファクトリおよび宛先の論理名を宣言する手順は、次のとおりです。

- コネクション・ファクトリの論理名を <resource-ref> 要素を使用して宣言します。
  - コネクション・ファクトリの論理名を <res-ref-name> 要素に指定します。これは、図 4-1 の矢印番号 1 および 6 のリンク・キーです。
  - コネクション・ファクトリのクラス・タイプを、次のいずれかを使用して <res-type> 要素に指定します。
    * javax.jms.ConnectionFactory
    * javax.jms.QueueConnectionFactory
    * javax.jms.TopicConnectionFactory
  - 認証方式（Container または Application）を <res-auth> 要素に指定します。
  - 共有範囲（Shareable または Unshareable）を <res-sharing-scope> 要素に指定します。
- JMS 宛先の論理名を <message-destination-ref> 要素を使用して宣言します。
  - 宛先の論理名を <message-destination-ref-name> 要素に指定します。これは、図 4-1 の矢印番号 2 および 8 のリンク・キーです。
  - 宛先タイプを、javax.jms.Queue または javax.jms.Topic のいずれかを使用して <message-destination-ref-type> 要素に指定します。
  - クライアントでこの宛先へのメッセージを生成するか、この宛先からのメッセージを使用するか、またはその両方を行うかを、Produces、Consumes、ConsumesProduces のいずれかを使用して <message-destination-usage> 要素に指定します。

例

次の例では、キューの論理名を指定する方法を示します。この例では、jms/PlayerConnectionFactory がコネクション・ファクトリの論理名であり、jms/PlayerCommandDestination が宛先キューの論理名です。この例は、デモ・セットの application-client.xml ファイルからの抜粋です。

```
<resource-ref>
  <res-ref-name>jms/PlayerConnectionFactory</res-ref-name>
  <res-type>javax.jms.ConnectionFactory</res-type>
  <res-auth>Application</res-auth>
</resource-ref>
<message-destination-ref>
  <message-destination-ref-name>jms/PlayerCommandDest</message-destination-ref-name>
  <message-destination-type>javax.jms.Queue</message-destination-type>
  <message-destination-usage>Produces</message-destination-usage>
</message-destination-ref>
```

論理名を指定する方法の説明コメント付きの詳細な例は、http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html のデモに含まれます。

/src/client/META-INF/application-client.xml

および

/src/ejb/META-INF/ejb-jar.xml というファイルを見つけてください。

How-To ドキュメントおよびデモ・セットと、その URL のリストは、4-4 ページの「JMS の How-To ドキュメントおよびデモ・セット」を参照してください。

明示的な JNDI ロケーションへの論理名のマッピング

作成した論理名は、リソースの JNDI ロケーションにマップする必要があります。通常は、デプロイヤがこれらを設定します。このマッピングは、次のいずれかの OC4J デプロイメント・ディスクリプタ・ファイルで定義します。

- スタンドアロン Java クライアントの場合、そのクライアントの `orion-application-client.xml` ファイルに定義します。
- EJB の場合、その EJB の `orion-ejb-jar.xml` ファイルに定義します。
- JSP またはサーブレットの場合、その JSP またはサーブレットの `orion-web.xml` ファイルに定義します。

アプリケーション・コンポーネントのデプロイメント・ディスクリプタで論理名をマップする手順は、次のとおりです。

- コネクション・ファクトリの場合、`<resource-ref>` 要素に宣言されている論理名を、`<resource-ref-mapping>` 要素を使用して JNDI ロケーションにマップします。これは、図 4-1 の矢印番号 6 および 5 で示されます。
- 宛先の場合、`<message-destination-ref>` 要素を使用して宣言されている論理名を、`<message-destination-ref-mapping>` 要素を使用して JNDI ロケーションにマップします。これは、図 4-1 の矢印番号 8 および 7 で示されます。

OEMS JMS の 3 つのサービス品質オプションにおけるマッピングと、アプリケーション・コンポーネントでこのネーミング規則を使用する方法の詳細は、次の項を参照してください。

- OEMS JMS のメモリー内およびファイル・ベースのリソース名
- OEMS JMS のデータベース・オプションのリソース名
- Java アプリケーション・クライアントに対する JNDI ネーミング・プロパティの設定
- 論理名を使用したクライアントからの JMS メッセージ送信

Java アプリケーション・クライアントに対する JNDI ネーミング・プロパティの設定

OC4J スタンドアロン環境では、Java アプリケーション・クライアントは、`jndi.properties` ファイルの次のプロパティ定義に基づいて OEMS JMS の宛先オブジェクトにアクセスします。

```
java.naming.factory.initial=
   com.evermind.server.ApplicationClientInitialContextFactory
java.naming.provider.url=ormi://myhost/myApplicationDeploymentName
java.naming.security.principal=oc4jadmin
java.naming.security.credentials=welcome
```

ポートを指定する場合は、オプションの `:port` 要素を次のように使用します。

```
java.naming.provider.url=ormi://myhost:port/myApplicationDeploymentName
```

デフォルトの RMI ポートは、`23791` です。

次の操作が必要です。

- `ApplicationClientInitialContextFactory` を初期コンテキスト・ファクトリ・オブジェクトとして使用します。
- スタンドアロン OC4J のホストとポート、およびアプリケーション・デプロイ名をプロバイダ URL に指定します。

Oracle Application Server 環境では、Java アプリケーション・クライアントは、`jndi.properties` ファイルの次のプロパティ定義に基づいて OEMS JMS の宛先オブジェクトにアクセスします。

```
java.naming.factory.initial=
   com.evermind.server.ApplicationClientInitialContextFactory
java.naming.provider.url=opmn:ormi://$HOST:$OPMN_REQUEST_PORT:$OC4J_INSTANCE/myApplicat
```

```
ionDeploymentName
java.naming.security.principal=oc4jadmin
java.naming.security.credentials=welcome
```

- `ApplicationClientInitialContextFactory` を初期コンテキスト・ファクトリ・オブジェクトとして使用します。
- OPMN のホストとポート、OC4J インスタンスおよびアプリケーション・デプロイ名をプロバイダ URL に指定します。

`jndi.properties` ファイルの詳細な例は、`http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html` を参照してください。

次のファイルを見つけてください。

`/src/client/jndi.properties`

論理名を使用したクライアントからの JMS メッセージ送信

リソースを定義して JNDI プロパティを構成した後、クライアントでは、次の手順に従って JMS メッセージを送信する必要があります。この手順は、4-24 ページの「JMS メッセージの送受信」に記載されている手順を要約したものです。

1. JNDI ルックアップを使用して、構成済の JMS 宛先とそのコネクション・ファクトリを取得します。
2. コネクション・ファクトリから接続を作成します。
3. 接続を使用してセッションを作成します。
4. 取得済の JMS 宛先を指定して、メッセージ・プロデューサを作成します。
5. メッセージを作成します。
6. メッセージ・プロデューサを使用してメッセージを送信します。
7. 接続をクローズします。

JMS コネクタのルックアップを使用するアプリケーション・クライアントに必要なクラスパス

JMS コネクタをアプリケーション・クライアントから使用する場合、表 4-12「JMS コネクタのルックアップに必要なクライアント・サイドの JAR ファイル」にリストされたクラスパスに各 JAR ファイルが含まれている必要があります。

表 4-12 JMS コネクタのルックアップに必要なクライアント・サイドの JAR ファイル

JAR	ORACLE_HOME パス
oc4jclient.jar	/j2ee/<instance>
jta.jar	/j2ee/<instance>/lib
jms.jar	/j2ee/<instance>/lib
jndi.jar	/j2ee/<instance>/lib
javax77.jar	/j2ee/<instance>/lib
adminclient.jar	/j2ee/<instance>/lib
oc4j-internal.jar	/j2ee/<instance>/lib
connector.jar	/j2ee/<instance>/lib
jmxri.jar	/j2ee/<instance>/lib
jazncore.jar	/j2ee/<instance>
oc4j.jar	/j2ee/<instance>
dms.jar	/lib

OEMS JMS での高可用性およびクラスタリングの使用

可用性の高い JMS サーバーは、JMS 要求がソフトウェアまたはハードウェア障害による中断なしで処理されるという保証を提供します。高可用性を実現する方法の 1 つはフェイルオーバー機能を使用することです。サーバー・インスタンスの 1 つに障害が発生すると、ソフトウェア、ハードウェアおよびインフラストラクチャ・メカニズムの組合せにより、別のサーバー・インスタンスが要求の処理を引き継ぎます。

高可用性の詳細は、『Oracle Application Server 高可用性ガイド』を参照してください。

表 4-13「高可用性の概要」に、OEMS JMS での高可用性のサポートを示します。

表 4-13 高可用性の概要

機能	データベース	メモリー内およびファイル・ベース
高可用性	RAC + OPMN	OPMN
構成	RAC 構成、リソース・プロバイダ構成	専用 JMS サーバー、jms.xml 構成、opmn.xml 構成
メッセージ・ストア	RAC データベース上	専用 JMS サーバー / 永続性ファイル内
フェイルオーバー	同一または異なるマシン（RAC による）	Oracle Application Server Cold Failover Cluster（中間層）内の同一または異なるマシン

OEMS JMS のクラスタリング環境にデプロイされた JMS アプリケーションは、複数の OC4J インスタンスまたはプロセス間で JMS リクエストのロード・バランシングを実行できます。ステートレス・アプリケーションのクラスタリングは一般的に行われます。アプリケーションは複数のサーバーにデプロイされ、ユーザー・リクエストはそのうちの 1 つにルーティングされます。

JMS は、ステートフル API です。JMS クライアントと JMS サーバーの両方に、相互の状態（接続、セッション、永続サブスクリプションに関する情報など）が保持されます。ユーザーは、

その環境を構成し、クラスタ対応アプリケーションを記述するときに少数の単純なテクニックを使用できます。

次の項では、OEMS JMS で高可用性とクラスタリングを実現する方法について説明します。

- OEMS JMS のメモリー内およびファイル・ベースの高可用性の構成
- OEMS JMS のデータベース・オプションの高可用性の構成
- RAC データベース使用時のフェイルオーバー
- 接続リカバリのためのサンプル・コード
- クラスタリングのベスト・プラクティス

OEMS JMS のメモリー内およびファイル・ベースの高可用性の構成

OEMS JMS のクラスタリングでは、通常、この種の環境にデプロイされたアプリケーションが、複数の OC4J インスタンス間でメッセージのロード・バランシングを実行できます。また、この環境では、コンテナ・プロセスを複数のノード間で分散できるため、ある程度の高可用性が実現します。いずれかのプロセスまたはノードが停止した場合は、代替ノード上のプロセスがメッセージの処理を続行します。

この項には、OEMS JMS のクラスタリングに関する次の項目が含まれます。

- 用語
- 分散宛先

  この構成では、すべてのファクトリと宛先がすべての OC4J インスタンス上で定義されます。各 OC4J インスタンスには、それぞれの宛先の個別コピーがあります。

  この構成は、OC4J インスタンス間でリクエストのロード・バランシングを必要とする高スループットのアプリケーションに適しています。この使用例の場合、構成変更は不要です。

- Cold Failover Cluster

  この構成は 2 ノード・クラスタです。常にアクティブなノードは 1 つのみです。2 つ目のノードは、1 つ目のノードに障害が発生した場合にアクティブになります。

- 専用 JMS サーバー

  この構成では、1 つの OC4J インスタンス内の 1 つの JVM が専用 JMS サーバーとなります。JMS クライアントをホスティングする他のすべての OC4J インスタンスは、JMS 要求を専用 JMS サーバーに転送します。

  この構成は、メンテナンスとトラブルシューティングが最も容易であり、大多数の JMS アプリケーション、特にメッセージの順序付けを必要とするアプリケーションに適しています。

- カスタム・トポロジ

  この項では、カスタム・トポロジを使用する理由について説明し、様々なトポロジ・オプションの要件とその効果をリストします。また、カスタム・トポロジの作成方法について説明します。

  カスタム・トポロジは、本質的に、正しく理解して使用することが通常より難しく、構成の手間もかかります。カスタム・トポロジは、標準の構成が適切でない場合にのみ使用してください。

用語

ここで説明する用語の詳細は、『Oracle Application Server 高可用性ガイド』および『Oracle Process Manager and Notification Server 管理者ガイド』を参照してください。

- OHS: Oracle HTTP Server
- OracleAS クラスタ : 同じように構成された Oracle Application Server インスタンスのグループ
- Oracle Application Server インスタンス : Oracle Application Server のインストール環境（つまり、*ORACLE_HOME*）
- OC4J インスタンス : 1 つの Oracle Application Server インスタンス内には、複数の OC4J インスタンスが存在する場合があります。
- ファクトリ : JMS コネクション・ファクトリ
- 宛先 : JMS 宛先

分散宛先

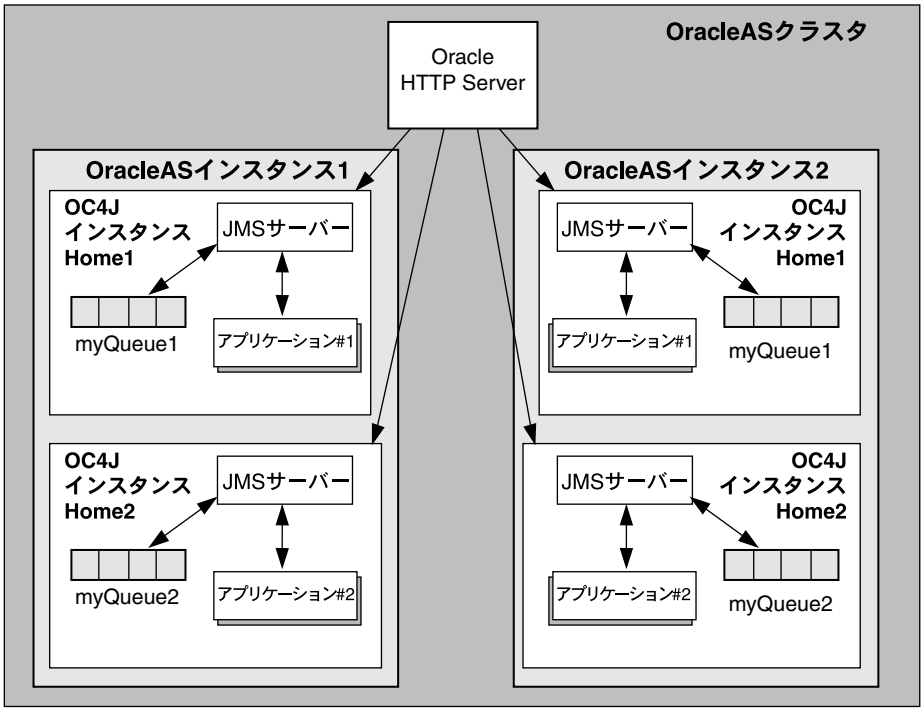
この構成では、OHS は、HTTP リクエストを処理し、Oracle Application Server クラスタ内の 2 つ以上の Oracle Application Server インスタンス間でリクエストのロード・バランシングを実行します。

宛先のコピーはそれぞれ一意で、OC4J インスタンス間ではレプリケートまたは同期化されません。宛先は、永続的でもメモリー内でもかまいません。1 つの OC4J インスタンスにエンキューされたメッセージは、その OC4J インスタンスからでなければデキューできません。

この種のデプロイメントには、次のような多数のメリットがあります。

- アプリケーションと JMS サーバーの両方が同じ JVM 内で実行され、プロセス間通信が不要であるため、高スループットが実現します。
- ロード・バランシングにより、高スループットと高可用性が促進されます。
- シングル・ポイント障害が発生しません。OC4J プロセスが 1 つでも使用可能であれば、リクエストを処理できます。
- JMS 操作に影響を与えずに Oracle Application Server インスタンスをクラスタリングできます。
- 宛先オブジェクトは、メモリー内またはファイル・ベースのいずれでもかまいません。

図 4-2 分散宛先



分散宛先クラスタの構成

各 Oracle Application Server インスタンス内に、2 つの OC4J インスタンスが定義されています。これらの OC4J インスタンスでは、それぞれ別のアプリケーションが実行されます。つまり、OC4J インスタンス #1（`Home1`）はアプリケーション #1 を実行し、OC4J インスタンス #2（`Home2`）はアプリケーション #2 を実行します。各 OC4J インスタンスは複数の JVM を実行するよう構成できるため、アプリケーションは複数の JVM 間でスケーラビリティを持つことができます。

Oracle Application Server クラスタ内では、各 Oracle Application Server インスタンスの構成情報は（ホスト名やポート番号など、インスタンス固有の情報を除いて）同一です。つまり、Oracle Application Server インスタンス #1 の OC4J インスタンス #1 にデプロイされたアプリケーション #1 は、Oracle Application Server インスタンス #2 の OC4J インスタンス #1 にもデプロイされます。このタイプのアーキテクチャでは、複数の Oracle Application Server インスタンス間でメッセージのロード・バランシングを実行できます。特に、ハードウェア障害への対策として Oracle Application Server インスタンス #2 が別のノードにデプロイされている場合、JMS アプリケーションの高可用性が実現します。

各アプリケーションのセンダーとレシーバは、1 つの OC4J インスタンスにともにデプロイする必要があります。つまり、ある OC4J プロセスで JMS サーバーにエンキューされたメッセージは、その OC4J プロセスからでなければデキューできません。

すべてのファクトリと宛先は、すべての OC4J プロセス上で定義されます。各 OC4J プロセスには、それぞれの宛先の個別コピーがあります。宛先のコピーは、レプリケートまたは同期化されません。そのため、この図では、アプリケーション #1 は `myQueue1` という宛先に書込みを行います。この宛先は、2 つの場所（Oracle Application Server インスタンス #1 および #2）に物理的に存在し、各 OC4J インスタンス内でそれぞれの JMS サーバーにより管理されます。

この種の JMS デプロイメントは、特定のタイプの JMS アプリケーションにのみ適していることに注意してください。メッセージの順序が重要でない場合、メッセージは同じ名前を持つ分散キューにエンキューされます。JMS の Point-to-Point メッセージングのセマンティクスでは、メッセージは複数のキュー間で複製できません。前述の例では、メッセージはロード・バランシング・アルゴリズムにより決定されたキューに送信され、着信時に MDB によりデキューされます。

Cold Failover Cluster

この構成は 2 ノード・クラスタです。常にアクティブなノードは 1 つのみです。2 つ目のノードは、1 つ目のノードに障害が発生した場合にアクティブになります。より大規模なクラスタでは、次の項で説明する専用 JMS サーバー構成と Cold Failover Cluster を組み合せて使用できます。この場合、2 つのノードを専用 JMS サーバーとして構成し、任意の時点で 1 つのノードのみがアクティブになるよう設定します。コールド・フェイルオーバーの詳細は、『Oracle Application Server 高可用性ガイド』を参照してください。

Cold Failover Cluster の構成

次の例に示すように、両方のノードを同じ構成にします。両方の OC4J インスタンスの `jms.xml` ファイルを変更します。`jms-server` の host パラメータを次のように設定します。

```
<jms-server host=vmt.my.company.com port="9127">
….
….
</jms-server>
```

ファイル・ベースのメッセージの永続性をキューに使用する場合、両方のノードからアクセスできる共有ディスク上にファイルを置く必要があります。共有ディスクは、一方のノードから他方のノードにフェイルオーバーする際に、仮想 IP を使用する必要があります。次のように `persistence-file` を構成します。

```
<queue name="Demo Queue" location="jms/demoQueue"
persistence-file="/path/to/shared_file_system/demoQueueFile">
        <description>A dummy queue</description>
</queue>
```

停止と起動

各ノードでは、次のコマンドを使用して停止と起動を行います。

```
$ORACLE_HOME/opmn/bin/opmnctl stopall
$ORACLE_HOME/opmn/bin/opmnctl startall
```

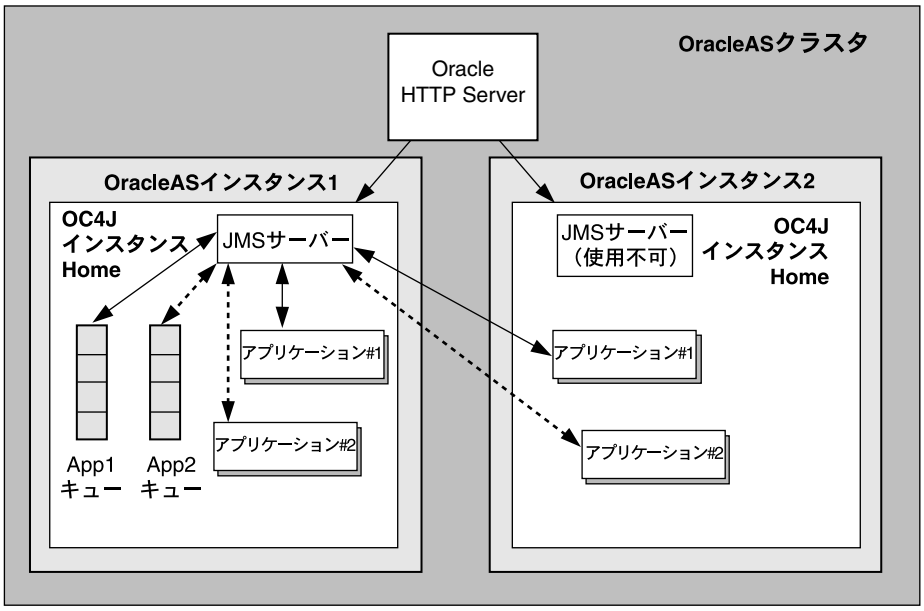
専用 JMS サーバー

この構成では、Oracle Application Server クラスタ環境で 1 つの OC4J インスタンスが専用 JMS サーバーとして構成されます。この OC4J インスタンスはすべてのメッセージを処理するため、常にメッセージの順序付けが維持されます。すべての JMS アプリケーションは、この専用サーバーを使用してコネクション・ファクトリと宛先をホスティングし、エンキューおよびデキューのリクエストを処理します。

専用 JMS プロバイダとして機能する OC4J JVM は、クラスタ内のすべての JMS アプリケーションに対して 1 つのみです。そのために、`opmn.xml` ファイル内の JMS ポート範囲は専用 OC4J インスタンス用の 1 つのポートのみに制限されます。

次の図は、OC4J の Home インスタンス内のアクティブな JMS サーバーを示していますが、JMS プロバイダは独自の OC4J インスタンス内でホスティングすることをお薦めします。たとえば、Home は Oracle Application Server のインストール後に実行されるデフォルトの OC4J インスタンスですが、Oracle Enterprise Manager 10*g* Application Server Control コンソールを使用して 2 つ目の OC4J インスタンスを作成する必要があります。後述の opmn.xml ファイルの例では、JMSserver という OC4J インスタンスを作成しています。

図 4-3 専用 JMS サーバー



OC4J インスタンス JMSserver を作成した後、この Oracle Application Server インスタンスについて opmn.xml ファイルに次の 2 つの変更を加える必要があります。

1. この OC4J インスタンス（JMSserver）用に起動される JVM が 1 つのみであることを確認します。

   OC4J インスタンス内の JVM を 1 つに限定することで、他の JVM が同じ永続性ファイル・セットを使用しないことが保証されます。

2. このインスタンスの JMS ポート範囲の値を 1 つのみ指定します。

   ポート値を 1 つにすると、OPMN では、常にこの値が専用 JMS サーバーに割り当てられます。このポート値を使用して、jms.xml ファイル内でコネクション・ファクトリを定義できます。他の OC4J インスタンスは、これを専用 JMS サーバーへの接続に使用します。

OPMN と動的ポート割当ての詳細は、『Oracle Process Manager and Notification Server 管理者ガイド』を参照してください。

OPMN 構成の変更

次の XML は、opmn.xml ファイルからの抜粋であり、必要な変更内容と変更する場所を示しています。

- Application Server Control コンソールを使用して OC4J インスタンス JMSserver が作成されたと仮定すると、(1) で示されている行は、JMSserver 定義の開始位置を示しています。
- (2) で示されている行は、OC4J JVM に JMS ポートを割り当てるときに OPMN で使用される JMS ポート範囲です。JMS プロバイダとして機能させる必要のある専用 OC4J インスタンスについて、この範囲を 1 つの値に限定します。この例では、元の範囲は 3701-3800 でした。今回のコネクション・ファクトリ定義では、この値を 3701-3701 に構成してポートを使用します。

> **注意：** 専用 JMS サーバーに割り当てられたポートは、このホストのその他のアプリケーションで使用されないようにしておく必要があります。また、このポートはこのホストを共有するすべての OC4J インスタンスに対する、全 OC4J コンポーネントのポート範囲から除外する必要があります。

- (3) で示されている行は、JMSserver のデフォルト・グループに含まれる JVM 数の定義です。この値は、デフォルトで 1 に設定されています。この値は、常に 1 である必要があります。

```
<ias-component id="OC4J">
  (1) <process-type id="JMSserver" module-id="OC4J" status="enabled">
     <module-data>
       <category id="start-parameters">
         <data id="java-options" value="-server
           -Djava.security.policy=$ORACLE_HOME/j2ee/home/config/java2.policy
           -Djava.awt.headless=true
         "/>
       </category>
       <category id="stop-parameters">
         <data id="java-options"
           value="-Djava.security.policy=
             $ORACLE_HOME/j2ee/home/config/java2.policy
           -Djava.awt.headless=true"/>
       </category>
    </module-data>
    <start timeout="600" retry="2"/>
    <stop timeout="120"/>
    <restart timeout="720" retry="2"/>
    <port id="ajp" range="3000-3100"/>
    <port id="rmi" range="3201-3300"/>
    (2) <port id="jms" range="3701-3701"/>
    (3) <process-set id="default_group" numprocs="1"/>
  </process-type>
</ias-component>
```

OEMS JMS の構成 この使用例で前述したように、OC4J インスタンスの 1 つは JMS サーバー専用です。他の OC4J インスタンスおよび OC4J 外部で実行されるスタンドアロン JMS クライアントは、JMS 要求を専用 JMS サーバーに転送するように設定する必要があります。すべてのコネクション・ファクトリと宛先は、JMS サーバー・インスタンスの `jms.xml` ファイル内で定義されます。この `jms.xml` ファイルを、JMS サーバーと通信する他のすべての OC4J インスタンスにコピーする必要があります。

専用 JMS サーバー上の `jms.xml` ファイル内で構成するコネクション・ファクトリでは、サーバーのホスト名とポート番号を明示的に指定する必要があります。`jms.xml` のホスト名とポート番号は、前述のとおり専用サーバー用に OPMN で定義されているホスト名および単一のポート番号と一致する必要があります。他のすべての OC4J インスタンスでも同じコネクション・ファクトリ構成を使用します。これにより、すべての OC4J インスタンスが、操作時に専用 JMS サーバーを参照することになります。

したがって、専用 JMS サーバーが `host1` のポート `3701` で実行される場合、クラスタ内の各 OC4J インスタンス用の `jms.xml` ファイルに定義されるすべてのコネクション・ファクトリは、`host1` のポート `3701` を指し示す必要があります。このポートは、専用 JMS サーバーのための専用 OC4J インスタンス（この例では `JMSserver`）で使用される `opmn.xml` ファイルに記述された単一のポートです。

専用 JMS サーバー上の `jms.xml` ファイル内で構成されている宛先は、他のすべての OC4J インスタンス上でも構成する必要があります。ただし、これらの宛先の物理ストアは専用 JMS サーバー上にあります。

キュー・コネクション・ファクトリ定義の例

次に、専用サーバーの `jms.xml` ファイル内でキュー・コネクション・ファクトリを定義する例を示します。

```
<!-- Queue connection factory -->
<queue-connection-factory name="jms/MyQueueConnectionFactory"
          host="host1" port="3701"
          location="jms/MyQueueConnectionFactory"/>
```

管理上の変更（新規宛先オブジェクトの追加）は、専用 JMS サーバーの `jms.xml` ファイルに加える必要があります。次に、JMS アプリケーションを実行する他のすべての OC4J インスタンスの `jms.xml` ファイル内で、同じ変更を実行します。この変更には、手動で実行する方法と、専用 JMS サーバーの `jms.xml` ファイルを他の OC4J インスタンスにコピーする方法があります。

JMS コネクタのデプロイ JMS コネクタは、専用 JMS サーバーのメッセージを送受信するアプリケーションのホストである各 OC4J インスタンスにデプロイされます。この JMS コネクタにより、専用 JMS サーバーを参照する、ローカルに構成された JMS コネクション・ファクトリの JNDI マッピングが提供されます。OC4J インスタンスにデプロイされたアプリケーションは、JMS コネクタ実装を介して基礎となる JMS 管理オブジェクトにアクセスできます。JMS コネクタの使用方法の詳細は、4-51 ページの「JMS コネクタ」を参照してください。

JMS コネクタ用のキュー・コネクション・ファクトリの定義

次の例では、事前に（前の例を参照）jms.xml ファイルに定義されているキュー・コネクション・ファクトリ（jms/MyQueueConnectionFactory）をマップする oc4j-ra.xml ファイルにキュー・コネクション・ファクトリを定義しています。

```
<connector-factory location="MyJMSConnector/QCF"
   connector-name="MyJMSConnector">
   <config-property name="jndiLocation" value="jms/MyQueueConnectionFactory"/>
   <connection-pooling use="private">
      <property name="waitTimeout" value="300" />
      <property name="scheme" value="fixed_wait" />
      <property name="maxConnections" value="50" />
      <property name="minConnections" value="0" />
   </connection-pooling>
   <security-config use="principal-mapping-entries">
      <principal-mapping-entries>
         <principal-mapping-entry>
            <initiating-user>servletuser</initiating-user>
            <res-user>jmsuser</res-user>
            <res-password>->jmsuser</res-password>
         </principal-mapping-entry>
      </principal-mapping-entries>
   </security-config>
   <connectionfactory-interface>
      javax.jms.QueueConnectionFactory</connectionfactory-interface>
</connector-factory>
```

この例で説明すると、OC4J インスタンスにデプロイされたアプリケーションはロケーション MyJMSConnector/QCF を使用して専用サーバーのコネクション・ファクトリをルックアップしています。このロケーションは、jms.xml ファイルに定義されている実際の JMS キュー・コネクション・ファクトリの JNDI ロケーション（jms/MyQueueConnectionFactory）にマップされています。リソース・アダプタの構成やデプロイ、および JMS コネクタの構成の詳細は、4-51 ページの「JMS コネクタ」を参照してください。

アプリケーションのデプロイ JMS アプリケーションが実際にデプロイされる場所は、ユーザーが決定します。専用 JMS サーバーは、JMS 要求を処理する一方で、デプロイされた JMS アプリケーションも実行できます。また、JMS アプリケーションは他の OC4J インスタンス（つまり Home）にもデプロイできます。

専用 JMS サーバーの jms.xml ファイルは、JMS アプリケーションがデプロイされる OC4J インスタンスすべてに伝播させる必要があることに注意してください。JMS アプリケーションは、別の JVM で実行中のスタンドアロン JMS クライアントにデプロイすることもできます。

高可用性 OPMN には、専用 JMS サーバーの稼働を維持するために、フェイルオーバー・メカニズムが用意されています。なんらかの理由で JMS サーバーに障害が発生すると、OPMN はそれを検出して JVM を再起動します。ハードウェア障害が発生した場合、メッセージをリカバリする唯一の方法は、永続する宛先がネットワーク・ファイル・システム上でホスティングされるようにすることです。OC4J インスタンスを起動し、これらの永続するファイルを指すように構成できます。

OPMN による Oracle Application Server プロセスの管理の詳細は、『Oracle Process Manager and Notification Server 管理者ガイド』を参照してください。

カスタム・トポロジ

前述の構成に加え、OEMS JMS は、任意のユーザー定義トポロジとなるように構成できます。

メカニズム

この項では、カスタム・トポロジを作成して機能させるためのメカニズムについて説明します。これらのメカニズムは、このマニュアルの別の項でもすでに定義されていますが、この項では、カスタム・トポロジを使用する場合に理解する必要のあるより高度な内容について説明します。

ローカルの OC4J JVM のサーバー以外に JMS サーバーを使用する場合、目的の JMS サーバーを参照するコネクション・ファクトリが必要です。この構成には、ホストおよびポートのマッピングを使用します。

最初に、JMS サーバーをホストとポートに関連付ける必要があります。通常、ホストは、JMS サーバーが稼働するマシンの単一のホスト（IP アドレス）です。複数のホスト（IP アドレス）を持つマシンの場合、特定のアドレスを `jms.xml` ファイルの `<jms-server>` 要素の `host` 属性を使用して指定しないかぎり、すべてのアドレスが JMS サーバーに関連付けられます。`port` は、`port` 属性を使用して異なる値を指定しないかぎり、`9127` です。4-19 ページの表 4-1「構成要素」の `<jms-server>` エントリと、4-24 ページの「ポートの構成」を参照してください。OPMN が使用される場合、`port` 属性は無視され、ポート割当ては `opmn.xml` ファイルに設定されたポート範囲の値に基づいて自動的に処理されます。4-73 ページの「OPMN 構成の変更」を参照してください。

各 JMS サーバーにホストとポートを指定したら、特定の JMS サーバーを参照するためのコネクション・ファクトリを作成できます。コネクション・ファクトリで特定の JMS サーバーを参照するための最初の手順では、コネクション・ファクトリ要素の `host` 属性を、特定の JMS サーバーのホスト（IP アドレス）と同じ値に設定します。4-19 ページの表 4-1「構成要素」で、`<connection-factory>` や `<xa-connection-factory>` などのコネクション・ファクトリ要素を参照してください。単一のホスト（IP アドレス）を持つ同じマシン上の JMS サーバーをコネクション・ファクトリで参照する場合、この手順は省略できます（`host` 属性は無視できます）。コネクション・ファクトリで特定の JMS サーバーを参照するための次の手順では、コネクション・ファクトリ要素の `port` 属性を、特定の JMS サーバーのポートと同じ値に設定します。ポートが `9127` の場合、この手順は省略できます（`port` 属性は無視できます）。

ここまでの手順によって JMS サーバー JMS1 を参照するためのコネクション・ファクトリ CF1 が構成された場合、CF1 から作成される接続は、JMS1 への接続になります。この接続を通じて実行されるすべての操作は、JMS1 を対象とします。この接続を使用して他の JMS サーバーと通信することはできません。（JMS1 を対象とするものでないかぎり、この接続を使用してローカルの JVM で稼働する JMS サーバーと通信することもできません。）同様に、この接続を使用してセッションを作成し、そのセッションでメッセージ・プロデューサを作成すると、そのセッションとメッセージ・プロデューサは両方とも JMS1 に接続されます。このメッセージ・プロデューサを通じて送信されるすべてのメッセージは、JMS1 を対象とし、JMS1 によって（メモリー内またはファイルに）格納され、（JMS1 に接続されたメッセージ・コンシューマまたはキュー・ブラウザを使用して）JMS1 からのみ受信または参照可能となります。ただし、この JMS サーバーとの関連付けは、`javax.jms.Message` オブジェクトには拡張されません。ある JMS サーバーと関連付けられたセッションやメッセージ・プロデューサから作成（または受信）されたメッセージ・オブジェクトは、任意のメッセージ・プロデューサを使用して送信できます。つまり、アプリケーションでは、JMS サーバー JMS1 で管理されている物理宛先から（JMS1 に関連付けられたメッセージ・コンシューマを使用して）メッセージを受信した後に、そのメッセージを JMS サーバー JMS2 で管理されている物理宛先に（JMS2 に関連付けられたメッセージ・プロデューサを使用して）送信することが可能です。

複数の JMS サーバーを扱う場合は、物理宛先（メッセージが格納される物理的な場所）、宛先構成（`jms.xml` の要素と属性）および宛先管理オブジェクト（JNDI でルックアップされる Java オブジェクト）を区別する必要があります。

JMS サーバーの観点から検討すると、次のようになります。

- 名前は（宛先構成または宛先管理オブジェクトのいずれにおける名前でも）、特定の JMS サーバーによって個別に管理されている一意の物理宛先を示します。
- `persistence-file` 属性（またはこの属性の欠如）は、JMS サーバーに対し、特定の物理宛先のメッセージを永続化する場所を示します。

- location 属性は、JMS サーバーに対し、(name 属性の値を含む) 宛先管理オブジェクトをバインドするローカル JVM の JNDI ロケーションを示します。

JMS クライアントの観点から検討すると、次のようになります。

- 宛先管理オブジェクトには、JNDI ロケーションは含まれません。かわりに、それらは JNDI ロケーションでルックアップされます。
- 宛先管理オブジェクトには、永続性ファイルの設定は含まれません。かわりに、特定の物理宛先で使用される永続性ファイルは、その物理宛先を管理する JMS サーバーによって決定されます (永続性ファイルが存在する場合)。
- 宛先管理オブジェクトには、名前のみが含まれ、それらの名前は物理宛先を一意に識別しません。JMS サーバー JMS1 と通信する場合、宛先管理オブジェクトは、JMS1 によって管理される物理宛先への参照とみなされます。JMS サーバー JMS2 と通信する場合、同じ宛先管理オブジェクトは、JMS2 によって管理される別の物理宛先への参照とみなされます。(JMS サーバーとの通信時に、特定の JMS サーバーの物理宛先名と一致しない名前を持つ宛先管理オブジェクトが使用されると、操作は失敗します。)

たとえば、JMS サーバー JMS1 の jms.xml ファイルが次のように記述されているとします。

```
<queue name="queue1"
       location="jms/alpha"
       persistence-file="foo">
</queue>
```

また、JMS サーバー JMS2 の jms.xml ファイルが次のように記述されているとします。

```
<queue name="queue1"
       location="jms/omega"
       persistence-file="bar">
</queue>

<queue name="queue2"
       location="jms/alpha">
</queue>
```

4-77 ページの表 4-14「様々なメッセージ使用例の結果」は、宛先が列 1 のサーバーから列 2 の JNDI ロケーションを使用してルックアップされ、列 3 のサーバーに接続されたメッセージ・プロデューサを通じて PERSISTENT メッセージが送信された場合に、列 4 の結果となることを示しています。

表 4-14 様々なメッセージ使用例の結果

宛先がルックアップされる JVM 内のサーバー	宛先のルックアップに使用される JNDI ロケーション	メッセージ・プロデューサの接続先	結果 (メッセージが受信される物理宛先またはエラー)
JMS1	jms/alpha	JMS1	JMS1 の queue1、ファイル foo に永続化
JMS1	jms/alpha	JMS2	JMS2 の queue1、ファイル bar に永続化
JMS2	jms/alpha	JMS2	JMS2 の queue2、JMS2 のメモリー内に格納
JMS2	jms/alpha	JMS1	エラー (JMS1 に queue2 はないため)
JMS1	jms/omega	任意	JNDI での jms/omega のルックアップ・エラー
JMS2	jms/omega	JMS1	JMS1 の queue1、ファイル foo に永続化
JMS2	jms/omega	JMS2	JMS2 の queue1、ファイル bar に永続化

この例 (および分散宛先構成でのコネクション・ファクトリ) では、様々な JVM で同じ JNDI ロケーションに異なる値をバインドしているか、異なる値を必要とするため、ある JVM から別の JVM に JNDI 値を自動的に伝播できません。

次の組合せを使用する場合について検討します。

- OPMN（自動 JMS ポート割当て）
- 1 つ以上の専用 JMS サーバー
- 1 つのマシンにおける複数の OC4J インスタンス

この組合せでは、次の要件に注意する必要があります。

- 使用可能な JMS ポートの範囲が、特定のマシン上の JMS サーバーの数に対して十分であること。これにより、専用 JMS サーバーのポート番号が、使用可能な JMS サーバーのいずれかに実際に割り当てられることが保証されます。
- 専用 JMS サーバーによって管理される 1 つ以上の宛先の永続性ファイルが存在する場合、絶対パスを使用して `jms.xml` ファイルに指定すること。パスが OC4J インスタンスに対する相対パスである場合、OPMN が専用 JMS サーバーのポート番号を前回の再起動時とは異なる OC4J インスタンスの JMS サーバーに割り当てると、以前に永続化されたメッセージはサーバーの再起動後に失われます。

考慮事項

専用 JMS サーバー構成と分散宛先構成は、任意の JMS アプリケーションの各インスタンスが常に単一の JMS サーバーとのみ通信することが保証されている構成です。この項で説明する考慮事項は、これらの使用例には適用されません。同様に、JMS アプリケーション・インスタンスが常に単一の JMS サーバーとのみ通信することが保証される他の使用例にも適用されません。

たとえば、アプリケーション A のすべてのインスタンスで JMS サーバー #1 を使用し、アプリケーション B のすべてのインスタンスで JMS サーバー #2 を使用する場合、次の考慮事項は適用されません。

ただし、単一の JMS アプリケーション・インスタンスが 2 つ以上の JMS サーバーと通信するその他の使用例では、次のようにいくつかの考慮事項があります。

- アプリケーションが複雑化する問題

  JMS サーバーは、コネクション・ファクトリ（およびその導出オブジェクト）に基づいて選択されるため、1 つのアプリケーション・インスタンスで複数の JMS サーバーを使用する場合、複数のコネクション・ファクトリ（およびセッション、コンシューマ、プロデューサなどの導出オブジェクト）を使用する必要があります。たとえば、アプリケーションでプロデューサ A を使用してメッセージを JMS サーバー #1 にエンキューする場合、同じプロデューサ A を使用してメッセージを JMS サーバー #2 にエンキューすることはできません。かわりに、プロデューサ A が導出されたコネクション・ファクトリとは別のコネクション・ファクトリから導出されたプロデューサを使用する必要があります。（具体的には、JMS サーバー #2 に関連付けられたコネクション・ファクトリから導出されたプロデューサを使用します。）つまり、アプリケーションがこの前提条件にまだ対応していない場合や、対応するように変更できない場合は、既存の JMS アプリケーションで複数の JMS サーバーを使用できない可能性があります。

- パフォーマンス上の問題

  2 つの異なる JMS サーバーが 2 つの異なるリソース・マネージャに相当するため、2 つの JMS サーバーに関連するグローバル・トランザクションでは、常に 2 フェーズ・コミットを必要とします（JDBC などの他のリソース・タイプがトランザクション中に使用されない場合でも同様です）。特定のトランザクションですでに 2 フェーズ・コミットを必要としていた場合でも、トランザクションに参加するリソース・マネージャの増加は、トランザクションの（パフォーマンス上の）コストの増大につながります。ただし、複数の JMS サーバーを使用することによるパフォーマンス上のメリットも多くあります。複数の JMS サーバーの使用により、潜在的なボトルネック（単一の専用 JMS サーバーなど）に起因する作業負荷を軽減し、並列性と局所性の両方を向上することが可能となるためです。

これらの考慮事項は、単一の JMS アプリケーション・インスタンスで 2 つの異なる JMS リソース・プロバイダを使用する場合にも適用されます。たとえば、アプリケーションでは、メモリー・ベースまたはファイル・ベースの永続性を確保するために OEMS JMS のメモリー内のオプションを使用し、同時に、データベースに基づく永続性を確保するために OEMS JMS のデータベースのオプションを使用する可能性があります。

使用例

この項では、アプリケーション固有のメッセージ要件が存在する 2 つの使用例と、各使用例でスループットを改善するために使用できるカスタム・トポロジについて説明します。これらの例は、実際の状況的にも作成可能なトポロジ的にも、完全なものではありません。

使用例 1: 一部の宛先のみでグローバル整合性を必要とする場合

一部の宛先では専用 JMS サーバー構成に基づくグローバル整合性を必要とする一方で、他の宛先にはそのような要件はないことがあります。この場合、グローバル整合性（単一の JMS サーバーを通じたすべてのメッセージのルーティング）を必要としない宛先でその要件を無理に満たす必要はありません。かわりに、分散宛先構成などを使用してローカルに分割することが可能です。

たとえば、グローバル整合性を必要とする宛先 A および B と、必要としない宛先 C および D があるとします。この場合、すべての jms.xml ファイルで、4 つのすべての宛先を定義します。宛先 A および B 用に、専用 JMS サーバーとして機能する 1 つのマシンを選択します。コネクション・ファクトリで参照できるように、専用 JMS サーバーには、固定のホスト値とポート値を割り当てる必要があります。jms.xml ファイルには、少なくとも 2 つのコネクション・ファクトリを定義します。1 つのコネクション・ファクトリは（固定のホスト値とポート値を持つ）専用 JMS サーバーを参照し、もう 1 つはローカル JVM 内の（デフォルトのホスト値とポート値を持つ）JMS サーバーを参照します。宛先 A および B にアクセスするには（グローバル整合性を確保するには）、Java コードで、専用 JMS サーバーを参照するコネクション・ファクトリを使用します。宛先 C および D にアクセスするには、ローカル JVM 内の JMS サーバーを参照するコネクション・ファクトリを使用します。

使用例 2: ロード・バランシングのために複数の専用 JMS サーバーを使用する場合

前の例では、2 つの宛先（A および B）がグローバル整合性を必要としていました。この場合、宛先 A と宛先 B の両方に専用 JMS サーバーを用意する必要がありますが、これら 2 つの JMS サーバーは同じである必要はありません。グローバル整合性を必要とする複数の宛先を頻繁に使用する場合は、それらの宛先に対する負荷を複数の専用 JMS サーバー間で分散すると効果的である可能性があります。特にこの方法が有効なのは、宛先が一般的なトランザクションにあまり関与しない場合や、複数の宛先を処理する専用 JMS サーバーがシステムのボトルネックとなっている場合です。

この場合、すべての jms.xml ファイルで、4 つのすべての宛先を定義します。宛先 A 用に、専用 JMS サーバーとして機能する 1 つのマシンを選択します。宛先 B 用に、専用 JMS サーバーとして機能するもう 1 つのマシン（または同じマシン上の別の JVM）を選択します。コネクション・ファクトリで参照できるように、これら 2 つの専用 JMS サーバーには、固定のホスト値とポート値を割り当てる必要があります。jms.xml ファイルには、少なくとも 3 つのコネクション・ファクトリを定義します。1 つのコネクション・ファクトリは宛先 A の専用 JMS サーバーを、1 つは宛先 B の専用 JMS サーバーを、1 つはローカル JVM 内の JMS サーバーをそれぞれ参照します。宛先 A にアクセスするには、Java コードで、宛先 A の専用 JMS サーバーを参照するコネクション・ファクトリを使用します。宛先 B にアクセスするには、宛先 B の専用 JMS サーバーを参照するコネクション・ファクトリを使用します。宛先 C および D にアクセスするには、ローカル JVM の JMS サーバーを参照するコネクション・ファクトリを使用します。

OEMS JMS のデータベース・オプションの高可用性の構成

OEMS JMS のデータベース・オプションで高可用性を実現するには、次のようにします。

- AQ キューおよびトピックを含む Oracle データベースを RAC モードで実行します。これにより、データベースの高可用性が保証されます。
- Oracle Application Server を OPMN モードで実行します。これにより、アプリケーション・サーバー（およびそこにデプロイされたアプリケーション）の高可用性が保証されます。

Oracle Application Server クラスタ内の各アプリケーション・インスタンスは、OC4J リソース・プロバイダを使用して、RAC モードで稼働しているバックエンド Oracle データベースを参照します。これらのリソース・プロバイダから導出されたオブジェクトで起動される JMS 操作は、Real Application Clusters（RAC）データベースに送られます。

アプリケーション障害が発生すると、そのアプリケーション内の状態情報は失われます（つまり、接続、セッションおよびメッセージの状態はコミットされていません）。アプリケーション・サーバーが再起動されると、アプリケーションは JMS 状態を適切に再作成して操作を再開する必要があります。

バックエンド・データベース（非 RAC データベース）のネットワーク・フェイルオーバーが発生すると、サーバー内の状態情報は失われます（つまり、トランザクションの状態はコミットされていません）。また、アプリケーション内部の JMS オブジェクト（コネクション・ファクトリ、宛先オブジェクト、接続、セッションなど）も無効になることがあります。

データベース障害の発生後に、アプリケーション・コードでこれらの JMS オブジェクトを使用すると、常に例外がスローされます。リカバリするためには、このような例外をスローしているすべての JMS オブジェクトをアプリケーションで再作成する必要があります。この処理には、JNDI を使用したコネクション・ファクトリの再ルックアップも含まれます。

RAC データベース使用時のフェイルオーバー

RAC データベースを使用するアプリケーションでは、データベース・フェイルオーバーを使用する必要があります。この項で説明されているように、フェイルオーバーには 2 つのタイプがあります。

- RAC ネットワーク・フェイルオーバー
- 透過的アプリケーション・フェイルオーバー（TAF）

RAC ネットワーク・フェイルオーバー

RAC データベースに対して実行するスタンドアロン・クライアントでは、API `oracle.oc4j.sql.DataSourceUtils.isOracleFatalError()` を起動して接続オブジェクトが無効であるかどうかを判断し、接続を再取得するためのコードを記述する必要があります。その後、必要に応じてデータベース接続を再確立します。`isOracleFatalError()` メソッドは、ネットワーク・フェイルオーバー中にデータベースによりスローされた SQL エラーが致命的エラーであるかどうかを検出します。このメソッドは SQL エラーとデータベース接続を取得して、致命的エラーの場合は `true` を戻します。`true` の場合は、トランザクションを積極的にロールバックし、JMS の状態（失われた接続、セッションおよびメッセージなど）を再作成する必要があります。

次の例にこのロジックの概略を示します。

```
Message getMessage(QueueSession session, Queue rcvrQueue) {
   try {
      Message msgRec = null;
      QueueReceiver rcvr = session.createReceiver(rcvrQueue);
      msgRec = rcvr.receive();
      rcvr.close();
      return msgRec;
   }
   catch (Exception ex) {
      if (ex instanceof JMSException) {
         Exception exLinked = ((JMSException) ex).getLinkedException();
```

```
            if (exLinked instanceof SQLException) {
               if (DataSourceUtils.isOracleFatalError((SQLException) exLinked) {
                  // failover logic
               }
            }
         }
      }
   }
```

透過的アプリケーション・フェイルオーバー（TAF）

TAF が構成されているほとんどの場合、アプリケーションは他のデータベース・インスタンスへのフェイルオーバーが発生したことを認識しません。そのため、障害からリカバリするために何かする必要はありません。

ただし、障害の発生時に OC4J から ORA エラーがスローされる場合があります。JMS クライアントは、これらのエラーを関連する SQL 例外とともに `JMSException` としてユーザーに渡します。この場合は、次の 1 つ以上の操作を実行してください。

- エラーが致命的エラーであるかどうかは、`isOracleFatalError()` メソッドを使用して判断できます。4-80 ページの「RAC ネットワーク・フェイルオーバー」を参照してください。致命的エラーでない場合、クライアントは短時間だけスリープした後、現行の操作を再試行してリカバリします。
- 少し待ってから JMS コネクションの使用を試行することで、不完全なフェイルオーバーにより発生したフェイルバックと一時的なエラーからリカバリできます。待っている間に、データベースはフェイルオーバーして障害からリカバリし、データベース自体を元の状態に戻します。
- トランザクション例外（「トランザクションをロールバックしてください。」（ORA-25402）、「トランザクションのステータスが不明です。」（ORA-25405）など）の場合は、現行の操作をロールバックし、前回のコミット後の操作をすべて再試行する必要があります。例外の原因が解消されるまで、接続は使用できません。この再試行に失敗した場合は、すべての接続をクローズして再作成し、コミットされていない操作をすべて再試行します。

接続リカバリのためのサンプル・コード

次の例は、任意の永続性オプションで使用できる OEMS JMS のアプリケーション・コードです。このキューにより、ネットワークの障害、サーバーの再起動、JMS サーバーのフェイルオーバーなどの状況で発生することのある断続的な接続に対応できます。

```
while (!shutdown) {
       Context ctx = new InitialContext();

       // get the queue connection factory
       QueueConnectionFactory qcf =
                      (QueueConnectionFactory) ctx.lookup(QCF_NAME);

       // get the queue
       Queue q = (Queue) ctx.lookup(Q_NAME);

       ctx.close();

       QueueConnection qc = null;
       try {
           // create a queue connection, session, sender and receiver
           qc = qcf.createQueueConnection();
           QueueSession qs = qc.createQueueSession(true, 0);
           QueueSender snd = qs.createSender(q);
           QueueReceiver rcv = qs.createReceiver(q);

           // start the queue connection
           qc.start();
```

```
            // receive requests using the queue receiver and send
            // replies using the queue sender
            while (!done) {
                Message request = rcv.receive();
                Message reply = qs.createMessage();

                // put code here to process request and construct reply

                snd.send(reply);
                qs.commit();
            }
        } catch (JMSException ex) {
            if (transientServerFailure) {
                // retry
            } else {
                shutdown = true;
            }
        } finally {
           // close the queue connection
           if (qc != null) qc.close();
        }
    }
```

接続リカバリのための J2CA 構成

接続プーリングでは、アプリケーションによって接続がクローズされる場合、実際には物理接続はコネクタによってクローズされず、接続プールに戻されることに注意してください。接続に失敗すると、無効な接続がプールに戻される可能性があります。続けて新規接続を作成すると、無効な接続が取得されることがあります。その場合、前述のコードの有効性は失われます。フェイルオーバーの信頼性を保証するには、接続を作成するコネクション・ファクトリの接続プーリングを無効にする必要があります。これを行うには、`oc4j-ra.xml` ファイルのコネクション・ファクトリの `<connector-factory>` 要素を次のように変更します。

```
<connection-pooling use="none">
</connection-pooling>
```

クラスタリングのベスト・プラクティス

- JMS クライアント・サイドの状態を最小限に抑えます。
  - 処理をトランザクション済セッションで実行します。
  - 完全なリカバリ可能性を得るために、中間的なプログラムの状態を JMS キューまたはトピックに保存するか、チェックポイントを設定します。
  - J2EE アプリケーションの状態が JVM 境界にまたがってシリアライズ可能またはリカバリ可能であるかどうかに依存しません。JMS オブジェクトには常に一時的なメンバー変数を使用し、JMS の状態を適切に保存してリカバリする受動 / 能動およびシリアライズ / デシリアライズ関数を記述します。
- トピックへの非永続サブスクリプションを使用しません。
  - トピックへの非永続サブスクリプションでは、アクティブなサブスクライバごとにメッセージが複製されます。クラスタリングやロード・バランシングにより、複数のアプリケーション・インスタンスが作成されます。アプリケーションで非永続サブスクライバが作成されると、各メッセージの複製がトピックにパブリッシュされます。これは非効率的であるか、セマンティック的に無効です。
  - トピックには永続サブスクリプションのみを使用します。できるだけキューを使用してください。

- 永続サブスクリプションがアクティブな期間を延長しません。
  - 常にアクティブであることが可能な永続サブスクリプションのインスタンスは 1 つのみです。クラスタリングやロード・バランシングにより、複数のアプリケーション・インスタンスが作成されます。アプリケーションで永続サブスクリプションが作成されると、クラスタ内のアプリケーションのインスタンスは 1 つのみ成功し、他のすべては `JMSException` が発生して失敗します。
  - 永続サブスクリプションは短時間または少数のコード範囲で作成、使用してクローズし、サブスクリプションがアクティブになっている期間を最短に抑えます。
  - クラスタリング（クラスタ内で実行されているアプリケーションの他のインスタンスが現在同じコード・ブロック内にあること）による永続サブスクリプションの作成失敗に対応するアプリケーション・コードを記述し、適切なバックオフ方法をプログラミングします。永続サブスクリプションの作成失敗は、常に致命的エラーとして処理しないでください。

JMS ルーター

複雑な異機種間エンタープライズ・コンピューティング環境では、様々なメッセージング・システムと統合して相互運用するメッセージング・インフラストラクチャが必要です。JMS ルーターにより、次の宛先間をブリッジできる信頼性の高い非同期 JMS メッセージ・ルーティング・サービスが提供されます。

- 同一の JMS プロバイダの宛先
- 異なる JMS プロバイダの宛先

JMS ルーターにより、次の動作が保証されます。

- 永続的な JMS メッセージの場合、必ず 1 回のメッセージ配信が保証されます。
- 非永続的な JMS メッセージの場合、最大 1 回のメッセージ配信が保証されます。
- ソース宛先から JMS ルーターが受信した順序でのメッセージ配信が保証されます。

ユーザーは、ルーター・ジョブの作成時に、メッセージを選択的にルーティングするメッセージ・セレクタを指定できます。セレクタの条件を満たすメッセージのみが、JMS ルーターによってソース宛先からターゲット宛先にルーティングされます。メッセージ・セレクタは、有効な JMS メッセージ・セレクタである必要があります。JMS キューおよびトピック用のメッセージ・セレクタの構文とセマンティックは、JMS 1.1 仕様を参照してください。

JMS ルーターでは、JMS ルーター・ジョブのソースまたはターゲットが JMS トピックである場合、関連する JMS プロバイダが JMS 1.1 をサポートしている必要があります。

JMS 1.1 仕様は、`http://java.sun.com/products/jms/docs.html` で入手できます。

JMS の宛先をブリッジする JMS ルーターには、次のメリットがあります。

- あるメッセージ・システムでメッセージを送信するアプリケーションと別のメッセージ・システムでメッセージを受信するアプリケーションを分離できます。2 つのアプリケーションでは、メッセージが異なるメッセージ・システムを経由していることを考慮する必要はありません。
- 複数のメッセージ・システムを介してメッセージを送受信するアプリケーションでは、1 つのメッセージ・システムに接続するだけで済み、JMS ルーターを使用して複数のメッセージ・システムにメッセージを配信できます。これにより、複数のメッセージ・システムへの接続、2 フェーズ・コミットによるグローバル・トランザクションの管理、メッセージの変換処理などを行う必要がなくなるため、アプリケーション・コードが簡略化されます。
- JMS ルーターの自動リカバリおよびメッセージ配信保証機能により、すべての関連メッセージ・システムが同時に稼働する必要がなくなるため、アプリケーションおよびアプリケーション統合の信頼性が向上します。

JMS プロバイダ

JMS ルーターでは、JMS コネクタを使用して次の JMS プロバイダにアクセスします。

- OEMS JMS のメモリー内、ファイル・ベースおよびデータベースの永続性オプション
- IBM WebSphere MQ V6.0 JMS および Fix Pack 8（CSD08）を適用した MQ V5.3 JMS
- TIBCO Enterprise Message Server バージョン 4.3.0
- SonicMQ 6.0 JMS

JMS ルーターでは、OC4J 付属パッケージのスタンドアロン JMS コネクタ・インスタンスである OracleASjms を使用して、JMS ルーターと同じコンテナ内で稼働する OEMS JMS のメモリー内およびファイル・ベースのオプションにアクセスします。したがって、OEMS JMS のメモリー内またはファイル・ベースの宛先を対象にメッセージをルーティングする場合、追加のアダプタをデプロイする必要はありません。

OracleAS JMS 以外の JMS プロバイダを対象にメッセージをルーティングする場合、その JMS プロバイダ用のスタンドアロン JMS コネクタをデプロイする必要があります。

JMS ルーターでは、JMS プロバイダの認証と認可に、J2CA アダプタによる宣言的なコンテナ管理のサインオン・メカニズムを使用します。JMS ルーターは、ロール `jmsRouter` のかわりに実行されます。したがって、JMS ルーターに使用されるすべてのリソース・アダプタには、`<default-mapping>` 要素、または `oc4j-ra.xml` の `<initiating-user>` に `jmsRouter` を保持する `<principal-mapping-entry>` 要素に基づく有効なプリンシパル・マッピングが含まれる必要があります。

OEMS JMS で JMS ルーターを使用する方法の詳細は、`http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html` のドキュメントを参照してください。

構成

この項では、次の JMS ルーター・オブジェクトに関する構成について説明します。

- ルーター・ジョブ
- グローバル・ルーター・パラメータ
- サブスクリプション
- ログ・キューおよび例外キュー

JMS ルーターの主要な管理インタフェースは、JMS ルーター MBean です。この MBean により、構成の管理、個々の JMS ルーター・ジョブの起動と停止、およびジョブのステータスの監視を行うことができます。

JMS ルーターは、`jms.xml` ファイルを通じて静的に構成することも可能です。

ルーター・ジョブ

JMS ルーター・ジョブは、ソース宛先からターゲット宛先にメッセージを移動するメッセージ・ルーティング・タスクとして定義されます。ルーター・ジョブを処理する場合、JMS ルーターは、ソース宛先からメッセージをデキューし、ターゲット宛先にそのメッセージをエンキューします。

ソース宛先とターゲット宛先は、JMS キューまたはトピックのいずれかです。ソースが JMS キューである場合、JMS ルーターは、すべてのメッセージをキューから移動します。ソースがトピックである場合、JMS ルーターは、JMS 永続サブスクライバが作成されてからそのトピックにパブリッシュされたすべてのメッセージを移動します。

注意： JMS ルーター・ジョブでは、ソースがキューであるか永続サブスクライバであるかにかかわらず、そのソースを共有できません。

JMS ルーター・ジョブでは、4-87 ページの表 4-16「JMS ルーター設定」に記載されているオブジェクトを使用します。

JMS ルーターでは、JMS コネクタを使用して、JMS の宛先とコネクション・ファクトリにアクセスします。JMS ルーターでは、すべての JMS ルーター・ジョブのソース宛先およびターゲット宛先が、アクセス先の JMS プロバイダに存在し、適切に構成された JMS コネクタ内で設定されている必要があります。キュー、トピックおよびコネクション・ファクトリを構成する方法の詳細は、4-17 ページの「宛先オブジェクトおよびコネクション・ファクトリの構成」を参照してください。リソース・アダプタを構成する方法の詳細は、4-51 ページの「JMS コネクタ」、および『Oracle Containers for J2EE リソース・アダプタ管理者ガイド』を参照してください。

グローバル・ルーター・パラメータ

同時ジョブの数は、JMS ルーター属性 `maxLocalConcurrency` で構成できます。この属性により、多くのアクティブなジョブを処理する JMS ルーターが OC4J J2EE コンテナを占有しないよう調整できます。また、この属性では、JMS ルーターが使用する JMS コネクタで作成される JMS セッションの上限数も設定できます。

サブスクリプション

ソースが JMS トピックである場合、トピックの永続サブスクライバ名を指定できます。実際の永続サブスクライバは、関連するルーター・ジョブの作成前に作成できます。永続サブスクライバ名を指定しない場合、JMS ルーターにより、ジョブ名に基づいたサブスクライバ名が作成されます。JMS ルーターがジョブの処理を開始したときに、サブスクライバ名に関連付けられた永続サブスクライバが存在しない場合、ルーターにより永続サブスクライバが作成されます。JMS ルーターは、永続サブスクライバが作成されてからそのトピックにパブリッシュされたすべてのメッセージを移動します。

デフォルトでは、JMS ルーターは、JMS ルーター・ジョブが削除された段階で永続サブスクライバも削除するよう試みます。JMS ルーターが永続サブスクライバの削除に失敗した場合は、不要なメッセージが蓄積されないように、ユーザーがメッセージ・システムの管理インタフェースを使用してその永続サブスクライバを削除する必要があります。オプションで、JMS ルーター・ジョブの削除時に永続サブスクライバを削除しないよう JMS ルーターを設定することも可能です。

ログ・キューおよび例外キュー

この項では、ログ・キューと例外キューについて説明します。

ログ・キュー

JMS ルーターでは、内部ロギングにキューを使用します。各ルーター・ジョブには、ソース・ログ・キューとターゲット・ログ・キューが必要です。これらのキューは、ルーター・ジョブを処理する際のサービス品質を確保するために使用されます。OEMS JMS のメモリー内およびファイル・ベースのオプションでは、ログ・キューは JMS ルーターによって作成されます。OEMS JMS のデータベースのオプションやサポートされる他の Oracle 以外の JMS プロバイダでは、JMS キューをそのシステムで作成し、キューの名前をルーター・ジョブの作成時に指定する必要があります。ログ・キューは、複数のジョブで共有可能です。

JMS ルーターのログ・キューは、アクセス先のメッセージ・システムに存在し、適切なリソース・アダプタ内で構成されている必要があります。

例外キュー

ターゲット宛先に対するメッセージの送信に失敗した場合、JMS ルーターは、メッセージの順序を維持するために関連するルーター・ジョブの処理をブロックします。

問題のあるメッセージを含むジョブの処理を JMS ルーターで継続するには、例外キューを使用してジョブを構成します。JMS ルーターでは、ソース宛先の残りのメッセージを処理するために、問題のあるメッセージを例外キューに移動できます。

JMS ルーター・ジョブごとに 1 つの例外キューを割り当てます。この例外キューは、ソース宛先と同じメッセージ・システム内にある JMS キューで、ソース宛先と同じコネクション・ファクトリを使用してアクセスできる必要があります。例外キューの物理的な JMS 宛先は、JMS ルーター・ジョブでの使用前に存在している必要があります。OEMS JMS のメモリー内およびファイル・ベースのオプションでは、`Oc4jJmsExceptionQueue` という例外キューがデフォルトで定義されています。

JMS ルーターで問題のあるメッセージを例外キューに移動するためには、関連するルーター・ジョブの `useExceptionQueue` フラグを `true` に設定する必要があります。

JMS ルーターの例外キューをオプションで構成する場合、その例外キューは、アクセス先のメッセージ・システムに存在し、適切なリソース・アダプタ内で構成されている必要があります。

JMS ルーターとそのオブジェクトの構成

JMS ルーターの主要な管理インタフェースは、JMS ルーター MBean です。この MBean を使用して、構成の管理、個々の JMS ルーター・ジョブの起動と停止、およびジョブのステータスの監視を行います。

この項では、JMS ルーター MBean で使用できる操作および設定について説明します。

JMS ルーター MBean へのパス :

「OC4J: ホーム」→「管理」タブ→「サービス」→「JMS プロバイダ」→アイコンをクリック→適切なタブを選択→「関連リンク : OracleAS JMS Router」

注意 : JMS ルーターは、OC4J インスタンスごとに 1 つの JMS ルーター MBean をエクスポートします（`jmsrouter:j2eeType=OracleASJMSRouter`）。

次の表に、JMS ルーター MBean で使用可能な操作とその説明を示します。すべての操作は JMS ルーターを動的に更新し、その変更は `jms.xml` に永続化されます。

表 4-15 JMS ルーター MBean の操作

操作	説明
`addRouterJob`	JMS ルーター・ジョブを構成します。
`alterRouterJob`	JMS ルーター・ジョブの属性を変更します。alterRouterJob のすべてのパラメータのデフォルト値は、パラメータを変更しないままの状態です。
`configureRouter`	JMS ルーターの特定のグローバル・パラメータ（`maxLocalConcurrency` など）を構成します。
`pauseRouterJob`	JMS ルーター・ジョブの実行を一時停止します。
`removeRouterJob`	JMS ルーター・ジョブを削除します。
`resetRouterJob`	JMS ルーター・ジョブの失敗数を 0（ゼロ）にリセットします。
`resumeRouterJob`	JMS ルーター・ジョブの実行を再開します。

表 4-16「JMS ルーター設定」に、JMS ルーター MBean でのルーター設定およびルーター・ジョブ設定とその説明を示します。表の最初の列は、JMS ルーター MBean での設定の名前です。JMS ルーター MBean で実行した設定は、jms.xml ファイルに永続化されます。表の 2 番目の列は、jms.xml ファイルでの対応する要素の名前です。

表 4-16 JMS ルーター設定

MBean 設定	XML エンティティ	説明
jobName addRouterJob 操作により編集可能。	<job-name> ルーター・ジョブ要素	このルーター・ジョブの名前。構成対象の OC4J インスタンスに対して一意である必要があります。
messageSource addRouterJob 操作により編集可能。	<message-source> ルーター・ジョブ要素	メッセージのソースとして使用する宛先（トピックまたはキュー）の JNDI ロケーション。 自動宛先ラッピングを使用する場合、名前には次の形式を使用できます。 <JNDIsubcontext>/providerName ここで、<JNDIsubcontext> は、oc4j-connectors.xml で指定された自動宛先ラッピングの JNDI サブコンテキストです。 たとえば、自動宛先ラッピングを JMS コネクタと組み合せて使用する場合、名前は次の形式になります。 OracleASjms/Queues/jms/queue_name または OracleASjms/Topics/jms/topic_name
sourceConnectionFactory addRouterJob 操作により編集可能。	<source-connection-factory> ルーター・ジョブ要素	メッセージ・ソースへのアクセスに使用するコネクション・ファクトリの JNDI ロケーション。 たとえば、JMS コネクタを使用する場合、名前は次のようになります。 OracleASjms/MyCF メッセージ・ソースが JMS トピックの場合、この名前には、JMS キューおよびトピックの両方をサポートするコネクション・ファクトリを指定する必要があります。
messageTarget addRouterJob 操作により編集可能。	<message-target> ルーター・ジョブ要素	メッセージ伝播のターゲットとして使用する宛先（トピックまたはキュー）の JNDI ロケーション。 自動宛先ラッピングを使用する場合、名前には次の形式を使用できます。 <JNDIsubcontext>/providerName ここで、<JNDIsubcontext> は、oc4j-connectors.xml で指定された自動宛先ラッピングの JNDI サブコンテキストです。 たとえば、自動宛先ラッピングを JMS コネクタと組み合せて使用する場合、名前は次の形式になります。 OracleASjms/Queues/jms/queue_name または OracleASjms/Topics/jms/topic_name

表 4-16 JMS ルーター設定（続き）

MBean 設定	XML エンティティ	説明
targetConnection Factory	<target-connection-factory>	messageTarget へのアクセスに使用するコネクション・ファクトリの JNDI ロケーション。
addRouterJob 操作により編集可能。	ルーター・ジョブ要素	たとえば、JMS コネクタを使用する場合、名前は次のようになります。 OracleASjms/MyCF メッセージ・ソースが JMS トピックの場合、この名前には、JMS キューおよびトピックの両方をサポートするコネクション・ファクトリを指定する必要があります。
sourceLogQueue addRouterJob 操作により編集可能。	<source-log-queue> ルーター・ジョブ要素	JMS ルーターがソース・メッセージ・システムの内部ロギングに使用するキューの JNDI ロケーション。ログ・キューは、次の名前で識別されるコネクション・ファクトリを通じてアクセスできる必要があります。 sourceConnectionFactory OEMS JMS のメモリー内またはファイル・ベースの宛先を使用している場合、このパラメータはオプションです。指定しない場合、JMS ルーターでは次のキューが使用されます。 OracleASRouter_LOGQ このキューが存在しない場合、JMS ルーターにより作成されます。
targetLogQueue addRouterJob 操作により編集可能。	<target-log-queue> ルーター・ジョブ要素	JMS ルーターがターゲット・メッセージ・システムの内部ロギングに使用するキューの JNDI ロケーション。ログ・キューは、次の名前で識別されるコネクション・ファクトリを通じてアクセスできる必要があります。 targetConnectionFactory OEMS JMS のメモリー内またはファイル・ベースの宛先を使用している場合、このパラメータはオプションです。指定しない場合、JMS ルーターでは次のキューが使用されます。 OracleASRouter_LOGQ このキューが存在しない場合、JMS ルーターにより作成されます。
messageSelector addRouterJob 操作により編集可能。	<message-selector> ルーター・ジョブ要素	オプション。messageSource からメッセージを選択的に受信するためのメッセージ・セレクタ。デフォルトは none です。
subscriberName addRouterJob 操作により編集可能。	<subscriber-name> ルーター・ジョブ要素	オプション。messageSource がトピックの場合、使用する永続サブスクライバの名前。指定した永続サブスクライバが存在せず、messageSource がトピックの場合、JMS ルーターにより作成されます。 デフォルト値は次のとおりです。 OracleASRouter_*jobName* ここで、*jobName* は、ルーター・ジョブの名前です。

表 4-16 JMS ルーター設定（続き）

MBean 設定	XML エンティティ	説明
exceptionQueue addRouterJob および alterRouterJob 操作により編集可能。	<exception-queue> ルーター・ジョブ要素	配信できないメッセージを配置するキューの JNDI ロケーション。例外キューは、sourceConnectionFactory からアクセスできる必要があります。 このパラメータは、useExceptionQueue が true の場合にのみ指定する必要があります。useExceptionQueue が true のときに、このパラメータがオプションになるのは、メッセージ・プロバイダが OEMS JMS のメモリー内またはファイル・ベースの永続性オプションの場合です。指定しない場合、JMS ルーターでは、OEMS JMS の例外キューである Oc4jJmsExceptionQueue が使用されます。このキューはすでに存在しているため、個別に作成する必要はありません。 alterRouterJob を使用する場合、デフォルトは null です。
maxRetries addRouterJob および alterRouterJob 操作により編集可能。	max-retries ルーター・ジョブ属性	オプション。JMS ルーターでこのジョブの実行を一時停止する前にメッセージの配信を試行する回数。 値は、整数値文字列である必要があります。文字列が整数を示していない場合、無視されてデフォルトが使用されます。 addRouterJob を使用する場合、デフォルトは 16 です。alterRouterJob を使用する場合、デフォルトは null です。
pollingInterval addRouterJob および alterRouterJob 操作により編集可能。	polling-interval ルーター・ジョブ属性	オプション。メッセージがメッセージ・ソースに存在しない場合に、メッセージ・ソースを再度チェックする前に待機する最低限の秒数。 値は、整数を示す文字列である必要があります。文字列が整数を示していない場合、無視されてデフォルトが使用されます。 addRouterJob を使用する場合、デフォルトは 5 です。alterRouterJob を使用する場合、デフォルトは null です。
useExceptionQueue addRouterJob および alterRouterJob 操作により編集可能。	use-exception-queue ルーター・ジョブ属性	オプション。値が true に設定され、例外キューが使用できる場合、配信できないメッセージは例外キューに配置されます。それ以外の場合、例外キューは使用されません。 addRouterJob を使用する場合、デフォルトは false です。alterRouterJob を使用する場合、デフォルトは null です。
pauseJob addRouterJob 操作により編集可能。	pause-job ルーター・ジョブ属性	オプション。true の場合、ジョブは、非アクティブ化モードで追加されます。 ジョブを開始するには、resumeJob を起動します。 true ではない場合、ジョブは、アクティブ化モードで作成されます。 デフォルトは false です。
batchSize addRouterJob および alterRouterJob 操作により編集可能。	batch-size ルーター・ジョブ属性	オプション。1 回のトランザクションでデキューおよびエンキューするメッセージの数。 addRouterJob を使用する場合、デフォルトは 30 です。alterRouterJob を使用する場合、デフォルトは null です。

表 4-16 JMS ルーター設定（続き）

MBean 設定	XML エンティティ	説明
removeSubscriber removeRouterJob 操作により編集可能。	remove-subscriber ルーター・ジョブ属性	true で、かつルーター・ジョブで永続サブスクライバを使用している場合、JMS ルーターによりその永続サブスクライバが削除されます。 false の場合、永続サブスクライバは削除されません。JMS ルーターによる永続サブスクライバの削除に失敗した場合、ユーザーが手動で削除する必要があります。 デフォルトは true です。
maxLocalConcurrency configureRouter 操作により編集可能。	maxlocalconcurrency グローバル JMS ルーター属性	オプション。可能にするデキュー操作の最大同時実行数。この引数では、JMS ルーターがメッセージをデキューするために一度に使用できるスレッドの数を制限します。このパラメータは、JMS ルーターによって使用されるコンテナ・リソースの量を制限します。 デフォルト値は -1 で、同時に処理できるルーター・ジョブの数に制限がないことを示します。

jms.xml での JMS ルーターの構成

J2EE_HOME/config/jms.xml ファイルは、JMS ルーター・ジョブおよびグローバル構成を永続化するのに使用されます。

JMS ルーターは、jms.xml ファイルの <jms-router> 要素で構成します。<jms-router> 要素には、0 個以上の <router-job> 要素が含まれます。JMS ルーター・ジョブは、<router-job> 要素で定義します。

jms.xml ファイルの JMS ルーター要素とその説明は、表 4-16「JMS ルーター設定」を参照してください。

最小構成の例 : jms.xml での JMS ルーター構成

この例では、単一の JMS ルーター・ジョブの最小構成を示します。このジョブは、OEMS JMS のメモリー内のキューをソースおよびターゲットとして使用し、グローバルにデプロイされた JMS コネクタ・インスタンスの OracleASjms を JMS オブジェクトとして利用します。

```
<?xml version="1.0"?>
<jms xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchema
Location="http://www.oracle.com/technology/oracleas/schema/jms-server-10_1.xsd"
schema-major-version="10" schema-minor-version="1">
  <!-- OracleAS JMS server configuration -- omitted for brevity  -->
  <jms-server>
     ...
  </jms-server>

  <!-- JMS Router configuration -->
  <jms-router max-local-concurrency="-1" >

     <!-- Minimal configuration for a JMS Router job.-->
     <router-job job-name="routerjob1">

        <!-- The name of a JMS Router destination -->
        <message-source>OracleASjms/Topics/jms/mySource</message-source>

        <!-- Connection factory for the message source. -->
        <source-connection-factory>OracleASjms/MyCF</source-connection-factory>

        <!-- The name of a JMS Router destination -->
        <message-target>OracleASjms/Queues/jms/myTarget</message-target>

        <!--Connection factory for the message target. -->
        <target-connection-factory>OracleASjms/MyCF</target-connection-factory>
```

```
    </router-job>
  </jms-router>

</jms>
```

構文の例 : jms.xml での JMS ルーター構成

この例では、使用可能なすべての属性値を設定するルーター・ジョブを定義して、jms.xml の JMS ルーターに関する構文を示します。この例では、単一の JMS ルーター・ジョブの構成を定義しますが、このジョブは、OEMS JMS のメモリー内のキューをソースとして、OEMS JMS のデータベースのキューをターゲットとして使用します。また、JMS コネクタ・インスタンスの OracleASjms をソース JMS オブジェクトとして、JMS コネクタ・インスタンスの ojmsaq をターゲット・オブジェクトとして利用します。

```
<?xml version="1.0"?>
<jms xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchema
Location="http://www.oracle.com/technology/oracleas/schema/jms-server-10_1.xsd"
schema-major-version="10" schema-minor-version="1">
  <!-- OracleAS JMS server configuration -- omitted for brevity  -->
  <jms-server>
     ...
  </jms-server>

  <!-- JMS Router configuration -->
  <jms-router max-local-concurrency="-1" >

      <router-job
          job-name="routerjob1"
          max-retries="16"
          polling-interval="5"
          pause-job="false"
          use-exception-queue="true"
          batch-size="30"
      >
          <!-- The name of an JMS Router source destination -->
          <message-source>OracleASjms/Topics/jms/mySource</message-source>

          <!-- Connection factory for the message source. -->
          <source-connection-factory>OracleASjms/MyCF</source-connection-factory>

          <!-- A message selector used at the message source -->
          <message-selector>color='blue'</message-selector>

          <!--This is the default subscriber name for this job as written to this file
by the JMS Router -->
          <subscriber-name>OracleASRouter_routerjob1</subscriber-name>

          <!--There is no need to specify the log queue for OracleAS JMS, but the
default value will be written back to this file by the JMS Router -->
          <source-log-queue>OracleASjms/Queues/OracleASRouter_LOGQ</source-log-queue>

          <!-- An exception queue -->
          <exception-queue>OracleASjms/Queues/jms/myExQ</exception-queue>

          <!-- The name of an JMS Router target destination -->
          <message-target>ojmsaq/Queues/Queues/MyDBTarget</message-target>

          <!-Connection factory for the message target. -->
          <target-connection-factory>ojmsaq/CF</target-connection-factory>

          <!-- A log queue must be specified for all providers but OracleAS JMS. This
queue must already exist  ->
          <target-log-queue>ojmsaq/Queues/Queues/MyJMSRouterLog</target-log-queue>
```

```
      </router-job>
  </jms-router>

</jms>
```

ルーターの管理

この項では、JMS ルーターを管理するための JMS ルーター MBean の操作について説明します。

JMS ルーター MBean では、次の操作を実行できます。

- 各 JMS ルーター・ジョブの開始と停止
- ルーター・ジョブ・ステータスの監視
- JMS ルーターの監視と問合せ

JMS ルーター MBean の操作とその説明は、4-86 ページの表 4-15「JMS ルーター MBean の操作」を参照してください。

JMS ルーター MBean へのパス :
「OC4J: ホーム」→「管理」タブ→「サービス」→「JMS プロバイダ」→アイコンをクリック→適切なタブを選択→「関連リンク : OracleAS JMS Router」

ルーター・ロギング

JMS ルーターは、すべての重要なイベントおよびエラー・メッセージを OC4J の標準ログ・ファイルに記録します。ログ出力名は、`oracle.j2ee.jms.router` です。

JMS ルーターのステータス情報

JMS ルーターの実行時ステータス情報にアクセスするには、JMS ルーター MBean の `RouterJobsStatus` および `RouterGlobalStatus` 属性を使用します。

表 4-17 JMS ルーターおよびルーター・ジョブのステータス

統計	説明
`NumberJobs`	構成されているジョブの数
`RouterState`	JMS ルーターの状態を示す文字列
`Retries`	JMS ルーターがこのジョブでメッセージ配信に失敗した回数
`ExceptionQMessages`	このジョブにより例外キューに移動されたメッセージの数
`LastErrorTime`	このジョブがエラー状態にある場合、最後のエラーが発生した時刻
`TargetQMessages`	このジョブによりターゲット・キューに伝播されたメッセージの数
`JobState`	このジョブの状態を示す文字列

エラー処理

JMS ルーター・ジョブの処理中には、ソースまたはターゲット宛先への接続障害やメッセージ操作の失敗などの様々な障害が発生する可能性があります。JMS ルーターは、障害の性質やルーター・ジョブの構成に基づいて、障害を次のように処理します。

JMS ルーターが、メッセージの内容を原因としてターゲット宛先にメッセージをエンキューできない場合で、かつ

- 例外キューが存在しないか、関連するルーター・ジョブの `useExceptionQueue` フラグが `false` に設定されている場合、JMS ルーターは、ルーター・ジョブの処理を停止します。
- ジョブ用の例外キューが存在し、関連するルーター・ジョブの `useExceptionQueue` フラグが `true` に設定されている場合、JMS ルーターは、そのメッセージを例外キューに移動して後続のメッセージの処理を続けます。

メッセージがソース宛先から例外キューに移動される場合、エラー情報の維持を目的として、JMS ルーターにより元のメッセージに特定のメッセージ・プロパティが追加されます。表 4-18「例外キューのメッセージに追加されるプロパティ」に、これらのメッセージとその説明を示します。

表 4-18 例外キューのメッセージに追加されるプロパティ

JMS プロパティ	説明
`oraMsgRouter_origMsgid`	ソース・メッセージの ID
`oraMsgRouter_jobName`	ルーター・ジョブの名前
`oraMsgRouter_srcCF`	デキューまたはエンキューに使用されたコネクション・ファクトリの名前
`oraMsgRouter_srcQName`	メッセージが取得されたソース・キューの名前
`oraMsgRouter_moveReason`	メッセージが例外キューに移動された理由
`oraMsgRouter_moveTime`	メッセージが例外キューに移動された時刻

障害の原因がメッセージの内容ではない場合、JMS ルーターは、次の式に基づいて徐々に間隔を空けながら失敗した操作を自動的に再試行します。

```
(2^n) * (pollingInterval),
```

この間隔は、最大 15 分です。この再試行操作は、成功するまで、または `maxRetries` 属性（4-87 ページの表 4-16「JMS ルーター設定」を参照）で指定された構成可能な最大再試行回数に到達してルーター・ジョブの処理が停止されるまで継続されます。

`maxRetries` 設定で指定された再試行回数に到達してルーター・ジョブの処理が停止された場合に、ジョブの処理を再開するには、`resetJob` をコールして失敗回数を `0` にリセットします。

各ジョブの失敗回数は、メモリーに格納されます。そのため、JMS ルーターが再起動すると、すべてのジョブの失敗回数は `0` にリセットされます。

ジョブの一時停止と再開

JMS ルーター・ジョブには、4-87 ページの表 4-16「JMS ルーター設定」に記載した `pauseJob` 設定で指定できる `activated` モードと `deactivated` モードがあります。

`deactivated` モードのジョブは、JMS ルーターで処理されません。

ジョブを `deactivated` モードに移行するには、JMS ルーター MBean の `pauseJob` 操作を使用します。

ジョブを `activated` モードに移行するには、JMS ルーター MBean の `resumeJob` 操作を使用します。

デフォルトでは、ジョブは `activated` モードで作成されます。

OC4J クラスタ環境での実行

OC4J クラスタ環境では、JMS ルーター・インスタンスは、OC4J インスタンスごとに実行できます。これらの JMS ルーター・インスタンスは、個別に構成され、相互に独立して実行されます。ある OC4J インスタンス上に構成された JMS ルーター・ジョブは、その OC4J インスタンスで稼働する JMS ルーター・インスタンスによってのみ処理されます。

- JMS ルーター・ジョブは、特定の OC4J インスタンスの JMS ルーター MBean または jms.xml ファイルを使用して作成または管理する必要があります。
- OC4J インスタンスに定義された JMS ルーター・ジョブにより参照されるすべての JMS コネクション・ファクトリおよび JMS 宛先は、その OC4J インスタンスからアクセスできる必要があります。
- OC4J インスタンスが停止すると、その OC4J インスタンス上で稼働する JMS ルーター・インスタンスに定義されたジョブは、処理されません。

原則として、異なる JMS ルーター・ジョブでは、一般的なソース（キューまたは永続サブスクライバ）を共有することはできません。違反した場合、JMS ルーター・ジョブにリカバリ不可能な障害が発生する可能性があります。OC4J クラスタ環境では、クラスタ内のすべての OC4J インスタンスにこの原則が適用されます。

- OEMS JMS のメモリー内またはファイル・ベースの分散宛先ではない JMS キューは、OC4J クラスタ全体でただ 1 つの JMS ルーター・ジョブのソースとなることができます。
- OEMS JMS のメモリー内またはファイル・ベースの分散宛先ではない JMS トピックは、OC4J クラスタ全体で複数の JMS ルーター・ジョブのソースとなることができます。ただし、関連する永続サブスクライバの名前は、クラスタ内で一意である必要があります。具体的には、JMS ルーター・ジョブのサブスクライバ名を指定するときに、それぞれ異なる名前を付けます。サブスクライバ名を指定しない場合、JMS ルーター・ジョブの名前に基づいて永続サブスクライバ名が生成されるため、JMS ルーター・ジョブの名前は、ソースとして同じトピックを共有するジョブに関してクラスタ全体で一意である必要があります。
- クラスタ内の OC4J インスタンスに存在する OEMS JMS のメモリー内またはファイル・ベースの分散宛先の各コピーは、JMS ルーターにより個別の JMS 宛先として扱われます。OEMS JMS のメモリー内またはファイル・ベースの分散宛先からメッセージをルーティングするには、その宛先をソースとして使用する JMS ルーター・ジョブを各 OC4J インスタンスに定義する必要があります。

リモート宛先を使用したルーティング

JMS ルーターでは、OEMS JMS のリモート・コネクション・ファクトリを使用して、ある OC4J インスタンスから OEMS JMS を介して別の OC4J インスタンスにメッセージをルーティングできます。詳細は、4-76 ページの「カスタム・トポロジ」を参照してください。

リモート OEMS JMS インスタンスのコネクション・ファクトリを使用して JMS ルーター・ジョブを定義する場合、そのコネクション・ファクトリを使用してアクセスされる任意の JMS 宛先は、リモートとローカル両方の OC4J インスタンスで定義される必要があります。ソース・コネクション・ファクトリがリモートである場合、これらの JMS 宛先には、メッセージ・ソース、JMS ルーターのソース・ログ・キュー、および（オプションの）例外キューが含まれます。ターゲット・コネクション・ファクトリがリモートである場合、適切な JMS 宛先には、メッセージ・ターゲットと、JMS ルーターのターゲット・ログ・キューが含まれます。

メッセージ・ソースまたはメッセージ・ターゲットがリモートである場合、JMS ルーター・ジョブの作成時に、JMS ルーター・ログ・キューをリモート・インスタンス上に手動で作成して指定する必要があります。

OEMS JMS の HTTP トネリングを使用したリモート宛先へのルーティング

OEMS JMS では、JMS メッセージがファイアウォールを通過できるよう HTTP トネリングがサポートされています。この機能は、Web で稼働している OC4J インスタンス上に JMS HTTP トンネル・プロバイダを構成することで有効化されます。Web で稼働する OC4J インスタンスは、ファイアウォールの内側に配置されている場合があります。この機能は、OEMS JMS のメモリー内およびファイル・ベースでのみ使用可能です。リモート宛先へのルーティングの詳細は、4-94 ページの「リモート宛先を使用したルーティング」および 4-76 ページの「カスタム・トポロジ」を参照してください。

JMS HTTP トンネル・プロバイダが構成されている場合、JMS アプリケーションは `tunnel`、`host` および `port` 属性が定義されているコネクション・ファクトリをルックアップして HTTP トネリングを使用します。`tunnel` 属性は、JMS HTTP トンネル・プロバイダの URL を指定します。`host` および `port` 属性は、JMS リクエストを処理するターゲット JMS サーバーを指定します。前述のコネクション・ファクトリを使用して JMS アプリケーションにより JMS 接続が作成されると、ターゲット JMS サーバーへの通信は JMS HTTP トンネル・プロバイダ経由でルーティングされます。アプリケーションと JMS HTTP トンネル・プロバイダ間の通信には、HTTP（または SSL を使用するよう構成されている場合は HTTPS）が使用されます。JMS HTTP トンネル・プロバイダとターゲット JMS サーバー間の通信には、TCP が使用されます。

JMS HTTP トンネル・プロバイダの構成

`<http-tunnel>` 要素が OC4J インスタンスの `jms.xml` 構成ファイルに含まれている場合、OC4J インスタンスは、JMS HTTP トンネル・プロバイダとして機能します。`<http-tunnel>` 要素は、`<jms-server>` 要素のオプションのサブ要素です。各 OC4J インスタンスに定義できる JMS HTTP トンネルは最大で 1 つです。

注意： OC4J インスタンスの `jms.xml` 構成ファイルで `<http-tunnel>` 要素が追加、変更または削除された場合には、OC4J インスタンスを再起動する必要があります。

次の表に、`<http-tunnel>` ノードの構成要素を示します。

表 4-19 OEMS JMS HTTP トンネルの構成

要素および属性	説明
`authentication`	`authentication` 属性には、値 `required` または `none` を指定します。デフォルト値は、`required` です。 値 `required` は、接続のユーザー / パスワードが、ターゲット JMS サーバーにリレーされる前に JMS HTTP トンネル・プロバイダによって認証されることを意味します。認証に失敗するとリクエストは拒否されます。 値 `none` は、（その他すべての設定で許可されている場合には）ユーザーは認証されずに、JMS HTTP トンネル・プロバイダによってターゲット JMS サーバーにリレーされることを意味します。このプロパティを `none` に設定すると、ターゲット JMS サーバーで実行される認証は抑制されません。
`tcp-relay`	`tcp-relay` 要素は、ターゲット JMS サーバーが JMS HTTP トンネル・プロバイダとは異なる JVM に存在する JMS リクエストが、（その他の設定で許可されている場合には）JMS HTTP トンネル・プロバイダによってターゲット JMS サーバーにリレーされることを指定します。 `tcp-relay` 要素が省略されている場合、ターゲット JMS サーバーが JMS HTTP トンネル・プロバイダとは異なる JVM に存在する JMS リクエストは、JMS HTTP トンネル・プロバイダによって拒否されます。

表 4-19 OEMS JMS HTTP トンネルの構成（続き）

要素および属性	説明
filter	tcp-relay 要素の filter 属性は list または auto で、デフォルトで list に設定されます。 値 list は、ターゲット JMS サーバーのアドレスがターゲット・アドレスのリストに含まれる場合には、JMS HTTP トンネル・プロバイダによりターゲット JMS サーバーへの JMS リクエストのみがリレーされることを意味します。その他すべての JMS リクエストは JMS HTTP トンネル・プロバイダによって拒否されます。リストの各ターゲット JMS サーバー・アドレスは、target サブ要素を使用して指定されます。 値 auto は、JMS HTTP トンネル・プロバイダにより、（コネクション・ファクトリ定義に host/port 属性で指定されているように）ターゲット JMS サーバー・アドレスが実際の JMS サーバーに対応しているかどうかが確認されることを意味します。プロバイダにより、指定された host/port にトネリング・プロトコルを使用してリクエストが送信されます（TCP が使用されます）。レスポンスが戻ってきた場合、その host/port は JMS リクエストに対する有効なターゲットであるとみなされ、リクエストがリレーされます（その他すべての設定で許可されている場合）。 確実に自動検出できる特定の host/port の組合せはキャッシュされるため、指定された host/port では自動検出は繰り返されません。host/port のリストを手動で作成する必要がないため、自動検出は便利です。ただし、安全性に関するコーディングが不十分な、JMS 以外の host/port におけるサーバー・プロセス・リスニングが、自動検出中の調査で不適切な反応をする可能性があります。有効なレスポンスが timeout の期間中に戻されない場合、自動検出は失敗したとみなされます。
timeout	tcp-relay 要素の timeout 属性は、JMS HTTP トンネル・プロバイダがターゲット JMS サーバーからのレスポンスを待機する期間を制限します。値はミリ秒単位で入力します。デフォルト値は、5000 です。
target	tcp-relay の下位の各 target サブ要素は、JMS HTTP トンネル・プロバイダが JMS リクエストをリレーするターゲット JMS サーバーのホストおよびポートをリストします。この要素は、tcp-relay フィルタが list の場合に使用され、フィルタが auto の場合には無視されます。複数の target 要素を指定できます。各ターゲット・アドレスは次の書式である必要があります。 `<target host="host" port="port"/>`

次の例では、jms.xml ファイルに構成された JMS HTTP トンネルを示します。

```
<jms>
   <jms-server>
   ...
      <http-tunnel authentication="required">
         <tcp-relay filter="list" timeout="5000">
            <target host="host1" port="9127"/>
            <target host="host2" port="9127"/>
         </tcp-relay>
      </http-tunnel>
   ...
   </jms-server>

   <jms-router>
   ...
   </jms-router>
</jms>
```

その他の HTTP トンネルの例

JMS HTTP トンネル・プロバイダは、JMS HTTP トンネル・プロバイダの OC4J インスタンスで JMS 接続のユーザー名 / パスワードが両方とも有効（認証が `required` に設定されているため）で、ターゲット JMS サーバーが JMS トネリング・プロトコルに肯定的に対応する場合に、JMS リクエストを JMS サーバーにリレーするように構成できます。

```
<http-tunnel authentication="none"/>
```

JMS HTTP トンネル・プロバイダは、JMS HTTP トンネル・プロバイダの OC4J インスタンスで JMS 接続のユーザー名 / パスワードが両方とも有効（認証が `required` に設定されているため）で、ターゲット JMS サーバーが JMS HTTP プロトコルに肯定的に対応する場合に、リクエストを任意の JMS サーバーに転送するように構成できます。

```
<http-tunnel authentication="required">
   <tcp-relay filter="auto"/>
</http-tunnel>
```

JMS HTTP トンネル・プロバイダは、`server1.xyz.com` port `9127` または `server2.xyz.com` port `2000` の JMS サーバー宛の JMS リクエストのみをリレーするように構成できます。

```
<http-tunnel authentication="none">
   <tcp-relay filter="list">
      <target host="server1.xyz.com" port="9127">
      <target host="server2.xyz.com" port="2000">
   </tcp-relay>
</http-tunnel>
```

SSL を使用した JMS HTTP トネリング

JMS HTTP トンネル・プロバイダの機能は Web アプリケーションとして構成され、Oracle Application Server で Oracle HTTP Server を使用する場合と Oracle HTTP Server を使用しないスタンドアロンの場合のどちらでも、Secure Socket Layer（SSL）通信での OC4J のサポートを利用できます。

SSL 通信の設定方法の詳細は、『Oracle Containers for J2EE セキュリティ・ガイド』を参照してください。

コネクション・ファクトリの構成

コネクション・ファクトリ構成に構成済の OEMS JMS HTTP トンネル・プロバイダの URL に設定された `tunnel` 属性が含まれる場合、OEMS JMS アプリケーションは JMS HTTP トンネルのみを通過します。

この章の表 4-1 ですでに説明したコネクション・ファクトリ属性に加え、JMS のコネクション・ファクトリ属性も設定されています。

注意： OC4J インスタンスの `jms.xml` 構成ファイルでコネクション・ファクトリが追加、変更または削除された場合には、OC4J インスタンスを再起動する必要があります。

表 4-20 コネクション・ファクトリの属性

属性	説明
tunnel	この属性は、JMS HTTP トンネル・プロバイダ・サーブレットの URL です。URL は、jms.xml 構成ファイルに <http-tunnel> 要素が定義されている JMS サーバーを指している必要があります。次の書式で指定する必要があります。 スタンドアロンの OC4J: ■ http:///hostname:port//jms（SSL 以外の HTTP トネリングの場合） ■ https:///hostname:port//jms（HTTP over SSL トネリングの場合） *i*AS: ■ http:///hostname:port/j2ee//jms（SSL 以外の HTTP トネリングの場合） ■ https:///hostname:port/j2ee//jms（HTTP over SSL トネリングの場合） HTTPS を介して JMS HTTP トネリングを使用している場合には、ポート番号が、SSL 対応の OC4J Web サイトを指定している必要があります。
keystore	この属性には、キーストアの名前およびパスが含まれます。キーストアは、SSL を使用して JMS HTTP トンネル・プロバイダ・サーブレットを通過する場合にのみ使用されます。絶対パスをお薦めします。
keystore-password	この属性はキーストアのパスワードです。keystore 属性が指定されている場合には、この属性は必須です。
truststore	この属性には、トラストストアの名前およびパスが含まれます。トラストストアを指定しない場合、OC4J ではキーストアがトラストストアとして使用されます。トラストストアは、SSL を使用して JMS HTTP トンネル・プロバイダ・サーブレットを通過する場合にのみ使用されます。絶対パスをお薦めします。
truststore-password	この属性はトラストストアのパスワードです。truststore 属性が指定されている場合には、この属性は必須です。
Provider	この属性は、Java セキュリティ API に java.security.Provider を実装するクラスの名前です。値が指定されていない場合、デフォルトのプロバイダ oracle.security.pki.OraclePKIProvider が使用されます。Provider 属性は、SSL を使用して JMS HTTP トンネル・プロバイダ・サーブレットを通過する場合にのみ使用されます。

コネクション・ファクトリの構成例

次の例では、JNDI ロケーション jms/SomeQCF でルックアップされる OEMS JMS の QueueConnectionFactory を作成します。JMS 接続の作成に使用される場合、接続は、ホスト jmstunnelhost に存在する SSL 以外の HTTP トンネルを通過します。

```
<queue-connection-factory
   name="jms/SomeQCF"
   location="jms/SomeQCF"
   host="server1.abc.com" port="9127"
   tunnel="http://jmstunnelhost:8888/jms"/>
```

次の例では、JNDI ロケーション jms/AnXaTCF でルックアップされる OEMS JMS の XATopicConnectionFactory を作成します。JMS 接続の作成に使用される場合、keystore および keystore-password が有効であれば、接続は、ホスト server1.xyz.com に存在する SSL の HTTPS トンネルを通過します。

```
<xa-topic-connection-factory
   name="jms/AnXaTCF"
   location="jms/AnXaTCF"
   host="server1.xyz.com" port="2000"
   tunnel="https://server1.xyz.com:4443/jms"
   keystore="/location/of/keystore"
   keystore-password="keypass"/>
```

ホスト名の競合の解決

OEMS JMS のメモリー内およびファイル・ベースのオプションでは、識別子（メッセージ ID など）が一意である必要があります。識別子の一部は、JMS サーバーおよびクライアントのマシンのホスト名から構成されます。同じ JMS サーバーと通信している異なる 2 つのマシンのホスト名が同一である場合、ID は一意ではなくなり、JMS の機能が予期したとおりに機能しない可能性があります。

たとえば、イントラネット A の JMS クライアント A で、イントラネット B の JMS サーバーへのアクセスに JMS HTTP トンネル・プロバイダを使用していて、イントラネット B の JMS クライアント B もその JMS サーバーにアクセスしている場合、JMS クライアント A が稼働しているマシンのホスト名は、JMS クライアント B が稼働しているマシンのホスト名とは異なる必要があります。2 つのイントラネットへの接続に JMS HTTP トネリングが使用されている場合、これらのイントラネットの管理者は、ホスト名の競合を避けるためにそれぞれを調整する必要があります。

注意： 同じマシン（つまりホスト名が同じ）の同一または異なる JVM で 2 つ以上の JMS サーバー（または JMS クライアント、あるいはその両方）が稼働していても、JVM 全体および個々の JVM 内で一意の ID が生成されるため特に問題はありません。ホスト名の競合は、2 つの異なるマシンに同じホスト名がある場合にのみ問題になります。

JMS のロギングの構成

この項では、JMS 固有のロギングを構成するための様々なオプションを説明します。OEMS は OC4J ロギング実装を使用して、JMS に関連するメッセージを記録します。通常、メッセージは、問題のトラブルシューティングや診断に使用されます。OC4J ロギングはデフォルトの設定を使用するように構成されます。ただし、これらの設定を JMS 用に変更して、さらに詳細なログ・メッセージを取得できます。OC4J ロギング実装の詳細は、『Oracle Containers for J2EE 開発者ガイド』を参照してください。

標準の JMS

標準の JMS 機能のトレースに使用可能なログ出力は 2 つあります。それらのログ出力は次のとおりです。

- `oracle.j2ee.jms`: 基本的な JMS ロギングに使用
- `com.evermind.server.jms.JMSServer`: 主な JMS サーバー・クラスからの高度な JMS サーバー・レベルのロギングに使用

標準の JMS ログ出力は、`J2EE_HOME/config/j2ee-logging.xml` ファイルに構成されます。たとえば、高度なサーバー・レベルのロギングを構成し、ログ・レベルを `FINEST` に設定するには、次のログ出力ノードを追加します。

```
<loggers>
   ...
   <logger name='com.evermind.jms.JMSServer' level='FINEST'
      useParentHandlers='false'>
      <handler name='oc4j-handler'/>
      <handler name='console-handler'/>
   </logger>
<loggers>
```

この例では、ログ出力からのメッセージは、`oc4j-handler` および `console-handler` ログ・ハンドラでの指定どおりに出力に送信されます。

JMS プロバイダ

`com.evermind.server.jms` ログ出力は、OEMS JMS ファイル全体の永続性プロバイダ実装のログ情報を取得するために使用されます。JMS プロバイダのログ出力は、`J2EE_HOME/config/j2ee-logging.xml` ファイルに構成されます。たとえば、ログ出力を構成してログ・レベルを `FINEST` に設定するには、次のログ出力ノードを追加します。

```
<loggers>
   ...
   <logger name='com.evermind.jms' level='FINEST'
      useParentHandlers='false'>
      <handler name='oc4j-handler'/>
      <handler name='console-handler'/>
   </logger>
<loggers>
```

この例では、ログ出力からのメッセージは、`oc4j-handler` および `console-handler` ログ・ハンドラでの指定どおりに出力に送信されます。

JMS コネクタ

JMS コネクタを使用する際には、すべてのコネクタのログ出力を設定するオプションと、各コネクタのログ出力を設定するオプションがあります。

すべてのコネクタ

oracle.j2ee.ra.jms.generic ログ出力は、すべてのコネクタに対して使用します。ログ出力は、J2EE_HOME/config/j2ee-logging.xml ファイルに構成されます。たとえば、ログ出力を構成してログ・レベルを FINEST に設定するには、次のログ出力ノードを追加します。

```
<loggers>
   ...
   <logger name='oracle.j2ee.ra.jms.generic' level='FINEST'
      useParentHandlers='false'>
      <handler name='oc4j-handler'/>
      <handler name='console-handler'/>
   </logger>
<loggers>
```

この例では、ログ出力からのメッセージは、oc4j-handler および console-handler ログ・ハンドラでの指定どおりに出力に送信されます。

各コネクタ

ログ出力は各コネクタに構成し、そのコネクタに関する特定のメッセージを指定できます。ログ出力は、J2EE_HOME/config/oc4j-connectors.xml ファイル、または ra.xml ファイルに構成されます。次に例を示します。

```
<config-property>
   <config-property-name>loggerName</config-property-name>
   <config-property-type>java.lang.String</config-property-type>
   <config-property-value>mylog</config-property-value>
</config-property>
<config-property>
   <config-property-name>logLevel</config-property-name>
   <config-property-type>java.lang.String</config-property-type>
   <config-property-value>FINEST</config-property-value>
</config-property>
```

メッセージ・エンドポイント

ログ・レベルは、LogLevel 構成プロパティを使用して、特定のメッセージ・エンドポイント用に調整できます。本番コードでは、このプロパティは設定しないでください。デバッグ目的で一時的にのみ設定してください。

このプロパティは、<message-driven-deployment> ノード内の orion-ejb-jar.xml ファイルに追加されます。次に例を示します。

```
<enterprise-beans>
   <message-driven-deployment ... >
      ...
      <config-property>
         <config-property-name>LogLevel</config-property-name>
         <config-property-value>FINEST</config-property-value>
      </config-property>
   </message-driven-deploymnet>
</enterprise-beans>
```

注意： LogLevel プロパティは、ejb-jar.xml ファイルにも設定できます。構成プロパティの使用方法の詳細は、「MDB エンドポイントの微調整」を参照してください。設定できる有効なログ・レベルの詳細なリストは、表 4-11「MDB の構成プロパティ」の LogLevel エントリを参照してください。

5
データソース

この章では、Oracle Containers for J2EE（OC4J）のデータソースについて説明します。この章には、次の項目が含まれます。

- データソース・タイプ
  - マネージド・データソース
  - ネイティブ・データソース
- データソースの定義
  - 接続プールの定義
  - マネージド・データソースの定義
  - ネイティブ・データソースの定義
  - パスワードの間接化の使用方法
- データソースからの接続の取得
- 接続
- 文
- トランザクション
- データソース・オブジェクトの構成
- 構成例
- 高可用性および Fast Connection Failover の使用方法
- JDBC ドライバの使用方法

タスク

この章では、次の OC4J データソース・タスクについて説明します。

- 接続プールの定義
- マネージド・データソースの定義
- ネイティブ・データソースの定義
- 致命的エラー・コードの定義
- パスワードの間接化の使用方法
- 接続の確立
- マネージド・データソースでの接続プールの使用方法
- マネージド・データソースでの接続プロキシの使用方法
- データソースからの接続の取得
- データソースでの JDBC 文のキャッシュ・サイズの設定

- データソース・オブジェクトの構成
- 高可用性および Fast Connection Failover の使用方法
- JDBC ドライバの使用方法

新機能

次の OC4J データソースの機能および動作は、今回のリリースの新機能です。

- Oracle Enterprise Manager 10*g* Application Server Control コンソールで、すべてのデータソース構成を実行できます。
- エミュレート、非エミュレートおよびネイティブの各 OC4J データソース・タイプは、マネージド・データソースとネイティブ・データソースに置き換えられました。
- OC4J では、デフォルトでローカル・トランザクションを追跡します。これにより、異なるデータベース・ドライバおよびベンダー間で一貫性を維持できます。さらに、この追跡によってインターリーブを原因とするトランザクションの破損を防止できます。
- Oracle データソース全体で統一化された新しい接続キャッシュ・メカニズムにより、Real Application Clusters（RAC）フェイルオーバーの統合サポートが提供されます。詳細は、5-26 ページの「接続プール」および 5-26 ページの表 5-3「接続プールの属性」を参照してください。

廃止予定

次の項目は、今回のリリースで廃止予定です。

- `OracleConnectionCacheImpl` クラスは廃止予定です。このクラスでは、複数のスキーマはサポートされず、1 人のユーザーのみがキャッシュされます。
- `stmt-cache-size` 属性。データソースに JDBC 文のキャッシング機能を構成するために、`stmt-cache-size` 属性ではなく `num-cached-statements` 属性を使用してキャッシュ・サイズを設定します。

追加ドキュメント

追加のデータソース情報は、次のドキュメントを参照してください。

- 『Oracle Database JDBC 開発者ガイドおよびリファレンス』
- コード例付きの How-To 概要ドキュメント（`http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html` で入手可能）
- 追加の資料（`http://www.oracle.com/technology/tech/java/oc4j/index.html` で入手可能）

データソース・タイプ

データソースは、javax.sql.DataSource インタフェースを実装する Java オブジェクトです。データソースにより、ベンダーに依存しない移植可能なメソッドを使用して、データベースへの接続を作成できます。特定のデータベースを指定するために、データソース・オブジェクトのプロパティを設定します。アプリケーションでは、データソースのプロパティを変更することで様々なデータベースを使用できるため、アプリケーション・コードは変更せずに済みます。

OC4J のデータソースには、マネージド・データソースとネイティブ・データソースという 2 つのタイプがあります。

マネージド・データソース

マネージド・データソースは、OC4J によって管理されます。つまり、グローバル・トランザクション管理、接続プーリング、エラー処理などの重要なシステム・インフラストラクチャは、OC4J によって提供されます。マネージド・データソースは、JDBC ドライバまたはデータソースのラッパーとして機能する javax.sql.DataSource インタフェースの OC4J による実装です。J2EE コンポーネントでは、データソース実装がラッパーであると認識することなく、JNDI を使用してマネージド・データソースにアクセスできます。

マネージド・データソースとネイティブ・データソースの相違点は、次のとおりです。

- マネージド・データソースから取得された接続は、グローバル・トランザクションに参加できます。
- マネージド・データソースは、OC4J の接続プールおよび文キャッシュを使用します。
- マネージド・データソースから戻される接続は、OC4J の接続プロキシでラップされます。

ネイティブ・データソース

ネイティブ・データソースは、javax.sql.DataSource インタフェースを実装しており、JDBC ドライバ・ベンダー（オラクル社や DataDirect 社など）により提供されます。ネイティブ・データソースとマネージド・データソースの相違点は、次のとおりです。

- ネイティブ・データソースから取得された接続は、グローバル・トランザクションに参加できません。
- ネイティブ・データソースは、OC4J の接続プールまたは文キャッシュを使用しません。
- ネイティブ・データソースから戻される接続は、OC4J の接続プロキシでラップされません。

ネイティブ・データソースを構成する方法の詳細は、5-6 ページの「ネイティブ・データソースの定義」を参照してください。

データソースの定義

Application Server Control コンソールは、データソースを管理するための主要ツールであり、データソースと接続プールの作成、データソースと接続プールの削除、既存のデータソースと接続プールの変更などの操作を実行できます。

データソースの設定に関する有益な情報は、Application Server Control コンソールのオンライン・ヘルプを参照してください。

Application Server Control コンソールでデータソースを変更すると、データソースの設定は、即座にそのアプリケーションの `data-sources.xml` ファイルに永続化されます。

デフォルト・アプリケーションのデータソース構成ファイルは、`$J2EE_HOME/config/data-sources.xml` です。

このファイルの各 `<data-source>` タグは、JNDI にバインドされていてクライアント・コンポーネント（サーブレットや EJB など）からアクセスできる 1 つのデータソースを示しています。

この項では、`data-sources.xml` 構成ファイルのデータソース定義の例を示します。

データソースを定義する方法の詳細は、5-20 ページの「データソース・オブジェクトの構成」を参照してください。

`data-sources.xml` ファイルの例は、5-31 ページの「構成例」を参照してください。

構成に関する注意事項

- データソースの追加、編集または削除を直接 `data-sources.xml` ファイルで行った場合、その変更を実装するために OC4J を再起動する必要があります。
- データソースの追加、編集または削除を Application Server Control コンソールで行った場合、その変更を実装するために OC4J を再起動する必要はありません。Application Server Control コンソールでデータソース設定を保存するたびに、新規の `data-sources.xml` ファイルが生成され、コメントは失われます。
- `data-sources.xml` ファイルには、サンプルのデータソース・エントリおよび接続プール・エントリが含まれます。これらのエントリはサンプルとしてのみ使用され、削除できません。5-8 ページの「サンプル・データソースの削除」を参照してください。
- データソースの `jndi-name` は、アプリケーションに対して一意である必要があります。同じ `data-sources.xml` ファイルに重複する JNDI 名を含めることはできません。JNDI ロケーションが `data-sources.xml` ファイル内で重複していると、OC4J によって例外がスローされます。グローバルな `data-sources.xml` ファイルですでに指定済のアプリケーションの `data-sources.xml` ファイルに、`jndi-name` を指定することも可能です。この場合、そのアプリケーションの `data-sources.xml` に指定された JNDI ロケーションにより、アプリケーション・コンテキストの JNDI ロケーションが上書きされます。
- データソースを定義するには、`<data-source>` タグを使用します。`class` 属性には、`javax.sql.DataSource` インタフェースを実装するオブジェクトのフルパスのクラス名を設定できます。
- 一部の明確なプロパティ（`user`、`password`、`url` など）もこのタグで指定します。
- OC4J によって認識されないプロパティは、`<property>` タグで定義します。
- OC4J 固有の実装には、設定可能な追加のプロパティはありません。つまり、OC4J のすべてのプロパティは、`<data-source>` タグの値を通じて設定します。
- Oracle 固有の実装（`oracle.jdbc.pool.OracleDataSource` など）には、`<data-source>` タグの `<property>` サブタグを通じて設定可能なプロパティがあります。
- Oracle JDBC ドライバの詳細は、『Oracle Database JDBC 開発者ガイドおよびリファレンス』を参照してください。
- 通常は、Oracle 以外のベンダー固有の実装（DB2、Sybase、SQLServer など）にも、`<data-source>` タグで定義するプロパティ以外に、`<property>` タグを使用して定義できるプロパティがあります。ユーザーは、Oracle 以外のデータソース実装のプロパティを決定できます。

接続プールの定義

マネージド・データソースでは、接続を効率的に管理するため、接続プールを使用します。マネージド・データソースを作成する場合、少なくとも 1 つの接続プールと、そのコネクション・ファクトリを定義する必要があります。

OC4J には、再利用可能な物理接続のキャッシュを保持することで効率性を高める接続プール機能があります。クライアントによって接続がクローズされると、その接続は、別のクライアントで使用できるようプールに戻されます。接続プールにより、複数のクライアントで少数の物理接続を共有できるため、パフォーマンスとスケーラビリティが向上します。

注意： 接続プールという語と接続キャッシュという語は、同じ意味です。

接続プールの性質および用途の詳細は、5-9 ページの「マネージド・データソースでの接続プールの使用方法」を参照してください。

Application Server Control コンソールでのコネクション・ファクトリおよび接続プール設定へのパス

「OC4J: ホーム」→「管理」タブ→「タスク名 : サービス」→「JDBC リソース」→「タスクに移動」→目的の設定にドリルダウン。

次の例で、Application Server Control コンソールではなく `data-sources.xml` ファイルで接続プールを定義する方法を示します。

```
<connection-pool name="myConnectionPool">
   <connection-factory
      factory-class="oracle.jdbc.pool.OracleDataSource"
      user="scott"
      password="tiger"
      url="jdbc:oracle:thin:@//localhost:1521/
         oracle.regress.rdbms.dev.us.oracle.com"/>
      <property name="foo" value="bar"/>
   </connection-factory>
</connection-pool>
```

`<connection-pool>` 要素には、接続プールを一意に識別する `name` 属性が含まれます。他の属性では、接続プールで保持する接続の最大数など、接続プールのパラメータを定義します。接続プールは、`<connection-factory>` 要素で定義されたコネクション・ファクトリを使用してデータベースから物理接続を取得します。

`<connection-factory>` 要素には、JDBC ドライバがデータベースに接続するための URL と、データベースから接続を取得するためのオプションのデフォルト・ユーザーおよびパスワードが含まれます。`factory-class` 属性では、接続を取得する JDBC ドライバにより提供される実装クラスを定義します。実装クラスは、次のいずれかのインタフェースを実装する必要があります。

- `java.sql.Driver`
- `javax.sql.DataSource`
- `javax.sql.XADataSource`
- `javax.sql.ConnectionPoolDataSource`

接続プールおよびコネクション・ファクトリ設定の詳細は、5-26 ページの表 5-3「接続プールの属性」を参照してください。

マネージド・データソースの定義

1 つ以上の接続プールを定義したら、マネージド・データソースを定義できます。

マネージド・データソース設定へのパス

「OC4J: ホーム」→「管理」タブ→「タスク名 : サービス」→「JDBC リソース」→「タスクに移動」→「データソース」→「作成」→「アプリケーションの選択」→「データソース・タイプの選択」→「マネージド」

次の例で、前に定義した接続プールを使用して、data-sources.xml ファイルでマネージド・データソースを定義する方法を示します。

```
<managed-data-source
    jndi-name="jdbc/ManagedDS"
    name="Managed DataSource Name"
    connection-pool-name="myConnectionPool" />
```

name 属性では、一意のマネージド・データソースを指定します。jndi-name 属性では、このデータソースを配置する JNDI ロケーションを定義します。connection-pool-name 属性では、このマネージド・データソースが接続を取得するために通信する接続プールを指定します。この接続プール名は、前項の「接続プールの定義」の例に記載されている <connection-pool> 要素の name 属性の値に対応します。

ネイティブ・データソースの定義

ネイティブ・データソースは、接続プールに依存しません。そのため、ネイティブ・データソースの定義には、基礎となるデータベースと通信するのに必要なデータが含まれます。

ネイティブ・データソース設定へのパス

「OC4J: ホーム」→「管理」タブ→「タスク名 : サービス」→「JDBC リソース」→「タスクに移動」→「データソース」→「作成」→「アプリケーションの選択」→「データソース・タイプの選択」→「ネイティブ」

次の例で、data-sources.xml ファイルでネイティブ・データソースを定義する方法を示します。

```
<native-data-source
    name="nativeDataSource"
    jndi-name="jdbc/nativeDS"
    data-source-class="com.acme.DataSourceImpl"
    user="frank"
    password="frankpw"
    url="jdbc:acme:@localhost:5500:acme" />
```

データソース構成の追加の例は、
http://www.oracle.com/technology/tech/java/newsletter/articles/oc4j_datasource_config.html にある資料「Data Source Configuration in Oracle Application Server 10*g*」を参照してください。

name 属性では、一意のネイティブ・データソースを指定します。jndi-name 属性では、このデータソースを配置する JNDI ロケーションを定義します。data-source-class 属性では、ネイティブ・データソースの実装クラスを定義します。このクラスは、javax.sql.DataSource を実装している必要があります。user および password 属性では、デフォルトのユーザーとパスワードを定義します。url 属性では、データソースがデータベースと通信するための URL を定義します。

致命的エラー・コードの定義

data-sources.xml で定義したデータソースごとに、そのデータソースの通信対象であるバックエンド・データベースにアクセスできないことを示す致命的エラー・コードを定義できます。OC4J でこれらのエラー・コードのいずれかが検出されると（JDBC ドライバによって SQLException がスローされた場合）、その接続プールは削除されます。つまり、接続プールのすべての接続がクローズされます。Oracle で事前定義されている致命的エラー・コードは、3113、3114、1033、1034、1089 および 1090 です。

Oracle 以外のデータベースに致命的エラー・コードを定義する場合や、Oracle データベースに別の致命的エラー・コードを追加する場合は、次の手順を使用します。

<connection-factory> 要素のサブタグである <fatal-error-codes> 要素を使用します。<fatal-error-codes> 要素で、子要素の <error-code> を使用して 1 個の致命的エラー・コードを定義します。1 つの <fatal-error-codes> 要素に対して、0 ～ n 個の <error-code> 要素を定義できます。たとえば、致命的エラー・コードとして 10、20、30 を設定する場合、データソース定義は次のようになります。

```
<connection-pool name="myConnectionPool">
   <connection-factory factory-class="oracle.jdbc.pool.OracleDataSource"
      user="scott"
      password="tiger"
      url="jdbc:oracle:thin:@//localhost:1521/
         oracle.regress.rdbms.dev.us.oracle.com">
      <fatal-error-codes>
         <error-code code='10'/>
         <error-code code='20'/>
         <error-code code='30'/>
      </fatal-error-codes>
   </connection-factory>
</connection-pool>
```

パスワードの間接化の使用方法

data-sources.xml ファイルには、認証用のパスワードが必要です。これらのパスワードをなんらかの形式で不明瞭化することなく埋め込むと、セキュリティ上のリスクが生じます。この問題を回避するため、OC4J では、パスワードの間接化をサポートしています。

間接パスワードは、特殊な間接化記号（->）とユーザー名（またはユーザー名とレルム）で構成されます。OC4J は、間接パスワードを検出すると、ユーザー・マネージャが提供するセキュリティ・ストアから、指定のユーザーに関連付けられているパスワードを取得します。

ユーザーとパスワードの作成およびユーザー・マネージャの操作の詳細は、『Oracle Containers for J2EE セキュリティ・ガイド』のパスワード管理に関する項を参照してください。

たとえば、次のようなネイティブ・データソースのエントリがあるとします。

```
<native-data-source
   name="nativeDataSource"
   jndi-name="jdbc/nativeDS"
   data-source-class="com.acme.DataSourceImpl"
   user="frank"
   password="frankpw"
   url="jdbc:acme:@localhost:5500:acme" />
```

この場合、パスワード frankpw は、間接化記号（->）とユーザー名（frank）を使用して、password="->frank" のように置き換えることができます。この操作は、ユーザー名 frank とそのパスワード frankpw がユーザー・マネージャに作成されていることを前提とします。

パスワードの間接化は、Application Server Control コンソールで構成できます。

データソースの間接パスワードを data-sources.xml ファイルに直接構成するには、password 属性の値を -> で始め、続けてユーザー名（またはスラッシュ（/）で区切ったレルムとユーザー名）を指定します。次に例を示します。

```
<native-data-source
   name="nativeDataSource"
   jndi-name="jdbc/nativeDS"
   data-source-class="com.acme.DataSourceImpl"
   user="frank"
   password="->frank"
   url="jdbc:acme:@localhost:5500:acme" />
```

<managed-data-source> および <connection-factory> 要素には、password 属性もあります。

サンプル・データソースの削除

data-sources.xml ファイルには、サンプルの接続プール、およびデフォルトのデータソースとして使用されるマネージド・データソースのエントリが含まれます。これらの例は説明を目的として使用され、データソースを理解するために用意されています。

接続プールおよびデータソースの削除は、Application Server Control コンソール（**「OC4J: ホーム」→「管理」タブ→「タスク名 : サービス」→「JDBC リソース : タスクに移動」**）を使用するか、data-sources.xml ファイルから手動でエントリを削除することで実行できます。コンソールを使用する場合は、データソースを削除してから接続プールを削除します。

サンプルのデータソースを削除したら、default-data-source 属性を使用して、新しいデフォルトのデータソースを J2EE_HOME/application.xml ファイルに割り当てます。実行時に適切なデータソース構成が検出されない場合、この OC4J インスタンスにデプロイされたすべてのアプリケーションは、デフォルトでこのデータソースに設定されます。ただし、OC4J インスタンスを実行するためにはデフォルトのデータソースは不要です。

注意 : 個々のアプリケーション・レベル（独自の application.xml）に定義されているデフォルトのデータソースは、このグローバルのデフォルトのデータソースより優先されます。

接続

OC4J のデータソースは、次の 2 つのタイプの接続を戻します。

- **マネージド接続 :** マネージド・データソースから戻される接続は、接続プールから取得され、プロキシ・クラスでラップされます。そのため、OC4J では、SQL エラーを処理することや、これらの接続をグローバル・トランザクションに登録することができます。5-9 ページの「マネージド・データソースでの接続プロキシの使用方法」を参照してください。
- **ネイティブ接続 :** ネイティブ・データソースから戻された接続は、OC4J では操作できません。つまり、この接続はラップされていない状態であり、関連付けられたプロキシもなく、グローバル・トランザクションにも参加できません。

接続の確立

データソースでは、データベースと通信することで接続を生成します。通常、データソースでは、データベースとの通信に使用するマシン、ポート、データベース名などの識別に URL を使用します。

マネージド・データソースの URL は、そのマネージド・データソースの接続プールのコネクション・ファクトリで定義します。コネクション・ファクトリの `url` 属性により、データベースとの通信に使用する URL を定義します。JDBC ドライバでは、URL の書式を定義しています。このドキュメント全体を通じて、いくつかの URL の例が記載されています。

ネイティブ・データソースの URL は、`<native-data-source>` 要素の `url` 属性で定義します。

マネージド・データソースでの接続プールの使用方法

基本的なデータソース実装では、クライアントの接続オブジェクトと物理接続の間に 1 対 1 の対応関係があります。通常、クライアントが接続をクローズすると、物理接続は削除されます。この場合、クライアントがデータソースから接続を取得するたびに、多くのオーバーヘッドが生じます。

OC4J のマネージド・データソースでは、接続プールを頻繁に使用します。

接続プールは、複数のマネージド・データソースによって使用できます。つまり、複数のマネージド・データソースで同じ接続プールを共有できます。

Oracle10*g* JDBC ドライバを使用するようデータソースを定義すると、OC4J では、このドライバに付属する Implicit Connection Cache（ICC）により提供される高機能な接続プールが使用されます。OC4J データソースでは、自動的に ICC が使用されます。ICC を無効化する方法は、5-31 ページの「ICC の無効化」を参照してください。特に指定のないかぎり、5-26 ページの表 5-3「接続プールの属性」に記載されたすべての接続プール属性が ICC に適用されます。一部の属性は、Implicit Connection Cache 対応のデータソース（`OracleDataSource` および `OracleXADataSource`）にのみ適用されます。ICC を使用するために追加の構成を実行する必要はありません。

接続プールの構成設定の詳細は、5-26 ページの表 5-3「接続プールの属性」を参照してください。

接続プールは、Oracle 以外の JDBC ドライバや旧リリースの Oracle JDBC ドライバでも使用できます。接続プール構成の例は、5-31 ページの「構成例」を参照してください。

マネージド・データソースでの接続プロキシの使用方法

マネージド・データソースを使用する場合、接続プールから取得される各接続は、OC4J によりプロキシ・オブジェクトでラップされます。このプロキシにより、OC4J では、トランザクション登録、例外処理およびロギングが可能になります。

ベンダー固有の拡張

クライアントでは、`java.sql` インタフェースのベンダー実装により提供される拡張インタフェースも使用できます。たとえば、`java.sql.Connection` インタフェースの Oracle 拡張は、`oracle.jdbc.OracleConnection` です。このインタフェースにより、`java.sql.Connection` インタフェースには含まれない Oracle 固有の API が提供されます。OC4J には、プロキシが実装するインタフェースを制限するための構成要素があり、クライアントのアクセス先をこれらの API に限定できます。この構成要素は、任意の `java.sql.*` インタフェースの追加インタフェースを指定するために使用できます。デフォルトでは、プロキシは、基礎となるオブジェクトにより実装されている任意の public インタフェースを実装します。

プロキシの設定方法の詳細は、5-26 ページの表 5-3「接続プールの属性」を参照してください。

次の例で、Application Server Control コンソールではなく data-sources.xml ファイルで接続プールの接続プロキシと文プロキシを定義する方法を示します。

```
<connection-pool name="myConnectionPool">
   <connection-factory
      factory-class="com.acme.AcmeDataSource"
      user="scott"
      password="tiger"
      url="jdbc:acme:@localhost:1234:acme">
      <property name="foo" value="bar"/>
      <proxy-interface sql-object="Connection"
         interface="com.acme.AcmeConnection"/>
      <proxy-interface sql-object="CallableStatement"
         interface="com.acme.AcmeCallableStatement"/>
   </connection-factory>
</connection-pool>
```

この例において、Connection オブジェクト用に生成されるプロキシは、基礎となる接続オブジェクトによって実装されるインタフェースにかかわらず、com.acme.AcmeConnection インタフェースのみを公開します。同様に、Statement オブジェクト用に生成されるプロキシは、com.acme.AcmeStatement インタフェースのみを公開します。データソースのデプロイヤは、この方法により、プロキシ・オブジェクトによって公開されるインタフェースを制限できます。

データソースからの接続の取得

この項では、データソースから接続を取得して文を実行するためのサンプル・コードを示します。

注意：

例外がスローされる場合でも、データソースから取得した接続は必ずクローズしてください。

ルックアップの実行に使用する文字列は、マネージド・データソースまたはネイティブ・データソースの JNDI ロケーションを示す jndi-name 設定の値と一致している必要があります。

```
Connection connection = null;
try {
   InitialContext context = new InitialContext();
   DataSource ds = (DataSource) context.lookup( "jdbc/ManagedDS" );
   connection = ds.getConnection();
   Statement statement = connection.createStatement();
   statement.execute( "select * from dual" );
   statement.close();
}
catch( Exception exception ) {
   // process exception
}
finally {
   if ( connection != null )
   {
      try {
         connection.close();
      }
      catch( SQLException sqlException ){}
   }
}
```

注意：

ユーザーとパスワードを渡さずに `getConnection()` を使用した場合、コネクション・ファクトリの定義で指定されたユーザーとパスワードが自動的に使用され、正常に接続が作成されます。コネクション・ファクトリで指定されたユーザーとパスワードのペアが、属性とプロパティで異なる場合、`getConnection()` ではプロパティに指定されたユーザーとパスワードが使用されます。

コネクション・ファクトリでユーザーとパスワードを指定する方法は、5-30 ページの表 5-4「コネクション・ファクトリの属性」を参照してください。

コネクション・ファクトリのプロパティの詳細は、5-24 ページの「コネクション・ファクトリのプロパティ」を参照してください。

再試行

一定の状況下では、データソースにより接続を戻すことができない場合があります。この最も一般的な原因は、接続プールのすべての接続が使用中であることです。データソースを一定時間待機させた後に接続プールをチェックして、接続を戻す前に使用可能な接続があるかどうかを確認できます。

接続プールの構成設定には、プールの接続が使用できない場合の待機時間を制御する設定と、接続プールに接続が存在するかどうかを問い合せる試行の回数を制御する設定の 2 つがあります。

`max-connect-attempts` 設定では、（接続プールのすべての接続が使用中の場合に）マネージド・データソースで接続プールから接続を取得する操作を再試行する回数を定義します。`connection-retry-interval` 設定では、接続プールからの接続の取得に最後に失敗した後、次の取得操作を再試行するまでに待機する間隔を秒単位で指定します。これらの設定の詳細は、5-26 ページの表 5-3「接続プールの属性」を参照してください。

次の例で、Application Server Control コンソールではなく `data-sources.xml` で再試行を構成する方法を示します。この例では、`max-connect-attempts` を `5` 回に、`connection-retry-interval` を `3` 秒に設定します。

```
<connection-pool name="myConnectionPool"
   max-connect-attempts="5"
   connection-retry-interval="3">
   <connection-factory
      factory-class="oracle.jdbc.pool.OracleDataSource"
      user="scott"
      password="tiger"
      url="jdbc:oracle:thin:@//localhost:1521/
         oracle.regress.rdbms.dev.us.oracle.com"/>
</connection-pool>
```

Oracle JDBC ネイティブ・データソースを使用したプロキシ認証

OC4J では、Oracle JDBC ネイティブ・データソースを使用したプロキシ認証がサポートされています。Oracle データベースではプロキシ認証がサポートされています。プロキシ認証を使用すると、クライアント・ユーザーはアプリケーション・サーバーを介して、プロキシ・ユーザーとしてデータベースに接続できます。アプリケーション・サーバーは Oracle データベースを使用して、プロキシ・ユーザーとして自身を認証しますが、クライアント・ユーザーは、アプリケーション・サーバーを使用して自身を認証します。この方法で確立されたプロキシ接続では、データベースに接続するまで同じクライアント・ユーザー名が使用されます。

Oracle JDBC ネイティブ・データソースを使用したプロキシ認証の構成および使用方法の主要な手順を説明し、構成済の Oracle データベースへのプロキシ接続を JSP コードでテストするデモが公開されています。デモの文書とソース・コードは次のサイトで公開されています。

```
http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/
index.html
```

プロキシ認証の動作

現在、OC4J の観点からすると、中間層ではグローバル・トランザクションのコンテキストでプロキシ・セッションをサポートすることはできません。主な問題は、プロキシ・セッションが既存の JDBC 接続ハンドル上に確立され、この接続ハンドルの基礎となる状態の変更が中間層のトランザクション・マネージャに認識されないことです。そのため、OC4J トランザクション・マネージャでプロキシ・セッションと通常のセッションを区別することができません。プロキシ・セッションの作成時に既存の JDBC 接続ハンドルが再利用されるため、トランザクション・マネージャがトランザクションの関連付けを終了するようリソース・マネージャに通知するまで、中間層では接続ハンドルでの処理は既存のグローバル・トランザクションの一部であるとみなされます。

OC4J の旧リリースでは、JDBC プロキシ・セッションの実装がサポートされるかどうかは、2 セッションのモデルに依存していました。このモデルでは、まず信頼できるユーザー（通常はアプリケーション）セッションが確立され、その後、プロキシ処理が発生するとターゲット・ユーザーのその他のセッションが確立されます。どちらのセッションもデータベースに保持されます。このデータベースでは、信頼できるユーザー・セッションはプロキシ・セッションがアクティブなかぎり一時停止されます。このモデルは、単一インスタンスのデータベースへの接続や、複数のターゲット・ユーザーとしてのプロキシをアプリケーションで保持する必要がある場合に適しています。ただし、いくつかのデメリットがあります。

- プロキシする必要のあるデータベース・ユーザー（独自のデータベース・ユーザー / スキーマなど）が 1 つのみのアプリケーションでは非効率的です。この場合、元のアプリケーション・セッションを維持する必要はありません。
- RAC の 2 セッションのモデルのブレーク。これらのブレークでは、それぞれのセッションは異なる RAC インスタンスで終了します。たとえば、アプリケーションにより 1 つ目の信頼できるセッションが取得され、RAC の inst1 で終了します。その後、プロキシ・セッションが取得され、RAC の inst2 で終了します。

JDBC/OC4J では、プロキシ・セッションの作成を可能にする、Connection の openProxySession API が公開されています。この JDBC API が既存の信頼できるユーザー接続 / セッションで起動されると、バックエンドで 2 つ目のプロキシ・セッションが作成され、その信頼できるユーザーのセッションが一時停止されます。中間層のサブシステムの多くで大規模な変更が必要なため、OC4J/TM で XA および 2 セッション・アーキテクチャのプロキシ・セッションをサポートすることはできません。これらの変更は間違いやすく、リスクも伴います。

サンプルのプロキシ認証

```
InitialContext ic = new InitialContext();
   DataSource nativeDS =
       (DataSource) ic.lookup(NATIVE_DS_NAME);
   OracleConnection oconn = (OracleConnection)
       nativeDS.getConnection(PROXY_USER_NAME, PROXY_USER_PWD);
   strBuf.append("Obtained a connection using getConnection(" +
                  PROXY_USER_NAME + ", " + PROXY_USER_PWD + ")\n");

   strBuf.append("isProxySession: " + oconn.isProxySession() + "\n");
   strBuf.append("Check user name before opening the proxy session\n");
   strBuf.append(checkUser(oconn));

   // Specify the user that connects through the proxy user and its roles
   Properties prop = new Properties();
   prop.put(OracleConnection.PROXY_USER_NAME, USER_NAME);
   prop.put(OracleConnection.PROXY_ROLES, roles);
   // Open the proxy session (DB-authenticated users)
   oconn.openProxySession(OracleConnection.PROXYTYPE_USER_NAME, prop);

   strBuf.append("Opened a proxy session for " + USER_NAME +
                 " through " + PROXY_USER_NAME + " using
                   openProxySession()\n");
   strBuf.append("isProxySession: " + oconn.isProxySession() + "\n");

   // Now test using the proxy-connection
   strBuf.append("### Testings using the proxy session ###\n");
   String resultStr = testAndGetResultString(oconn);
   strBuf.append(resultStr);

   strBuf.append("Closing the proxy session ...\n");
   oconn.close(OracleConnection.PROXY_SESSION);

   strBuf.append("isProxySession: " + oconn.isProxySession() + "\n");
   strBuf.append("Check user name after closing the proxy session\n");
   strBuf.append(checkUser(oconn));

   strBuf.append("Closing the original connection ...\n");
   oconn.close();
```

文

マネージド・データソースには、文の操作をより効率的に実行するためのキャッシング機能とプロキシ機能があります。

マネージド・データソースでの文キャッシュ

文キャッシュにより、繰り返し使用される実行可能文がキャッシュされるため、パフォーマンスが向上します。また、プログラマは、プリコンパイルされた文を明示的に再利用する必要がなくなります。文キャッシュを使用することで、カーソル作成の繰返しや文の解析と作成の繰返しに起因するオーバーヘッドを排除できると同時に、アプリケーション・サーバーとデータベース・サーバー間の通信により発生するオーバーヘッドも削減できます。文キャッシュの特徴は、次のとおりです。

- 文のキャッシュと再利用は、アプリケーションには透過的に動作します。
- 各文キャッシュは、物理接続に関連付けられます。つまり、物理接続ごとに独自の文キャッシュを保持します。
- 文の一致条件は、次のとおりです。
  - 文の SQL 文字列は、キャッシュ内の文字列と同一である必要があります（大 / 小文字も区別されます）。
  - 文のタイプ（`prepared` または `callable`）は、同一である必要があります。
  - 文により生成される結果セットのスクロール可能タイプ（`forward-only` または `scrollable`）は、同一である必要があります。
  - キャッシュされる文の最大数。

データソースでの JDBC 文のキャッシュ・サイズの設定

カーソル作成の繰返しや文の解析と作成の繰返しによるオーバーヘッドを削減するため、データベース文で文キャッシュを使用できます。JDBC 文のキャッシング機能を有効化して繰り返し使用される実行可能文をキャッシュするには、文キャッシュを使用するようデータソースを構成します。JDBC 文のキャッシュは、データソースで管理されている特定の物理接続と関連付けられます。文キャッシュは、データソースに関連付けられていないため、すべての物理接続で共有することはできません。JDBC 文のキャッシュは、データベース・サーバーではなく中間層で管理されます。

文キャッシュの有効化と無効化をプログラムで動的に切り替えるには、接続オブジェクトの `setStmtCacheSize()` メソッドを使用します。

データソースで JDBC 文のキャッシング機能を構成するには、`num-cached-statements` 属性を使用してキャッシュのサイズを設定します。この属性では、キャッシュに配置する文の最大数を設定します。`num-cached-statements` 属性を指定しないか、`0` に設定した場合、文キャッシュは無効化されます。

次の XML では、文のキャッシュ・サイズを 200 文に設定します。

```
<data-source>
 ...
num-cached-statements="200"
</data-source>
```

`num-cached-statements` 属性を設定するには、まずアプリケーションがデータベースに発行する個別の文の数を決定します。次に、キャッシュのサイズをその数に設定します。アプリケーションがデータベースに発行する文の数がわからない場合は、JDBC のパフォーマンス・メトリックを使用すると文のキャッシュ・サイズの決定に役立ちます。文メトリックを使用するには、OC4J で Java プロパティの `oracle.jdbc.DMSStatementMetrics` を `true` に設定する必要があります。

注意： データソースで JDBC 文のキャッシング機能を構成するには、num-cached-statements 属性を使用してキャッシュのサイズを設定します。stmt-cache-size 属性は、廃止予定です。

文のキャッシュ・サイズのリソース問題

データソースに num-cached-statements を指定しても、文はデータソースや接続プールごとにではなく、接続ごとにキャッシュされます。つまり、特定のデータソースの num-cached-statements が 0 より大きい場合、そのデータソースから取得されたマネージド接続ごとに独自の文キャッシュが保持されます。

接続の文キャッシュに保持されている文は、データベース・リソースを解放しない可能性があることに注意してください。オープンしている接続の数と、接続ごとにキャッシュされている文の数の合計が、データベースに許可されたオープン・カーソルの制限を超える可能性があります。num-cached-statements の値を減らすか、データベースに許可されたオープン・カーソルの制限を増やすことで、この問題を回避できる場合があります。

マネージド・データソースでの文プロキシ

java.sql.* インタフェース（マネージド・データソース）のすべての実装は、OC4J によりプロキシ・オブジェクトでラップされます。これには、文オブジェクト（java.sql.Statement、java.sql.PreparedStatement および java.sql.CallableStatement）も含まれます。

プロキシの設定方法の詳細は、5-26 ページの表 5-3「接続プールの属性」に記載されているコネクション・ファクトリの「プロキシ・インタフェース」タブの説明を参照してください。

一定の状況下では、接続プロキシが新しい物理接続にリバインドされる場合があります。これは、接続プロキシが 1 つのトランザクション全体で使用される場合などに発生します。リバインドが発生すると、接続プロキシを通じて取得された文オブジェクトは、古い物理接続を使用して作成されているため、すべて無効になります。このため、物理接続から取得される文オブジェクトに対しても、プロキシが配置されます。これらの文プロキシは、基礎となる物理接続との関連を監視できるように、取得元の接続プロキシと関連付けられます。文プロキシにより、接続プロキシに関連付けられた物理接続が変化したと判断されると、接続プロキシから新しい物理文が取得されます。

java.sql.Statement、java.sql.PreparedStatement および java.sql.CallableStatement インタフェースのベンダー固有の拡張も、接続と同じ方法によりクライアントで使用できます。

トランザクション

J2EE では、次の 2 種類のトランザクションがサポートされます。

- **ローカル・トランザクション：**ローカル・トランザクションは、単一のリソースの内部で実行されます。
- **グローバル・トランザクション：**グローバル・トランザクションは、外部のトランザクション・マネージャによって作成され、複数のリソースを操作対象として実行されます。

ローカル・トランザクションとグローバル・トランザクションを含むトランザクション・サポートの詳細は、第 5 章「データソース」を参照してください。

ローカル・トランザクション

ローカル・トランザクション用に構成されたマネージド・データソースは、ローカル・トランザクションに参加できるがグローバル・トランザクションには参加できない接続を戻します。つまり、その接続は、グローバル・トランザクションに登録されません。データソースは、取得した接続の自動コミットを true に設定します。ただし、ローカル・トランザクションでその接続をどのように使用するかを決定するのは、クライアントです。クライアントは、接続に対して `setAutoCommit()` を使用することで、自動コミット・モードを変更できます。

マネージド・データソースをローカル・トランザクション用に設定する方法は、5-20 ページの表 5-1「マネージド・データソースの設定」に記載されている「トランザクション・レベル」の設定を参照してください。

Application Server Control コンソールではなく `data-sources.xml` ファイルでデータソースをローカル・トランザクション用に構成するには、`tx-level` 属性を `local` に設定します（デフォルト値は `global` です）。次に例を示します。

```
<managed-data-source
   jndi-name="jdbc/ManagedDS"
   name="Managed DataSource Name"
   connection-pool-name="myConnectionPool"
   tx-level="local" />
```

ネイティブ・データソースは、ローカル・トランザクションにのみ参加できるため、ネイティブ・データソース用のトランザクション・サポートの設定はありません。

`local` としてマーク付けされた接続を、グローバル・トランザクションの内部で使用することも可能です。このケースは、JDBC 仕様で具体的に言及されているわけではありませんが、仕様には、接続が分散トランザクションに参加しない場合、その接続はローカル接続のように振る舞うという意味の記載があります。接続がグローバル・トランザクションに登録されない場合、その接続はグローバル・トランザクションに参加しません。したがって、`local transaction` で構成されたデータソースにより生成される接続は、たとえグローバル・トランザクションの内部でその作業が実行される場合でも、ローカル・トランザクションの内部に存在するかのように扱われます。つまり、自動コミットが `false` に設定されている場合、このような接続で実行される作業は、その接続で commit や rollback がコールされるまでコミットまたはロールバックされません。この動作は、分散トランザクションでコミットまたはロールバックが実行される場合でも同様です。このケースでは、接続は、構文的にトランザクション境界を伴って出現するだけで、意味的にはそのトランザクションに参加しません（つまり、この接続は登録されません）。

例：

- ローカル・トランザクション用に構成されたデータソースから接続 `lc` を取得します。
- グローバル・トランザクションを開始します。
- グローバル・トランザクション用に使用できるデータソースから接続 `gc` を取得します。
- 両方の接続上で作業を実行します。

  `lc` 上で処理される作業は、その作業が実行されるか（自動コミットが `true` の場合）、`lc` 上で commit や rollback がコールされると（自動コミットが `false` の場合）コミットされます。
- グローバル・トランザクションをコミットまたはロールバックします。

  `gc` 上の作業は、この時点でコミットまたはロールバックされます。`lc` 上の作業は、コミットされません。
- `lc` 上の作業は、その接続で `commit` または `rollback` をコールすることでコミットまたはロールバックできます（自動コミットが `false` に設定されている場合）。

次のコード例では、前述の手順を実装します。

```
Connection lc = localTxDataSource.getConnection();
userTransaction.begin();
Connection gc = globalTxDataSource.getConnection();
lc.doWork();
gc.doWork();
userTransaction.commit();
// At this point work done on gc is now committed.
//The work done on lc is NOT yet committed.
lc.commit();
// At this point work done on lc is now committed.
```

ローカル・トランザクション管理

通常、ローカル・トランザクションは、クライアントが接続上で autoCommit を false に設定すると開始し、クライアントが同じ接続上で commit() または rollback() をコールすると終了します。autoCommit が false で、トランザクション作業が実行されていない場合、接続で明示的に commit() または rollback() をコールする必要はありません（トランザクション作業が実行されていないためにコミットやロールバックが不要であることは、ドライバにより正しく認識されます）。

OC4J は、autoCommit が false であり、かつ commit()、rollback()、setAutoCommit(true) または close() 以外の任意のメソッドが接続上でコールされると、アクティブなローカル・トランザクションがその接続に存在すると判断します（ただし、OC4J は、接続上で処理される作業が実際にトランザクション形式の作業であるかどうかは判断できません）。commit() や rollback() のコール、または autoCommit の値の変更が発生すると、現在のローカル・トランザクションは終了します。autoCommit が false であり、かつ commit()、rollback()、setAutoCommit(true) または close 以外のメソッドが引き続き接続上でコールされると、OC4J は、それを新しいローカル・トランザクションの開始と判断します。

クライアントが接続で（commit()、rollback() または setAutoCommit(true) をコールして）明示的にローカル・トランザクションを終了しない場合については、次の 2 つのケースを検討する必要があります。

- 第 1 のケース : アクティブなローカル・トランザクションが存在し、接続がユーザーによってクローズされる場合。
- 第 2 のケース : アクティブなローカル・トランザクションが存在し、接続がグローバル・トランザクションで使用される場合。

OC4J では、次の 2 つの方法でこれらのケースを処理できます。

- 接続がクローズされるか、グローバル・トランザクションで使用される場合に、OC4J でローカル・トランザクションを管理および調整します。具体的には、OC4J は、commit() または rollback() をコールして暗黙的にローカル・トランザクションを終了します。また、OC4J では、接続がクローズされるか、グローバル・トランザクションで使用される場合に例外をスローできます。
- OC4J でローカル・トランザクションを管理せず、これらのケースを調整しません。具体的には、接続がクローズされるか、グローバル・トランザクションで使用される場合に、リソース側でローカル・トランザクションを終了する方法（または終了しない方法）を決定する必要があります。ローカル・トランザクションを管理しないよう OC4J を構成している場合、接続が接続プールに戻されたときに、コミットされていないローカル・トランザクションがアクティブになっている可能性があることに注意してください。

グローバル・トランザクション（XA）

グローバル・トランザクション用に構成されたマネージド・データソースは、グローバル・トランザクションに参加できる接続を戻します。グローバル・トランザクション（分散トランザクション）では、トランザクションに複数のリソースが登録されます。

未処理の J2CA ローカル・トランザクションが存在する場合に、トランザクション・マネージャでグローバル・トランザクションを処理する方法の詳細は、5-17 ページの「ローカル・トランザクション管理」を参照してください。

トランザクションの詳細は、第 3 章「OC4J トランザクション・サポート」を参照してください。

マネージド・データソースをグローバル・トランザクション用に設定する方法は、5-20 ページの表 5-1「マネージド・データソースの設定」に記載されている「トランザクション・レベル」の設定を参照してください。

Application Server Control コンソールではなく `data-sources.xml` ファイルでデータソースをグローバル・トランザクション用に構成するには、`tx-level` 属性（デフォルト値は `global`）を含めないか、`tx-level` 属性を `global` に設定します。次に例を示します。

```
<managed-data-source
   jndi-name="jdbc/ManagedDS"
   name="Managed DataSource Name"
   connection-pool-name="myConnectionPool"
   tx-level="global" />
```

XA リカバリ

グローバル・トランザクションが失敗した場合、トランザクション・マネージャで XA リカバリを実行する必要があります。この場合、リカバリするリソースごとにいくつかの情報を定義しておく必要があります。データソースでは、factory-class に `javax.sql.XADataSource` を使用するコネクション・ファクトリごとに、リカバリ用の `username` と `password` を定義します。

5-26 ページの表 5-3「接続プールの属性」に記載されている「ユーザー」および「パスワード」の設定を参照してください。

次の例で、Application Server Control コンソールではなく `data-sources.xml` ファイルで XA リカバリを構成する方法を示します。`xa-recovery-config` ノードに注意してください。

```
<connection-pool name="myConnectionPool">
   <connection-factory
      factory-class="oracle.jdbc.xa.client.OracleXADataSource"
      user="scott"
      password="tiger"
      url="jdbc:oracle:thin:@//localhost:1521/
         oracle.regress.rdbms.dev.us.oracle.com">
      <xa-recovery-config>
         <password-credential>
            <username>system</username>
            <password>manager</password>
         </password-credential>
      </xa-recovery-config>
   </connection-factory>
</connection-pool>
```

注意：

- factory-class が java.sql.Driver、javax.sql.DataSource または javax.sql.ConnectionPoolDataSource のインスタンスである場合、XA リカバリ構成は不要です。詳細は、5-19 ページの「XA のエミュレート」を参照してください。
- OracleXADataSource も、javax.sql.XADataSource のインスタンスです。

XA のエミュレート

エミュレートされた XAResource は、XA プロトコルのセマンティックをエミュレートする javax.sql.XAResource の実装です。グローバル・トランザクション中、エミュレートされた XAResource に関連付けられた接続は、グローバル作業ユニットのサブセットとして明示的に制御されるトランザクション・ブランチではなく、ローカル・トランザクションを使用して XA のセマンティックに準拠します。

javax.sql.XADataSource の実装を提供しない JDBC ドライバをサポートする場合は、XAResource をエミュレートする必要があります。エミュレートされた XAResource は、実際の 2 フェーズ・コミットに関連するオーバーヘッドによるパフォーマンス上の影響を受けないため、本物の XAResource より高速に動作します。

エミュレートされた XAResource を使用する場合、複数の XAResource を登録し、少なくともその 1 つをエミュレートすると、一貫性のない状態やリカバリ不可能な状態が発生することがあります。エミュレートされた XAResource は、準備フェーズ中にローカル・トランザクションを使用するため、正しい準備を実行できないのがその理由です。これが問題となる 1 つの例としては、エミュレートされた XAResource で commit がコールされたときに、そのローカル・トランザクションがすでにタイムアウトしており、結果としてローカル・トランザクションのコミットに失敗し、トランザクション全体が一貫性のない状態に陥ることがあります。

OC4J では、XA 動作をエミュレートする状況が自動的に判別されます。これは、コネクション・ファクトリの factory-class オブジェクトを内部監視することで実行されます（factory-class 属性では、接続プール用の接続を作成するためのコネクション・ファクトリで使用するオブジェクトを指定します）。このオブジェクトが javax.sql.XADataSource のインスタンスであれば、XA はエミュレートされません。このオブジェクトが java.sql.Driver、javax.sql.DataSource または javax.sql.ConnectionPoolDataSource のインスタンスである場合、このデータソースで XA 動作がエミュレートされます。

データソース・オブジェクトの構成

この項では、Application Server Control コンソールで設定するか、data-sources.xml ファイルで直接設定するかを問わず、様々なデータソース関連オブジェクトの構成設定とその説明を示します。

各設定は、次の表に記載されています。

- 表 5-1「マネージド・データソースの設定」
- 表 5-2「ネイティブ・データソースの設定」
- 表 5-3「接続プールの属性」

データソース・オブジェクトを構成する方法の詳細は、5-4 ページの「データソースの定義」を参照してください。

マネージド・データソース

マネージド・データソースを定義するには、1 つ以上の接続プールを定義する必要があります。

マネージド・データソース設定へのパス

「OC4J: ホーム」→「管理」タブ→「タスク名 : サービス」→「JDBC リソース」→「タスクに移動」→「データソース」→「作成」→「アプリケーションの選択」→「データソース・タイプの選択」→「マネージド」

- 1 つの <managed-data-source> タグで、1 つのマネージド・データソースを定義します。
- 各属性の詳細は、5-20 ページの表 5-1「マネージド・データソースの設定」を参照してください。

次の例で、data-sources.xml のマネージド・データソースの定義を示します。

```
<managed-data-source
   jndi-name="jdbc/ManagedDS"
   name="Managed DataSource Name"
   connection-pool-name="myConnectionPool"/>
```

表 5-1 マネージド・データソースの設定

Application Server Control コンソールのプロパティ	<managed-data-source> の属性	説明
名前	name	**必須**。データソースの名前。この値は、一意である必要があります。 この名前は、JDBCDataSource 管理オブジェクトの name キー・プロパティとして使用されます (j2eeType=JDBCDataSource,name=data source name)。
JNDI ロケーション	jndi-name	**必須**。データソース・オブジェクトの JNDI 論理名。OC4J では、データソースのインスタンスが、この値を持つアプリケーション JNDI ネームスペースにバインドされます。
接続プール	connection-pool-name	**必須**。このマネージド・データソースで接続を取得するのに使用する接続プールの名前。
スキーマ	schema	EJB の Orion CMP 実装の使用時における、このデータソースのデータベース・スキーマへのパス。これは、下位互換性を目的として提供されています。

表 5-1 マネージド・データソースの設定（続き）

Application Server Control コンソールのプロパティ	<managed-data-source> の属性	説明
トランザクション・レベル	tx-level	このマネージド・データソースでサポートされるトランザクション・レベル。 値 local を指定すると、このデータソースと生成される接続は、ローカル・トランザクションにのみ参加できます。 値 global を指定すると、このデータソースと生成される接続は、ローカルおよびグローバル・トランザクションに参加できます。 オプション設定です。デフォルトは、global です。
ローカル・トランザクション管理	manage-local-transactions	OC4J でローカル・トランザクションを管理するかどうかを指定します。 ■ この設定が true で、接続の autoCommit が false であると、次のようになります。 接続で close() がコールされると、OC4J は、接続で commit() をコールしてから close() をコールします。 接続がグローバル・トランザクションで使用されている場合、OC4J は、接続で rollback() をコールし、例外をスローします。 ■ この設定が false で､接続の autoCommit が false であると、次のようになります。 接続で close() がコールされると、OC4J は、接続で commit() をコールせずに close() をコールします。ローカル・トランザクションの処理方法は、JDBC ドライバにより決定されます。 接続がグローバル・トランザクションで使用されている場合、OC4J は、接続で rollback() をコールせず、例外もスローしません。ローカル・トランザクションの処理方法は、JDBC ドライバにより決定されます。 オプション設定です。デフォルトは、true です。

表 5-1 マネージド・データソースの設定（続き）

Application Server Control コンソールのプロパティ	<managed-data-source> の属性	説明
SQL オブジェクト管理	manage-sql-objects	OC4J で java.sql.* オブジェクトを管理する方法を決定します。これらのオブジェクトのいずれかを管理する場合、そのオブジェクトは、OC4J によってプロキシでラップされます。また、オブジェクトで起動されるメソッドは、OC4J の割込みを受けます。 値 all を指定すると、すべての java.sql.* オブジェクトが OC4J により管理されます。 値 basic を指定すると、java.sql.Connection、java.sql.Statement、java.sql.PreparedStatement および java.sql.CallableStatement オブジェクトのみ OC4J により管理されます。 値 minimal を指定すると、プールの管理および制限付きのトランザクション登録用の java.sql.Connection オブジェクトのみが OC4J により管理されます。Connection がグローバル・トランザクションのコンテキストで取得された場合、その接続は OC4J によりグローバル・トランザクションに登録されます。接続 / トランザクションの関連を管理するために Connection メソッドは捕捉されないため、Connection はトランザクション境界全体でキャッシュされません。そのため Connection は、それが取得されたトランザクション・コンテキストでのみ使用し、トランザクションが完了したら終了する必要があります。Statement オブジェクトはこの構成ではラップされないため、Statement オブジェクトでのメソッドの起動に関連するオーバーヘッドはありません。 オプション設定です。デフォルトは、basic です。
ログイン・タイムアウト	login-timeout	データベースへの接続試行時にこのデータソースが待機する最大時間（秒単位）。値 0（ゼロ）を指定すると、デフォルトのシステム・タイムアウト時間が存在する場合はその時間となり、存在しない場合はタイムアウトなしとなります。 オプション設定です。デフォルトは、0 です。
ユーザー	user	データベースへの接続に使用するデフォルト・ユーザー。 オプション設定です。デフォルトはありません。
パスワード	password	データベースへの接続に使用するデフォルト・パスワード。 オプション設定です。デフォルトはありません。

ネイティブ・データソース

ネイティブ・データソース設定へのパス

「OC4J: ホーム」→「管理」タブ→「タスク名 : サービス」→「JDBC リソース」→「タスクに移動」→「データソース」→「作成」→「アプリケーションの選択」→「データソース・タイプの選択」→「ネイティブ」

- ネイティブ・データソースは、接続プールに依存しません。そのため、ネイティブ・データソースの定義には、基礎となるデータベースと通信するのに必要なデータが含まれます。
- 1 つの `<native-data-source>` タグで、1 つのネイティブ・データソースを定義します。
- ネイティブ・データソースの設定は、5-23 ページの表 5-2「ネイティブ・データソースの設定」を参照してください。
- 1 つの `native-data-source` タグには、0 個以上の `property` タグを指定できます。各 `property` タグでは、ネイティブ・データソース・インスタンスのプロパティを定義します。ネイティブ・データソース・オブジェクトでは、プロパティ値の設定にリフレクションが使用されます。プロパティ名は、そのプロパティを設定するための setter メソッドの名前と一致する必要があります（大 / 小文字も区別されます）。たとえば、コネクション・ファクトリ・オブジェクトに `MyProp` というプロパティが存在する場合、そのプロパティを設定するために `setMyProp()` というメソッドがコールされます。したがって、プロパティを正しく設定するためには、property タグの名前が `MyProp` である必要があります。

```
<native-data-source
   name='My Native DataSource'
   jndi-name='jdbc/nativeDs'
   data-source-class='com.acme.DataSourceImpl'
   user='frank'
   password='frankpw'
   url='jdbc:acme:@localhost:5500:acme'>
   <property name="foo" value="bar" />
</native-data-source>
```

データソース構成の追加の例は、`http://www.oracle.com/technology/tech/java/newsletter/articles/oc4j_datasource_config.html` にある資料「Data Source Configuration in Oracle Application Server 10*g*」を参照してください。

表 5-2 ネイティブ・データソースの設定

Application Server Control コンソールのプロパティ	<native-data-source> の属性	説明
名前	`name`	**必須**。データソースの名前。この値は、一意である必要があります。
JNDI ロケーション	`jndi-name`	**必須**。データソース・オブジェクトの JNDI 論理名。OC4J では、データソースのインスタンスが、この値を持つアプリケーション JNDI ネームスペースにバインドされます。
データ・ソース・クラス	`data-source-class`	**必須**。データソース・クラス実装の名前とパス。このクラスは、`javax.sql.DataSource` を実装している必要があります。
URL	`url`	**必須**。データベースに接続するために JDBC ドライバにより使用される URL。URL では、通常、データベース・ホスト・マシン、ポートおよびデータベース名を指定します。たとえば、`jdbc:acme:@localhost:1234:acme` と指定します。

表 5-2 ネイティブ・データソースの設定（続き）

Application Server Control コンソールのプロパティ	<native-data-source> の属性	説明
ログイン・タイムアウト	login-timeout	データベースへの接続試行時にこのデータソースが待機する最大時間（秒単位）。値 0（ゼロ）を指定すると、デフォルトのシステム・タイムアウト時間が存在する場合はその時間となり、存在しない場合はタイムアウトなしとなります。 オプション設定です。デフォルトは、0 です。
ユーザー	user	データソースへの接続に使用するデフォルト・ユーザー。 オプション設定です。デフォルトはありません。
パスワード	password	データソースへの接続に使用するデフォルト・パスワード。 オプション設定です。デフォルトはありません。

接続プールとコネクション・ファクトリ

コネクション・ファクトリおよび接続プール設定へのパス
「OC4J: ホーム」→「管理」タブ→「タスク名 : サービス」→「JDBC リソース」→「タスクに移動」→「接続プール」→目的の設定にドリルダウン。

コネクション・ファクトリ

connection-factory タグでは、データソースの接続を作成するためのコネクション・ファクトリを定義します。

factory-class が javax.sql.XADataSource の実装である場合、コネクション・ファクトリから取得された接続は、グローバル・トランザクションに参加できます。この接続は、XA 機能をエミュレートしません。factory-class が javax.sql.XADatatSource の実装ではない場合、コネクション・ファクトリから取得された接続は、グローバル・トランザクションへの参加時に XA 動作をエミュレートします。

コネクション・ファクトリのプロパティ 1 つの <connection-factory> タグには、0 個以上の <property> タグを指定できます。各 <property> タグでは、コネクション・ファクトリ・インスタンスのプロパティを定義します。

コネクション・ファクトリが java.sql.Driver の実装である場合、これらの各ドライバ・プロパティは、データベースからの接続の取得時にドライバによって使用される java.util.Properties オブジェクトに配置されます。

コネクション・ファクトリが javax.sql.DataSource、javax.sql.ConnectionPoolDataSource または javax.sql.XADataSource の実装である場合、プロパティ値を設定するためにコネクション・ファクトリ・オブジェクトでリフレクションが使用されます。

プロパティ名は、そのプロパティを設定するための setter メソッドの名前と一致する必要があります（大 / 小文字も区別されます）。たとえば、コネクション・ファクトリ・オブジェクトに MyProp というプロパティが存在する場合、そのプロパティを設定するために setMyProp() というメソッドがコールされます。したがって、プロパティを正しく設定するためには、property タグの名前が MyProp である必要があります。

注意：

コネクション・ファクトリでは、次の例のように、属性とプロパティに 2 つの異なるユーザーとパスワードの組合せを指定できます。この例の場合、getConnection() では、属性に指定されたユーザーとパスワードではなく、プロパティに指定されたユーザーとパスワード（scott2 と tiger2）が使用されます。

```
<connection-factory user="scott1" password="tiger1" ...>
    <property name="user" value="scott2" />
    <property name="password" value="tiger2" />
</connection-factory>
```

コネクション・ファクトリのプロキシ・インタフェース 1 つの <connection-factory> タグには、0 個以上の <proxy-interface> タグを指定できます。

各プロキシ・インタフェースは、コネクション・ファクトリから戻される接続オブジェクトと、それらの接続オブジェクトにより作成される java.sql.* オブジェクトをラップするプロキシによって実装されます。

SQL オブジェクト設定で、プロキシ・インタフェースの定義対象となる java.sql.* オブジェクトを指定します。これは、次のいずれかである必要があります。

- Array
- Blob
- CallableStatement
- Connection
- DatabaseMetaData
- ParameterMetaData
- PreparedStatement
- Ref
- resultSet
- ResultSetMetaData
- Savepoint
- SQLData
- SQLInput
- SQLOutput
- Struct
- Statement

インタフェース属性で、このオブジェクトのプロキシが実装するインタフェースの完全修飾パスを定義します。

各 SQL オブジェクトに対して複数のプロキシ・インタフェースを定義できます。

<xa-recover-config> タグでは、グローバル・トランザクションの失敗時にトランザクション・マネージャでリカバリを実行するために必要な情報を定義します。<username> サブタグでは、リカバリの実行時に使用するユーザー名を定義します。<password> サブタグでは、リカバリの実行時に使用するパスワードを定義します。

接続プロパティ

<connection-properties> タグでは、コネクション・ファクトリが oracle.jdbc.pool.OracleDataSource のインスタンス（oracle.jdbc.pool.OracleDataSource から導出されるインスタンスを含む）である場合にコネクション・ファクトリに設定する接続プロパティを定義します。各接続プロパティは、<property> サブタグで定義します。<connection-properties> タグには、0 ～ N 個の <property> サブタグを定義できます。

接続プール

マネージド・データソースでは、接続を効率的に管理するため、接続プールを使用します。マネージド・データソースを作成する場合、少なくとも 1 つの接続プールと、そのコネクション・ファクトリを定義する必要があります。

<connection-pool> タグで、1 つの接続プールを定義します。

各 <connection-pool> タグには、<connection-factory> タグが 1 つ含まれる必要があります。

表 5-3 接続プールの属性

Application Server Control コンソールの設定	<connection-pool> の属性	説明
名前	name	**必須**。接続プールの名前。この値は、一意である必要があります。
最小接続数	min-connections	接続プールで保持する接続の最小数。 オプション設定です。デフォルトは、0 です。 min-connections 設定では、次のアクティビティが想定される場合に、任意の時点でプール内に保持する接続の最小数を指定します。 ■ 使用中の接続が 1 つも存在しない場合。使用中の接続がある場合、それらはプールに存在しないため、<min-connections> で指定した数の接続がプールに存在する可能性があります。 ■ すでに十分な数の接続が作成されており、それらの接続の作成に接続プールが必要とされるほど接続が同時に使用された場合。 たとえば、min-connections が 10 で、それまでに使用された接続が 2 つのみの場合、プール内で使用可能な接続の数は 2 です。OC4J では、不必要な接続は作成されません。
最大接続数	max-connections	接続プールで保持する接続の最大数。 値 0 を指定すると、次のようになります。 ■ データソースの接続プールは、無効になります。接続はプールされません。 ■ 他のすべての接続プール設定は、無視されます。 負の値を指定すると、接続プールは有効になり、最大数の制限がなくなります。 オプション設定です。デフォルトは、無制限です。 セッションの開始時に、max-connections の設定値が min-connections より小さいと、min-connections は max-connections の値に再設定されます。

表 5-3 接続プールの属性（続き）

Application Server Control コンソールの設定	<connection-pool> の属性	説明
接続キャッシュの初期サイズ	initial-limit	キャッシュが最初に作成される場合、またはキャッシュが再初期化される場合の接続キャッシュのサイズ。このプロパティに 0 より大きい値を設定すると、複数の接続が事前に作成されていつでも使用できる状態になります。このパラメータは、通常、最適なキャッシュ・サイズを準備するまでの時間を短縮するために使用します。 オプション設定です。デフォルトは、0 です。 接続プールの initial-limit が 1 より大きいが、connection-factory で user/password が指定されていない場合、OC4J は起動に失敗し、エラーをスローします。5-30 ページの表 5-4「コネクション・ファクトリの属性」の下にある「注意」を参照してください。 セッションの開始時に、initial-limit が max-connections より大きい場合（たとえば、initial-limit=10 に対して max-connections=5 など）、max-connections の数（5）の接続のみが初期化されます。 セッションの開始時に、initial-limit が min-connections より小さい場合（たとえば、initial-limit=10 に対して min-connections=15 など）、initial-limit の数（10）の接続のみが初期化されます。その後、さらに多くの接続がコールされた場合、min-connections の数の接続がプールで保持されます。
使用接続タイムアウトの待機時間	used-connection-wait-timeout	使用中の接続がクライアントから解放されるまで待機する時間（秒単位）。 このパラメータは、データソースから最大数の接続が取得されて使用中となっている場合にのみ適用されます。クライアントがプールから接続を取得しようとして、すべての接続が使用中の場合、接続プールでは、接続がプールに戻されるのを待機します。 オプション設定です。デフォルトは、0 です。 タイムアウト設定を適用するには、property-check-interval がタイムアウト設定未満である必要があります。
非アクティブのタイムアウト	inactivity-timeout	プールから削除する前に、使用されていない接続を非アクティブにして待機する時間（秒単位）。 オプション設定です。デフォルトは、60 です。 タイムアウト設定を適用するには、property-check-interval がタイムアウト設定未満である必要があります。
ログイン・タイムアウト	login-timeout	データベースへの接続試行時にこのデータソースが待機する最大時間（秒単位）。 値 0 を指定すると、デフォルトのシステム・タイムアウト時間が存在する場合はその時間となり、存在しない場合はタイムアウトなしとなります。 オプション設定です。デフォルトは、0 です。 タイムアウト設定を適用するには、property-check-interval がタイムアウト設定未満である必要があります。

表 5-3 接続プールの属性（続き）

Application Server Control コンソールの設定	<connection-pool> の属性	説明
接続再試行間隔	connection-retry-interval	失敗した接続試行を再実行するまでの待機間隔（秒単位）。 このパラメータは、max-connect-attempts と組み合せて使用します。 オプション設定です。デフォルトは、1 です。
最大接続試行回数	max-connect-attempts	接続の再試行回数。 このパラメータは、connection-retry-interval と組み合せて使用します。 オプション設定です。デフォルトは、3 です。
接続の妥当性チェック	validate-connection	Oracle Implicit Connection Cache 専用。 プールから取得した接続をデータベースに対して検証するかどうかを指定します。この検証は、validate-connection-statement パラメータの値で指定された SQL 文によって実行されます。 値 true を指定すると、接続が接続プールから取得されるときに、その接続の有効性を検証する SQL 文が実行されます。 オプション設定です。デフォルトは、false（検証なし）です。
検証用の SQL 文	validate-connection-statement	Oracle Implicit Connection Cache 専用。 validate-connection が true の場合、接続がプールから取得されるときにこの SQL 文が実行されます。 オプション設定です。デフォルトはありません。
キャッシュする文の最大数	num-cached-statements	各接続でキャッシュする SQL 文の最大数。0 より大きい値を指定すると、データソースの文キャッシュが自動的に有効化されます。 オプション設定です。デフォルトは、0 です。 詳細は、5-14 ページの「データソースでの JDBC 文のキャッシュ・サイズの設定」を参照してください。
使用接続の最大アクティブ時間	time-to-live-timeout	Oracle Implicit Connection Cache 専用。 使用中の接続をアクティブにする最大時間（秒単位）。 このタイムアウト時間を超えると、使用中の接続は無条件にクローズされ、関連する文ハンドルは取り消され、接続は接続プールに戻されます。 オプション設定です。デフォルトは、-1（無効）です。 タイムアウト設定を適用するには、property-check-interval がタイムアウト設定未満である必要があります。

表 5-3 接続プールの属性（続き）

Application Server Control コンソールの設定	<connection-pool> の属性	説明
中止接続タイムアウト	abandoned-connection-timeout	Oracle データベース専用。 プールから削除する前に、使用されていない論理接続を非アクティブにして待機する時間（秒単位）。 このパラメータは、inactivity-timeout に類似していますが、対象となるのはユーザーによってキャッシュから取得された論理接続です。設定すると、JDBC により、この論理接続の SQL データベース・アクティビティが監視されます。 たとえば、この接続で stmt.execute() が起動されると、この接続がアクティブであることを通知するハートビートが登録されます。ハートビートは、（監視コストを抑えるため）データベースの実行操作がコールされる場所でのみ監視されます。 接続が非アクティブの状態が指定した時間続くと、基礎となる PooledConnection は回収され、再利用のためにキャッシュに戻されます。 オプション設定です。デフォルトは、-1（無効）です。 タイムアウト設定を適用するには、property-check-interval がタイムアウト設定未満である必要があります。
サーバー接続プールの無効化（チェック・ボックス）	disable-server-connection-pooling	アプリケーション・サーバーの接続プールを無効にするかどうかを指定します。 このパラメータは、ドライバ内部に接続プールを保持する一部の JDBC ドライバ用に提供されています。 JDBC ドライバが Oracle であり、ドライバで Implicit Connection Cache を使用している場合、このパラメータは無視されます。 オプション設定です。デフォルトは、false（プール有効）です。
タイムアウト制限の強制間隔	property-check-interval	Oracle データベース専用。 常に Oracle データベースと組み合せて使用します。キャッシュ・デーモン・スレッドでタイムアウト制限を強制する時間間隔（秒単位）です。 オプション設定です。デフォルトは、900 です。 タイムアウト設定を適用するには、property-check-interval がタイムアウト設定未満である必要があります。
プールの下限しきい値	lower-threshold-limit	Oracle データベース専用。 常に Oracle データベースと組み合せて使用します。max-connections の値に対する割合で示される接続プールの下限しきい値です。 オプション設定です。デフォルトは、20 パーセントです。

表 5-4 に、コネクション・ファクトリの属性とその説明を示します。

表 5-4 コネクション・ファクトリの属性

Application Server Control コンソールの設定	<connection-factory> の属性	説明
コネクション・ファクトリ・クラス	factory-class	**必須**。データソースの接続の作成に使用するコネクション・ファクトリ・クラスの名前とパス。このクラスは、JDBC ドライバにより提供されます。たとえば、com.acme.AcmeDataSource と指定します。 このクラスは、java.sql.Driver、javax.sql.DataSource、javax.sql.ConnectionPoolDataSource または javax.sql.XADataSource を実装している必要があります。 factory-class が javax.sql.XADataSource の実装である場合、コネクション・ファクトリから取得された接続は、グローバル・トランザクションに参加できます。この接続は、XA 機能をエミュレートしません。factory-class が javax.sql.XADatatSource の実装ではない場合、コネクション・ファクトリから取得された接続は、グローバル・トランザクションへの参加時に XA 動作をエミュレートします。
URL	url	**必須**。データベースに接続するために JDBC ドライバにより使用される URL。URL では、通常、データベース・ホスト・マシン、ポートおよびデータベース名を指定します。たとえば、jdbc:acme:@localhost:1234:acme と指定します。
ユーザー	user	データベースへの接続に使用するデフォルト・ユーザー。 オプション設定です。デフォルトはありません。 connection-pool の initial-limit が 1 より大きいが、connection-factory で user/password が指定されていない場合、OC4J は起動に失敗し、エラーをスローします。この表の下にある「注意」を参照してください。
パスワード	password	データベースへの接続に使用するデフォルト・パスワード。 オプション設定です。デフォルトはありません。
ログイン・タイムアウト	login-timeout	データベースへの接続試行時にこのデータソースが待機する最大時間（秒単位）。 値 0 を指定すると、デフォルトのシステム・タイムアウト時間が存在する場合はその時間となり、存在しない場合はタイムアウトなしとなります。 オプション設定です。デフォルトは、0 です。

注意： connection-pool の initial-limit が 1 より大きいが、connection-factory で user/password が指定されていない場合、OC4J は起動に失敗し、エラーをスローします。

重大 : コネクタの初期化中にエラーが発生しました。例外 : データソースの接続プールの作成中にエラーが発生しました。例外 : oracle.oc4j.sql.DataSourceException: ConnectionCacheManager のインスタンスを取得 / 作成できませんでした。例外 : ユーザー資格証明が com.evermind.server.ApplicationStateRunning initConnector の既存の資格証明と一致しません。

このエラーは、プールの接続を初期化するためのユーザーとパスワードが存在しない場合に発生します。

ICC の無効化 oracle.jdbc.pool.OracleDataSource が data-sources.xml ファイルに connection-factory の factory-class として指定されている場合、OracleDataSource によって提供される ICC は OC4J により有効化されます。ICC が不要な場合には、connection-factory の connectionCachingEnabled プロパティを false に設定することで無効化できます。次に例を示します。

```
<connection-pool name="myConnectionPool"
   min-connections="10"
   max-connections="100"
   max-connect-attempts="5"
   connection-retry-interval="3">
   <connection-factory
      factory-class="oracle.jdbc.pool.OracleDataSource"
      user="scott"
      password="tiger"
      url="jdbc:oracle:thin:@//localhost:1521/
         oracle.regress.rdbms.dev.us.oracle.com">
      <property name="connectionCachingEnabled"  value="false"/>
   </connection-factory>
</connection-pool>
```

構成例

この項では、data-sources.xml 構成ファイルのデータソース定義の例を示します。

デフォルト・アプリケーションのデータソース構成ファイルは、$J2EE_HOME/config/data-sources.xml です。

各アプリケーションには、それぞれ独自の data-sources.xml ファイルを割り当てることができます。アプリケーションに独自ファイルを割り当てると、そのファイルは、アプリケーションのルート・ディレクトリに配置されます。

この項には、次の項目が含まれます。

data-sources.xml ファイルの構文

データソース設定は、エンタープライズ・アプリケーションの data-sources.xml ファイルで永続化されます。このファイルの各 <data-source> タグは、JNDI にバインドされていてクライアント・コンポーネント（サーブレットや EJB など）からアクセスできる 1 つのデータソースを示しています。

次の例は、data-sources.xml ファイルの構文を示しています。詳細は、スキーマを参照してください。

```
<managed-data-source
   attr1="val1"
   attr2="val2"
   … />

<native-data-source
   attr1="val1"
   attr2="val2"
   … >
   <property name="propertyName" value="propertyValue" />
   …
</native-data-source>

<connection-pool
   attr1="val1"
   attr2="val2"
   … >
   <connection-factory
      attr1="val1"
      attr2="val2"
      … >
      <proxy-interface sql-object="javaSQLObject" interface=""/>
      …
      <property name="propertyName" value="propertyValue"/>
      …
      <xa-recover-config>
         <password-credential>
            <username></username>
            <password></password>
         </password-credential>
      </xa-recovery-config>
      <fatal-error-codes>
         <error-code code="integerCode"/>
          …
      </fatal-error-codes>
      <connection-properties>
         <property name="propertyName value="propertyValue"/>
         …
      </connection-properties>
   </connection-factory>
</connection-pool
```

移入例

次に、data-sources.xml の定義の移入例を示します。

```
<?xml version="1.0" standalone="yes"?>
<data-sources>
   <connection-pool name="myConnectionPool" max-connections="30">
      <connection-factory
         factory-class="oracle.jdbc.pool.OracleDataSource"
         user="scott"
         password="tiger"
         url="jdbc:oracle:thin:@//localhost:1521/
            oracle.regress.rdbms.dev.us.oracle.com" />
   </connection-pool>

   <managed-data-source
      jndi-name="jdbc/ManagedDS"
      name="Managed DataSource Name"
      connection-pool-name="myConnectionPool" />

   <native-data-source
      name="nativeDataSource"
      jndi-name="jdbc/nativeDS"
      data-source-class="com.acme.DataSourceImpl"
      user="frank"
      password="frankpw"
      url="jdbc:acme:@localhost:5500:acme" />
</data-sources>
```

例 : データソースの構成

この項では、データソースの構成例を示します。

データソース構成の追加の例は、http://www.oracle.com/technology/tech/java/newsletter/articles/oc4j_datasource_config.html にある資料「Data Source Configuration in Oracle Application Server 10*g*」を参照してください。

例 : ネイティブ・データソース

```
<native-data-source
   name='My Native DataSource'
   jndi-name='jdbc/nativeDs'
   data-source-class='com.acme.DataSourceImpl'
   user='frank'
   password='frankpw'
   url='jdbc:acme:@localhost:5500:acme'>
   <property name="foo" value="bar"/>
</native-data-source>
```

例 : XADataSource コネクション・ファクトリを使用したマネージド・データソース

このデータソースは、XA 動作をエミュレートしません。XA 動作のエミュレートの詳細は、5-19 ページの「XA のエミュレート」を参照してください。

```
<managed-data-source
   name='My Managed DataSource'
   jndi-name='jdbc/managedDs_1'
   connection-pool-name='myConnectionPool'/>

<connection-pool
   name='myConnectionPool'
   min-connections='5'
   max-connections='25'>
   <connection-factory
      factory-class='oracle.jdbc.xa.client.OracleXADataSource'
      user='scott'
      password='tiger'
      url='jdbc:oracle:thin:@//localhost:1521/
         oracle.regress.rdbms.dev.us.oracle.com' />
</connection-pool>
```

例 : DataSource コネクション・ファクトリを使用したマネージド・データソース

このデータソースは、XA 動作をエミュレートします。XA 動作のエミュレートの詳細は、5-19 ページの「XA のエミュレート」を参照してください。

```
<managed-data-source
   name='My Managed DataSource'
   jndi-name='jdbc/managedDs_1'
   connection-pool-name='myConnectionPool'/>

<connection-pool
   name='myConnectionPool'
   min-connections='5'
   max-connections='25'>
   <connection-factory
      factory-class='oracle.jdbc.pool.OracleDataSource'
      user='scott'
      password='tiger'
      url='jdbc:oracle:thin:@//localhost:1521/
         Oracle.regress.rdbms.dev.us.oracle.com' />
</connection-pool>
```

例 : Driver コネクション・ファクトリを使用したマネージド・データソース

このデータソースは、XA 動作をエミュレートします。XA 動作のエミュレートの詳細は、5-19 ページの「XA のエミュレート」を参照してください。

```
<managed-data-source
   name='My Managed DataSource'
   jndi-name='jdbc/managedDs_1'
   connection-pool-name='myConnectionPool'/>

<connection-pool
   name='myConnectionPool'
   min-connections='5'
   max-connections='25'>
   <connection-factory
      factory-class='oracle.jdbc.OracleDriver'
      user='scott'
      password='tiger'
      url='jdbc:oracle:thin:@//localhost:1521/
         oracle.regress.rdbms.dev.us.oracle.com' />
</connection-pool>
```

例 : プロキシ・インタフェースの定義

```
<connection-pool
   name='myConnectionPool'
   min-connections='5'
   max-connections='25'>
   <connection-factory
      factory-class='oracle.jdbc.pool.OracleDataSource'
      user='scott'
      password='tiger'
      url='jdbc:oracle:thin:@//localhost:1521/
         oracle.regress.rdbms.dev.us.oracle.com'>
      <proxy-interface sql-object="Connection"
         interface="oracle.jdbc.internal.OracleConnection"/>
      <proxy-interface sql-object="Statement"
         interface="oracle.jdbc.OracleStatement"/>
      <proxy-interface sql-object="CallableStatement"
         interface="oracle.jdbc.OracleCallableStatement"/>
      <proxy-interface sql-object="ResultSet"
         interface="oracle.jdbc.OracleResultSet"/>
      <proxy-interface sql-object="PreparedStatement"
         interface="oracle.jdbc.OraclePreparedStatement"/>
    </connection-factory>
</connection-pool>
```

例 : XA リカバリの定義

```
<connection-pool
   name='myConnectionPool'
   min-connections='5'
   max-connections='25'>
   <connection-factory
      factory-class='oracle.jdbc.xa.client.OracleXADataSource'
      user='scott'
      password='tiger'
      url='jdbc:oracle:thin:@//localhost:1521/
         oracle.regress.rdbms.dev.us.oracle.com'>
      <xa-recovery-config>
          <password-credential>
              <username>system</username>
              <password>manager</password>
          </password-credential>
      </xa-recovery-config>
   </connection-factory>
</connection-pool>
```

例 : 接続プロパティ

```
<managed-data-source
   jndi-name="jdbc/managedDs_1"
   name="Managed DataSource"
   connection-pool-name="myConnectionPool" />

<connection-pool
   name="myConnectionPool">
   <connection-factory
      factory-class="oracle.jdbc.pool.OracleDataSource"
      user="scott"
      password="tiger"
      url='jdbc:oracle:thin:@//localhost:1521/
         oracle.regress.rdbms.dev.us.oracle.com'>
      <connection-properties>
         <property name="oracle.jdbc.RetainV9LongBindBehavior"
            value="true"/>
      </connection-properties>
   </connection-factory>
</connection-pool>
```

接続プロパティの詳細は、5-26 ページの「接続プロパティ」を参照してください。

例 : トランザクション・レベルの構成

グローバル

次の例では、data-sources.xml ファイルの tx-level 属性を global に設定して、マネージド・データソースをグローバル・トランザクション用に構成する方法を示します。

```
<managed-data-source
   jndi-name="jdbc/ManagedDS"
   name="Managed DataSource Name"
   connection-pool-name="myConnectionPool"
   tx-level="global" />
```

ローカル

次の例では、data-sources.xml ファイルの tx-level 属性を local に設定して、マネージド・データソースをローカル・トランザクション用に構成する方法を示します。デフォルト値は、global です。

```
<managed-data-source
   jndi-name="jdbc/ManagedDS"
   name="Managed DataSource Name"
   connection-pool-name="myConnectionPool"
   tx-level="local" />
```

例 : Fast Connection Failover の構成

次に、Fast Connection Failover のコネクション・ファクトリを構成する例を示します。

シン

```
<connection-factory
   factory-class="oracle.jdbc.pool.OracleDataSource"
   user="scott"
   password="tiger"
   url="jdbc:oracle:thin:@(DESCRIPTION=
 (ADDRESS=(PROTOCOL=TCP)(HOST=cluster_alias) (PORT=1521))
 (CONNECT_DATA=(SERVICE_NAME=service_name)))"
          <property name="connectionCachingEnabled" value="true"/>
          <property name="fastConnectionFailoverEnabled" value="true" />
</connection-factory>
```

OCI

次に、OCI を使用するコネクション・ファクトリ定義の例を示します。

```
<connection-factory
   factory-class="oracle.jdbc.pool.OracleDataSource"
   user="scott"
   password="tiger"
   url="jdbc:oracle:oci:@myAlias" />
```

高可用性および Fast Connection Failover の使用方法

OC4J のデータソースは、Oracle10*g* の JDBC ドライバと完全に統合されているため、自動的に高可用性（HA）および Fast Connection Failover（FCF）機能を使用できます。

高可用性と Fast Connection Failover の詳細は、次のドキュメントを参照してください。

- 『Oracle Database 高可用性概要』
- 『Oracle Database JDBC 開発者ガイドおよびリファレンス』の第 VI 部「高可用性」
- 追加情報 : `http://www.oracle.com/technology/tech/java/oc4j/index.html`

Fast Connection Failover は、JDBC Implicit Connection Cache に実装された RAC/FaN クライアントです。この機能の主な目的は、接続の有効性と可用性を保証することです。クライアント・サイドの Fast Connection Failover により、次の機能が提供されます。

- Implicit Connection Cache 内の無効接続の高速検出（DCD）
- 失効または無効接続のキャッシュからの削除
- 上位レイヤーでの再試行を容易にする、コール元へのエラーの伝播
- 新規 RAC インスタンス追加時の接続の再配布

Fast Connection Failover メカニズムを有効化するには、`OracleDataSource` オブジェクトの `<connection-factory>` タグで次のプロパティおよび属性を設定する必要があります。

表 5-5 Fast Connection Failover の設定

設定	説明
`connectionCachingEnabled`	これは、ブール型プロパティであり、`true` に設定すると接続キャッシュが有効化されます。デフォルトでは、接続キャッシュは無効であり、プロパティ値は `false` に設定されています。
`fastConnectionFailoverEnabled`	このプロパティを `true` に設定すると、Fast Connection Failover メカニズムが有効化されます。デフォルトでは、Fast Connection Failover は無効であり、プロパティ値は `false` に設定されています。
`url`	これは、<connection-factory> タグの属性です。Fast Connection Failover を有効化する場合、サービス名の構文を使用して URL を設定する必要があります。接続 URL に指定されたサービス名は、接続キャッシュをサービスにマップするために使用されます。Fast Connection Failover が有効な場合に SID を URL に指定すると、例外がスローされます。

次の例では、Fast Connection Failover 用の接続キャッシュ設定における URL の用例として、有効な構文と無効な構文を示します。

有効な URL の用例

```
url="jdbc:oracle:oci:@TNS_ALIAS"

url="jdbc:oracle:oci:@(DESCRIPTION=
   (LOAD_BALANCE=on)
   (ADDRESS=(PROTOCOL=TCP)(HOST=host1) (PORT=1521))
   (ADDRESS=(PROTOCOL=TCP)(HOST=host2)(PORT=1521))
   (CONNECT_DATA=(SERVICE_NAME=service_name)))"

url="jdbc:oracle:oci:@(DESCRIPTION=
   (ADDRESS=(PROTOCOL=TCP)(HOST=cluster_alias) (PORT=1521))
   (CONNECT_DATA=(SERVICE_NAME=service_name)))"
```

```
url = "jdbc:oracle:thin@//host:port/service_name"

url = "jdbc:oracle:thin@//cluster-alias:port/service_name"

url="jdbc:oracle:thin:@(DESCRIPTION=
   (LOAD_BALANCE=on)
   (ADDRESS=(PROTOCOL=TCP)(HOST=host1) (PORT=1521))
   (ADDRESS=(PROTOCOL=TCP)(HOST=host2)(PORT=1521))
   (CONNECT_DATA=(SERVICE_NAME=service_name)))"

url = "jdbc:oracle:thin:@(DESCRIPTION=
   (ADDRESS=(PROTOCOL=TCP)(HOST=cluster_alias) (PORT=1521))
   (CONNECT_DATA=(SERVICE_NAME=service_name)))"
```

無効な URL の用例

```
url = "jdbc:oracle:thin@host:port:SID"
```

data-sources.xml ファイルでの Fast Connection Failover の有効化

次に、`data-sources.xml` ファイルでネイティブ・データソース用に Fast Connection Failover を有効化する例を示します。

```
<native-data-source
   name="nativeDataSource"
   jndi-name="jdbc/nativeDS"
   description="Native DataSource"
   data-source-class="oracle.jdbc.pool.OracleDataSource"
   user="scott"
   password="tiger"
   url="jdbc:oracle:thin:@(DESCRIPTION=
   (LOAD_BALANCE=on)
   (ADDRESS=(PROTOCOL=TCP)(HOST=host1) (PORT=1521))
   (ADDRESS=(PROTOCOL=TCP)(HOST=host2)(PORT=1521))
   (CONNECT_DATA=(SERVICE_NAME=service_name)))">
     <property name="connectionCacheName" value="ICC1"/>
     <property name="connectionCachingEnabled" value="true"/>
     <property name="fastConnectionFailoverEnabled" value="true"/>
</native-data-source>
```

JDBC ドライバの使用方法

この項には、次の項目が含まれます。

Oracle JDBC ドライバ

この項では、Oracle JDBC OCI ドライバと Oracle JDBC シン・ドライバについて説明します。

Oracle JDBC ドライバの詳細は、『Oracle Database JDBC 開発者ガイドおよびリファレンス』を参照してください。

OCI

この章の Oracle データソース定義の例では、Oracle JDBC シン・ドライバが使用されています。ただし、Oracle JDBC OCI ドライバを使用することもできます。OC4J サーバーを起動する前に、次の作業を実行します。

1. OC4J がインストールされているシステムに Oracle Client をインストールします。
2. `OLE_HOME` 変数を設定します。
3. `LD_LIBRARY_PATH`（または使用中のオペレーティング・システムの対応する環境変数）に `$OLE_HOME/lib` を設定します。
4. `TNS_ADMIN` に、適切な `tnsnames.ora` ファイルを含む有効な Oracle 管理ディレクトリを設定します。

<connection-factory> 要素定義の url 属性で使用する URL は、次のいずれかの形式です。

- `jdbc:oracle:oci:@` - この TNS エントリは、クライアントと同じシステム上にあるデータベース用で、クライアントは IPC モードでデータベースに接続します。
- `jdbc:oracle:oci:@TNS_service_name` - TNS サービス名は、インスタンスの tnsnames.ora ファイルのエントリです。
- `jdbc:oracle:oci:@full_TNS_listener_description` - TNS の詳細は、『Oracle Database Net Services 管理者ガイド』を参照してください。

次の例では、OCI を使用するコネクション・ファクトリの定義を示します。

```
<connection-factory
    factory-class="oracle.jdbc.pool.OracleDataSource"
    user="scott"
    password="tiger"
    url="jdbc:oracle:oci:@myAlias" />
```

シン

この章の Oracle データソース定義の例では、Oracle JDBC シン・ドライバが使用されています。

Oracle Application Server での Oracle JDBC-OCI ドライバのアップグレードに関する注意事項

クライアント・ライブラリの互換性に制限があることから、任意のバージョンの Oracle JDBC-OCI ドライバにアップグレードすることはできません。サポートされるのは、Oracle Application Server 10*g* リリース 2（10.1.2）内にインストールされている Oracle Client ライブラリと一致する OCI ドライバ・バージョンへのアップグレードです。たとえば、Oracle JDBC 10.1.x ドライバはサポートされますが、Oracle JDBC 9.2.x ドライバはサポートされません。

Oracle Application Server で JDBC-OCI の使用をサポートする場合、ORACLE_HOME およびライブラリ・パスの適切な値を起動環境に伝播するため、各 OC4J インスタンス用の `opmn.xml` エントリも必要です。

環境変数 `ORACLE_HOME` は、すべてのプラットフォームに共通ですが、ライブラリ・パスを指定する環境変数の名前は、次のようにオペレーティング・システムごとに異なります。

- `LD_LIBRARY_PATH`（Solaris）
- `SLIB_PATH`（AIX）
- `SHLIB_PATH`（HP-UX）
- `PATH`（Windows）

`opmn.xml` でライブラリ・パスを指定する一般的な構文は、次のとおりです。

```
<prop name="<LIB_PATH_VARIABLE>" value="<LIB_PATH_VARIABLE_VALUE>"/>
```

ここで、`<LIB_PATH_VARIABLE>` は、ライブラリ・パスを指定するプラットフォーム固有の適切な変数名で置き換えます。また、`<LIB_PATH_VARIABLE_VALUE>` は、その変数の値で置き換えます。

次の例では、Solaris オペレーティング・システムの場合を示します。

```
<process-type id="OC4J_SECURITY" module-id="OC4J">
   <environment>
      <variable id="ORACLE_HOME" value="/u01/app/oracle/product/inf10120"/>
      <variable id="LD_LIBRARY_PATH"
         value="/u01/app/oracle/product/inf10120/lib"/>
   </environment>
```

Oracle 以外のデータベースの JDBC ドライバ

異種データベースに接続する必要のあるアプリケーションでは、DataDirect JDBC ドライバを使用できます。DataDirect JDBC ドライバは、Oracle データベース用ではありませんが、DB2、Sybase、Informix、SQLServer などの Oracle 以外のデータベースに接続する場合に使用できます。

DataDirect JDBC ドライバの使用方法

『DataDirect Connect for JDBC User's Guide and Reference』の説明に従って DataDirect JDBC ドライバをインストールします。

DataDirect JDBC ドライバを使用するには、次のようにします。

1. DataDirect JDBC ドライバの内容をディレクトリに解凍します。このディレクトリは、*`DDJD_INSTALL`* と呼ばれます。
2. Oracle Enterprise Manager 10*g* コンソールで共有ライブラリを作成（**「管理」→「共有ライブラリ」**）し、*`DDJD_INSTALL`*`/lib` にある DataDirect JDBC ライブラリを共有ライブラリにアップロードします。DataDirect ドライバを使用するには、`YMbase.jar` および `YMutil.jar` ライブラリが必要です。
3. 「DataDirect のデータソース・エントリの例」で説明されているように、データベースのデータソース定義を `ORACLE_HOME/config/data-sources.xml` ファイルに追加します。

4. 手順 2 で作成した共有ライブラリをインポートしたことを確認し、Oracle Enterprise Manager 10*g* コンソールから、アプリケーションをデプロイ（**「アプリケーション」→「デプロイ」**）します。

注意： 共有ライブラリ作成の詳細は、『Oracle Containers for J2EE 開発者ガイド』の第 3 章を参照してください。

DataDirect のデータソース・エントリの例

この項では、次の Oracle 以外の各データベースについてデータソース・エントリの例を示します。

- DataDirect DB2
- DataDirect Sybase
- DataDirect Informix
- DataDirect SQLServer

ベンダー固有のデータソースをクラス属性で直接使用することもできます。つまり、クラス属性で OC4J 固有のデータソースを使用する必要はありません。

詳細は、『DataDirect Connect for JDBC User's Guide and Reference』を参照してください。

データソース構成の追加の例は、`http://www.oracle.com/technology/tech/java/newsletter/articles/oc4j_datasource_config.html` にある資料「Data Source Configuration in Oracle Application Server 10*g*」を参照してください。

DataDirect DB2

注意： DataDirect JDBC ドライバを使用して DB2 に接続する場合、次の制限事項があります。

- 次の項目は、例外的な動作です。

  `com.oracle.ias.jdbcx.db2.DB2DataSource` をコネクション・ファクトリ・クラスとして使用する場合、`<url>` は必須要素ですが、url 属性で渡される値は無視されます。このため、url のみを設定すれば十分です。

  データソースが正しく動作するためには、`<property>` 要素に `serverName`、`portNumber` および `databaseName` を設定する必要があります。

- `com.oracle.ias.jdbc.db2.DB2Driver` をコネクション・ファクトリ・クラスとして使用する場合、<property> 要素に `serverName`、`portNumber` および `databaseName` を設定する必要はありません。`serverName`、`portNumber` および `databaseName` は、url 属性に設定できます。url は完全に解釈され、url 属性で渡される値も読み取られます。

```
<managed-data-source
   name="db2"
   jndi-name="jdbc/db2"
   connection-pool-name="db2 Connection Pool"/>

<connection-pool name="db2 Connection Pool">
   <connection-factory
      factory-class="com.oracle.ias.jdbcx.db2.DB2DataSource"
      user="user1"
      password="user1"
      url="jdbc:oracle:db2://localhost:50000;
```

```
        DatabaseName=sample;PackageName=JDBCPKG">
        <property name="databaseName" value="sample"/>
        <property name="packageName" value="JDBCPKG"/>
        <property name="serverName" value="localhost"/>
        <property name="portNumber" value="50000"/>
        <xa-recovery-config>
           <password-credential>
              <username>system</username>
              <password>manager</password>
           </password-credential>
        </xa-recovery-config>
    </connection-factory>
</connection-pool>
```

DataDirect Sybase

```
<managed-data-source
   name="Sybase"
   jndi-name="jdbc/Sybase"
   connection-pool-name="Sybase Connection Pool"/>

<connection-pool name=" Sybase Connection Pool">
   <connection-factory
      factory-class="com.oracle.ias.jdbcx.sybase.SybaseDataSource"
      user="user1"
      password="password"
      url="jdbc:oracle:sybase://localhost:4101">
         <property name="serverName" value="localhost"/>
         <property name="portNumber" value="4101"/>
         <xa-recovery-config>
            <password-credential>
               <username>system</username>
               <password>manager</password>
            </password-credential>
         </xa-recovery-config>
   </connection-factory>
</connection-pool>
```

DataDirect Informix

```
<managed-data-source
   name="Informix"
   jndi-name="jdbc/Informix"
   connection-pool-name="Informix Connection Pool"/>

<connection-pool name=" Informix Connection Pool">
   <connection-factory
      factory-class="com.oracle.ias.jdbc.informix.InformixDriver"
      user="user1"
      password="password"
      url="jdbc:oracle:informix://localhost:3900;
         informixServer=gtw93;DatabaseName=gatewaydb">
         <xa-recovery-config>
            <password-credential>
               <username>userid</username>
               <password>pword</password>
            </password-credential>
         </xa-recovery-config>
   </connection-factory>
</connection-pool>
```

DataDirect SQLServer

この項では、SQLServer のマネージド・データソースの例とネイティブ・データソースの例を示します。

SQLServer のマネージド・データソース

```
<connection-pool name="ConnectionSqlserver" max-connections="20"
   min-connections="1">
   <connection-factory
      factory-class="com.oracle.ias.jdbcx.sqlserver.SQLServerDataSource"
      user="msuser"
      password="mspass"
      url="jdbc:oracle:sqlserver://myserver\
         myinstance;User=msuser;Password=mspass">
         <property name="serverName" value="myserver\myinstance" />
         <xa-recovery-config>
            <password-credential>
               <username>msuser</username>
               <password>mspass</password>
            </password-credential>
         </xa-recovery-config>
   </connection-factory>
</connection-pool>

<managed-data-source connection-pool-name="ConnectionSqlserver"
   jndi-name="jdbc/mysqlserver"
   name="mysqlserver" />
```

SQLServer のネイティブ・データソース

```
<native-data-source
   jndi-name="jdbc/mysqlserver"
   name="mysqlserver"
   data-source-class="com.oracle.ias.jdbcx.sqlserver.SQLServerDataSource"
   user="msuser"
   password="mspass"
   url="jdbc:oracle:sqlserver://myserver\
      myinstance;User=msuser;Password=mspass" >
      <property name="serverName" value="myserver\myinstance" />
</native-data-source>
```

追加のデータソース構成例

次の追加のデータソース構成例では、データソース・タイプ、コネクション・ファクトリ・タイプ、および他の変数の様々な組合せを示します。これらの例は、`http://www.oracle.com/technology/tech/java/newsletter/articles/oc4j_datasource_config.html` にある資料「Data Source Configuration in Oracle Application Server 10*g*」からの抜粋です。

ネイティブ・データソース : Oracle JDBC から Oracle データベースへ

```
<native-data-source
   name="nativeDataSource"
   jndi-name="jdbc/nativeDS"
   description="Native DataSource"
   data-source-class="oracle.jdbc.pool.OracleDataSource"
   user="scott"
   password="tiger"
   url="jdbc:oracle:thin:@//dbhost:1521/dbservicename">
</native-data-source>
```

ネイティブ・データソース : DataDirect JDBC から DB2 UDB へ

```
<native-data-source
   name="nativeDataSource"
   jndi-name="jdbc/nativeDS"
   description="Native DataSource"
   data-source-class="com.ddtek.jdbcx.db2.DB2DataSource"
   user="frank"
   password="frankpw"
   url="jdbc:datadirect:db2://server_name:50000;DatabaseName=your_database">
</native-data-source>
```

ネイティブ・データソース : DB2 Universal JDBC から DB2 UDB へ

```
<native-data-source
   name="nativeDataSource"
   jndi-name="jdbc/nativeDS"
   description="Native DataSource"
   data-source-class="com.ibm.db2.jcc.DB2DataSource"
   user="db2adm"
   password="db2admpwd"
   url="jdbc:db2://sysmvs1.stl.ibm.com:5021/
      dbname:user=db2adm;password=db2admpwd;" />
```

マネージド・データソース : XADataSource コネクション・ファクトリを使用

```
<managed-data-source
   jndi-name="jdbc/ManagedDS"
   description="Managed DataSource"
   connection-pool-name="myConnectionPool"/>

<connection-pool
   name="myConnectionPool"
   min-connections="10"
   max-connections="30"
   inactivity-timeout="30">
      <connection-factory
         factory-class="oracle.jdbc.xa.client.OracleXADataSource"
         user="scott"
         password="tiger"
         url="jdbc:oracle:thin:@//dbhost:1521/dbservicename" />
</connection-pool>
```

マネージド・データソース : DataSource コネクション・ファクトリを使用

```
<managed-data-source
   jndi-name="jdbc/ManagedDS"
   description="Managed DataSource">
   connection-pool-name="myConnectionPool"/>

<connection-pool
   name="myConnectionPool"
   min-connections="10"
   max-connections="30"
   inactivity-timeout="30">
      <connection-factory
         factory-class="oracle.jdbc.pool.OracleDataSource"
         user="scott"
         password="tiger"
         url="jdbc:oracle:thin:@//dbhost:1521/dbservicename" />
</connection-pool>
```

マネージド・データソース : Driver コネクション・ファクトリを使用

```
<managed-data-source
   jndi-name="jdbc/ManagedDS"
   description="Managed DataSource">
   connection-pool-name="myConnectionPool"/>

<connection-pool
   name="myConnectionPool"
   min-connections="10"
   max-connections="30"
   inactivity-timeout="30">
   <connection-factory
      factory-class="oracle.jdbc.OracleDriver"
      user="scott"
      password="tiger"
      url="jdbc:oracle:thin:@//dbhost:1521/dbservicename" />
</connection-pool>
```

レガシー構成

レガシー・データソース構成を使用する際には、次の事項を考慮する必要があります。

- `data-sources.xml` には、次の 2 つの構文があります。
  - 現行の構文は、`http://xmlns.oracle.com/oracleas/schema/data-sources-10_1.xsd` スキーマに準拠します。
  - レガシー構文は、`http://xmlns.oracle.com/ias/dtds/data-sources-9_04.dtd` スキーマに準拠します。

  どちらの構文も有効ですが、新規構文を使用することをお薦めします。
- Application Server Control コンソールで追加した変更内容が `data-sources.xml` に永続化される場合、その構文は、常に新規形式で記述されます。
- `data-sources.xml` ファイルの内容は、`admin.jar` ユーティリティを使用することで、レガシーの旧構文から新規構文に明示的に変換できます。`-convertDataSourceConfiguration` オプションを使用します。詳細は、5-46 ページの次項「既存のデータソースの変換」で説明されています。

既存のデータソースの変換

OC4J では、`data-sources.xml` ファイルのレガシー形式を解釈できます。既存のデータソースの変更やデータソースの作成または削除など、`data-sources.xml` ファイルの変更に Application Server Control コンソールが使用されている場合、古い形式の `data-sources.xml` ファイルが含まれるアプリケーションでは、OC4J によりその `data-sources.xml` ファイルが自動的に最新の形式に変換されます。

スタンドアロン環境でアクティブな OC4J インスタンスを使用すると、次の構文で `admin.jar` を使用し、レガシーの `data-sources.xml` ファイルを現行の形式に手動で変換することもできます。詳細は、『Oracle Containers for J2EE 構成および管理ガイド』を参照してください。このマニュアルでは、ORMI URL を指定できるのは OC4J の稼働中のみであること、または ORMI URL がオプションであることは説明されていません。

```
java -jar admin.jar ormi://oc4jHost:oc4jOrmiPort adminId adminPassword
-convertDataSourceConfiguration old-data-sources.xml new-data-sources.xml
```

OC4J インスタンスが稼働していなくても、デプロイ前に `data-sources.xml` ファイルを変換することも可能です。このオフライン変換の構文は次のとおりです。

```
java -jar admin.jar -convertDataSourceConfiguration old-data-sources.xml
new-data-sources.xml
```

注意：

- ORMI ポートを指定する場合は、OC4J が稼働している必要があります。OC4J が稼働していない場合は、admin.jar コマンドラインから ORMI URL を除外する必要があります。
- ORMI ポートを指定しない場合には、OC4J が稼働しているかどうかにかかわらず、admin.jar コマンドを使用できます。
- admin.jar ユーティリティを使用できるのは、スタンドアロン OC4J 環境のみです。このユーティリティは Oracle Application Server 環境にインストールされますが、OPMN 管理環境では使用できません。
- 新しい admin_client.jar ユーティリティは、スタンドアロン環境でも管理対象の Oracle Application Server 環境でも使用できます。ただし、admin_client.jar ユーティリティでは、data-sources.xml ファイルは変換されません。

アプリケーションと新しい data-sources.xml ファイル間の整合性の確認

手動または自動での変換後、新しい `data-sources.xml` ファイルを開いて調査し、アプリケーションと新しいファイルとの間に、データソースの参照に使用される JNDI ロケーションの整合性があることを確認します。新しいファイルには使用されないデータソース定義が含まれている場合があるので、この確認を行うことをお薦めします。

これは、古い形式で複数のロケーション属性（`location`、`ejb-location`、`xa-location` など）が使用されていることが原因です。新しい形式への変換により、古い `data-sources.xml` ファイルに指定されているそれぞれのロケーション属性に対応する別のデータソースが、新しい `data-sources.xml` ファイルに作成されます。多くの場合、クライアント・アプリケーションでは、`location` または `ejb-location` 属性で定義されたデータソースのみが使用されます。ただし、これは保証されているわけではありません。そのため、変換された `data-sources.xml` にはアプリケーションで使用されない定義が含まれますが、ファイルから削除できます。

6

Remote Method Invocation の使用方法

この章には、次の項目が含まれます。

RMI とは

Java Remote Method Invocation（RMI）を使用すると、分散 Java ベース間アプリケーションを作成できます。この種のアプリケーションでは、リモート Java オブジェクトのメソッドを、異なるホスト上にあるものも含め、他の Java 仮想マシン（JVM）から起動できます。

OC4J は、独自仕様の Oracle Remote Method Invocation（ORMI）プロトコルを介した RMI と、標準の Internet Inter-ORB Protocol（IIOP）を介した RMI の両方をサポートします。RMI/ORMI を使用すると、OC4J インスタンス内で稼働するオブジェクトを相互起動できます。RMI/IIOP を使用すると、異なる J2EE コンテナ間（OC4J と BEA WebLogic サーバー間など）でオブジェクトを相互起動できます。

追加ドキュメント

- 次のサイトにある How-To ドキュメント「How-To Propagate a transaction context between OC4J instances」と ZIP ファイルには、ORMI を使用する OC4J インスタンス間でのトランザクション・コンテキストの伝播に関する情報および使用例が含まれます。

  ```
  http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/
  how-to-transaction-propagation/doc/how-to-transaction-
  propagation.html
  ```

- 次のサイトにある How-To ドキュメントでは、OC4J の機能に関する追加情報（各機能の概要や機能に関連するコードの抜粋など）を提供しています。

  ```
  http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/
  index.html
  ```

RMI/ORMI または RMI/IIOP の選択

RMI/ORMI と RMI/IIOP の選択基準は、次のとおりです。

- OC4J インスタンス内または OC4J インスタンス間でリモート・オブジェクトのメソッドを起動するには、RMI/ORMI を使用します。

  詳細は、6-2 ページの「Oracle Remote Method Invocation（RMI/ORMI）の使用方法」を参照してください。

- OC4J と Oracle 以外の J2EE コンテナ（BEA WebLogic など）間でリモート・オブジェクトのメソッドを起動するには、RMI/IIOP を使用します。

  詳細は、6-20 ページの「ORMI から IIOP トランスポートへの切替え」を参照してください。

Oracle Remote Method Invocation（RMI/ORMI）の使用方法

この項では、独自仕様の Oracle RMI（ORMI）プロトコルを介した Remote Method Invocation（RMI）を使用して、Oracle Containers for J2EE（OC4J）のサーバー・インスタンス間で EJB などのオブジェクトを相互起動できるようにする OC4J のサポート機能について説明します。

この項には、次の項目が含まれます。

- RMI/ORMI の概要
- スタンドアロン OC4J インストール環境での RMI の構成
- Oracle Application Server 環境での RMI の構成
- RMI/ORMI 使用時のクライアント・サイドの要件

RMI/ORMI の概要

Oracle Remote Method Invocation（ORMI）は、OC4J 用に最適化された、RMI プロトコルの Oracle 固有の実装です。

デフォルトでは、OC4J サーバー・インスタンス内の EJB は、ORMI を介して RMI コールを交換します。かわりに、IIOP を介して RMI コールを交換するよう EJB を変換して、異なるベンダーの EJB コンテナ間（OC4J と BEA WebLogic など）で EJB を相互起動することも可能です。詳細は、6-20 ページの「ORMI から IIOP トランスポートへの切替え」を参照してください。

注意： OC4J 10*g* リリース 3（10.1.3.1.0）の実装では、ロード・バランシングとフェイルオーバーがサポートされるのは ORMI の場合のみで、IIOP の場合はサポートされません。

ORMI の機能

ORMI は OC4J 向けに拡張されており、次の機能が用意されています。

- RMI メッセージ・スループットの増大
- スレッド化のサポートの拡張
- 同じ場所に配置されているオブジェクトのサポート

RMI メッセージ・スループットの増大 ORMI を使用すると、OC4J は高トランザクション・レートで処理できます。これは、Oracle の SpecJ Application Server ベンチマークに反映されています。`http://www.spec.org/` を参照してください。

一方向 ORMI は、IIOP メッセージよりも小さいメッセージを使用して、このパフォーマンスを達成します。メッセージが小さいため、送受信用の帯域幅が小さく、エンコードとデコードの処理時間が短縮されます。

ORMI のメッセージ・サイズは、クライアントとサーバーの間でやり取りされる状態情報の量を最適化することで、さらに小さくなります。ORMI を使用すると、一部の状態はサーバー上でキャッシュされるため、RMI コールのたびに送信する必要がありません。フェイルオーバー時には、クライアント・コードにより新規サーバーに必要な状態情報がすべて再送されるため、これはステートレスであるという RMI 要件に違反しません。

スレッド化のサポートの拡張 ORMI は OC4J のスレッド化モデルと密結合され、そのキューイング、プーリングおよびステージング機能を最大限に利用します。

ORMI はクライアントごとに 1 つずつスレッドを使用します。マルチスレッド化されたクライアントの場合、OC4J は 1 つの接続を介して各コールを多重化します。ただし、OC4J は各コールをシリアライズしないため、複数のスレッドが相互をブロックすることはありません。

この機能により、各クライアント（単一スレッドまたはマルチスレッド）からリモート・サーバーに対する接続は必ず 1 つになります。

同じ場所に配置されているオブジェクトのサポート 同じ場所に配置されているオブジェクトの場合、RMI/ORMI は同じ場所に配置された使用例を検出し、余分で不要なソケット・コールを回避します。

これは、JNDI レジストリが同じ場所に配置されている場合にも当てはまります。

リリース 9.0.4.x および 10.1.2.x の互換性パッチ

OC4J RMI 実装はリリースごとに異なります。そのため、起動元のオブジェクトと起動先のオブジェクトがそれぞれ異なるリリースの OC4J 上で実行されている場合、リモート・オブジェクトのメソッドを起動するにはパッチが必要です。旧リリースから新規リリースのメソッドを起動する場合でも、新規リリースから旧リリースのメソッドを起動する場合でも、パッチが旧リリースにインストールされている必要があります。

例 :

- OC4J リリース 9.0.4.3 上で稼働するサーブレットが、OC4J リリース 10.1.3 上で稼働する EJB のメソッドを起動する場合
- OC4J リリース 10.1.3 上で稼働する EJB が、OC4J リリース 10.1.2.0.2 上で稼働する JMS オブジェクトを起動する場合
- OC4J リリース 10.1.2 上で稼働する JSP が、OC4J リリース 10.1.3 上で稼働する EJB のメソッドを起動する場合

> **注意 :** リリース 9.0.4.x と 10.1.2.x 間での起動操作には、パッチは必要ありません。これらのリリースには互換性があります。

パッチは、`http://metalink.oracle.com` からダウンロードできます。次の表に、パッチの適用対象となる旧リリースと、対応するパッチ識別子を示します。

パッチ適用対象の OC4J リリース	パッチ識別子
9.0.4.3	BUG 4712885
10.1.2.2	BUG 4712552
10.1.2.0.0	BUG 4742351
10.1.2.0.2	BUG 4740687

スタンドアロン OC4J インストール環境での RMI の構成

スタンドアロン OC4J インストール環境では、RMI 構成ファイルの `rmi.xml` に RMI サーバー・データを指定する必要があります。また、このファイルの場所を OC4J 構成ファイルの `server.xml` に指定する必要があります。

`rmi.xml` ファイルと `server.xml` ファイルは、デフォルトで `ORACLE_HOME/j2ee/home/config` にインストールされます。

1. RMI 構成ファイルの `rmi.xml` へのパスを、OC4J サーバー構成ファイルの `server.xml` の `<rmi-config>` 要素に指定します。

   構文は次のとおりです。

   ```
   <rmi-config path="RMI_PATH" />
   ```

   通常、*`RMI_PATH`* の値は `./rmi.xml` です。

2. リモート RMI サーバーと通信し、接続を取得するために使用するホスト、ポート、ユーザー名およびパスワードを指定する `<rmi-server>` 要素を OC4J インスタンスの `rmi.xml` ファイルに追加します。このファイルは、デフォルトで `ORACLE_HOME/j2ee/home/config` にインストールされます。

   次に例を示します。

   ```
   <rmi-server host="hostname" port="port">
   </rmi-server>
   ```

<rmi-server> 要素の属性は、次のとおりです。

- host: RMI サーバーが RMI リクエストを受信するホスト名または IP アドレス。この属性を省略すると、MRI サーバーは、すべてのホストで RMI リクエストを受信します。
- port: RMI サーバーが RMI リクエストをリスニングするポート番号。OC4J スタンドアロン環境では、この属性を省略すると、デフォルトの 23791 が使用されます。

3. オプションで、アプリケーションから RMI 経由で接続できるリモート（Point-to-Point）RMI サーバーを個別に指定する 1 つ以上の <server> 要素で、<rmi-server> 要素を構成します。

   次に例を示します。

```
<rmi-server host="hostname" port="port">
    <server host="serverhostname" username="username" port="serverport"
    password="password"/>
</rmi-server>
```

   host 属性は必須、その他の属性はオプションです。server 要素のユーザーが置き換えることができる属性は次のとおりです。

   - *serverhostname*: リモート RMI サーバーが RMI リクエストをリスニングするホスト名または IP アドレス
   - *username*: リモート RMI サーバーの有効なプリンシパルのユーザー名
   - *serverport*: リモート RMI サーバーが RMI リクエストをリスニングするポート番号
   - *password*: プリンシパル *username* が使用するパスワード

アクセス制限

ORMI と ORMIS では、<access-mask> 要素を使用して rmi.xml 内に ACL マスクを定義することで、受信 IP アクセスを制限できます。

アクセス制御には、排他モードと包含モードがあります。

- 排他モードでは、アクセス権を IP アドレスまたはホスト名に明示的に付与します。アクセス・マスク default="deny" のデフォルト・モードは、アクセス制御が排他的であることを示します。
- 包含モードでは、アクセス権がすべてのオブジェクトに付与されるため、例外を個別に指定する必要があります。アクセス・マスク default="allow" のデフォルト・モードは、アクセス制御が包含的であることを示します。

デフォルトのアクセス・モードに対する例外を指定するには、<host-access> および <ip-access> サブ要素を使用します。

詳細は、『Oracle Containers for J2EE サーブレット開発者ガイド』の付録 A「Web モジュールの管理」を参照してください。

次に、localhost と 192.168.1.* サブネットのみにアクセスを許可する排他モード構成の例を示します。

```
<rmi-server
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
   "http://xmlns.oracle.com/oracleas/schema/rmi-server-10_0.xsd"
  port="23791"
  ssl-port="23943"
  schema-major-version="10"
  schema-minor-version="0">

      <access-mask default="deny" >
         <host-access domain="localhost" mode="allow"/>
         <ip-access ip="192.168.1.0" netmask="255.255.255.0" mode="allow"/>
      </access-mask>
```

```
<log>
  <file path="../log/rmi.log" />
</log>

<ssl-config
  keystore="../wallets/wallet-server-a/ewallet.p12"
  keystore-password="serverkey-a"
 />

 </rmi-server>
```

RMI/ORMI 使用時のクライアント・サイドの要件

この項では、Oracle および J2EE 標準の JAR ファイルをインストールするための ZIP ファイルのリストを示します。これらのファイルにより、RMI/ORMI を使用した EJB と JMS のルックアップが可能になります。

次の ZIP ファイルは、`http://www.oracle.com/technology/software/products/ias/htdocs/utilsoft.html` で入手できます。

- ORMI を使用した EJB のルックアップを可能にするには、`oc4j_client_<version>.zip` をダウンロードして解凍します。
- JMS などの他のルックアップを可能にするには、`oc4j_client.zip` のかわりに `oc4j_extended.zip` をダウンロードして解凍します。

適切な ZIP ファイルを解凍したら、必ず `oc4jclient.jar` を CLASSPATH に含めます。

ZIP ファイルには、クライアントに必要なすべての JAR ファイルが含まれます。これらの JAR ファイルには、クライアントの対話に必要なクラスが格納されています。クライアントに必要な他のすべての JAR ファイルは、`oc4jclient.jar` のマニフェスト・クラスパスで参照されるため、`oc4jclient.jar` のみを `CLASSPATH` に追加する必要があります。

EJB のルックアップを可能にするには、次の JAR ファイルをクライアント・サイドに含めます。

表 6-1 EJB のルックアップに必要なクライアント・サイドの JAR ファイル

JAR	ORACLE_HOME パス
oc4jclient.jar	/j2ee/<instance>
ejb.jar	/j2ee/<instance>/lib

リソース・アダプタのルックアップを可能にするには、次の JAR ファイルをクライアント・サイドに含めます。

表 6-2 JMS コネクタのルックアップに必要なクライアント・サイドの JAR ファイル

JAR	ORACLE_HOME パス
oc4jclient.jar	/j2ee/<instance>
jta.jar	/j2ee/<instance>/lib
jms.jar	/j2ee/<instance>/lib
jndi.jar	/j2ee/<instance>/lib
javax77.jar	/j2ee/<instance>/lib
adminclient.jar	/j2ee/<instance>/lib
oc4j-internal.jar	/j2ee/<instance>/lib
connector.jar	/j2ee/<instance>/lib
jmxri.jar	/j2ee/<instance>/lib

表 6-2 JMS コネクタのルックアップに必要なクライアント・サイドの JAR ファイル（続き）

JAR	ORACLE_HOME パス
jazncore.jar	/j2ee/<instance>
oc4j.jar	/j2ee/<instance>

内部的な OEMS JMS のメモリー内およびファイル・ベースのルックアップを可能にするには、表 6-3「OEMS JMS のメモリー内およびファイル・ベースのルックアップに必要なクライアント・サイドの JAR ファイル」にリストされている JAR ファイルをクライアント・サイドに含めます。

表 6-3 OEMS JMS のメモリー内およびファイル・ベースのルックアップに必要なクライアント・サイドの JAR ファイル

JAR	ORACLE_HOME パス
oc4jclient.jar	/j2ee/<instance>
jta.jar	/j2ee/<instance>/lib
jms.jar	/j2ee/<instance>/lib
jndi.jar	/j2ee/<instance>/lib
javax77.jar	/j2ee/<instance>/lib
optic.jar （opmn:ormi 接頭辞が Oracle Application Server 環境で使用される場合にのみ必須）	/opmn/lib

アプリケーション・クライアントから内部的な OEMS JMS のデータベース・オプションの直接ルックアップを可能にするには、表 6-4「OEMS JMS のデータベース・オプションのルックアップに必要なクライアント・サイドの JAR ファイル」にリストされている JAR ファイルをクライアント・サイドに含めます。

表 6-4 OEMS JMS のデータベース・オプションのルックアップに必要なクライアント・サイドの JAR ファイル

JAR	ORACLE_HOME パス
oc4jclient.jar	/j2ee/<instance>
ejb.jar	/j2ee/<instance>/lib
jta.jar	/j2ee/<instance>/lib
jms.jar	/j2ee/<instance>/lib
jndi.jar	/j2ee/<instance>/lib
javax77.jar	/j2ee/<instance>/lib
adminclient.jar	/j2ee/<instance>/lib
ojdbc14dms.jar	/j2ee/<instance>/../../oracle/jdbc/lib
dms.jar	/j2ee/<instance>/../../oracle/lib
bcel.jar	/j2ee/<instance>/lib
optic.jar （opmn:ormi 接頭辞が Oracle Application Server 環境で使用される場合にのみ必須）	/opmn/lib

Oracle Application Server 環境での RMI の構成

Oracle Application Server 環境では、opmn.xml ファイルを編集して、ローカル RMI サーバーが RMI リクエストをリスニングするポート範囲を指定する必要があります。

Oracle Application Server 環境の構成ファイルの変更内容は、OC4J インスタンスごとに手動で更新する必要があります。

opmn.xml ファイルを構成する手順は、次のとおりです。

1. opmn.xml ファイルから抜粋した次の例のように、port id="rmi" 要素を使用して rmi のポート範囲を構成します。

```
<ias-component id="OC4J">
    <process-type id="home" module-id="OC4J">
        <port id="default-web-site" range="12501-12600" protocol="ajp" />
        <port id="rmi" range="12401-12500" />
        <port id="rmis" range="12801-12900" />
        <port id="jms" range="12601-12700" />
        <process-set id="default_group" numprocs="1"/>
    </process-type>
</ias-component>
```

   opmn.xml ファイルの構成方法は、『Oracle Application Server 管理者ガイド』を参照してください。

2. 次の opmnctl コマンドを実行して変更を適用します。

```
opmnctl reload
```

RMI/ORMI を使用したリモート・オブジェクトのルックアップ

オブジェクトのメソッドを起動するには、最初にオブジェクトの場所を検索できる必要があります。

- RMI 用の JNDI プロパティの設定
- ORMI リクエストのロード・バランシングの構成
- ORMI を使用したルックアップの例

RMI 用の JNDI プロパティの設定

次の RMI/ORMI プロパティをクライアントの jndi.properties ファイルに指定します。

- java.naming.provider.url（6-8 ページの「Java ネーミング・プロバイダ URL の設定」を参照）
- java.naming.factory.initial（6-11 ページの「コンテキスト・ファクトリの指定」を参照）

Java ネーミング・プロバイダ URL の設定

次の構文を使用して、java.naming.provider.url の値として 1 つ以上の OC4J ホストを定義します。

```
<protocol>://<host>:[<port>]:<oc4j_instance>/<appName>
```

たとえば、Oracle Application Server 内で稼働するアプリケーションの場合、次のように指定します。

```
java.naming.provider.url=opmn:ormi://myHost:home/ejbsamples
```

表 6-5 に、この構文で使用される引数を示します。

注意：

- HTTP トンネリングを使用するアプリケーションの `java.naming.provider.url` を設定する方法の詳細は、6-14 ページの「HTTP を介した ORMI トンネリングの構成」を参照してください。
- ORMI over SSL を使用するアプリケーションの `java.naming.provider.url` の値は、『Oracle Containers for J2EE セキュリティ・ガイド』の「CSIv2」の章を参照してください。

表 6-5 ネーミング・プロバイダ URL

パラメータ	説明
`protocol`	■ Oracle Application Server のアプリケーションの場合： `opmn:ormi://` を使用します。 ■ スタンドアロン OC4J のアプリケーションの場合： `ormi://` を使用します。 ■ OC4J 以外のコンテナと相互運用する必要のあるアプリケーションの場合： `corbaname` を使用します（6-22 ページの「corbaname の URL の指定」を参照）。
`host`	■ Oracle Application Server のアプリケーションの場合： `opmn.xml` ファイルに定義された OPMN ホストの名前を指定します。通常、OPMN は OC4J インスタンスと同じノードにありますが、別のマシンにある場合はホスト名を指定します。 ■ スタンドアロン OC4J のアプリケーションの場合： `rmi.xml` ファイルの `<rmi-server>` 要素の `host` 属性に定義された OC4J ホスト名を指定します。
`port`	■ Oracle Application Server のアプリケーションの場合： OPMN `request` ポートを指定します。`opmn` プロセスは、適切な OC4J インスタンス用に選択した RMI ポートに RMI リクエストを転送します（6-10 ページの「Oracle Application Server での opmn リクエスト・ポートの指定」を参照）。省略すると、デフォルトの OPMN `request` ポート値の 6003 が使用されます。 ■ Oracle Application Server 10*g* リリース 2（10.1.2）以下のアプリケーションの場合： `opmn` によって OC4J インスタンス用に選択された RMI ポートを指定します（6-10 ページの「Oracle Application Server 10g リリース 2（10.1.2）以下での RMI ポートの指定」を参照）。 ■ スタンドアロン OC4J のアプリケーションの場合： `rmi.xml` の `<rmi-server>` 要素の `port` 属性に定義されたポート番号を指定します。 ■ OC4J 以外のコンテナと相互運用し、`corbaname` 接頭辞を使用する必要のあるアプリケーションの場合： 指定するポートの詳細は、6-22 ページの「corbaname の URL の指定」を参照してください。 ポートは省略可能であり、その場合はプロトコルによって決定されます。 ORMI プロトコルのデフォルト・ポートは、`23791` です。 ORMIS プロトコルのデフォルト・ポートは、`23943` です。

表 6-5 ネーミング・プロバイダ URL（続き）

パラメータ	説明
oc4j_instance	■ Oracle Application Server のアプリケーションの場合 : OC4J インスタンスの名前です。デフォルト OC4J インスタンスの名前は、home です。 ■ スタンドアロン OC4J のアプリケーションの場合 : この変数は適用されません。
appName	アプリケーション名です。

Oracle Application Server での opmn リクエスト・ポートの指定

Oracle Application Server では、opmn.xml ファイルで構成されている notification-server 要素の port 要素の request 属性に定義されたポートを指定できます（デフォルトは 6003 です）。opmn が request ポートで RMI リクエストを受信すると、それを該当する OC4J インスタンス用に選択した RMI ポートに転送します。

たとえば、次の opmn.xml ファイルの抜粋を考えてみます。

```
<notification-server>
    <port local="6100" remote="6200" request="6004"/>
    <log-file path="$OLE_HOME/opmn/logs/ons.log" level="4"
        rotation-size="1500000"/>
    <ssl enabled="true" wallet-file="$OLE_HOME/opmn/conf/ssl.wlt/default"/>
</notification-server>
```

この例では、notification-server 要素の port 要素の request 属性に定義されたポートは 6004 であるため、JNDI ネーミング・プロバイダ URL のポートとして 6004 を使用します。

この URL の使用例は、6-13 ページの「Oracle Application Server の OC4J」を参照してください。

Oracle Application Server 10*g* リリース 2（10.1.2）以下での RMI ポートの指定

Oracle Application Server の旧リリースの場合、OC4J インスタンスごとに opmn によって割り当てられた RMI ポートを指定します。割り当てられた RMI ポートを取得するには、OC4J ホストで次の opmnctl コマンドを使用します。

```
opmnctl status -l
```

これにより、OC4J インスタンスごとに 1 行を含むデータ表が出力されます。

次に例を示します（読みやすいように一部の列は省略されています）。

```
Processes in Instance: server817.company.us.com
-------------------+--------------------+-------+ ... +------
ias-component      | process-type       |   pid | ... | ports
-------------------+--------------------+-------+ ... +------
OC4J               | home               |  2012 | ... | iiop:12701,jms:12601,rmi:12401
HTTP_Server        | HTTP_Server        | 28818 | ... | http2:7200,http1:7778,http:7200
```

この例では、OC4J インスタンスの RMI 用として、opmn によってポート 12401 が選択されています。この OC4J インスタンスの JNDI ネーミング・プロバイダ URL のポートには、この値を使用します。

コンテキスト・ファクトリの指定

初期コンテキスト・ファクトリは、クライアント用の初期コンテキスト・クラスを作成します。

java.naming.factory.initial プロパティを次のいずれかに設定してください。

- oracle.j2ee.naming.ApplicationClientInitialContextFactory
- com.evermind.server.ApplicationInitialContextFactory
- oracle.j2ee.rmi.RMIInitialContextFactory

注意 : 次の初期コンテキスト・ファクトリは、廃止予定です。

- com.evermind.server.ApplicationClientInitialContextFactory
- com.evermind.server.RMIInitialContextFactory

ApplicationClientInitialContextFactory は、スタンドアロン・アプリケーション・クライアントからリモート・オブジェクトをルックアップするときに使用されます。このコンテキスト・ファクトリは、application-client.xml および orion-application-client.xml で検出した refs および ref-mappings を使用します。これは、初期コンテキストが Java アプリケーションでインスタンス化される場合のデフォルトの初期コンテキスト・ファクトリです。

RMIInitialContextFactory は、ORMI プロトコルを使用して異なる OC4J サーバー・インスタンス間でリモート・オブジェクトをルックアップするときに使用されます。

使用する初期コンテキスト・ファクトリのタイプは、次のようにクライアントに応じて異なります。

- クライアントが OC4J コンテナ外部の Pure Java クライアントの場合は、ApplicationClientInitialContextFactory クラスを使用します。
- クライアントが OC4J コンテナ内の EJB またはサーブレット・クライアントの場合は、ApplicationInitialContextFactory クラスを使用します。これはデフォルト・クラスであるため、初期コンテキスト・ファクトリ・クラスを指定せずに新規 InitialContext を作成するたびに、クライアントでは ApplicationInitialContextFactory クラスが使用されます。
- クライアントが JNDI ツリーを操作または横断する管理クラスの場合は、RMIInitialContextFactory クラスを使用します。
- クライアントが DNS ロード・バランシングを使用する場合は、RMIInitialContextFactory クラスを使用します。

ORMI リクエストのロード・バランシングの構成

OC4J では、クラスタ化された複数の OC4J インスタンスにデプロイされた EJB に対するクライアント ORMI リクエストのロード・バランシングがサポートされます。

デフォルトの動作では、クライアントは、コールのたびに同じ OC4J インスタンスに接続します。具体的には、クライアントが同じ user/password/provider URL の組合せを使用して InitialContext() をコールするたびに、クライアントの最初の起動時に作成されたキャッシュ済の Context オブジェクトが戻されます。そのため、クライアントは、この Context オブジェクトによって定義される同じ OC4J インスタンスにリクエストを送信することになります。

クライアント・コールの数が明らかに少ない状況では、この方法での接続は効率的であり、最大のパフォーマンスを得ることができます。

ただし、大量のリクエストが予想される状況では、OC4J インスタンス間でリクエストのロード・バランシングを実行する方が適切である可能性があります。ロード・バランシングは、oracle.j2ee.rmi.loadBalance プロパティを使用して構成できます。このプロパティは、クライアントの jndi.properties ファイル、またはクライアント・コードのハッシュテーブルで設定します。次の表に、このプロパティの値とその説明を示します。

表 6-6 oracle.j2ee.rmi.loadBalance プロパティの値

値	説明
client	指定した場合、クライアントは、通信全体に対する最初のルックアップで初期選択された OC4J プロセスと対話します。
context	クラスタ化された OC4J 環境の EJB にアクセスする Web クライアント（サーブレットまたは JSP）で使用されます。 指定した場合、InitialContext() が起動するたびに、ランダムに選択された OC4J インスタンスの新規 Context オブジェクトが戻されます。
lookup	クラスタ化された OC4J 環境の EJB にアクセスするスタンドアロン・クライアントで使用されます。 指定した場合、クライアントが Context.lookup() をコールするたびに、ランダムに選択された OC4J インスタンスの新規 Context オブジェクトが作成されます。

次に、Hashtable でこのプロパティを lookup に設定する例を示します。

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,"com.evermind.server.rmi.RMIInitialContextFactory");
env.put(Context.SECURITY_PRINCIPAL, "jazn.com/admin");
env.put(Context.SECURITY_CREDENTIALS, "password");
env.put(Context.PROVIDER_URL,"opmn:ormi://<hostname>:oc4j_inst1/ejbsamples");
env.put("oracle.j2ee.rmi.loadBalance","lookup");
```

ORMI を使用したルックアップの例

この項では、次の環境で ORMI を使用して EJB をルックアップする例を示します。

- スタンドアロン OC4J
- Oracle Application Server の OC4J
- 旧リリースの Oracle Application Server の OC4J

スタンドアロン OC4J

次の例に、スタンドアロン OC4J インスタンスにデプロイされた J2EE アプリケーション `ejbsamples` で `MyCart` という EJB をルックアップする方法を示します。アプリケーションは、RMI ポート 23792 でリスニングするよう構成されたノード上に配置されています。

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,"oracle.j2ee.rmi.RMIInitialContextFactory");
env.put(Context.SECURITY_PRINCIPAL, "admin");
env.put(Context.SECURITY_CREDENTIALS, "password");
env.put(Context.PROVIDER_URL, "ormi://<hostname>:23792/ejbsamples");

Context context = new InitialContext(env);
Object homeObject = context.lookup("MyCart");
CartHome home = (CartHome)PortableRemoteObject.narrow(homeObject,CartHome.class);
```

Oracle Application Server の OC4J

Oracle Application Server では、Oracle Application Server 環境の EJB をルックアップする場合、URL で次のタイプのルックアップを使用できます。

次の例に、Oracle Application Server 環境の J2EE アプリケーション `ejbsamples` で `MyCart` という EJB をルックアップする方法を示します。この起動とスタンドアロンの起動の違いは、`ormi` に対する `opmn` 接頭辞、EJB アプリケーションがデプロイされている OC4J インスタンス名 `oc4j_inst1` の指定、および RMI ポートの指定が不要という点です。

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,"oracle.j2ee.rmi.RMIInitialContextFactory");
env.put(Context.SECURITY_PRINCIPAL, "jazn.com/admin");
env.put(Context.SECURITY_CREDENTIALS, "password");
env.put(Context.PROVIDER_URL,"opmn:ormi://<hostname>:oc4j_inst1/ejbsamples");

Context context = new InitialContext(env);
Object homeObject = context.lookup("MyCart");
CartHome home = (CartHome)PortableRemoteObject.narrow(homeObject,CartHome.class);
```

旧リリースの Oracle Application Server の OC4J

Oracle Application Server 環境の OC4J インスタンスでは、RMI ポートが動的に割り当てられます。デフォルトのユーザー・マネージャは、JAZNUserManager です。

Oracle Application Server の旧リリースでは、Oracle Application Server の EJB にアクセスする場合、opmn によって割り当てられた RMI ポートを確認する必要があります。OC4J インスタンス用の JVM が 1 つのみの場合は、RMI のポート範囲を特定の番号（3101 ～ 3101 など）に限定する必要があります。

次の例に、J2EE アプリケーション ejbsamples で MyCart という EJB をルックアップする方法を示します。アプリケーションは、RMI ポート 12401 でリスニングするよう構成されたノード上に配置されています。

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,"com.evermind.server.rmi.RMIInitialContextFacto
ry");
env.put(Context.SECURITY_PRINCIPAL, "jazn.com/admin");
env.put(Context.SECURITY_CREDENTIALS, "welcome");
env.put(Context.PROVIDER_URL, "ormi://<hostname>:12401/ejbsamples");

Context context = new InitialContext(env);
Object homeObject = context.lookup("MyCart");
CartHome home = (CartHome)PortableRemoteObject.narrow(homeObject,CartHome.class);
```

HTTP を介した ORMI トンネリングの構成

ファイアウォールを通じた EJB との通信を可能にするため、ORMI では、HTTP トンネリングを使用して HTTP POST リクエスト内に RMI コールをラップします。このリクエストは、ターゲット OC4J インスタンス内の default Web アプリケーションに転送されます。トンネリングは、RMI/ORMI でのみサポートされます。RMI/IIOP を使用した HTTP トンネリングは実行できません。

java.naming.provider.url の値として設定される HTTP トンネリング URL の構文は、次のとおりです。

ormi:http[s]://*<hostname>:<port>/<appName>*

表 6-7 HTTP トンネリング URL の構文

パラメータ	説明
hostname	リクエストを受信するサーバー・ホスト名。
port	サーバーのポート。この値はオプションです。省略すると、デフォルトの 80 が使用されます。 Oracle Application Server 環境に接続する際には、OPMN に登録されているのと同じポートを使用します。
appName	ターゲット・アプリケーションの名前。

HTTP トラフィックがプロキシ Web サーバーを経由する場合は、次のようにコマンドラインで proxyHost および（オプションで）proxyPort を指定し、EJB クライアントを起動します。

-Dhttp.proxyHost=*<proxy_host>* -Dhttp.proxyPort=*<proxy_port>*

proxy_port を省略すると、デフォルトの 80 が使用されます。

OC4J での ORMI/SSL（ORMIS）の使用方法

ORMI over SSL（ORMIS）は、Oracle Application Server 10*g* リリース 3（10.1.3.1.0）の新機能です。この機能により、OC4J では、複数の OC4J サーバー・インスタンスのオブジェクト間通信で Secure Sockets Layer（SSL）を介した RMI がサポートされます。

ORMIS は、OC4J ではデフォルトで無効化されています。この機能を使用するには、クライアントおよびサーバーでキーストアを作成する必要があります。

詳細は、『Oracle Containers for J2EE セキュリティ・ガイド』で次の内容に関する情報を参照してください。

- キーストアおよびウォレットの作成
- OC4J での ORMI/SSL（ORMIS）の有効化
- ORMIS を使用するための OC4J の構成
- ORMIS を使用するためのクライアントの構成
- EJB サーバー・セキュリティのプロパティ（internal-settings.xml）
- CSIv2 セキュリティのプロパティ
- CSIv2 セキュリティのプロパティ（internal-settings.xml）
- CSIv2 セキュリティのプロパティ（ejb_sec.properties）
- CSIv2 セキュリティのプロパティ（orion-ejb-jar.xml）
- EJB クライアント・セキュリティのプロパティ（ejb_sec.properties）

J2EE 相互運用性の使用方法（RMI/IIOP）

この項では、標準の Internet Inter-Orb Protocol（IIOP）を介した Remote Method Invocation（RMI）を使用して、異なる J2EE コンテナ間（OC4J と BEA WebLogic サーバー間など）で EJB やその他のオブジェクトを相互起動できるようにする OC4J のサポート機能について説明します。

この項には、次の項目が含まれます。

- RMI/IIOP の概要
- ORMI から IIOP トランスポートへの切替え
- 相互運用性のための OC4J の構成

RMI/IIOP の概要

Java RMI を使用すると、分散 Java ベース間アプリケーションを作成できます。この種のアプリケーションでは、リモート Java オブジェクトのメソッドを、異なるホスト上にあるものも含め、JVM 間で起動できます。

バージョン 2.0 の EJB 仕様では、異なるコンテナ間で EJB ベースのアプリケーションを容易に相互起動できる機能が導入されました。コードを変更せずに Bean のプロパティを編集して再デプロイするだけで、既存の EJB を相互運用可能にできます。再デプロイの詳細は、6-20 ページの「ORMI から IIOP トランスポートへの切替え」を参照してください。

EJB 相互運用性は、次のもので構成されています。

- トランスポート相互運用性 : CORBA IIOP（Internet Inter-ORB Protocol。ORB は Object Request Broker）を使用。
- ネーミング相互運用性 : CORBA CosNaming Service（OMG CORBA Object Service 仕様の一部である CORBA Object Service Naming）を使用。
- セキュリティ相互運用性 : Common Secure Interoperability Version 2（CSIv2）を使用。
- トランザクション相互運用性 : CORBA Transaction Service（OTS）を使用。

OC4J は、トランスポート相互運用性、ネーミング相互運用性およびセキュリティ相互運用性を提供します。

トランスポート

デフォルトでは、OC4J の EJB は、独自仕様のプロトコルである RMI/Oracle Remote Method Invocation（ORMI）を使用して通信を行います（6-2 ページの「Oracle Remote Method Invocation（RMI/ORMI）の使用方法」を参照）。

ただし、異なる EJB コンテナ間で EJB を相互起動するため、RMI/IIOP を使用するよう EJB を変換することも簡単です。この項では、RMI/IIOP を構成および使用する方法について説明します。

注意 : OC4J 10*g* リリース 3（10.1.3.1.0）の実装では、ロード・バランシングとフェイルオーバーがサポートされるのは ORMI の場合のみで、IIOP の場合はサポートされません。

ネーミング

OC4J は、CORBA CosNaming サービスをサポートします。OC4J では、`EJBHome` オブジェクト参照を CosNaming サービスで公開できます。また、アプリケーションが CORBA を使用して JNDI 名をルックアップできる JNDI CosNaming 実装が提供されます。JNDI API または CosNaming API のいずれかを使用してアプリケーションを作成できます。

セキュリティ

OC4J は Common Secure Interoperability Version 2（CSIv2）をサポートします。CSIv2 は、様々な準拠レベルを指定します。OC4J は、準拠レベル 0（ゼロ）を必要とする EJB 仕様に準拠しています。

セキュリティの詳細は、『Oracle Containers for J2EE セキュリティ・ガイド』の「CSIv2」の章を参照してください。

トランザクション

EJB2.0 仕様では、オプションのトランザクション相互運用性機能が規定されています。仕様に準拠した実装では、次のいずれかを選択する必要があります。

- トランザクション相互運用可能 : 異なる J2EE コンテナでホスティングされている Bean 間でトランザクションがサポートされます。
- トランザクション相互運用不可 : 同じコンテナ内の Bean 間でのみトランザクションがサポートされます。

このリリースの OC4J は、トランザクション相互運用不可です。つまり、トランザクションが複数の EJB コンテナにまたがると、指定の例外が発生します。

rmic.jar コンパイラ

CORBA オブジェクトの起動または CORBA オブジェクトによる起動のためには、RMI オブジェクトに、対応するスタブ、スケルトンおよび Interface Description Language（IDL）が必要です。`rmic.jar` コンパイラを使用して、Java クラスからスタブとスケルトンを生成するか、IDL を生成します。

RMI/IIOP で使用する場合は、必ず `-iiop` オプションを使用してコンパイルしてください。

相互運用性のための OC4J の構成

EJB に相互運用性サポートを追加するには、相互運用性プロパティを指定する必要があります。これらのプロパティの一部は OC4J の起動時に指定され、その他はデプロイメント・ファイルに指定されています。

相互運用性 OC4J フラグ

次の OC4J 起動フラグにより、RMI の相互運用性がサポートされます。

- `-DGenerateIIOP=true`: アプリケーションが再デプロイされるたびに、新規のスタブとスケルトンを生成します。
- `-Diiop.debug=true`: デプロイ時のデバッグ・メッセージを生成します。その大部分はコード生成に関連しています。
- `-Diiop.runtime.debug=true`: 実行時のデバッグ・メッセージを生成します。

相互運用性構成ファイル

EJB による通信を可能にするには、表 6-8 に示す構成ファイルのパラメータを構成する必要があります。

表 6-8 相互運用性構成ファイル

コンテキスト	ファイル	説明
サーバー	server.xml	このファイルの <sep-config> 要素は、サーバー拡張プロバイダのプロパティに対するパス名（通常は internal-settings.xml）を指定します。次に例を示します。 <sep-config path="./internal-settings.xml">
	internal-settings.xml	このファイルでは、RMI/IIOP 固有のサーバー拡張プロバイダのプロパティを指定します。
アプリケーション	orion-ejb-jar.xml	<session-deployment> および <entity-deployment> エンティティの <ior-security-config> サブエンティティは、サーバーの Common Secure Interoperability Version 2（CSIv2）セキュリティのプロパティを指定します。
	ejb_sec.properties	このファイルでは、EJB に対するクライアント・サイド・セキュリティのプロパティを指定します。
	jndi.properties	このファイルでは、クライアントが使用する初期ネーミング・コンテキストの URL を指定します。詳細は、6-18 ページの「相互運用に関する JNDI プロパティ（jndi.properties）」を参照してください。

セキュリティの詳細は、『Oracle Containers for J2EE セキュリティ・ガイド』の「CSIv2」の章を参照してください。

相互運用に関する JNDI プロパティ（jndi.properties）

次の RMI/IIOP プロパティは、クライアントの jndi.properties ファイルで制御されます。

- Bean を相互運用可能にするには、java.naming.provider.url に OPMN または corbaname の URL を使用できます。

  クライアントの JNDI プロパティ java.naming.provider.url を構成して OPMN の URL を使用する場合、OPMN による割当てポートは OC4J にレポートされないため、そのクライアントでは ssl-port ポートや ssl-client-server-auth-port ポートに接続できません。

  corbaname の URL の詳細は、6-22 ページの「corbaname の URL の指定」を参照してください。OPMN の URL の詳細は、6-23 ページの「OPMN の URL の指定」を参照してください。

- contextFactory は、ApplicationClientInitialContextFactory またはクラス IIOPInitialContextFactory です。

  アプリケーションに application-client.xml ファイルがある場合は、contextFactory を ApplicationClientInitialContextFactory に設定したままにします。アプリケーションに application-client.xml ファイルがない場合は、contextFactory を IIOPInitialContextFactory に変更します。

コンテキスト・ファクトリの使用

- oracle.j2ee.naming.ApplicationClientInitialContextFactory は、スタンドアロン・アプリケーション・クライアントからリモート・オブジェクトをルックアップするときに使用されます。このコンテキスト・ファクトリは、application-client.xml および orion-application-client.xml で検出した refs および ref-mappings を使用します。これは、初期コンテキストが Java アプリケーションでインスタンス化される場合のデフォルトの初期コンテキスト・ファクトリです。
- oracle.j2ee.iiop.IIOPInitialContextFactory は、IIOP プロトコルを使用して異なるコンテナ間でリモート・オブジェクトをルックアップするときに使用されます。

IIOP 使用時のクライアント・サイドの要件

この項では、Oracle および J2EE 標準の JAR ファイルをインストールするための ZIP ファイルのリストを示します。これらのファイルにより、RMI/IIOP を使用した EJB と JMS のルックアップが可能になります。

次の ZIP ファイルは、
http://www.oracle.com/technology/software/products/ias/index.html で入手できます。

- IIOP を使用した EJB のルックアップを可能にするには、oc4j_iiop_client.zip をダウンロードして解凍します。
- JMS などの他のルックアップを可能にするには、oc4j_extended.zip もダウンロードして解凍します。

これらの ZIP ファイルを解凍したら、必ず oc4jclient.jar を CLASSPATH に含めます。

ZIP ファイルには、クライアントに必要なすべての JAR ファイルが含まれます。これらの JAR ファイルには、クライアントの対話に必要なクラスが格納されています。クライアントに必要な他のすべての JAR ファイルは、oc4jclient.jar のマニフェスト・クラスパスで参照されるため、oc4jclient.jar のみを CLASSPATH に追加する必要があります。

IIOP クライアントを実行する場合、IIOP クライアント用に次のプロパティを設定する必要があります。

- -Dorg.omg.CORBA.ORBInitialHost=${orb.host}
- -Dorg.omg.CORBA.ORBInitialPort=${orb.port}
- -Djavax.rmi.CORBA.PortableRemoteObjectClass=com.sun.corba.ee.impl.javax.rmi.PortableRemoteObject
- -Dcom.oracle.CORBA.OrbManager=com.oracle.corba.ee.impl.orb.ORBManagerImpl

ORMI から IIOP トランスポートへの切替え

OC4J では、EJB は独自仕様のプロトコルである RMI/ORMI を使用して通信を行います（6-2 ページの「Oracle Remote Method Invocation（RMI/ORMI）の使用方法」を参照）。ただし、異なるベンダーの EJB コンテナ間（OC4J と BEA WebLogic など）で EJB を相互起動するため、RMI/IIOP を使用するよう EJB を変換することも可能です。

> **注意：** RMI/IIOP のサポートは CORBA 2.3.1 仕様に基づいています。以前のリリースの CORBA を使用してコンパイルされたアプリケーションは、正しく動作しない可能性があります。

次の項で、この変換について詳しく説明します。

- スタンドアロン OC4J 環境で相互運用性を確保するための EJB の構成
- Oracle Application Server 環境で相互運用性を確保するための EJB の構成
- corbaname の URL の指定
- OPMN の URL の指定
- 例外マッピング
- 非 OC4J コンテナからの OC4J ホスティング Bean の起動

スタンドアロン OC4J 環境で相互運用性を確保するための EJB の構成

スタンドアロン OC4J 環境で RMI/IIOP を使用するよう EJB を変換する手順は、次のとおりです。

1. `orion-ejb-jar.xml` および `internal-settings.xml` で、Bean の CSIv2 セキュリティ・ポリシーを指定します。

   セキュリティの詳細は、『Oracle Containers for J2EE セキュリティ・ガイド』の「CSIv2」の章を参照してください。

2. `-DGenerateIIOP=true` フラグを指定して OC4J を再起動します。

3. `admin.jar` を使用してアプリケーションをデプロイします。`-iiopClientJar` スイッチを使用して、クライアントのスタブ JAR ファイルを取得する必要があります。次に例を示します。

   ```
   java -jar $J2EE_HOME/admin.jar ormi://localhost admin welcome -deploy -file
   filename -deployment_name application_name -iiopClientJar stub_jar_filename
   ```

   > **注意：** デプロイするアプリケーションで相互運用性（IIOP）を有効化するために、`-iiopClientJar` スイッチを使用する必要があります。OC4J では、相互運用性はアプリケーションごとに有効化されます。

4. `-iiopClientJar` スイッチを指定して `admin.jar` を実行し、クライアントの `classpath` を変更して、デプロイ中に取得されたスタブ JAR ファイルを組み込みます。

   OC4J により生成されたスタブ JAR ファイルのコピーは、次のようにサーバーのデプロイメント・ディレクトリに置かれている場合もあります。

   application_deployment_directory/appname/ejb_module/`_iiopClient.jar`

   > **注意：** 実行時に行われる ORMI スタブの生成とは異なり、IIOP スタブおよび Tie クラス・コードの生成はデプロイ時に行われます。このため、管理者自身が JAR ファイルを `classpath` に追加する必要があります。

5. ormi の URL のかわりに corbaname の URL を使用するように、クライアントの JNDI プロパティ java.naming.provider.url を編集します。corbaname の URL の詳細は、6-22 ページの「corbaname の URL の指定」を参照してください。

6. （オプション）CORBA アプリケーションから Bean にアクセスできるようにするには、rmic.jar を実行して、そのインタフェースを記述する IDL を生成します。

Oracle Application Server 環境で相互運用性を確保するための EJB の構成

この項では、Oracle Application Server 環境で RMI/IIOP を使用するよう EJB を変換する方法について説明します。

1. orion-ejb-jar.xml および internal-settings.xml で、Bean の CSIv2 セキュリティ・ポリシーを指定します。

   セキュリティの詳細は、『Oracle Containers for J2EE セキュリティ・ガイド』の「CSIv2」の章を参照してください。

2. デフォルトでは、Oracle Application Server 環境で RMI/IIOP は無効化されています。RMI/IIOP を有効化するには、OPMN の構成ファイル *J2EE_HOME*/opmn/conf/opmn.xml で、OPMN の管理対象となる OC4J インスタンスごとに一意の iiop、iiops1 および iiops2 ポート（またはポート範囲）が存在することを確認します。これらは次のポートを意味します。

   iiop: 標準 IIOP ポート

   iiops1: サーバー・サイド認証専用の IIOP/SSL ポート

   iiops2: クライアント・サイドおよびサーバー・サイド認証用の IIOP/SSL ポート

   > **注意：** CSIv2 を使用した相互運用性を有効化する OC4J インスタンスごとに、iiop、iiops1 および iiops2 ポート（またはポート範囲）を指定する必要があります。これを指定しないと、OC4J では IIOP リスナーが構成されないため、OC4J の internal-settings.xml ファイルでの構成に関係なく、相互運用性は自動的に無効化されます。

   次に例を示します。

```
<ias-component id="OC4J">
    <process-type id="home" module-id="OC4J">
        <port id="ajp" range="3000-3100"/>
        <port id="rmi" range="23791-23799"/>
        <port id="jms" range="3201-3300"/>
        <port id="iiop" range="3401-3500"/>
        <port id="iiops1" range="3501-3600"/>
        <port id="iiops2" range="3601-3700"/>
        <process-set id="default_group" numprocs="1"/>
    </process-type>
</ias-component>
```

   > **注意：** クライアントの JNDI プロパティ java.naming.provider.url を構成して OPMN の URL を使用する場合、OPMN による割当てポートは OC4J にレポートされないため、そのクライアントでは iiops1 ポートや iiops2 ポートに接続できません。

3. opmnctl を使用して、OPMN で管理される OC4J インスタンスをすべて再起動します。-DGenerateIIOP=true フラグを使用します。

```
opmnctl -DGenerateIIOP=true startall
```

4. -enableIIOP オプションを指定してアプリケーションをデプロイします。デプロイの詳細は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。

5. クライアントの classpath を変更して、OC4J によって生成されたスタブ JAR ファイルを組み込みます。通常、このファイルはサーバーのデプロイメント・ディレクトリにあります。

   *application_deployment_directory/appname/ejb_module/*_iiopClient.jar

   注意： 実行時に行われる ORMI スタブの生成とは異なり、IIOP スタブおよび Tie クラス・コードの生成はデプロイ時に行われます。このため、管理者自身が JAR ファイルを classpath に追加する必要があります。

6. ormi の URL のかわりに OPMN または corbaname の URL を使用するように、クライアントの JNDI プロパティ java.naming.provider.url を編集します。corbaname の URL の詳細は、6-22 ページの「corbaname の URL の指定」を参照してください。OPMN の URL の詳細は、6-23 ページの「OPMN の URL の指定」を参照してください。

7. （オプション）CORBA アプリケーションから Bean にアクセスできるようにするには、rmic.jar を実行して、そのインタフェースを記述する IDL を生成します。

corbaname の URL の指定

相互運用を可能にするには、EJB で CosNaming を使用して他の Bean をルックアップする必要があります。つまり、ルート NamingContext をルックアップするための URL には、ormi の URL スキームのかわりに、corbaname の URL スキームを使用する必要があります。この項では、EJB 開発者が一般的に使用する corbaname のサブセットについて説明します。corbaname スキームの詳細は、CORBA Naming Service 仕様の第 2.5.3 項を参照してください。corbaname スキームは、corbaloc スキームに基づいています。これについては、CORBA 仕様の第 13.6.10.1 項で説明されています。

corbaname の URL スキームの最も一般的な形式は、次のとおりです。

```
corbaname::host[:port]
```

この corbaname の URL によって、従来型の DNS ホスト名または IP アドレスとポート番号が指定されます。次に例を示します。

```
corbaname::example.com:8000
```

corbaname の URL では、ホストとポートの後に # および文字列表現による NamingContext を記述することによって、ネーミング・コンテキストを指定することもできます。指定したホストの CosNaming サービスが、ネーミング・コンテキストを解析します。

```
corbaname::host[:port]#namingcontext
```

次に例を示します。

```
corbaname::example.com:8000#Myapp
```

OPMN の URL の指定

この項では、RMI/IIOP 固有の OPMN の URL の詳細を説明します。OPMN の URL の概要は、6-8 ページの「RMI 用の JNDI プロパティの設定」を参照してください。

Oracle Application Server 環境では、各 Oracle Application Server インスタンス内のすべての OC4J プロセスの IIOP ポートは、OPMN により動的に管理されます。このため、OC4J プロセスがアクティブに IIOP リクエストをリスニングしているポートをクライアントが認識することはできません。Oracle Application Server 環境のクライアントが、すべてのアクティブな OC4J プロセスの IIOP ポートを認識することなく、RMI/IIOP リクエストを正常に発行できるようにするには、次の形式の URL を使用して（クライアントの `jndi.properties` ファイルの）`jndi.naming.provider.url` プロパティを変更します。

opmn:corbaname::*host*[:*port*][#*instance-name*]#*appname*

次に例を示します。

```
opmn:corbaname::dlsun74:6003#oc4j_inst1#ejbsamples
```

注意：

- ロード・バランシングとフェイルオーバーがサポートされるのは ORMI の場合のみで、IIOP の場合はサポートされません。
- OPMN の URL を使用すると、クライアントは `iiops1` または `iiops2`（`ssl-port` または `ssl-client-server-auth-port`）ポートに接続できません。

例外マッピング

EJB が IIOP を介して起動される場合、OC4J ではシステム例外を CORBA 例外にマップする必要があります。表 6-9 に、例外マッピングを示します。

表 6-9 Java 例外と CORBA 例外のマッピング

OC4J システム例外	CORBA システム例外
`javax.transaction.TransactionRolledbackException`	`TRANSACTION_ROLLEDBACK`
`javax.transaction.TransactionRequiredException`	`TRANSACTION_REQUIRED`
`javax.transaction.InvalidTransactionException`	`INVALID_TRANSACTION`
`java.rmi.NoSuchObjectException`	`OBJECT_NOT_EXIST`
`java.rmi.AccessException`	`NO_PERMISSION`
`java.rmi.MarshalException`	`MARSHAL`
`java.rmi.RemoteException`	`UNKNOWN`

非 OC4J コンテナからの OC4J ホスティング Bean の起動

OC4J でホスティングされていない EJB では、OC4J ホスティング EJB を起動するために、`oc4j_interop.jar` を `classpath` で検索できる必要があります。OC4J では、ホスト・コンテナが `java:comp/HandleDelegate` の JNDI ネームスペースで `HandleDelegate` オブジェクトを使用可能にすることを前提としています。`oc4j_interop.jar` ファイルには、ホーム・ハンドル、リモート・ハンドルおよびメタデータ・オブジェクトの移植可能な標準の実装が含まれています。

7

Java Object Cache

この章では、Oracle Containers for J2EE（OC4J）の Java Object Cache（JOC）について、そのアーキテクチャとプログラミング機能も含めて説明します。この章には、次の項目が含まれます。

新機能

次の OC4J JOC の機能および動作は、今回のリリースの新機能です。

- DMS のサポート
- 拡張クラス・ローダーのサポート：

  リージョン固有のクラス・ローダーを使用して、オブジェクト名とキャッシュされたオブジェクト自体をロードできます。

- 拡張リモート・キャッシュ・アクセス：

  `CacheAccess.replaceRemote` および `CacheLoader.getFromRemote` をそれぞれ使用して、特定のリモート・キャッシュにオブジェクトを追加することや、リモート・キャッシュからオブジェクトを取得することが可能です。

  `CacheAccess.getAllCached` を使用すると、システム内のすべてのキャッシュから特定の名前付きオブジェクトを取得できます。この結果はリストとして戻されます。この機能は、各キャッシュから統計を取得する場合に特に役立ちます。

  以前は、すべてのリスナーが非同期に実行されていました。この場合、リクエストは、イベント・リスナーの実行前に戻されるのが普通です。同期式の最新リスナーが追加されたため、個別のスレッドではなくイベントの一部としてリスナーを実行できます。

- **廃止予定**

  次の項目は、今回のリリースで廃止予定です。

  - `Attributes.logging` のレベルは、`java.util.logging.Levels` の指定レベルで置き換えられました。

Java Object Cache の概念

Oracle Application Server 10*g* は、動的に生成されるコンテンツに関する Web サイトのパフォーマンスの問題を管理する E-Business を支援するために、Java Object Cache を提供しています。Java Object Cache により、Oracle Application Server 10*g* で動作する Web サイトのパフォーマンス、スケーラビリティおよび可用性が向上します。

Java Object Cache では、頻繁にアクセスするオブジェクトや作成にコストがかかるオブジェクトをメモリーまたはディスクに格納することによって、Java プログラム内で情報を繰り返し作成したりロードする必要がなくなります。Java Object Cache は、コンテンツをすばやく取得するため、アプリケーション・サーバーの負荷が大幅に軽減されます。

Oracle Application Server 10*g* のキャッシュ・アーキテクチャには、次のキャッシュ・コンポーネントが含まれています。

- **Oracle Application Server Web Cache:** Web Cache は、アプリケーション・サーバー（Web サーバー）の手前に位置しており、サーバーのコンテンツをキャッシュし、そのコンテンツをリクエストしている Web ブラウザに提供します。Web サイトへのアクセス時に、ブラウザは、HTTP リクエストを Web Cache に送信します。Web Cache は、アプリケーション・サーバーに対する仮想サーバーとして動作します。リクエストしたコンテンツが変更された場合、Web Cache はアプリケーション・サーバーから新しいコンテンツを取得します。

  Web Cache は、アプリケーションの外部でメンテナンスされる HTTP レベルのキャッシュで、高速なキャッシュ操作を行います。純粋なコンテンツ・ベースのキャッシュであるため、静的データ（HTML、GIF、JPEG ファイルなど）または動的データ（サーブレットや JSP の結果など）をキャッシュできます。このキャッシュが、アプリケーション外部にフラットなコンテンツ・ベースのキャッシュとして存在する場合は、オブジェクト（Java オブジェクトや XML DOM（Document Object Model）オブジェクトなど）を構造化された形式でキャッシュすることはできません。さらに、キャッシュされたデータに対する後処理機能がかなり制限されます。

- **Java Object Cache**: Java Object Cache は、アプリケーション・サーバーが Java プログラムを使用してコンテンツを提供する場合に、コストがかかる、あるいは頻繁に使用される Java オブジェクトのキャッシングを提供します。キャッシュされた Java オブジェクトは、生成されたページを含んでいたり、新規コンテンツの作成に役立つプログラム内のサポート・オブジェクトを提供する場合もあります。Java Object Cache は、Java アプリケーションの指定どおりに、オブジェクトを自動的にロードおよび更新します。

- **Web Object Cache:** Web Object Cache は、Web アプリケーション・レベルのキャッシング機能です。アプリケーション・レベルのキャッシュは、Java の Web アプリケーション内に埋め込まれ、メンテナンスされます。Web Object Cache は、Web ベースとオブジェクト・ベースの両方を組み合せたハイブリッド・キャッシュです。Web Object Cache を使用すると、アプリケーションは、Application Program Interface（API）コール（サーブレットの場合）またはカスタム・タグ・ライブラリ（JSP の場合）を使用してプログラムでキャッシュできます。Web Object Cache は通常、Web Cache の補足機能として使用されます。デフォルトでは、Web Object Cache は、そのリポジトリとして Java Object Cache を使用します。

  カスタム・タグ・ライブラリまたは API を使用すると、ユーザーはページ・フラグメントの境界を定義でき、JSP ページやサーブレットの実行の中間結果と部分的な結果を、キャッシュされたオブジェクトとして取得、格納、再使用、処理および管理できます。ブロックごとに独自の結果に基づいたキャッシュ・オブジェクトを作成できます。キャッシュされたオブジェクトは、HTML または XML のテキスト・フラグメント、XML DOM オブジェクトまたは Java のシリアライズ可能オブジェクトなどです。これらのオブジェクトは、HTTP セマンティックと関連付けると簡単にキャッシュできます。また、HTTP 外部で再利用できます。たとえば、キャッシュ内の XML オブジェクトを Simple Mail Transfer Protocol（SMTP）、Java Message Service（JMS）、Advanced Queueing（AQ）または Simple Object Access Protocol（SOAP）を介して出力する場合に再利用できます。

注意： この章では、主に Java Object Cache について説明します。3 種類のキャッシュとその相違点の詳細は、『Oracle Containers for J2EE JSP タグ・ライブラリおよびユーティリティ・リファレンス』を参照してください。

Java Object Cache の基本アーキテクチャ

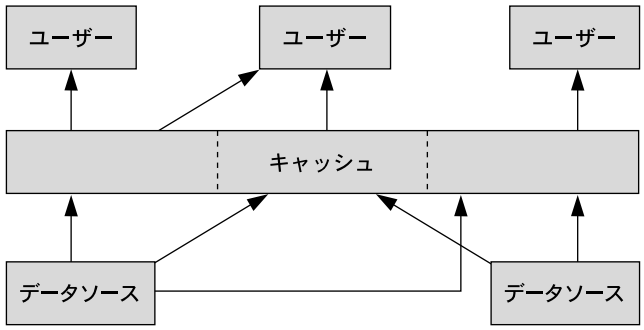
図 7-1「Java Object Cache の基本アーキテクチャ」は、Java Object Cache の基本アーキテクチャを示しています。キャッシュは、ユーザー・プロセスに情報を配信します。プロセスとは、HTML ページを生成するサーブレット・アプリケーションまたはその他の Java アプリケーションです。

Java Object Cache は、一般的なアプリケーションを対象とするインプロセスのプロセス全体のキャッシング・サービスです。つまり、オブジェクトはプロセスのメモリー領域内にキャッシュされ、Java Object Cache は単一のサービスとして、そのプロセス内で実行されるすべてのスレッドにより共有されます。これは、他のプロセス内で実行されるサービスとは異なります。Java Object Cache では、すべての Java オブジェクトを管理できます。キャッシュされたオブジェクトを共有しやすいように、キャッシュ内のオブジェクトはすべて名前でアクセスされます。キャッシング・サービスでは、キャッシュされるオブジェクトの構造に制限はありません。オブジェクトの名前、構造、タイプおよび元のソースは、すべてアプリケーションにより定義されます。

システム・リソースを最大限に活用するために、キャッシュ内のオブジェクトはすべて共有されます。共有されていても、キャッシュされたオブジェクトへのアクセスがアクセス・ロックによりシリアライズされることはなく、高水準の同時アクセスが可能です。オブジェクトが無効化または更新されると、そのオブジェクトの無効バージョンは、そのバージョンへの参照が存在するかぎりキャッシュに残っています。そのため、キャッシュにはオブジェクトの複数バージョンが同時に存在する可能性がありますが、有効なバージョンが複数存在することはありません。古いバージョンまたは無効なバージョンのオブジェクトは、無効化される前にそのバージョンを参照していたアプリケーションでのみ参照可能です。オブジェクトが更新されると、そのオブジェクトの新規コピーがキャッシュ内で作成され、古いバージョンには無効を示すマークが付けられます。

オブジェクトは、ユーザー提供の `CacheLoader` オブジェクトを使用してキャッシュにロードされます。このローダー・オブジェクトは、ユーザー・アプリケーションがキャッシュに存在しないオブジェクトをリクエストした時点で Java Object Cache によりコールされます。
図 7-1 に、このアーキテクチャを示します。アプリケーションはキャッシュとやり取りしてオブジェクトを取得し、キャッシュは `CacheLoader` を介してデータソースとやり取りします。このプロセスにより、オブジェクトの作成と使用が明確に分割されます。

図 7-1 Java Object Cache の基本アーキテクチャ



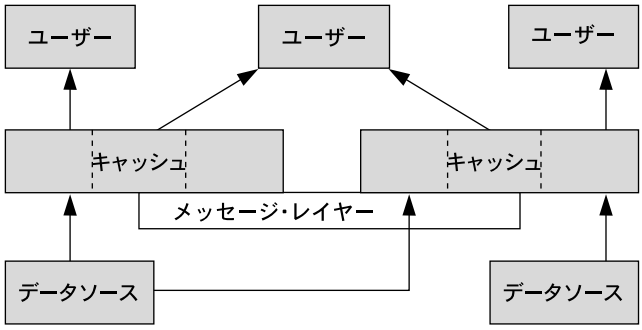
分散オブジェクト管理

Java Object Cache は、複数の Java プロセスが同じアプリケーションを実行したり、同じアプリケーションのかわりに動作している環境で使用できます。この環境の場合、同じオブジェクトを異なるプロセスでキャッシュできると便利です。簡易性、可用性およびパフォーマンスのために、Java Object Cache は各プロセス固有のものです。プロセスへのオブジェクトのロードは、集中管理されません。ただし、Java Object Cache では、プロセス間でオブジェクトの更新および無効化が調整されます。あるプロセス内でオブジェクトが更新または無効化された場合は、他のすべての関連プロセスでもそのオブジェクトが更新または無効化されます。この分散管理によって、集中管理によるオーバーヘッドを発生させることなく複数プロセスのシステムの同期が維持されます。

図 7-2「Java Object Cache の分散アーキテクチャ」に、次の動作を示します。

- アプリケーションが Java Object Cache とのやり取りによりオブジェクトを取得する方法
- Java Object Cache がデータソースとやり取りする方法
- Java Object Cache のキャッシュがキャッシュ・メッセージ・システムを介してキャッシュ・イベントを調整する方法

図 7-2 Java Object Cache の分散アーキテクチャ



Java Object Cache の動作

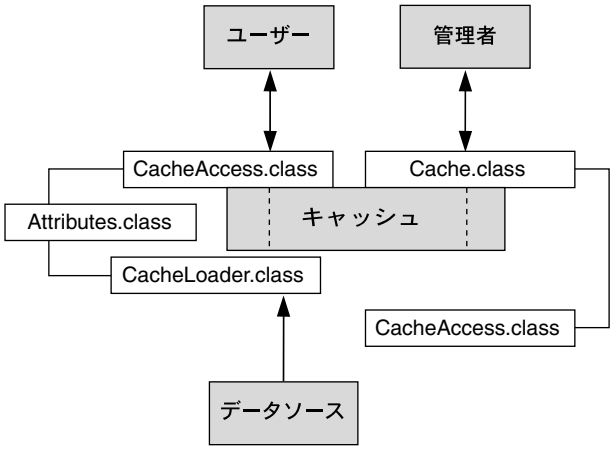
Java Object Cache は、プロセス内、プロセス間およびローカル・ディスク上の Java オブジェクトを管理します。Java Object Cache は、Java オブジェクトのローカル・コピーを管理することによって、Java のパフォーマンスを大幅に向上させる、強力で柔軟な使いやすいサービスを提供します。キャッシュできる Java オブジェクトの型や、オブジェクトの元のソースに関する制限はほとんどありません。プログラマは、Java Object Cache を使用して、取得や作成にコストがかかるオブジェクトを、キャッシュ・アクセスなしに管理します。

Java Object Cache は、新規および既存のアプリケーションに簡単に統合できます。オブジェクトは、ユーザー定義オブジェクト `CacheLoader` を使用してオブジェクト・キャッシュにロードでき、`CacheAccess` オブジェクトを使用してアクセスできます。`CacheAccess` オブジェクトは、ローカル・オブジェクト管理および分散オブジェクト管理をサポートします。Java Object Cache のほとんどの機能が、管理または構成を必要としません。拡張機能によって、`Cache` クラスの管理 API を使用した構成がサポートされます。管理には、ローカル・ディスク領域のネーミングやネットワーク・ポートの定義など、構成オプションの設定が含まれます。管理機能を使用すると、アプリケーションと Java Object Cache を完全に統合できます。

キャッシュされた各 Java オブジェクトには一連の属性が関連付けられており、オブジェクトをキャッシュにロードする方法、オブジェクトの格納場所、およびオブジェクトを無効化する方法が制御されます。キャッシュされたオブジェクトは、時間制御または明示的なリクエストに基づいて無効化されます（オブジェクトが無効化されたときに通知を行うことができます）。オブジェクトは、グループ単位または個別に無効化できます。

図 7-3「Java Object Cache の基本 API」は、Java Object Cache の基本 API を示しています。
図 7-3「Java Object Cache の基本 API」は、管理については示していません。

図 7-3 Java Object Cache の基本 API



キャッシュの編成

Java Object Cache は、次のように編成されます。

- **キャッシュ環境 :** キャッシュ環境には、キャッシュ・リージョン、サブリージョン、グループおよび属性が含まれます。キャッシュ・リージョン、サブリージョンおよびグループは、オブジェクトとオブジェクトの集合を関連付けます。属性は、キャッシュ・リージョン、サブリージョン、グループおよび個々のオブジェクトに関連付けられます。属性は、Java Object Cache によるオブジェクトの管理方法に影響を与えます。
- **キャッシュ・オブジェクト・タイプ :** キャッシュ・オブジェクト・タイプには、メモリー・オブジェクト、ディスク・オブジェクト、プール・オブジェクトおよび `StreamAccess` オブジェクトがあります。

表 7-1「キャッシュの組織化された構成メンバー」に、キャッシュ環境およびキャッシュ・オブジェクト・タイプの構成メンバーの要約を示します。

表 7-1 キャッシュの組織化された構成メンバー

キャッシュの構成メンバー	説明
属性	キャッシュ・リージョン、グループおよび個々のオブジェクトに関連付けられる機能。属性は、Java Object Cache によるオブジェクトの管理方法に影響を与えます。
キャッシュ・リージョン	Java Object Cache 内のキャッシュ・オブジェクトの集合を保持する組織化されたネームスペース。
キャッシュ・サブリージョン	親リージョン、サブリージョンまたはグループ内のキャッシュ・オブジェクトの集合を保持する組織化されたネームスペース。
キャッシュ・グループ	オブジェクト間の関連付けを定義するために使用される組織化された構成メンバー。リージョン内のオブジェクトは、グループ単位で無効化できます。グループ内のオブジェクトには、共通の属性を関連付けることができます。
メモリー・オブジェクト	メモリーに格納され、メモリーからアクセスされるオブジェクト。
ディスク・オブジェクト	ディスクに格納され、ディスクからアクセスされるオブジェクト。
プール・オブジェクト	Java Object Cache が管理する同一オブジェクトのセット。オブジェクトは、プールからチェックアウトされ、使用された後に戻されます。

表 7-1 キャッシュの組織化された構成メンバー（続き）

キャッシュの構成メンバー	説明
StreamAccess オブジェクト	Java の OutputStream を使用してロードされ、Java の InputStream を使用してアクセスされるオブジェクト。オブジェクトのサイズおよびキャッシュ容量に応じて、メモリーまたはディスクからアクセスします。

Java Object Cache の機能

Java Object Cache には、次の機能があります。

- オブジェクトを更新または無効化できます。
- オブジェクトを明示的に無効化するか、有効時間またはアイドル時間を指定する属性を使用して無効化できます。
- オブジェクトをプロセス間で調整できます。
- オブジェクトのロードと作成を自動化できます。
- オブジェクトのロードをプロセス間で調整できます。
- オブジェクトを、類似する特性によってキャッシュ・リージョンまたはグループで関連付けることができます。
- イベント処理や特別な処理に対してキャッシュ・イベント通知を提供します。
- キャッシュ管理属性を、オブジェクトごとに指定するか、キャッシュ・リージョンまたはグループに適用できます。

Java Object Cache のオブジェクト・タイプ

この項では、Java Object Cache で管理されるオブジェクト・タイプについて説明します。

- メモリー・オブジェクト
- ディスク・オブジェクト
- StreamAccess オブジェクト
- プール・オブジェクト

注意： オブジェクトは、任意の Java オブジェクトの名前で識別されます。名前の識別に使用される Java オブジェクトは、デフォルト Java オブジェクトの `equals` メソッドと `hashcode` メソッドをオーバーライドします。オブジェクトが分散されていて、更新されたりディスクに保存される可能性がある場合は、`Serializable` インタフェースを実装する必要があります。

メモリー・オブジェクト

メモリー・オブジェクトは、Java Object Cache で管理される Java オブジェクトです。メモリー・オブジェクトは、Java 仮想マシン（JVM）のヒープ領域に Java オブジェクトとして格納されます。メモリー・オブジェクトには、HTML ページ、データベース問合せの結果または Java オブジェクトとして格納できる、任意の情報を格納できます。

メモリー・オブジェクトは通常、アプリケーションが提供するローダーを使用して Java Object Cache にロードされます。メモリー・オブジェクトのソースは、外部にある場合があります（Oracle9*i* Database Server の表のデータを使用する場合など）。アプリケーションが提供するローダーは、ソースにアクセスしてメモリー・オブジェクトを作成または更新します。Java Object Cache がない場合、アプリケーションは、ローダーを使用せずにソースに直接アクセスする必要があります。

メモリー・オブジェクトを更新するには、メモリー・オブジェクトのプライベート・コピーを取得し、コピーに変更を加えた後、更新したオブジェクトを（`CacheAccess.replace()` メソッドを使用して）キャッシュに戻します。これにより、元のメモリー・オブジェクトが置き換えられます。

`CacheAccess.defineObject()` メソッドは、属性をオブジェクトに関連付けます。属性が定義されていない場合、オブジェクトは、その関連リージョン、サブリージョンまたはグループからデフォルト属性を継承します。

アプリケーションは、メモリー・オブジェクトをローカル・ディスクにスプールするようにリクエストできます（`SPOOL` 属性を使用）。この属性を設定すると、サイズが大きい、あるいは再作成のコストが高いためほとんど更新しないメモリー・オブジェクトを Java Object Cache で処理できます。ディスク・キャッシュがメモリー・キャッシュよりかなり大きいサイズに設定されている場合、ディスク上のオブジェクトはメモリー内のオブジェクトより長い時間ディスク・キャッシュに存在します。

ローカル・ディスクにスプールされたメモリー・オブジェクトを `DISTRIBUTE` 属性の分散機能と組み合せると、オブジェクトの永続性が提供されます（Java Object Cache が分散モードで動作している場合）。オブジェクトの永続性により、JVM の再起動後もオブジェクトが存続できます。

ディスク・オブジェクト

ディスク・オブジェクトは、ローカル・ディスクに格納され、Java Object Cache を使用してアプリケーションによってディスクから直接アクセスされます。ディスク・オブジェクトは、指定のディスク位置と `DISTRIBUTE` 属性の設定（および Java Object Cache が分散モードとローカル・モードのどちらで動作しているか）に応じて、Java Object Cache プロセス間で共有される場合と、特定のプロセスに限定される場合があります。

ディスク・オブジェクトは、明示的に無効化するか、`TimeToLive` 属性または `IdleTime` 属性を設定して無効化できます。Java Object Cache で追加領域が要求されたとき、参照されていないディスク・オブジェクトはキャッシュから削除される場合があります。

StreamAccess オブジェクト

StreamAccess オブジェクトは、Java の InputStream および OutputStream クラスを使用してアクセスされるように設定されている特殊なキャッシュ・オブジェクトです。StreamAccess オブジェクトへのアクセス方法は、オブジェクトのサイズとキャッシュの容量に基づいて、Java Object Cache によって決定されます。サイズの小さいオブジェクトはメモリーからアクセスされ、サイズの大きいオブジェクトはディスクから直接ストリームされます。streamAccess オブジェクトはすべてディスクに格納されます。

StreamAccess オブジェクトに対するキャッシュ・ユーザーのアクセスは、InputStream を使用して行われます。メモリー・オブジェクトおよびディスク・オブジェクトに適用される属性はすべて、StreamAccess オブジェクトにも適用されます。StreamAccess オブジェクトは、ストリームを管理する機能を提供しません。たとえば、StreamAccess オブジェクトはソケットのエンドポイントを管理できません。InputStream オブジェクトと OutputStream オブジェクトは、固定サイズの、潜在的に非常に大きいオブジェクトへのアクセスに使用できます。

プール・オブジェクト

プール・オブジェクトは、Java Object Cache で管理される特別なクラスのオブジェクトです。プール・オブジェクトには、同一オブジェクト・インスタンスのセットが含まれます。プール・オブジェクト自体は共有オブジェクトですが、プール内のオブジェクトはプライベート・オブジェクトです。プール内の個々のオブジェクトは、チェックアウトして使用した後、不要になるとプールに戻すことができます。

TimeToLive や IdleTime などの属性をプール・オブジェクトに関連付けることができます。これらの属性はプール・オブジェクト全体に適用されます。

Java Object Cache は、アプリケーション定義のファクトリ・オブジェクトを使用してプール内のオブジェクトをインスタンス化します。プールのサイズは、必要に応じて、および TimeToLive 属性または IdleTime 属性の値に基づいて増減します。プールの最小サイズは、プールの作成時に指定されます。最小サイズ値は、最小保証値ではなく、リクエストとして解釈されます。プール・オブジェクト内のオブジェクトは、領域不足によりキャッシュから削除されるため、プールのサイズはリクエストした最小値より小さくなる場合があります。プールの最大サイズの値を設定すると、プールで使用可能なオブジェクト数に関して強い制限を設けることができます。

Java Object Cache 環境

Java Object Cache 環境には、次のものがあります。

- キャッシュ・リージョン
- キャッシュ・サブリージョン
- キャッシュ・グループ
- リージョンとグループのサイズ制御
- キャッシュ・オブジェクトの属性

この項では、これらの Java Object Cache 環境の構成メンバーについて説明します。

キャッシュ・リージョン

Java Object Cache は、キャッシュ・リージョン内のオブジェクトを管理します。キャッシュ・リージョンは、キャッシュ内のネームスペースを定義します。キャッシュ・リージョン内の各オブジェクトには一意の名前を付ける必要があり、キャッシュ・リージョン名とオブジェクト名の組合せで、オブジェクトが一意に識別される必要があります。したがって、キャッシュ・リージョン名は他のリージョン名と異なる必要があり、リージョン内のすべてのオブジェクトに、リージョンに関して一意の名前を付ける必要があります（異なるリージョンまたはサブリージョン内に存在する場合は、複数のオブジェクトに同じ名前を付けることができます)。

アプリケーションのサポートに必要な数のリージョンを定義できます。ただし、ほとんどのアプリケーションで、必要となるリージョンは 1 つのみです。Java Object Cache は、デフォルト・リージョンを提供しています。リージョンが指定されない場合、オブジェクトはデフォルト・リージョンに配置されます。

リージョンに対して属性を定義できます。属性は、リージョン内のオブジェクト、サブリージョンおよびグループによって継承されます。

キャッシュ・サブリージョン

Java Object Cache は、キャッシュ・リージョン内のオブジェクトを管理します。キャッシュ・リージョン内にサブリージョンを指定すると、子の階層が定義されます。キャッシュ・サブリージョンは、キャッシュ・リージョンまたは上位のキャッシュ・サブリージョン内のネームスペースを定義します。キャッシュ・サブリージョン内の各オブジェクトには一意の名前を付ける必要があり、キャッシュ・リージョン名、キャッシュ・サブリージョン名およびオブジェクト名の組合せで、オブジェクトが一意に識別される必要があります。

アプリケーションのサポートに必要な数のサブリージョンを定義できます。

サブリージョンの定義時に属性が定義されない場合、サブリージョンは、その親であるリージョンまたはサブリージョンから属性を継承します。サブリージョンの属性は、サブリージョン内のオブジェクトによって継承されます。サブリージョンの親リージョンが無効化または破棄されると、サブリージョンも無効化または破棄されます。

キャッシュ・グループ

キャッシュ・グループは、リージョン内のオブジェクト間の関連を作成します。キャッシュ・グループによって、関連オブジェクトをまとめて操作できます。オブジェクトは通常、まとめて無効化する必要があったり、共通の属性を使用するため、キャッシュ・グループで関連付けられます。同じリージョンまたはサブリージョン内のキャッシュ・オブジェクトのセットは、キャッシュ・グループを使用して関連付けることができ、キャッシュ・グループの中に別のキャッシュ・グループを含めることもできます。

Java Object Cache オブジェクトは、ある時点で 1 つのグループにのみ属することができます。オブジェクトをグループに関連付ける前に、グループを明示的に作成する必要があります。グループは名前で定義されます。グループには独自の属性を設定できます。その親であるリージョン、サブリージョンまたはグループから属性を継承することもできます。

グループ名は、個々のオブジェクトの識別には使用されませんが、なんらかの共通点があるオブジェクトのセット（集合）を定義するために使用されます。グループは、階層ネームスペースを定義しません。オブジェクト・タイプでは、名前付けの目的でオブジェクトが区別されることはありません。したがって、リージョン内に同じ名前のグループとメモリー・オブジェクトを含めることはできません。リージョン内に階層ネームスペースを定義するには、サブリージョンを使用する必要があります。

グループの中にグループを含めることができ、親と子の関係を設定できます。子グループは、親グループから属性を継承します。

リージョンとグループのサイズ制御

10*g* リリース 3（10.1.3.1.0）の Java Object Cache では、リージョンまたはグループの最大サイズを、そこに含まれるオブジェクトの数または最大許容バイト数として指定できます。リージョンの容量をバイト数で制御する場合は、リージョン内のすべてのオブジェクトのサイズ属性を設定します。この属性は、オブジェクトの作成時にユーザーが直接設定できます。また、`Attributes.MEASURE` 属性フラグを設定すると、自動的に設定されます。リージョンまたはグループのサイズは、リージョン・レベルとサブリージョン・レベル、リージョンまたは別のグループ内のグループ・レベルなど、ネーミング階層の複数のレベルで設定できます。

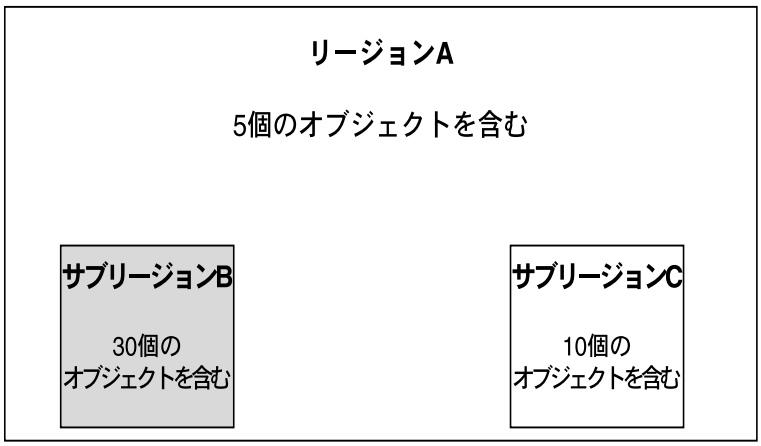
リージョンまたはグループの容量に達した場合、そのリージョンまたはグループに関連付けられている `CapacityPolicy` オブジェクトが定義されていれば、そのオブジェクトがコールされます。容量ポリシーが指定されていない場合は、デフォルトのポリシーが使用されます。デフォルトのポリシーでは、同等以下の優先順位を持ち参照されていないオブジェクトが見つかると、そのオブジェクトが新規オブジェクトのために無効化されます。オブジェクトの優先順位属性が設定されていない場合、優先順位は `Integer.MAX_VALUE` とみなされます。削除するオブジェクトの検索時には、そのリージョンまたはグループ内とすべてのサブリージョンおよびサブグループ内のオブジェクトがすべて検索されます。容量ポリシーに基づいて削除可能な最初のオブジェクトが削除されます。そのため、検索領域で最下位の優先順位を持つオブジェクトが削除されるとはかぎらない場合があります。

図 7-4「容量ポリシーの例」に、例を示します。

リージョン A の容量はオブジェクト 50 個に設定され、サブリージョン B とサブリージョン C の容量はそれぞれ 20 個に設定されています。リージョン A のオブジェクト数が 50 個に達し、そのうちの 10 個がリージョン A にあり、サブリージョン B および C にそれぞれ 20 個ずつある場合は、リージョン A に対する容量ポリシーがコールされます。削除されるオブジェクトは、リージョン A に含まれている場合と、サブリージョンのどちらかに含まれている場合があります。図 7-4「容量ポリシーの例」にこの状況を示します。

リージョン A の容量に達する前にサブリージョン B のオブジェクト数が 20 個になると、サブリージョン B に対する容量ポリシーがコールされ、サブリージョン B のオブジェクトのみが削除対象とみなされます。

図 7-4 容量ポリシーの例



キャッシュ・オブジェクトの属性

キャッシュ・オブジェクトの属性は、Java Object Cache によるオブジェクトの管理方法に影響を与えます。各オブジェクト、リージョン、サブリージョンおよびグループには、一連の属性が関連付けられます。オブジェクトに適用可能な属性は、デフォルトの属性値、オブジェクトの親であるリージョン、サブリージョンまたはグループから継承した属性値、あるいはそのオブジェクトに対してユーザーが設定した属性値のいずれかです。

属性は、次の 2 つのカテゴリに分けられます。

- 最初のカテゴリは、オブジェクトがキャッシュにロードされる前に定義する必要のある属性です。表 7-2「Java Object Cache の属性（オブジェクト作成時に設定）」に、これらの属性をまとめます。表 7-2「Java Object Cache の属性（オブジェクト作成時に設定）」に示されている各属性には、対応する `set` メソッドまたは `get` メソッドはありません（`LOADER` 属性を除く）。これらの属性を設定するには、`Attributes.setFlags()` メソッドを使用します。
- 第 2 のカテゴリは、オブジェクトがキャッシュに格納された後に変更できる属性です。表 7-3「Java Object Cache の属性」に、これらの属性をまとめます。

注意： 特定のタイプのオブジェクトには適用されない属性もあります。表 7-2「Java Object Cache の属性（オブジェクト作成時に設定）」および表 7-3「Java Object Cache の属性」の説明の「オブジェクト・タイプ」を参照してください。

オブジェクトのロード前に定義する属性の使用方法

表 7-2「Java Object Cache の属性（オブジェクト作成時に設定）」に示した属性は、オブジェクトのロード前に定義する必要があります。これらの属性は、オブジェクトの基本的な管理特性を決定します。

次のリストは、表 7-2「Java Object Cache の属性（オブジェクト作成時に設定）」に示した属性の設定に使用できるメソッドです（`Attributes` オブジェクト引数の値を設定します）。

- `CacheAccess.defineRegion()`
- `CacheAccess.defineSubRegion()`
- `CacheAccess.defineGroup()`
- `CacheAccess.defineObject()`
- `CacheAccess.getAccess()`
- `CacheAccess.getSubRegion()`
- `CacheAccess.put()`
- `CacheAccess.createPool()`
- `CacheLoader.createDiskObject()`
- `CacheLoader.createStream()`
- `CacheLoader.SetAttributes()`

注意： 表 7-2「Java Object Cache の属性（オブジェクト作成時に設定）」に示した属性は、`CacheAccess.resetAttributes()` メソッドを使用してリセットすることはできません。

表 7-2 Java Object Cache の属性（オブジェクト作成時に設定）

属性名	説明
DISTRIBUTE	オブジェクトがローカル・オブジェクトであるか分散オブジェクトであるかを指定します。Java Object Cache の分散キャッシング機能を使用している場合は、オブジェクトはローカル・オブジェクトとして設定されるため、更新および無効化はサイト内の他のキャッシュに伝播されません。 オブジェクト・タイプ : リージョン、サブリージョンまたはグループに設定されると、オブジェクトに固有の DISTRIBUTE 属性が明示的に設定されないかぎり、この属性は、リージョン、サブリージョンまたはグループ内のオブジェクトに対して DISTRIBUTE 属性のデフォルト値を設定します。プール・オブジェクトは常にローカルであるため、この属性は適用されません。 デフォルト値 : すべてのオブジェクトがローカルです。
GROUP_TTL_DESTROY	TimeToLive が期限切れになったときに、関連するオブジェクト、グループまたはリージョンを破棄することを示します。 オブジェクト・タイプ : リージョンまたはグループに設定されると、TimeToLive が期限切れになったときに、リージョンまたはグループ内のすべてのオブジェクト、およびリージョン、サブリージョンまたはグループ自体が破棄されます。 デフォルト値 : TimeToLive が期限切れになったとき、グループ・メンバー・オブジェクトのみが無効化されます。
LOADER	オブジェクトに関連付けられる CacheLoader を指定します。 オブジェクト・タイプ : リージョンまたはグループに設定されると、指定した CacheLoader が、そのリージョン、サブリージョンまたはグループのデフォルト・ローダーになります。LOADER 属性は、リージョンまたはグループ内のオブジェクトごとに指定します。 デフォルト値 : 設定されません。
ORIGINAL	オブジェクトが外部ソースからロードされたのではなく、キャッシュ内で作成されたことを示します。ORIGINAL オブジェクトは、参照件数が 0（ゼロ）になっても、キャッシュから削除されません。ORIGINAL オブジェクトは、不要になったときに明示的に無効化する必要があります。 オブジェクト・タイプ : リージョンまたはグループに設定されると、オブジェクトに固有の ORIGINAL 属性が設定されないかぎり、この属性は、リージョン、サブリージョンまたはグループ内のオブジェクトに対して ORIGINAL 属性のデフォルト値を設定します。 デフォルト値 : 設定されません。
REPLY	あるオブジェクトの更新または無効化のリクエストが完了した後、リモート・キャッシュから応答メッセージが送信されるように指定します。この属性は、キャッシュ間で高水準の整合性が必要な場合に設定してください。DISTRIBUTE 属性が設定されていない場合、あるいはキャッシュが非分散モードで開始された場合、REPLY は無視されます。 オブジェクト・タイプ : リージョンまたはグループに設定されると、オブジェクトに固有の REPLY 属性が明示的に設定されないかぎり、この属性は、リージョン、サブリージョンまたはグループ内のオブジェクトに対して REPLY 属性のデフォルト値を設定します。メモリー・オブジェクト、StreamAccess オブジェクトおよびディスク・オブジェクトの場合、この属性は、DISTRIBUTE 属性の値が DISTRIBUTE に設定されている場合にのみ適用されます。プール・オブジェクトは常にローカルであるため、この属性は適用されません。 デフォルト値 : 応答は送信されません。DISTRIBUTE がローカルに設定されている場合、REPLY 属性は無視されます。

表 7-2 Java Object Cache の属性（オブジェクト作成時に設定）（続き）

属性名	説明
SPOOL	領域を回復するために、キャッシュ・システムによってメモリーからメモリー・オブジェクトが削除されるときに、そのメモリー・オブジェクトを消去せずにディスクに格納するように指定します。この属性はメモリー・オブジェクトにのみ適用されます。オブジェクトが分散オブジェクトでもある場合、オブジェクトは、それをスプールしたプロセスの終了後も存続します。ローカル・オブジェクトにアクセスできるのは、それをスプールしているプロセスのみであるため、Java Object Cache が分散モードで動作していない場合、スプール・オブジェクトは、プロセスの終了時に失われます。 **注意 :** オブジェクトをスプールするには、シリアライズ可能であることが必要です。 オブジェクト・タイプ : リージョン、サブリージョンまたはグループに設定されると、オブジェクトに固有の SPOOL 属性が設定されないかぎり、この属性は、リージョン、サブリージョンまたはグループ内のオブジェクトに対して SPOOL 属性のデフォルト値を設定します。 デフォルト値 : メモリー・オブジェクトはディスクにスプールされません。
SYNCHRONIZE	このオブジェクトに対する更新を同期化する必要があることを示します。このフラグが設定されている場合、オブジェクトをロードまたは置換できるのは、そのオブジェクトの所有者のみとなります。所有権を取得するには、CacheAccess.getOwnership() メソッドを使用します。オブジェクトの所有者は CacheAccess オブジェクトです。SYNCHRONIZE 属性を設定することによって、ユーザーがオブジェクトの読取りまたは無効化を実行できなくなることはありません。 オブジェクト・タイプ : リージョン、サブリージョンまたはグループに設定すると、所有権の制限は、そのリージョン、サブリージョンまたはグループ全体に適用されます。プール・オブジェクトはこの属性を使用しません。 デフォルト値 : 更新は同期化されません。
SYNCHRONIZE_DEFAULT	リージョン、サブリージョンまたはグループ内のすべてのオブジェクトが同期化されることを示します。リージョン、サブリージョンまたはグループ内の各ユーザー・オブジェクトは、SYNCHRONIZE 属性でマーク付けされます。オブジェクトの所有権は、オブジェクトがロードまたは更新される前に取得する必要があります。 SYNCHRONIZE_DEFAULT 属性を設定することによって、ユーザーがオブジェクトの読取りまたは無効化を実行できなくなることはありません。したがって、SYNCHRONIZE 属性が設定されているオブジェクトの読取りまたは無効化には、所有権は必要ありません。 オブジェクト・タイプ : リージョン、サブリージョンまたはグループに設定すると、所有権は、そのリージョン、サブリージョンまたはグループ内の個々のオブジェクトに適用されます。プール・オブジェクトはこの属性を使用しません。 デフォルト値 : 更新は同期化されません。
ALLOWNULL	キャッシュが影響を受けるオブジェクトの有効な値として NULL を受け入れるように指定します。cacheLoader オブジェクトから戻される NULL オブジェクトは、ObjectNotFoundException を生成するのではなくキャッシュされます。 オブジェクト・タイプ : リージョン、サブリージョン、グループまたはプールに設定されると、オブジェクトに明示的に設定されないかぎり、この属性は、リージョン、サブリージョン、グループまたはプール内の各オブジェクトに個別に適用されます。 デフォルト値 : OFF（NULL は使用できません）。
MEASURE	オブジェクトがキャッシュにロードまたは置換されるときに、キャッシュされるオブジェクトのサイズ属性を自動的に計算することを指定します。キャッシュまたはリージョンの容量を、オブジェクトの数ではなくサイズに基づいて正確に制御できます。 オブジェクト・タイプ : リージョン、サブリージョンまたはグループに設定されると、オブジェクトに明示的に設定されないかぎり、この属性は、リージョン、サブリージョンまたはグループ内の各オブジェクトに個別に適用されます。 デフォルト値 : OFF（オブジェクトのサイズは自動的には計算されません）。

表 7-2 Java Object Cache の属性（オブジェクト作成時に設定）（続き）

属性名	説明
CapacityPolicy	リージョンまたはグループのサイズ制御に CapacityPolicy オブジェクトを使用するように指定します。この属性は、個々のオブジェクトに対して設定すると無視されます。 オブジェクト・タイプ : リージョン、サブリージョンまたはグループに設定すると、この属性はそのリージョンまたはグループ全体に適用されます。この属性は、個々のオブジェクトやプールには適用できません。 デフォルト値 : OFF（リージョンまたはグループに対する容量ポリシーは定義されません。リージョンまたはグループが容量に達すると、そのリージョンまたはグループ内の参照されていない最初のオブジェクトが無効化されます）。
Classloader	オブジェクトまたはオブジェクト名がディスクからインスタンス化される場合、またはそれらがネットワークを通じて別のキャッシュから受信される場合に使用するクラス・ローダーを指定します。 デフォルト値 : デフォルトでは、システム・クラス・ローダーが使用されます。これは、cache.jar をロードしたローダーです。

オブジェクトのロード前およびロード後に定義する属性の使用方法

一連の Java Object Cache の属性は、オブジェクトのロード前またはロード後に更新できます。表 7-3「Java Object Cache の属性」に、これらの属性を示します。これらの属性は、7-11 ページの「オブジェクトのロード前に定義する属性の使用方法」に示したリストのメソッドを使用して設定でき、CacheAccess.resetAttributes() メソッドを使用してリセットできます。

表 7-3 Java Object Cache の属性

属性名	説明
DefaultTimeToLive	リージョン、サブリージョンまたはグループ内のすべてのオブジェクトに対して個別に適用される TimeToLive 属性のデフォルト値を設定します。この属性は、リージョン、サブリージョンおよびグループにのみ適用されます。この値は、個々のオブジェクトに TimeToLive を設定すると上書きできます。 オブジェクト・タイプ : リージョン、サブリージョン、グループまたはプールに設定されると、オブジェクトに固有の TimeToLive が明示的に設定されないかぎり、この属性は、リージョン、サブリージョン、グループまたはプール内のすべてのオブジェクトに適用されます。 デフォルト値 : 自動的な無効化は行われません。
IdleTime	オブジェクトが無効化されるまでの、キャッシュに（参照件数 0（ゼロ）で）アイドル状態で存在する時間を指定します。TimeToLive 属性または DefaultTimeToLive 属性が設定されている場合、IdleTime 属性は無視されます。 オブジェクト・タイプ : リージョン、サブリージョン、グループまたはプールに設定されると、オブジェクトに IdleTime が明示的に設定されないかぎり、この属性は、リージョン、サブリージョン、グループまたはプール内の各オブジェクトに個別に適用されます。 デフォルト値 : IdleTime による自動的な無効化は行われません。
CacheEventListener	オブジェクトに関連付けられる CacheEventListener を指定します。 オブジェクト・タイプ : リージョン、サブリージョンまたはグループに設定されると、CacheEventListener がリージョン、サブリージョンまたはグループ内のオブジェクトに個別に指定されないかぎり、指定した CacheEventListener は、そのリージョン、サブリージョンまたはグループのデフォルトの CacheEventListener になります。 デフォルト値 : CacheEventListener は設定されません。

表 7-3 Java Object Cache の属性（続き）

属性名	説明
TimeToLive	オブジェクトが無効化されるまでにキャッシュに存在する最大時間を設定します。リージョン、サブリージョンまたはグループに関連付けられると、期限切れになった場合に、そのリージョン、サブリージョンまたはグループ内のすべてのオブジェクトが無効化されます。リージョン、サブリージョンまたはグループが破棄されない場合（つまり、GROUP_TTL_DESTROY が設定されていない場合）、TimeToLive の値はリセットされます。 オブジェクト・タイプ : リージョン、サブリージョン、グループまたはプールに設定されると、オブジェクトに固有の TimeToLive が明示的に設定されないかぎり、この属性は、リージョン、サブリージョン、グループまたはプール全体に適用されます。 デフォルト値 : 自動的な無効化は行われません。
Version	アプリケーションで、キャッシュ内のオブジェクトの各インスタンスに対して Version が設定される場合があります。Version は、アプリケーションの利便性および確認に利用できます。キャッシング・システムではこの属性は使用されません。 オブジェクト・タイプ : リージョン、サブリージョン、グループまたはプールに設定されると、オブジェクトに固有の Version が明示的に設定されないかぎり、この属性は、リージョン、サブリージョン、グループまたはプール内のすべてのオブジェクトに適用されます。 デフォルト値 : デフォルトの Version は 0 です。
Priority	容量に達した時点で、どのオブジェクトがキャッシュまたはリージョンから削除されるかを制御します。この属性は整数で、キャッシュ、リージョンまたはグループのサイズ制御に使用する CapacityPolicy オブジェクトで使用できます。数が大きいほど優先順位が高くなります。リージョンとグループの容量制御では、空き容量（特に他のオブジェクト用）を増やすためにオブジェクトが削除される場合、優先順位の低いオブジェクトをキャッシュに入れるために優先順位の高いオブジェクトが削除されることはありません。キャッシュの容量制御では、優先順位の高いオブジェクトをキャッシュに入れるために優先順位の低いオブジェクトが削除対象として選択されます。 オブジェクト・タイプ : リージョン、サブリージョン、グループまたはプールに設定されると、オブジェクトに明示的に設定されないかぎり、この属性は、リージョン、サブリージョン、グループまたはプール内の各オブジェクトに個別に適用されます。 デフォルト値 : integer.MAX_VALUE。
MaxSize	リージョンまたはグループに使用可能な最大バイト数を指定します。この属性をオブジェクトに対して指定すると無視されます。 オブジェクト・タイプ : リージョン、サブリージョンまたはグループに設定すると、この属性はそのリージョンまたはグループ全体に適用されます。この属性は、個々のオブジェクトやプールには適用できません。 デフォルト値 : 無制限。
MaxCount	リージョンまたはグループに格納できるオブジェクトの最大数を指定します。この属性をオブジェクトに対して指定すると無視されます。 オブジェクト・タイプ : リージョン、サブリージョンまたはグループに設定すると、この属性はそのリージョンまたはグループ全体に適用されます。この属性は、個々のオブジェクトやプールには適用できません。 デフォルト値 : 無制限。
ユーザー定義属性	属性をユーザーが定義できます。この種の属性は、オブジェクト、グループまたはリージョンに関連付けられる名前 / 値ペアです。CapacityPolicy オブジェクトとの併用を意図した属性ですが、必要に応じてキャッシュ・ユーザーが定義できます。 オブジェクト・タイプ : リージョン、サブリージョン、グループまたはプールに設定されると、オブジェクトに対して明示的にリセットされないかぎり、これらの属性は、リージョン、サブリージョン、グループまたはプール内の各オブジェクトに使用できます。 デフォルト値 : デフォルトでは、ユーザー定義属性は設定されません。

Java Object Cache を使用したアプリケーションの開発

この項では、Java Object Cache を使用するアプリケーションの開発方法を説明します。この項には、次の項目が含まれます。

Java Object Cache のインポート

Oracle インストーラによって、Java Object Cache の JAR ファイル `cache.jar` が、`$OLE_HOME/javacache/lib` ディレクトリ（UNIX の場合）または `%OLE_HOME%¥javacache¥lib` ディレクトリ（Windows の場合）にインストールされます。

Java Object Cache を使用するには、次のように `oracle.ias.cache` をインポートします。

```
import oracle.ias.cache.*;
```

キャッシュ・リージョンの定義

Java Object Cache へのアクセスはすべて、キャッシュ・リージョンに関連付けられている `CacheAccess` オブジェクトを使用して行われます。キャッシュ・リージョンは通常、static メソッド `CacheAccess.defineRegion()` を使用して、アプリケーション名に関連付けて定義します。キャッシュが初期化されていない場合は、`defineRegion()` によって Java Object Cache が初期化されます。

リージョンを定義するときに属性も設定できます。属性は、Java Object Cache によるオブジェクトの管理方法を指定します。`Attributes.setLoader()` メソッドは、キャッシュ・ローダーの名前を設定します。例 7-1「CacheLoader の名前の設定」にこの設定を示します。

例 7-1 CacheLoader の名前の設定

```
Attributes attr = new Attributes();
MyLoader mloader = new MyLoader;
attr.setLoader(mloader);
attr.setDefaultTimeToLive(10);

final static String APP_NAME_ = "Test Application";
CacheAccess.defineRegion(APP_NAME_, attr);
```

`defineRegion` の最初の引数は、`String` を使用してリージョン名を設定します。この static メソッドは、Java Object Cache 内にプライベート・リージョン名を作成します。2 番目の引数は、デフォルトのキャッシュ属性を使用して、新規リージョンの属性を定義します。

キャッシュ・グループの定義

キャッシュ内の複数のオブジェクト間の関連を作成する場合は、キャッシュ・グループを作成します。オブジェクトは通常、まとめて無効化する必要があったり、共通の属性セットを使用するため、キャッシュ・グループで関連付けられます。

同じリージョンまたはサブリージョン内のキャッシュ・オブジェクトのセットは、キャッシュ・グループを使用して関連付けることができ、このキャッシュ・グループには別のキャッシュ・グループを含めることもできます。オブジェクトをキャッシュ・グループに関連付ける前に、キャッシュ・グループを定義する必要があります。キャッシュ・グループは名前で定義されます。また、固有の属性を使用するか、あるいはその親であるキャッシュ・グループ、サブリージョンまたはリージョンから属性を継承できます。例 7-2「キャッシュ・グループの定義」のコードは、`Test Application` という名前のリージョン内にキャッシュ・グループを定義します。

例 7-2 キャッシュ・グループの定義

```
final static String APP_NAME_ = "Test Application";
final static String GROUP_NAME_ = "Test Group";
// obtain an instance of CacheAccess object to a named region
CacheAccess caccess = CacheAccess.getAccess(APP_NAME_);
// Create a group
caccess.defineGroup(GROUP_NAME_);
// Close the CacheAccess object
caccess.close();
```

キャッシュ・サブリージョンの定義

リージョン内またはすでに定義されているサブリージョン内にプライベート・ネームスペースを作成する場合は、サブリージョンを定義します。サブリージョンのネームスペースは、親のネームスペースから独立しています。リージョン内では、異なるサブリージョン内に存在する場合は、2 つのオブジェクトに同じ名前を付けることができます。

サブリージョンには、キャッシュ・オブジェクト、グループまたは別のサブリージョンなど、リージョンに含めることができるすべてのものを含めることができます。オブジェクトをサブリージョンに関連付ける前に、サブリージョンを作成する必要があります。キャッシュ・サブリージョンは名前で定義されます。また、固有の属性を使用するか、あるいはその親であるキャッシュ・リージョンまたはサブリージョンから属性を継承できます。サブリージョンの親を取得するには、`getParent()` メソッドを使用します。

例 7-3「キャッシュ・サブリージョンの定義」のコードは、`Test Application` という名前のリージョン内にキャッシュ・サブリージョンを定義します。

例 7-3 キャッシュ・サブリージョンの定義

```
final static String APP_NAME_ = "Test Application";
final static String SUBREGION_NAME_ = "Test SubRegion";
// obtain an instance of CacheAccess object to a named region
CacheAccess caccess = CacheAccess.getAccess(APP_NAME_);
// Create a SubRegion
caccess.defineSubRegion(SUBREGION_NAME_);
// Close the CacheAccess object
caccess.close();
```

キャッシュ・オブジェクトの定義と使用

状況によっては、個々のオブジェクトがロードされる前に、キャッシュ内のオブジェクトの管理方法を Java Object Cache に指示できます。CacheLoader.load() メソッド内に属性を設定することで、オブジェクトのロード時に管理オプションを指定できます。ただし、CacheAccess.defineObject() メソッドを使用して、属性をオブジェクトに関連付けることもできます。オブジェクトに対する属性が定義されていない場合、Java Object Cache は、そのオブジェクトが関連付けられているリージョン、サブリージョンまたはグループのデフォルトの属性セットを使用します。

例 7-4「キャッシュ属性の設定」は、キャッシュ・オブジェクトの属性の設定方法を示しています。この例は、リージョン APP_NAME_ がすでに定義されていることを前提としています。

例 7-4 キャッシュ属性の設定

```
import oracle.ias.cache.*;
final static String APP_NAME_ = "Test Application";
CacheAccess cacc = null;
try
{
   cacc = CacheAccess.getAccess(APP_NAME_);
// set the default IdleTime for an object using attributes
   Attributes attr = new Attributes();
// set IdleTime to 2 minutes
   attr.setIdleTime(120);

// define an object and set its attributes
   cacc.defineObject("Test Object", attr);

// object is loaded using the loader previously defined on the region
// if not already in the cache.
   result = (String)cacc.get("Test Object");
}  catch (CacheException ex){
     // handle exception
   } finally {
      if (cacc!= null)
         cacc.close();
}
```

CacheLoader オブジェクトの実装

Java Object Cache には、オブジェクトをキャッシュにロードするためのメカニズムが 2 つ用意されています。

- アプリケーションで CacheAccess.put() メソッドを使用してオブジェクトをキャッシュに直接入れることができます。
- CacheLoader オブジェクトを実装できます。

ほとんどの場合は、CacheLoader を実装する方法をお薦めします。キャッシュ・ローダーを使用すると、オブジェクトのリクエスト時に、オブジェクトをキャッシュにロードする必要があるかどうかが Java Object Cache によって自動的に判断されます。また、Java Object Cache では、オブジェクトが同時に複数のユーザーからリクエストされた場合に、ロードが調整されます。

CacheLoader オブジェクトをリージョン、サブリージョン、グループまたはオブジェクトに関連付けることができます。CacheLoader を使用すると、Java Object Cache でオブジェクトのロードをスケジュールおよび管理し、「オブジェクトがキャッシュ内に存在しない場合はロードする」というロジックを処理できます。

オブジェクトがキャッシュ内に存在しない場合、アプリケーションで CacheAccess.get() または CacheAccess.preLoad() メソッドがコールされると、キャッシュによって CacheLoader.load メソッドが実行されます。load メソッドが戻されると、Java Object Cache は戻されたオブジェクトをキャッシュに挿入します。CacheAccess.get() の使用時に

キャッシュがいっぱいの場合、オブジェクトはローダーから戻された後、すぐにキャッシュ内で無効化されます（したがって、キャッシュがいっぱいの状態で CacheAccess.get() メソッドを使用した場合、CacheFullException は生成されません）。

リージョン、サブリージョンまたはグループに対して定義されている場合、CacheLoader は、そのリージョン、サブリージョンまたはグループに関連付けられているすべてのオブジェクトのデフォルト・ローダーとみなされます。個々のオブジェクトに対して定義されている CacheLoader オブジェクトは、そのオブジェクトのロードにのみ使用されます。

注意： リージョン、サブリージョンまたはグループ、あるいは複数のキャッシュ・オブジェクトに対して定義されている CacheLoader オブジェクトは、同時アクセスを考慮して記述する必要があります。CacheLoader オブジェクトが共有されるため、実装はスレッド・セーフであることが必要です。

CacheLoader のヘルパー・メソッドの使用方法

CacheLoader は、load() メソッドの実装内で使用できるヘルパー・メソッドをいくつか提供しています。表 7-4「load() で使用される CacheLoader メソッド」に、使用可能な CacheLoader メソッドをまとめます。

表 7-4 load() で使用される CacheLoader メソッド

メソッド	説明
setAttributes()	ロードされるオブジェクトの属性を設定します。
netSearch()	使用可能な他のキャッシュを検索して、ロード対象のオブジェクトを探します。オブジェクトは、リージョン名、サブリージョン名およびオブジェクト名で一意に識別されます。
getName()	ロードされるオブジェクトの名前を戻します。
getRegion()	ロードされるオブジェクトに関連付けられたリージョンの名前を戻します。
createStream()	StreamAccess オブジェクトを作成します。
createDiskObject()	ディスク・オブジェクトを作成します。
getFromRemote()	指定されたリモート・キャッシュからオブジェクトを取得します。
exceptionHandler()	非キャッシュ例外を CacheExceptions に変換し、そのベースを元の例外に設定します。
log()	キャッシュ・サービスのログにメッセージを記録します。

例 7-5「CacheLoader の実装」は、ロードされるオブジェクトが分散 Java Object Cache キャッシュで使用可能かどうかを、cacheLoader.netSearch() メソッドを使用してチェックする CacheLoader オブジェクトを示しています。netSearch() によってオブジェクトが見つからない場合、ロード・メソッドは、よりコストがかかるコールを使用してオブジェクトを取得します（コストがかかるコールには、リモート Web サイトへの HTTP 接続や Oracle9*i* Database Server への接続などがあります）。この例の場合、Java Object Cache は、結果を String として格納します。

例 7-5 CacheLoader の実装

```
import oracle.ias.cache.*;
class YourObjectLoader extends CacheLoader{
      public YourObjectLoader () {
      }
      public Object load(Object handle, Object args) throws CacheException
     {
         String contents;
         // check if this object is loaded in another cache
         try {
            contents = (String)netSearch(handle, 5000);// wait for up to 5 scnds
            return new String(contents);
         } catch(ObjectNotFoundException ex){}

         try {
            contents =  expensiveCall(args);
            return new String(contents);
         } catch (Exception ex) {throw exceptionHandler("Loadfailed", ex);}
           }

    private String expensiveCall(Object args) {
        String str = null;
        // your implementation to retrieve the information.
        // str = ...
        return str;
    }
 }
```

キャッシュ・オブジェクトの無効化

オブジェクトをキャッシュから削除するには、オブジェクト、グループ、サブリージョンまたはリージョンに TimeToLive 属性を設定するか、オブジェクトを明示的に無効化または破棄します。

オブジェクトを無効化すると、そのオブジェクトに、キャッシュから削除されたことを示すマークが付けられます。リージョン、サブリージョンまたはグループを無効化すると、リージョン、サブリージョンまたはグループ内の個々のオブジェクトすべてが無効化されますが、すべてのグループ、ローダーおよび属性などの環境は使用可能なままキャッシュに残ります。オブジェクトを無効化してもオブジェクトの定義は解除されません。オブジェクト・ローダーはその名前に関連付けられたままになります。オブジェクトをキャッシュから完全に削除するには、CacheAccess.destroy() メソッドを使用します。

オブジェクトは、TimeToLive 属性または IdleTime 属性に基づいて自動的に無効化することもできます。TimeToLive または IdleTime が期限切れになったとき、デフォルトではオブジェクトは無効化され、破棄されません。

オブジェクト、グループ、サブリージョンまたはリージョンが分散として定義されている場合、無効化リクエストは分散環境内のすべてのキャッシュに伝播されます。

オブジェクト、グループ、サブリージョンまたはリージョンを無効化するには、次のように CacheAccess.invalidate() メソッドを使用します。

```
CacheAccess cacc = CacheAccess.getAccess("Test Application");
cacc.invalidate("Test Object");  // invalidate an individual object
cacc.invalidate("Test Group"); // invalidate all objects associated with a group
cacc.invalidate();     // invalidate all objects associated with the region cacc
cacc.close();          // close the CacheAccess handle
```

キャッシュ・オブジェクトの破棄

オブジェクトをキャッシュから削除するには、オブジェクト、グループ、サブリージョンまたはリージョンに TimeToLive 属性を設定するか、オブジェクトを明示的に無効化または破棄します。

オブジェクトを破棄すると、オブジェクトとその関連環境（関連するすべてのローダー、イベント・ハンドラおよび属性など）に、キャッシュから削除されたことを示すマークが付けられます。リージョン、サブリージョンまたはグループを破棄すると、リージョン、サブリージョンまたはグループに関連付けられているすべてのオブジェクト（関連環境も含む）に、削除されたことを示すマークが付けられます。

オブジェクトは、TimeToLive 属性または IdleTime 属性に基づいて自動的に破棄することもできます。デフォルトでは、オブジェクトは無効化され、破棄はされません。オブジェクトを破棄する必要がある場合は、属性 GROUP_TTL_DESTROY を設定します。リージョンを破棄すると、リージョンのアクセスに使用された CacheAccess オブジェクトもクローズされます。

オブジェクト、グループ、サブリージョンまたはリージョンを破棄するには、次のように CacheAccess.destroy() メソッドを使用します。

```
CacheAccess cacc = CacheAccess.getAccess("Test Application");
cacc.destroy("Test Object"); // destroy an individual object
cacc.destroy("Test Group");  // destroy all objects associated with
                             // the group "Test Group"

cacc.destroy();       // destroy all objects associated with the region
                      // including groups and loaders
```

複数のオブジェクトのロードおよび無効化

ほとんどの場合、オブジェクトはキャッシュに個別にロードされますが、複数のオブジェクトが 1 セットとしてキャッシュにロードされる場合があります。その主な例は、データベースからの 1 回の読取りでキャッシュされたオブジェクトが複数作成される場合です。この場合は、CacheLoader.load メソッドの 1 回のコールで複数のオブジェクトを作成する方が効率的です。

この使用例をサポートするために、抽象クラス CacheListLoader と CacheAccess.loadList メソッドが追加されています。CacheListLoader オブジェクトは、抽象メソッド loadList とヘルパー・メソッド getNextObject、getList、getNamedObject および saveObject を定義して CacheLoader オブジェクトを拡張します。キャッシュ・ユーザーは CacheListLoader.loadList メソッドを実装します。ヘルパー・メソッドを使用すると、ユーザーはオブジェクト・リストを反復してオブジェクトを個別に作成し、キャッシュに保存できます。CacheLoader に定義されているヘルパー・メソッドが CacheListLoader メソッドから使用される場合は、最初に getNextObject または getNamedObject をコールして正しいコンテキストを設定する必要があります。

CacheAccess.loadList メソッドは、ロードされるオブジェクト名の配列を引数として取ります。キャッシュは、このオブジェクト配列を処理します。現在キャッシュ内に存在しないオブジェクトは、キャッシュされたオブジェクトに対して定義されている CacheListLoader オブジェクトに渡されるリストに追加されます。オブジェクトに対して CacheListLoader オブジェクトが定義されていない場合、または異なる CacheListLoader オブジェクトが定義されている場合、各オブジェクトは定義済の CacheLoader.load メソッドを使用して個別にロードされます。

最善の方法は、CacheListLoader.loadList メソッドと CacheListLoader.load メソッドの両方を実装することです。どちらのメソッドがコールされるかは、ユーザーがキャッシュに対してリクエストする順序によって決定されます。たとえば、CacheAccess.loadList メソッドの前に CacheAccess.get メソッドがコールされると、CacheAccess.loadList メソッドではなく CacheListLoader.load メソッドが使用されます。

利便性を考慮して、無効化メソッドと破棄メソッドはオブジェクト配列も処理するようにオーバーロードされています。

例 7-6「サンプル CacheListLoader」にサンプル CacheListLoader を示し、例 7-7「使用例」にその使用例を示します。

例 7-6 サンプル CacheListLoader

```
Public class ListLoader extends CacheListLoader
{
   public void loadList(Object handle, Object args) throws CacheException
   {
      while(getNextObject(handle) != null)
      {
         // create the cached object based on the name of the object
         Object  cacheObject = myCreateObject(getName(handle));
         saveObject(handle, cacheObject);
      }
   }

   public Object load(Object handle, Object args) throws CacheException
   {
      return myCreateObject(getName(handle));
   }

   private Object myCreateObject(Object name)
   {
      // do whatever to create the object
   }
}
```

例 7-7 使用例

```
// Assumes the cache has already been initialized

CacheAccess   cacc;
Attributes    attr;
ListLoader    loader = new
ListLoader();
String        objList[];
Object        obj;

// set the CacheListLoader for the region
attr  = new Attributes();
attr.setLoader(loader);

//define the region and get access to the cache
CacheAccess.defineRegion("region name", attr);
cacc = CacheAccess.getAccess("region name");

// create the array of object names
objList = new String[3];
for (int j = 0; j < 3; j++)
      objList[j] = "object " + j;

// load the objects in the cache using the CacheListLoader.loadList method
cacc.loadList(objList);
```

```
// retrieve the already loaded object from the cache
obj = cacc.get(objList[0]);

// do something useful with the object

// load an object using the CacheListLoader.load method
obj = cache.get("another object")

// do something useful with the object
```

Java Object Cache の構成

Java Object Cache は、OC4J の起動時に自動的に初期化されません。次の例に示すように、opmn.xml で -Doracle.ias.jcache=true を使用して、OC4J に jcache を初期化させることができます。

```
   <ias-component id="OC4J">
            <process-type id="home" module-id="OC4J" status="enabled">
              <module-data>
                  <category id="start-parameters">
                    <data id="java-options" value="-server
-Djava.security.policy=$OLE_HOME/j2ee/home/config/java2.policy
 -Djava.awt.headless=true
-DApplicationServerDebug=true
-Ddatasource.verbose=true
-Djdbc.debug=true -Doracle.ias.jcache=true"/>
                  </category>
                  <category id="stop-parameters">
                    <data id="java-options"
value="-Djava.security.policy=$OLE_HOME/j2ee/home/config/java2.
policy -Djava.awt.headless=true"/>
                  </category>
              </module-data>
              <start timeout="600" retry="2"/>
              <stop timeout="120"/>
              <restart timeout="720" retry="2"/>
              <port id="ajp" range="3301-3400"/>
              <port id="rmi" range="3201-3300"/>
              <port id="jms" range="3701-3800"/>
              <process-set id="default_group" numprocs="1"/>
            </process-type>
        </ias-component>
```

OC4J ランタイムは、javacache.xml ファイルに定義されている構成設定を使用して、Java Object Cache を初期化します。ファイルのパスは、OC4J の server.xml ファイルの <javacache-config> タグに指定されます。server.xml では、javacache.xml の相対パスのデフォルト値は次のとおりです。

```
<javacache-config path="../../../javacache/admin/javacache.xml"/>
```

javacache.xml の記述ルールとデフォルトの構成値は、XML スキーマ内で指定されます。XML スキーマ・ファイル ora-cache.xsd とデフォルトの javacache.xml は、$OLE_HOME/javacache/admin ディレクトリ（UNIX の場合）および %OLE_HOME%¥javacache¥admin ディレクトリ（Windows の場合）にあります。

server.xml のリファレンス・ドキュメントは、『Oracle Containers for J2EE 構成および管理ガイド』の付録 B「OC4J で使用される構成ファイル」の「OC4J サーバー構成ファイル（server.xml）の概要」を参照してください。

以前のリリース（リリース 10*g*（9.0.4）より前）の Java Object Cache では、構成には javacache.properties ファイルを使用していました。リリース 10*g*（9.0.4）以上は、Java Object Cache の構成に javacache.xml を使用します。

注意： javacache.properties を使用するリリース（リリース 10*g* (9.0.4) より前）と javacache.xml を使用するリリース（リリース 10*g* (9.0.4) 以上）の両方を同じホストにインストールする場合は、javacache.xml の discovery-port 属性と javacache.properties の coordinatorAddress 属性が同じポートに構成されていないことを確認する必要があります。同じポートに構成されている場合は、どちらか一方の値を異なるポート番号に手動で変更してください。デフォルトの範囲は 7000 ～ 7099 です。

構成例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<cache-configuration
xmlns=http://www.oracle.com/oracle/ias/cache/configuration
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.oracle.com/oracle/ias/cache/configuration
ora-cache.xsd">
  <logging>
    <location>javacache.log</location>
    <level>ERROR</level>
  </logging>
  <communication>
    <isDistributed>true</isDistributed>
    <discoverer discovery-port="7000"/>
  </communication>
  <persistence>
    <location>diskcache</location>
    <disksize>32</disksize>
  </persistence>
  <max-objects>1000</max-objects>
  <max-size>48</max-size>
  <clean-interval>30</clean-interval>
</cache-configuration>
```

表 7-5「Java Object Cache の構成プロパティ」に、有効なプロパティ名と、各プロパティの有効な型を示します。

表 7-5 Java Object Cache の構成プロパティ

構成の XML 要素	説明	型
clean-interval	各キャッシュのクリーンの間隔を秒で指定します。Java Object Cache は、キャッシュのクリーンの間隔で、オブジェクトに関連付けられている TimeToLive 属性または IdleTime 属性によって無効化されたオブジェクトがないかチェックします（これらの属性については、表 7-3 を参照してください）。 デフォルト値：60	正の整数
ping-interval	リモート・キャッシュ・システムの可用性を判断するために、各キャッシュの終了が検出される間隔を秒単位で指定します。 デフォルト値：60	正の整数
max-size	Java Object Cache で使用可能なメモリーの最大サイズを MB で指定します。 デフォルト値：10	正の整数
max-objects	キャッシュで許可されるメモリー内オブジェクトの最大数を指定します。この件数には、グループ・オブジェクト、ディスクにスプールされているオブジェクトまたは現在メモリーにないオブジェクトは含まれません。 デフォルト値：5000	正の整数

表 7-5 Java Object Cache の構成プロパティ（続き）

構成の XML 要素	説明	型
preload-file	宣言的なキャッシュ構成ファイルへのフルパスを指定します。このファイルのフォーマットは、宣言的なキャッシュ・スキーマ（cache.xsd）に準拠する必要があります。宣言的なキャッシュ構成では、システムは Java Object Cache サービスの初期化時に、キャッシュのリージョン、グループ、オブジェクト、属性およびポリシーを事前に定義できます。宣言的なキャッシュの詳細は、7-27 ページの「宣言的なキャッシュ」を参照してください。7-26 ページの「例」も参照してください。 **注意 :** 宣言的なキャッシュの XML スキーマのファイル・パスは、*OLE_HOME*/javacache/admin/cache.xsd です。宣言的なキャッシュ・ファイルを記述する場合は、XML スキーマを参照してください。 デフォルト値 : 宣言的なキャッシュは使用されません。	String
communication	キャッシュが分散されているかどうかを示します。分散キャッシングの使用時に、Java Object Cache がキャッシング・システムに加わるために最初に接続する IP アドレスとポートを指定します。 distribute プロパティがオブジェクトに設定されている場合、そのオブジェクトの更新および無効化は、Java Object Cache で認識されている他のキャッシュに伝播されます。 communication 要素の isDistributed サブ要素が false に設定されていると、オブジェクトの属性セットが分散に設定されている場合でも、すべてのオブジェクトがローカルとして処理されます。7-26 ページの「例」を参照してください。 デフォルト値 : キャッシュは分散されません（isDistributed サブ要素は false に設定されます）。	複合（サブ要素あり）
logging	ログ・ファイル名やログ・レベルなどのログ出力属性を指定します。ログ・レベルに使用可能なオプションは、OFF、FATAL、ERROR、DEFAULT、WARNING、TE、INFO および DEBUG です。7-26 ページの「例」を参照してください。 これらのログ・レベルは、廃止予定です。Java キャッシュでは、現在、java.util.logging.Level の指定レベルが使用されています。 デフォルトのログ・ファイル名 : UNIX の場合 : $OLE_HOME/javacache/admin/logs/javacache.log Windows の場合 : %OLE_HOME%¥javacache¥admin¥logs¥javacache.log デフォルトのログ・レベル : DEFAULT	複合（サブ要素あり）
persistence	ディスク・キャッシュのルートへの絶対パスやディスク・キャッシュの最大サイズなど、ディスク・キャッシュ構成を指定します。ルート・パスを指定すると、ディスク・キャッシュのデフォルト最大サイズは 10MB となります。ディスク・キャッシュのサイズは MB 単位です。7-26 ページの「例」を参照してください。 デフォルト値 : ディスク・キャッシングは使用できません。	複合（サブ要素あり）

注意 : 構成プロパティは、Attributes クラスを使用して指定する Java Object Cache の属性とは異なります。

例

次の例に、<preload-file> 要素の使用方法を示します。

- 宣言的なキャッシュ構成ファイルを指定します。

```
<preload-file>/app/oracle/javacache/admin/decl.xml</preload-file>
```

次の例に、<communication> 要素の使用方法を示します。

- 分散キャッシュをオフにします。

```
<communication>
  <isDistributed>false</isDistributed>
</communication>
```

- ローカル・マシンの複数の JVM 間でキャッシュを分散します。

```
<communication>
  <isDistributed>true</isDistributed>
</communication>
```

- Java Object Cache がローカル・ノードのキャッシング・システムに加わるために最初に接続する初期検出ポートを指定します。

```
<communication>
  <isDistributed>true</isDistributed>
  <discoverer discovery-port="7000">
</communication>
```

- Java Object Cache がキャッシング・システムに加わるために最初に接続する IP アドレスと初期検出ポートを指定します。

```
<communication>
<isDistributed>true</isDistributed>
<discoverer ip="192.10.10.10" discovery-port="7000">
</communication>
```

- Java Object Cache がキャッシング・システムに加わるために最初に接続する、複数の IP アドレスと初期検出ポートを指定します。最初に指定したアドレスにアクセスできない場合は、次に指定したアドレスに接続されます。

```
<communication>
  <isDistributed>true</isDistributed>
  <discoverer ip="192.10.10.10" discovery-port="7000">
  <discoverer ip="192.11.11.11" discovery-port="7000">
  <discoverer ip="192.22.22.22" discovery-port="7000">
  <discoverer ip="192.22.22.22" discovery-port="8000">
</communication>
```

次の例に、<persistence> 要素の使用方法を示します。

- デフォルトのディスク・サイズを使用してディスク・キャッシュのルート・パスを指定します。

```
<persistence>
  <location>/app/9iAS/javacache/admin/diskcache</location>
</persistence>
```

- ディスク・サイズが 20MB のディスク・キャッシュのルート・パスを指定します。

```
<persistence>
  <location>/app/9iAS/javacache/admin/diskcache</location>
  <disksize>20</disksize>
</persistence>
```

次の例に、<logging> 要素の使用方法を示します。

- ログ・ファイル名を指定します。

```
<logging>
<location>/app/9iAS/javacache/admin/logs/my_javacache.log</location>
</logging>
```

- ログ・レベルとして INFO を指定します。

```
<logging>
<location>/app/9iAS/javacache/admin/logs/my_javacache.log</location>
<level>INFO</level>
</logging>
```

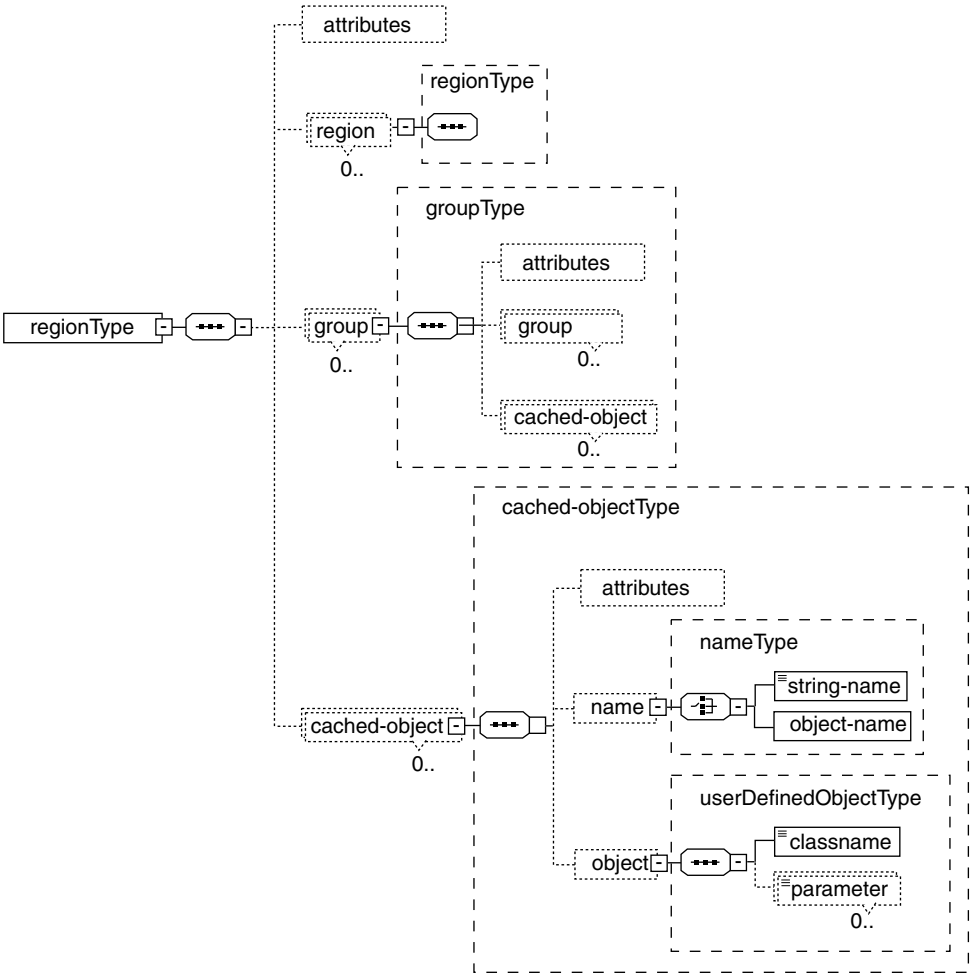
宣言的なキャッシュ

Java Object Cache 10*g* リリース 3（10.1.3.1.0）では、オブジェクト、グループ、リージョン、キャッシュ属性を宣言的に定義できます。宣言的キャッシュを使用する場合、アプリケーションでキャッシュ・オブジェクトと属性を定義するために Java コードを記述する必要はありません。

宣言的なキャッシュ・ファイルは、Java Object Cache の初期化時に自動的に読み取ることができます。宣言的なキャッシュ・ファイルの位置は、キャッシュ構成ファイルの <preload-file> 要素内で指定します。また、宣言的なキャッシュ・ファイルは、プログラムによってロードするか、oracle.ias.cache.Configurator.class のパブリック・メソッドを使用して明示的にロードできます。宣言的なキャッシュ・ファイルを複数使用することもできます。

図 7-5「宣言的なキャッシュのアーキテクチャ」に、宣言的なキャッシュを示します。

図 7-5 宣言的なキャッシュのアーキテクチャ



システムの初期化時に宣言的なキャッシュ・ファイルが自動的にロードされるように、Java Object Cache を設定できます。例 7-8「宣言的なキャッシュを自動的にロードする方法」にこの設定を示します。例 7-9「宣言的なキャッシュ・ファイルをプログラムで読み取る方法」には、宣言的なキャッシュ・ファイルをプログラムで読み取る方法を示します。

例 7-8 宣言的なキャッシュを自動的にロードする方法

```
<!-- Specify declarative cache file:my_decl.xml in javacache.xml -->
<cache-configuration>
  …
<preload-file>/app/9iAS/javacache/admin/my_decl.xml</preload-file>
  …
</cache-configuration>
```

例 7-9 宣言的なキャッシュ・ファイルをプログラムで読み取る方法

```
try {
  String filename = "/app/9iAS/javacache/admin/my_decl.xml";
Configurator config = new Configurator(filename);
Config.defineDeclarable();
} catch (Exception ex) {
}
```

宣言的なキャッシュ・ファイルの例

```
<?xml version="1.0" encoding="UTF-8"?>
<cache xmlns="http://www.javasoft.com/javax/cache"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.oracle.com/javax/cache">
  <region name="fruit">
    <attributes>
      <time-to-live>3000</time-to-live>
      <max-count>200</max-count>
      <capacity-policy>
        <classname>com.acme.MyPolicy</classname>
      </capacity-policy>
    </attributes>
    <group name="apple">
      <attributes>
        <flag>spool</flag>
        <flag>distribute</flag>
        <cache-loader>
          <classname>com.acme.MyLoader</classname>
          <parameter name="color">red</parameter>
        </cache-loader>
      </attributes>
    </group>
    <cached-object>
      <name>
        <string-name>theme</string-name>
      </name>
      <object>
        <classname>com.acme.DialogHandler</classname>
        <parameter name="prompt">Welcome</parameter>
      </object>
    </cached-object>
  </region>
</cache>
```

宣言的なキャッシュ・ファイルの形式

宣言的なキャッシュ・ファイルは XML 形式です。このファイルの内容は、Oracle Application Server 10*g* に付属する宣言的なキャッシュの XML スキーマに準拠する必要があります。この XML スキーマのファイル・パスは、*OLE_HOME*/javacache/admin/cache.xsd です。

表 7-6「宣言的なキャッシュのスキーマの説明（cache.xsd）」に、宣言的なキャッシュ・スキーマの要素、その子および各要素の有効な型を示します。ほとんどの要素の使用方法を示すコードについては、7-32 ページの「例」を参照してください。

表 7-6 宣言的なキャッシュのスキーマの説明（cache.xsd）

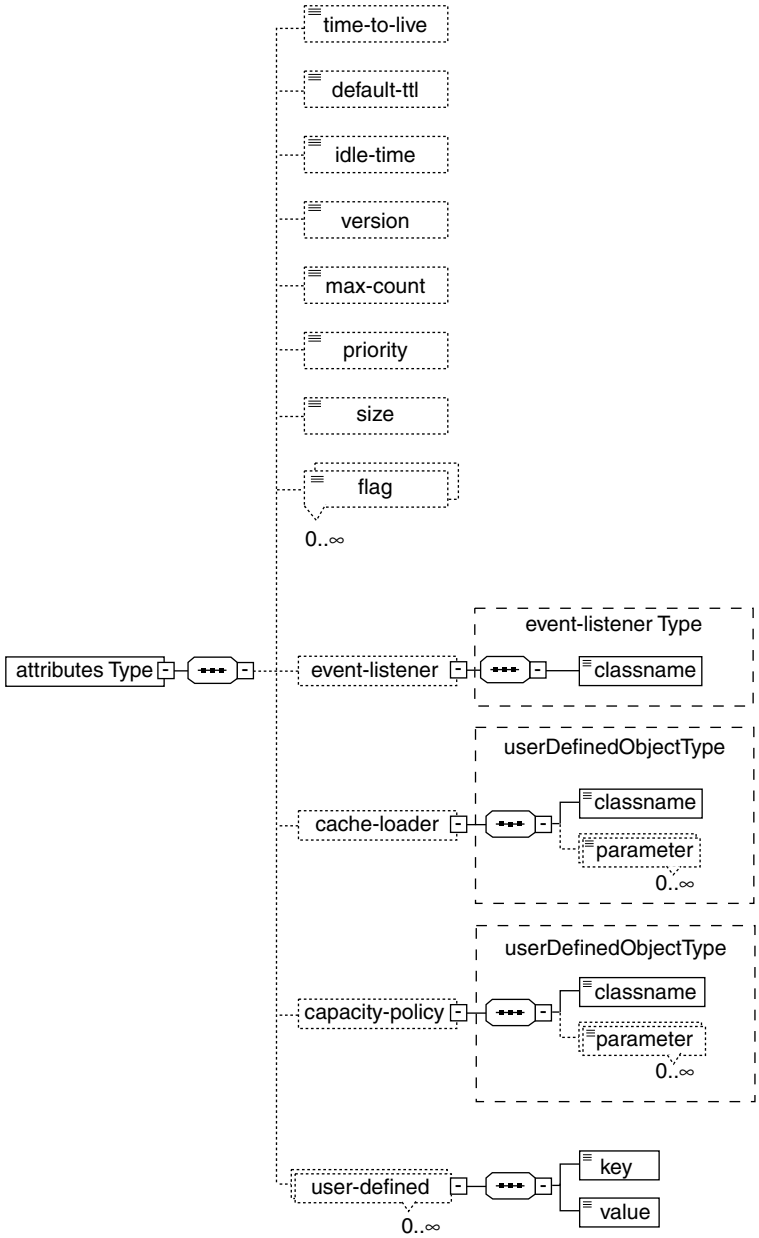
要素	説明	子	型
region	キャッシュのリージョンまたはサブリージョンを宣言します。	<attributes> <region> <group> <cached-object>	regionType
group	キャッシュのグループまたはサブグループを宣言します。	<attributes> <group> <cached-object>	groupType
cached-object	キャッシュ・オブジェクトを宣言します。	<attributes> <name> <object>	objectType

表 7-6 宣言的なキャッシュのスキーマの説明（cache.xsd）（続き）

要素	説明	子	型
name	キャッシュされるオブジェクトの名前を宣言します。この名前には、単純な文字列型または指定した Java オブジェクトの型を使用できます。	<string-name> <object-name>	nameType
object	ユーザー定義の Java オブジェクトを宣言します。指定するオブジェクトのクラスでは、`oracle.ias.cache` パッケージの宣言可能なインタフェースを実装する必要があります。	<classname> <parameter>	userDefinedObjectType
attributes	キャッシュ・リージョン、グループまたはキャッシュ・オブジェクトの attributes オブジェクトを宣言します。子要素は、それぞれ `oracle.ias.cache` パッケージの attributes クラスの各フィールドに対応します。詳細は、`Attributes.class` の Javadoc を参照してください。	<time-to-live> <default-ttl> <idle-time> <version> <max-count> <priority> <size> <flag> <event-listener> <cache-loader> <capacity-policy> <user-defined>	attributesType
event-listener	`CacheEventListener` オブジェクトを宣言します。	<classname>	event-listenerType
cache-loader	`CacheLoader` オブジェクトを宣言します。	<classname> <parameter>	userDefinedObjectType
capacity-policy	`CapacityPolicy` オブジェクトを宣言します。	<classname> <parameter>	userDefinedObjectType
user-defined	ユーザー定義の文字列型属性を宣言します。	<key> <value>	element

図 7-6「宣言的なキャッシュ・スキーマの属性」に、宣言的なキャッシュ・スキーマの属性を示します。

図 7-6 宣言的なキャッシュ・スキーマの属性



例

次の例に、表 7-6「宣言的なキャッシュのスキーマの説明（cache.xsd）」にリストした要素の使用方法を示します。

- <region> 要素を使用してキャッシュのリージョンとサブリージョンを宣言します。

```
<region name="themes">
    <region name="cartoon">
      <!-- sub region definition -->
    </region>
  <group name="colors">
    <!-- group definition -->
  </group>
</region>
```

- <group> 要素を使用してキャッシュのグループとサブグループを宣言します。

```
<group name="colors">
  <group name="dark">
    <!-- sub group definition -->
  </group>
</group>
```

- <cached-object> 要素を使用して、キャッシュされるオブジェクトを宣言します。

```
<cached-object>
  <name>
    <string-name>DialogHandler</string-name>
  </name>
  <object>
    <classname>com.acme.ConfigManager</classname>
    <parameter name="color">blue</parameter>
  </object>
</cached-object>
```

- <name> 要素で文字列を使用して、キャッシュされるオブジェクトの名前を宣言します。

```
<name>
  <string-name>DialogHandler</string-name>
</name>
```

<name> 要素でオブジェクトを使用して、キャッシュされるオブジェクトの名前を宣言します。

```
<name>
  <object-name>
    <classname>DialogHandler</classname>
    <parameter name="color">green</parameter>
  </object-name>
</name>
```

- <object> 要素を使用して、ユーザー定義の Java オブジェクトを宣言します。

```
<object>
  <classname>com.acme.CustomConfigManager</classname>
  <parameter name="color">blue</parameter>
</object>
```

```
// Implementation of CustomConfigManager.java
package com.acme;
import oracle.ias.cache.Declarable;
public class CustomConfigManager implements Declarable {
}
```

- <attributes> 要素を使用して、キャッシュ・リージョン、グループまたはキャッシュ・オブジェクトの attributes オブジェクトを宣言します。

```
<attributes>
  <time-to-live>4500</time-to-live>
  <default-ttl>6000</default-ttl>
  <version>99</version>
  <max-count>8000</max-count>
  <priority>50</priority>
  <flag>spool</flag>
  <flag>allownull</flag>
  <flag>distribute</flag>
  <flag>reply</flag>
  <cache-loader>
    <classname>MyLoader</classname>
    <parameter name="debug">false</parameter>
  </cache-loader>
</attributes>
```

- <user-defined> 要素を使用して、ユーザー定義の文字列型の属性を宣言します。

```
<attributes>
  <user-defined>
    <key>color</key>
    <value>red</value>
  </user-defined>
</attributes>
```

宣言可能なユーザー定義オブジェクト

キャッシュ・オブジェクト、オブジェクト属性およびユーザー定義オブジェクトのトポロジは、すべて宣言的なキャッシュ・ファイルに記述できます。宣言的なキャッシュ・ファイルで宣言されているユーザー定義の Java オブジェクト（CacheLoader、CacheEventListener および CapacityPolicy など）がシステムでロードおよびインスタンス化されるようにするには、そのオブジェクトを oracle.ias.cache.Declarable インタフェースのインスタンスにする必要があります。つまり、宣言的なキャッシュ・ファイルで宣言されているすべての Java オブジェクトについて、oracle.ias.cache.Declarable インタフェースを実装します。すべてのユーザー定義の Java オブジェクトは、アプリケーションのクラス・ローダーではなく JVM のデフォルト・クラス・ローダーによってロードされることに注意してください。宣言可能なオブジェクトがインスタンス化されると、システムにより、その init(Properties props) メソッドが暗黙的に起動されます。このメソッドは、宣言的なキャッシュ・ファイルに定義されているユーザー指定のパラメータ（名前 / 値ペア）を使用して、必要な初期化タスクを実行します。例 7-10「パラメータで宣言的に渡すことによるオブジェクトの定義」に、パラメータ（color = yellow）で宣言的に渡すことでオブジェクトを定義する方法を示します。

例 7-10 パラメータで宣言的に渡すことによるオブジェクトの定義

宣言的な XML ファイル内で次のように記述します。

```
<cached-object>
  <name>
    <string-name>Foo</string-name>
  </name>
  <object>
    <classname>com.acme.MyCacheObject</classname>
    <parameter name="color">yellow</parameter>
  </object>
</cached-object>
```

宣言可能なオブジェクトの実装は次のとおりです。

```
package com.acme;

import oracle.ias.cache.*;
import java.util.Properties;

public class MyCacheObject implements Declarable {

  private String color_;

  /**
    * Object initialization
    */
  public void init(Properties prop) {
        color_ = prop.getProperty("color");
  }
}
```

宣言可能な CacheLoader、CacheEventListener および CapacityPolicy

宣言的なキャッシュ・ファイル内で CacheLoader、CacheEventListener または CapacityPolicy オブジェクトを指定する場合は、そのオブジェクト自体が oracle.ias.cache.Declarable のインスタンスでもあることが必要です。この要件は、ユーザー定義オブジェクトの要件と同様です。必要な抽象クラスを拡張する他に、指定のオブジェクトごとに宣言可能なインタフェースを実装する必要があります。例 7-11「宣言可能な CacheLoader の実装」に、宣言可能な CacheLoader の実装を示します。

例 7-11 宣言可能な CacheLoader の実装

```
import oracle.ias.cache.*;
import java.util.Properties;

public class MyCacheLoader extends CacheLoader implements Declarable {

  public Object load(Object handle, Object argument) {
    // should return meaningful object based on argument
    return null;
  }

  public void init(Properties prop) {
  }
}
```

非 OC4J コンテナでの Java Object Cache の初期化

Java アプリケーション内で Java Object Cache を使用して非 OC4J ランタイムで実行するには、アプリケーション（Java クラス）が初期化される場所に次の参照を挿入する必要があります。

```
Cache.open(/path-to-ocnfig-file/javacache.xml);
```

コードでパラメータを指定せずに Cache.open() を起動すると、Java Object Cache では内部のデフォルト構成パラメータが使用されます。また、Cache.init(CacheAttributes) を起動して Java Object Cache を初期化することもできます。これにより、独自の構成ファイルから構成パラメータを導出するか、プログラム的に生成できます。

OC4J ランタイムで Java Object Cache が使用されない場合は、JVM が起動される CLASSPATH に cache.jar を組み込む必要があります。また、Cache.open(String *config_filename*) を起動するか（*config_filename* は有効な javacache.xml ファイルへのフルパス）、Cache.init(CacheAttributes) を起動して、Java Object Cache を明示的に初期化します。

次のいずれかのメソッド起動を使用して、非 OC4J コンテナ内で Java Object Cache を明示的に初期化します。

- `Cache.open();`

  `cache.jar` ファイルに格納されているデフォルトの Java Object Cache 構成を使用します。

- `Cache.open(/path-to-oracle-home/javacache/admin/javacache.xml);`

  `javacache.xml` ファイルに定義されている構成を使用します。

- `Cache.open(/path-to-user's-own-javacache.xml);`

  特定の `javacache.xml` ファイルに定義されている構成を使用します。

- `Cache.init(CacheAttributes);`

  `CacheAttributes` オブジェクト内で設定されている構成を使用します。

OC4J コンテナ内で実行される J2EE アプリケーションの場合、`javacache.xml` ファイルへのパスは OC4J の `server.xml` 構成ファイル内で構成できます。キャッシュは、OC4J プロセスの起動時に自動的に初期化できます。詳細は、OC4J の構成を参照してください。

非 OC4J コンテナでは、前述のメソッド起動を使用しない場合、`Cache.getAccess()` または `Cache.defineRegion()` を起動すると、Java Object Cache が（`cache.jar` に格納されているデフォルトの構成設定を使用して）暗黙的に初期化されます。

容量制御

容量制御機能を使用すると、キャッシュ・ユーザーは、キャッシュ、リージョンまたはグループの容量に達したときに、どのオブジェクトをキャッシュから削除するかを決定するためのポリシーを指定できます。ポリシーを指定するには、抽象クラス `CapacityPolicy` を拡張し、キャッシュ、リージョンまたはグループの属性としてインタンス化されたオブジェクトを設定します。

リージョンおよびグループの場合は、そのリージョンまたはグループが容量に達し、新規オブジェクトがロードされるときに、`CapacityPolicy` オブジェクトがコールされます。リージョンまたはグループ内で無効化するオブジェクトが見つからないと、新規オブジェクトはキャッシュに保存されません（ユーザーには戻されますが、即時に無効化されます）。

キャッシュの容量がなんらかの最高水位標（構成された最大使用率）に達した場合、キャッシュ全体に関連付けられている `CapacityPolicy` オブジェクトがコールされます。最高水位標に達すると、キャッシュはオブジェクトを削除してキャッシュ内のロードを最高水位標より 3% 下げようとします。この最高水位標は、`capacityBuffer` キャッシュ属性で指定されます。`capacityBuffer` が 5 に設定されている場合、キャッシュは使用率が 95%（100% - 5%）になるとオブジェクトの削除を開始し、使用率が 92%（95% - 3%）になるまで削除を続行します。`capacityBuffer` のデフォルト値は 15 です。

キャッシュには、特定のリージョンまたはグループに使用するものとは異なる容量ポリシーを使用できます。

デフォルトで、グループおよびリージョンに対する容量ポリシーでは、新規オブジェクトが追加されるときに容量に達している場合は、同等以下の優先順位を持つ参照されていないオブジェクトが削除されます。キャッシュの場合、デフォルト・ポリシーでは、オブジェクトの優先順位に従って過去 2 回のクリーン間隔中に参照されていないオブジェクトが削除されます。つまり、優先順位が低く最近参照されていないオブジェクトから順番に削除されます。

容量ポリシーを作成しやすいように、キャッシュ内のオブジェクトに関して多数の統計が保持され、キャッシュ、リージョンおよびグループ間で集計されます。この統計は `CapacityPolicy` オブジェクトで使用できます。キャッシュ・オブジェクトの場合は、次の統計が保持されます。

- 優先順位
- アクセス回数 : オブジェクトが参照された回数
- サイズ : オブジェクトのバイト数（使用可能な場合）
- 最終アクセス時間 : オブジェクトが最後にアクセスされた時間（ミリ秒）

- 作成時間 : オブジェクトが作成された時間（ミリ秒）
- ロード時間 : オブジェクトのミリ秒単位のロード所要時間（オブジェクトが CacheAccess.put でキャッシュに追加された場合、この値は 0（ゼロ）です。）

これらの統計とともに、オブジェクトに関連付けられている属性すべてを CapacityPolicy オブジェクトで使用できます。

キャッシュ、リージョンおよびグループの場合は、次の集計統計が保持されます。これらの統計ごとに、下限、上限および平均値が保持されます。これらの統計は、クリーン間隔ごと、または Cache.updateStats() のコール時に再計算されます。

- 優先順位
- アクセス回数 : オブジェクトが参照された回数
- サイズ : オブジェクトのバイト数（使用可能な場合）
- 最終アクセス時間 : オブジェクトが最後にアクセスされた時間（ミリ秒）
- ロード時間 : オブジェクトのミリ秒単位のロード所要時間（オブジェクトが CacheAccess.put でキャッシュに追加された場合、この値は 0（ゼロ）です。）

例 7-12「オブジェクト・サイズに基づくサンプル CapacityPolicy」に、オブジェクト・サイズに基づくリージョンのサンプル CapacityPolicy オブジェクトを示します。

例 7-12 オブジェクト・サイズに基づくサンプル CapacityPolicy

```
class SizePolicy extends CapacityPolicy
{
   public boolean policy (Object victimHandle, AggregateStatus aggStatus,
    long currentTime , Object newObjectHandle) throws CacheException
   {
      int             newSize;
      int             oldSize;

      oldSize = getAttributes(victimHandle).getSize();
      newSize = getAttributes(newObjectHandle).getSize();
      if (newSize >= oldSize)
         return true;
      return false;
   }
```

例 7-13「アクセス時間と参照回数に基づくサンプル CapacityPolicy」に、アクセス時間と参照回数に基づくキャッシュのサンプル CapacityPolicy を示します。オブジェクトの参照回数が平均値を下回っており、過去 30 秒アクセスされていないと、キャッシュから削除されます。

例 7-13 アクセス時間と参照回数に基づくサンプル CapacityPolicy

```
class SizePolicy extends CapacityPolicy
{
public boolean policy (Object victimHandle, AggregateStatus aggStatus, long
 currentTime , Object newObjectHandle) throws CacheException
{
   long          lastAccess;
   int           accessCount;
   int           avgAccCnt;

   lastAccess    = getStatus(victimHandle).getLastAccess();
   accessCount   = getStatus(victimHandle).getAccessCount();
   avgAccCnt     = aggStatus.getAccessCount(AggregateStatus.AVG);

   if (lastAccess + 30000 < currentTime && accessCount < avgAccCnt)
      return true;
   }

}
```

キャッシュ・イベント・リスナーの実装

キャッシュ内のオブジェクトのライフ・サイクルでは、オブジェクトの作成やオブジェクトの無効化など、多数のイベントが発生する可能性があります。この項では、キャッシュ・イベントが発生したときのアプリケーションへの通知方法について説明します。

オブジェクトの作成の通知を受信するには、cacheLoader の一部としてイベント通知を実装します。無効化または更新の通知の場合は、CacheEventListener を実装し、Attributes.setCacheEventListener() を使用して CacheEventListener をオブジェクト、グループ、リージョンまたはサブリージョンに関連付けます。

CacheEventListener は、java.util.EventListener を拡張するインタフェースです。キャッシュ・イベント・リスナーには、登録済のコールバック・メソッドを設定する機能があり、イベント発生時に実行されます。Java Object Cache では、イベント・リスナーは、キャッシュ内のオブジェクトが無効化または更新された場合に実行されます。

イベント・リスナーは、キャッシュ内のオブジェクト、グループ、リージョンまたはサブリージョンに関連付けられます。イベント・リスナーがグループ、リージョンまたはサブリージョンに関連付けられた場合、リスナーは、デフォルトでそのグループ、リージョンまたはサブリージョン自体が無効化されたときにのみ実行されます。その中のメンバーが無効化された場合、イベントはトリガーされません。Attributes.setCacheEventListener() のコールは Boolean 引数を使用します。この値が true の場合、イベント・リスナーは、リージョン、サブリージョンまたはグループ自体ではなく、リージョン、サブリージョンまたはグループの各メンバーに適用されます。この場合、リージョン、サブリージョンまたはグループ内のオブジェクトが無効化されると、イベントがトリガーされます。

CacheEventListener インタフェースには、1 つのメソッド handleEvent() があります。このメソッドは、1 つの引数 CacheEvent オブジェクト（java.util.EventObject を拡張します）を使用します。このオブジェクトには、イベント処理に役立つ次のような複数のメソッドがあります。

- getID(): イベントのタイプ（OBJECT_INVALIDATION、OBJECT_UPDATED または OBJECT_UPDATED_SYNC）を戻します。
- getSource(): 無効化されたオブジェクトを戻します。グループおよびリージョンの場合、getSource() メソッドはグループ名またはリージョン名を戻します。
- getName(): イベントに関連付けられたオブジェクトの名前を戻します。
- getRegion(): イベントに関連付けられたオブジェクトを含むリージョンを戻します。
- getReason(): イベントの原因を戻します。このメソッドは、現在のところ無効化イベントにのみ適用されます。

handleEvent() メソッドは、Java Object Cache が管理するバックグラウンド・スレッドのコンテキスト内で実行されます。必要なスレッド・コンテキストが使用可能でない場合があるため、このメソッドでは Java ネイティブ・インタフェース（JNI）コードを使用しないでください。

例 7-14「CacheEventListener の実装」に、CacheEventListener を実装し、オブジェクトまたはグループに関連付ける方法を示します。

例 7-14 CacheEventListener の実装

```
import oracle.ias.cache.*;

    // A CacheEventListener for a cache object
    class MyEventListener implements CacheEventListener
    {

        public void handleEvent(CacheEvent ev) throws CacheException
        {
           MyObject obj = (MyObject)ev.getSource();
           obj.cleanup();
        }
     }
```

```
class MyObject
{
  public void cleanup()
  {
    // do something
  }
}

import oracle.ias.cache.*;

   // A CacheEventListener for a group object
    class MyGroupEventListener implements CacheEventListener
    {
       public void handleEvent(CacheEvent ev) throws CacheException
       {
          String groupName = (String)ev.getSource();
          notify("group " + groupName + " has been invalidated");

       }
       void notify(String str)
       {
         // do something
       }
    }
```

Attributes.setCacheEventListener() メソッドを使用して、リージョン、サブリージョン、グループまたはオブジェクトの CacheEventListener を指定します。

例 7-15「オブジェクトのキャッシュ・イベント・リスナーの設定」に、オブジェクトにキャッシュ・イベント・リスナーを設定する方法を示します。また、例 7-16「グループのキャッシュ・イベント・リスナーの設定」に、グループにキャッシュ・イベント・リスナーを設定する方法を示します。

例 7-15 オブジェクトのキャッシュ・イベント・リスナーの設定

```
import oracle.ias.cache.*;

   class YourObjectLoader extends CacheLoader
   {
      public YourObjectLoader () {
      }

      public Object load(Object handle, Object args) {
         Object obj = null;
         Attributes attr = new Attributes();
         MyEventListener el = new MyEventListener();
         attr.setCacheEventListener(CacheEvent.OBJECT_INVALIDATED, el);

         // your implementation to retrieve or create your object

         setAttributes(handle, attr);
         return obj;
    }
}
```

例 7-16 グループのキャッシュ・イベント・リスナーの設定

```
import oracle.ias.cache.*;
try
{
   CacheAccess cacc = CacheAccess.getAccess(myRegion);
   Attributes attr = new Attributes ();

   MyGroupEventListener listener = new MyGroupEventListener();
   attr.setCacheEventListener(CacheEvent.OBJECT_INVALIDATED, listener);

   cacc.defineGroup("myGroup", attr);
   //....
   cacc.close();

}catch(CacheException ex)
{
   // handle exception
}
```

制限事項およびプログラミングに関する注意点

この項では、Java Object Cache を使用するときの制限事項およびプログラミングに関する注意点について説明します。

- CacheAccess オブジェクトは、スレッド間で共有しないでください。このオブジェクトは、キャッシング・システムに対するユーザーを表します。CacheAccess オブジェクトには、キャッシュに対するユーザー・アクセスの現在の状態（現在アクセスされているオブジェクト、現在所有されているオブジェクトなど）が含まれています。CacheAccess オブジェクトを共有する必要はなく、共有した場合の結果は予測できません。
- CacheAccess オブジェクトは、同時に 1 つのキャッシュ済オブジェクトに対する参照のみを保持します。複数のキャッシュ済オブジェクトが同時にアクセスされる場合は、複数の CacheAccess オブジェクトを使用します。メモリーに格納されるオブジェクトについては、この作業は重要ではありません。これは、Java では、キャッシュが参照されていない場合でも、キャッシュ済オブジェクトはガベージ・コレクションの対象とならないためです。ディスク・オブジェクトについては、キャッシュ参照がメンテナンスされない場合は、基礎となるファイルが別のユーザーや時間ベースの無効化によって削除され、予期しない例外が発生する可能性があります。リソース管理を最適化するには、キャッシュ済オブジェクトが使用されている間、キャッシュ参照をオープンのままにします。
- CacheAccess オブジェクトは、使用しなくなったときは必ずクローズしてください。CacheAccess オブジェクトはプールされます。これらは、ユーザーのかわりにキャッシュ・リソースを取得します。アクセス・オブジェクトが使用されなくなったときにクローズされないと、これらのリソースがプールに戻されず、JVM によってガベージ・コレクションの対象となるまでクリーン・アップされません。CacheAccess オブジェクトが絶えず割り当てられ、クローズされない場合は、パフォーマンスが低下することがあります。
- ローカル・オブジェクト（Attributes.DISTRIBUTE 属性が設定されていないオブジェクト）が CacheAccess.save() メソッドを使用してディスクに保存された場合は、プロセスの終了後、このオブジェクトは存続しません。定義により、ローカル・オブジェクトを参照できるのは、そのオブジェクトがロードされたキャッシュ・インスタンスのみです。そのキャッシュ・インスタンスがなんらかの理由で消失した場合、管理対象のオブジェクトは、ディスクに存在している場合でも失われます。プロセスの終了後もオブジェクトが存続する必要がある場合は、オブジェクトとキャッシュの両方を DISTRIBUTE として定義する必要があります。
- キャッシュ構成（キャッシュ環境とも呼ばれます）はキャッシュに固有で、リージョン、サブリージョン、グループおよびオブジェクトの定義が含まれます。キャッシュ構成はディスクには保存されず、他のキャッシュには伝播されません。アプリケーションの初期化時に、キャッシュ構成を定義します。

- CacheAccess.waitForResponse() または CacheAccess.releaseOwnership() メソッドのコールがタイムアウトになった場合は、正常に戻されるまでコールを繰り返す必要があります。CacheAccess.waitForResponse() に失敗した場合は、CacheAccess.cancelResponse をコールしてリソースを解放します。CacheAccess.releaseOwnership() に失敗した場合は、タイムアウト値に -1 を指定して CacheAccess.releaseOwnership をコールし、リソースを解放します。
- グループまたはリージョンが破棄または無効化されたときは、分散定義がローカル定義より優先されます。つまり、グループが分散されている場合、個々のオブジェクトまたは関連グループがローカルとして定義されている場合でも、グループまたはリージョン内のすべてのオブジェクトが、キャッシュ・システム全体で無効化または破棄されます。グループまたはリージョンがローカルとして定義されている場合、グループ内のローカル・オブジェクトはローカルで無効化され、分散オブジェクトはキャッシュ・システム全体で無効化されます。
- オブジェクトまたはグループが SYNCHRONIZE 属性を設定して定義されている場合、オブジェクトをロードまたは置換するには所有権が必要です。ただし、オブジェクトへの一般的なアクセスまたはオブジェクトの無効化には所有権は必要ありません。
- 通常、キャッシュに格納されるオブジェクトは、ユーザー定義のクラス・ローダーではなく、JVM の初期化時に classpath で定義されているシステム・クラス・ローダーでロードする必要があります。特に、アプリケーション間で共有しているオブジェクト、あるいはディスクに保存またはスプールされる可能性があるオブジェクトは、システムの classpath で定義する必要があります。この定義を行わない場合、ClassNotFoundException または ClassCastException が発生する場合があります。
- 一部のシステムでは、オープン・ファイル記述子がデフォルトで制限されている場合があります。これらのシステムでは、システム・パラメータを変更してパフォーマンスを向上させる必要があります。たとえば、UNIX システムでは、オープン・ファイル記述子の数の適切な値は、1024 以上です。
- ローカル・モードまたは分散モードで構成されている場合は、起動時に、1 つのアクティブな Java Object Cache キャッシュが JVM プロセス（Java Object Cache API を使用する JVM で動作しているプログラム）に作成されます。

ディスク・オブジェクトの操作

Java Object Cache は、メモリー内のオブジェクトのみでなく、ディスク上のオブジェクトも管理できます。

この項には、次の項目が含まれます。

ローカルおよび分散ディスク・キャッシュ・オブジェクト

この項には、次の項目が含まれます。

ローカル・オブジェクト

ローカル・モードで作業しているときは、オブジェクトに DISTRIBUTE 属性が設定されている場合でも、キャッシュ属性 isDistributed は設定されず、すべてのオブジェクトがローカル・オブジェクトとして処理されます。ローカル・モードでは、ディスク・キャッシュのオブジェクトはすべて、そのオブジェクトをロードした Java Object Cache キャッシュでのみ参照でき、プロセスの終了後は存続しません。ローカル・モードでは、ディスク・キャッシュに格納されたオブジェクトは、そのキャッシュを使用しているプロセスが終了すると失われます。

分散オブジェクト

キャッシュ属性 isDistributed が true に設定されている場合、キャッシュは分散モードで動作します。ディスク・キャッシュ・オブジェクトは、そのディスク・キャッシュをホスティングしているファイル・システムにアクセスできるすべてのキャッシュで共有できます。これは、構成されているディスク・キャッシュの位置により決定されます。この構成では、ディスク・リソースの使用率が高くなり、ディスク・オブジェクトは、Java Object Cache プロセスの終了後も存続します。

ディスク・キャッシュに格納されているオブジェクトは、diskPath 構成プロパティに指定されたパスと、ファイルの残りのパスを表す内部的に生成された String の組合せで識別されます。したがって、diskPath が物理ディスク上の同一ディレクトリを表し、Java Object Cache プロセスでアクセスできるかぎり、ディスク・キャッシュを共有するキャッシュのディレクトリ構造は異なっていてもかまいません。

ディスクに保存されているメモリー・オブジェクトが分散オブジェクトでもある場合、メモリー・オブジェクトは、それをスプールしたプロセスの終了後も存続できます。

ディスク・キャッシュへのオブジェクトの追加

Java Object Cache でディスク・キャッシュを使用するには、次のようにいくつかの方法があります。

オブジェクトの自動的な追加

Java Object Cache は、特定のオブジェクトをディスク・キャッシュに自動的に追加します。これらのオブジェクトは、メモリー・キャッシュまたはディスク・キャッシュに常駐します。ディスク・キャッシュ内のオブジェクトが必要な場合は、コピーされてメモリー・キャッシュに戻されます。ディスクへのスプーリングの操作は、Java Object Cache によって、メモリー・キャッシュに空き領域が必要であると判断された場合に行われます。オブジェクトのスプーリングが発生するのは、そのオブジェクトに対して SPOOL 属性が設定されている場合のみです。

オブジェクトの明示的な追加

1 つ以上のオブジェクトを Java Object Cache のディスク・キャッシュに強制的に書き込むこともできます。CacheAccess.save() メソッドを使用すると、リージョン、サブリージョン、グループまたはオブジェクトが、ディスク・キャッシュに書き込まれます（オブジェクトがディスク・キャッシュ内にすでに存在している場合は、再度書き込まれることはありません）。

注意： CacheAccess.save() を使用すると、オブジェクトに SPOOL 属性が設定されていない場合でも、オブジェクトがディスクに保存されます。

リージョン、サブリージョンまたはグループで CacheAccess.save() をコールすると、そのリージョン、サブリージョンまたはグループ内のすべてのオブジェクトがディスク・キャッシュに保存されます。CacheAccess.save() メソッドのコール時に、オブジェクトがシリアライズ可能ではない、または他の理由でディスクに書き込むことができない場合は、Java Object Cache ログにイベントが記録され、保存操作は次のオブジェクトで続行されます。各オブジェクトがディスクに書き込まれる場合、書込みは同期的に実行されます。グループまたはリージョンが保存される場合、書込みは非同期のバックグラウンド・タスクとして実行されます。

ディスク・キャッシュにのみ存在するオブジェクトの使用方法

ディスク・キャッシュからのみ直接アクセスされるオブジェクトは、CacheLoader.load() メソッドから CacheLoader.createDiskObject() をコールすると、ディスク・キャッシュにロードされます。createDiskObject() メソッドは、アプリケーションがディスク・オブジェクトのロードに使用できる File オブジェクトを戻します。そのディスク・オブジェクトに対してディスク・オブジェクトの属性が定義されていない場合は、createDiskObject() メソッドを使用して設定します。ローカル・ディスク・オブジェクトと分散ディスク・オブジェクトでは、管理方法が異なります。ローカルまたは分散のいずれであるかは、オブジェクトの作成時に、指定した属性に基づいて決定されます。

注意： 同じキャッシュ・システム内の分散キャッシュ間でディスク・キャッシュ・オブジェクトを共有する場合は、ディスク・キャッシュ・オブジェクトの作成時に DISTRIBUTE 属性を定義する必要があります。この属性は、オブジェクトの作成後は変更できません。

CacheAccess.get() がディスク・オブジェクトでコールされると、ファイルへのフルパス名が戻され、アプリケーションで必要に応じてファイルをオープンできます。

ディスク・オブジェクトはローカル・ディスクに格納され、Java Object Cache を使用してアプリケーションによってディスクから直接アクセスされます。ディスク・オブジェクトは、DISTRIBUTE 属性の設定（および Java Object Cache が分散モードとローカル・モードのどちらで動作しているか）に応じて、すべての Java Object Cache プロセスによって共有される場合と、特定のプロセスに限定される場合があります。

例 7-17「CacheLoader でのディスク・オブジェクトの作成」は、ディスク・オブジェクトをキャッシュにロードするローダー・オブジェクトを示しています。

例 7-17 CacheLoader でのディスク・オブジェクトの作成

```
import oracle.ias.cache.*;

class YourObjectLoader extends CacheLoader
{
   public Object load(Object handle, Object args) {
      File file;
      FileOutputStream = out;
      Attributes attr = new Attributes();

      attr.setFlags(Attributes.DISTRIBUTE);
      try
      // The distribute attribute must be set on the createDiskObject method
      {
         file = createDiskObject(handle, attr);
         out = new FileOutputStream(file);

         out.write((byte[])getInfofromsomewhere());
         out.close();
     }
     catch (Exception ex) {
       // translate exception to CacheException, and log exception
         throw exceptionHandler("exception in file handling", ex)
      }
      return file;
      }
   }
```

例 7-18「ディスク・オブジェクトを使用するアプリケーション・コード」に、Java Object Cache のディスク・オブジェクトを使用するアプリケーション・コードを示します。この例では、Stock-Market というリージョンが、例 7-17「CacheLoader でのディスク・オブジェクトの作成」でそのリージョンのデフォルト・ローダーとして設定された YourObjectLoader ローダーによって、すでに定義されていると仮定しています。

例 7-18 ディスク・オブジェクトを使用するアプリケーション・コード

```
import oracle.ias.cache.*;

try
{
   FileInputStream in;
   File file;
   String filePath;
   CacheAccess cacc = CacheAccess.getAccess("Stock-Market");

   filePath = (String)cacc.get("file object");
   file = new File(filePath);
   in = new FileInputStream(filePath);
   in.read(buf);

// do something interesting with the data
   in.close();
   cacc.close();
}
catch (Exception ex)
{
// handle exception
}
```

StreamAccess オブジェクトの操作

StreamAccess オブジェクトはストリームとしてアクセスされ、ディスク・キャッシュに自動的にロードされます。オブジェクトは OutputStream としてロードされ、InputStream として読み取られます。サイズの小さい StreamAccess オブジェクトにはメモリーまたはディスク・キャッシュからアクセスでき、サイズの大きい StreamAccess オブジェクトはディスクから直接ストリームされます。StreamAccess オブジェクトのアクセス方法は、オブジェクトのサイズおよびキャッシュの容量に基づいて、Java Object Cache によって自動的に決定されます。

ユーザーには常に、オブジェクトがファイル内にあるか、メモリー内にあるかに関係なく、ストリーム・オブジェクト（読取りの場合は InputStream、書込みの場合は OutputStream）が示されます。StreamAccess オブジェクトを使用すると、オブジェクトのサイズやリソースの可用性に関係なく、Java Object Cache ユーザーは常に同じ方法でオブジェクトにアクセスできます。

StreamAccess オブジェクトの作成

StreamAccess オブジェクトを作成するには、オブジェクトがキャッシュにロードされるときに、CacheLoader.load() メソッドから CacheLoader.createStream() メソッドをコールします。createStream() メソッドによって OutputStream オブジェクトが戻されます。OutputStream オブジェクトを使用すると、キャッシュにオブジェクトをロードできます。

オブジェクトに対する属性がまだ定義されていない場合は、createStream() メソッドを使用して設定します。ローカル・ディスク・オブジェクトと分散ディスク・オブジェクトでは、管理方法が異なります。ローカルまたは分散のいずれであるかは、オブジェクトの作成時に属性に基づいて決定されます。

> **注意：** 同じキャッシュ・システム内の分散キャッシュ間で StreamAccess オブジェクトを共有する場合は、StreamAccess オブジェクトの作成時に DISTRIBUTE 属性を定義する必要があります。この属性は、オブジェクトの作成後は変更できません。

例 7-19「キャッシュ・ローダーでの StreamAccess オブジェクトの作成」に、StreamAccess オブジェクトをキャッシュにロードするローダー・オブジェクトを示します。

例 7-19 キャッシュ・ローダーでの StreamAccess オブジェクトの作成

```
import oracle.ias.cache.*;

class YourObjectLoader extends CacheLoader
{
   public Object load(Object handle, Object args) {
     OutputStream = out;
     Attributes attr = new Attributes();
     attr.setFlags(Attributes.DISTRIBUTE);

     try
     {
        out = createStream(handle, attr);
        out.write((byte[])getInfofromsomewhere());
     }
     catch (Exception ex) {
        // translate exception to CacheException, and log exception
        throw exceptionHandler("exception in write", ex)
     }
     return out;
     }
}
```

プール・オブジェクトの操作

プール・オブジェクトは、Java Object Cache で管理される特別なキャッシュ・オブジェクトです。プール・オブジェクトには、同一オブジェクト・インスタンスのセットが含まれます。プール・オブジェクト自体は共有オブジェクトですが、プール内のオブジェクトは Java Object Cache により管理されるプライベート・オブジェクトです。ユーザーは、プール・アクセス・オブジェクトを使用して、プール内の個々のオブジェクトをチェックアウトしてアクセスし、その後不要になるとオブジェクトをプールに戻します。

この項には、次の項目が含まれます。

プール・オブジェクトの作成

プール・オブジェクトを作成するには、`CacheAccess.createPool()` を使用します。`CreatePool()` メソッドは、次の引数を使用します。

- `PoolInstanceFactory`
- `Attributes` オブジェクト
- 2 つの整数引数

整数引数では、最大プール・サイズと最小プール・サイズを指定します。`CreatePool()` に引数としてグループ名を指定すると、プール・オブジェクトがグループに関連付けられます。

`TimeToLive` や `IdleTime` などの属性をプール・オブジェクトに関連付けることができます。これらの属性は、`CacheAccess.createPool()` を使用して属性セットで指定されると、プール・オブジェクト自体に適用できます。またはプール内のオブジェクトに個別に適用することもできます。

`CacheAccess.createPool()` を使用して、整数引数で最小サイズと最大サイズを指定します。最初に最小サイズを指定してください。これにより、プール内に作成されるオブジェクトの最小数を設定します。最小サイズは、最小保証ではなく、リクエストとして解釈されます。プール・オブジェクト内のオブジェクトはリソースが不足した場合にキャッシュから削除されるため、プールのオブジェクト数がリクエストした最小値よりも小さくなる場合があります。最大プール・サイズは、プールで使用可能なオブジェクト数に関して強い制限を設けます。

注意： プール・オブジェクトおよびプール・オブジェクト内のオブジェクトは、常にローカル・オブジェクトとして処理されます。

例 7-20「プール・オブジェクトの作成」に、プール・オブジェクトの作成方法を示します。

例 7-20 プール・オブジェクトの作成

```
import oracle.ias.cache.*;

   try
   {
      CacheAccess cacc = CacheAccess.getAccess("Stock-Market");
      Attributes  attr = new Attributes();
      QuoteFactory poolFac = new QuoteFactory();

      // set IdleTime for an object in the pool to three minutes
      attr.setIdleTime(180);
      // create a pool in the "Stock-Market" region with a minimum of
      // 5 and a maximum of 10 object instances in the pool
```

```
        cacc.createPool("get Quote", poolFac, attr, 5, 10);
        cacc.close();
    }
    catch(CacheException ex)
    {
            // handle exception
    }
}
```

プール・オブジェクトの使用方法

プール内のオブジェクトにアクセスするには、PoolAccess オブジェクトを使用します。static メソッド PoolAccess.getPool() は、指定したプールへのハンドルを戻します。PoolAccess.get() メソッドは、プール内からオブジェクトのインスタンスを戻します（これによって、オブジェクトはプールからチェックアウトされます）。オブジェクトが不要になったときには、PoolAccess.returnToPool() メソッドを使用してプールに戻します。これによって、オブジェクトはプールにチェックインされます。最後に、プール・ハンドルが不要になったときに PoolAccess.close() メソッドをコールします。

例 7-21「PoolAccess オブジェクトの使用方法」に、PoolAccess オブジェクトを作成し、オブジェクトをプールからチェックアウトした後、プールにオブジェクトをチェックインし、PoolAccess オブジェクトをクローズするために必要なコールを示します。

例 7-21 PoolAccess オブジェクトの使用方法

```
PoolAccess pacc = PoolAccess.getPool("Stock-Market", "get Quote");
//get an object from the pool
GetQuote  gq = (GetQuote)pacc.get();
// do something useful with the gq object
// return the object to the pool
pacc.returnToPool(gq);
pacc.close();
```

プール・オブジェクトのインスタンス・ファクトリの実装

Java Object Cache は、アプリケーション定義のファクトリ・オブジェクト PoolInstanceFactory を使用して、プール内のオブジェクトをインスタンス化および削除します。PoolInstanceFactory は、実装が必要な 2 つのメソッド createInstance() および destroyInstance() を持つ抽象クラスです。

Java Object Cache は、プール内に蓄積されるオブジェクトのインスタンスを作成する場合に、createInstance() をコールします。Java Object Cache は、オブジェクトのインスタンスがプールから削除されるときに destroyInstance() をコールします（プール内のオブジェクト・インスタンスが destroyInstance() に渡されます）。

プール・オブジェクトのサイズ（プール内のオブジェクト数）は、これらの PoolInstanceFactory() メソッドを使用して管理されます。プール内のサイズおよびオブジェクト数は、必要に応じて、および TimeToLive 属性または IdleTime 属性の値に基づいて、自動的に増減します。

例 7-22「プール・インスタンスのファクトリ・メソッドの実装」に、PoolInstanceFactory の実装時に必要なコールを示します。

例 7-22 プール・インスタンスのファクトリ・メソッドの実装

```
import oracle.ias.cache.*;
   public class MyPoolFactory implements PoolInstanceFactory
   {
       public Object createInstance()
      {
         MyObject obj = new MyObject();
         obj.init();
         return obj;
```

```
        }
        public void destroyInstance(Object obj)
        {
            ((MyObject)obj).cleanup();
        }
    }
```

プール・オブジェクトのアフィニティ

オブジェクト・プールは、シリアルで再利用可能なオブジェクトの集合です。ユーザーはプールからオブジェクトをチェックアウトして機能を実行し、完了後にオブジェクトを元のプールにチェックインします。オブジェクトがチェックアウトされている間は、ユーザーがそのオブジェクト・インスタンスを排他的に使用することになります。オブジェクトのチェックイン後は、ユーザーはそのオブジェクトへのアクセスをすべて放棄します。オブジェクトのチェックアウト中に、ユーザーは現行のタスクを実行できるようにプール・オブジェクトに一時的な変更を適用（状態を追加）できます。このような変更を追加するにはある程度のコストが発生するため、ユーザーができるだけ変更を適用済のプールから同じオブジェクトを取得できるようにするとメリットが得られます。リリース 9.0.2 以降の Java Object Cache では、そのための唯一の方法はオブジェクトをチェックインしないことでしたが、これではプールの目的にそぐわないことになります。ここで説明したプール要件をサポートするために、Java Object Cache のプール管理には次の 2 つの段落で説明する機能が追加されています。

`PoolAccess` オブジェクトの `returnToPool` メソッドを使用してプールにチェックインされたオブジェクトは、そのオブジェクトを参照した最後の `PoolAccess` オブジェクトとの関連を維持します。`PoolAccess` ハンドルがオブジェクト・インスタンスをリクエストすると、前回と同じオブジェクトが戻されます。この関連は、`PoolAccess` ハンドルがクローズされるか、`PoolAccess.release` メソッドがコールされるか、またはオブジェクトが別のユーザーに与えられる時点で終了します。オブジェクトが別のユーザーに与えられる前にコールバックが実行され、ユーザーがオブジェクトとの関連を放棄するかどうかが判断されます。ユーザーが関連を解除しない場合は、新規ユーザーにはそのオブジェクトへのアクセスが与えられません。`PoolAffinityFactory` インタフェースは、`PoolInstanceFactory` インタフェースを拡張してコールバック・メソッド `affinityRelease` を追加します。このメソッドは、関連を解除できる場合は `true`、それ以外の場合は `false` を戻します。

プール全体が無効化される場合、`affinityRelease` メソッドはコールされません。その場合は、`PoolInstanceFactory.instanceDestroy` メソッドを使用してオブジェクト・インスタンスのクリーン・アップが実行されます。

ローカル・モードでの実行

ローカル・モードで実行中の Java Object Cache は、オブジェクトを共有せず、同じシステムでローカルに実行されているか、あるいはネットワークを介してリモートで実行されている他のキャッシュとは通信しません。ローカル・モードで実行されているときは、システムのシャットダウンやプログラム・エラー発生時のオブジェクトの永続性はサポートされません。

デフォルトでは、Java Object Cache はローカル・モードで実行され、キャッシュ内のすべてのオブジェクトがローカル・オブジェクトとして処理されます。Java Object Cache がローカル・モードで構成されている場合、キャッシュはすべてのオブジェクトの `DISTRIBUTE` 属性を無視します。

分散モードでの実行

分散モードで実行中の Java Object Cache は、オブジェクトを共有でき、同じシステムでローカルに実行されているか、あるいはネットワークを介してリモートで実行されている他のキャッシュと通信できます。オブジェクトの更新および無効化は、通信を行っているキャッシュ間で伝播されます。分散モードでは、システムのシャットダウンやプログラム・エラー発生時のオブジェクトの永続性がサポートされます。

この項には、次の項目が含まれます。

分散モード用のプロパティの構成

Java Object Cache を分散モードで実行するように構成するには、`javacache.xml` ファイルの `distribute` および `discoveryAddress` 構成プロパティの値を設定します。

distribute 構成プロパティの設定

Java Object Cache を分散モードで起動するには、構成ファイルで `isDistributed` 属性を `true` に設定します。設定方法は、7-23 ページの「Java Object Cache の構成」を参照してください。

discoveryAddress 構成プロパティの設定

分散モードでは、無効化、破棄および置換は、キャッシュのメッセージ・システムを介して伝播されます。メッセージ・システムでは、キャッシュが最初の初期化時にキャッシュ・システムに加わることができるように、既知のホスト名およびポート・アドレスが必要です。`javacache.xml` ファイルの communication セクションで `discoverer` 属性を使用すると、ホスト名とポート・アドレスのリストを指定できます。

デフォルトでは、Java Object Cache は、`discoverer` の値を `:12345` に設定します（これは、`localhost:12345` と同じです）。サイト上の他のソフトウェアと競合しないように、システム管理者に `discoveryAddress` の設定を依頼してください。

Java Object Cache が複数のシステムにまたがる場合は、ノードごとに *`hostname:port`* ペアを 1 つ指定して、複数の discoverer エントリを構成します。この結果、使用可能な特定のシステムまたはプロセスの起動順序に対する依存性が回避されます。7-23 ページの「Java Object Cache の構成」も参照してください。

注意： 同じキャッシュ・システムで同時に動作するすべてのキャッシュが、同じホスト名とポート・アドレスのセットを指定している必要があります。discoverer 属性で設定されるアドレス・リストは、特定のキャッシュ・システムを構成するキャッシュを定義します。アドレス・リストが異なると、キャッシュ・システムは別々のグループに分割され、キャッシュ間で不整合が発生する可能性があります。

分散オブジェクト、リージョン、サブリージョンおよびグループの使用方法

Java Object Cache が分散モードで実行されているとき、個々のリージョン、サブリージョン、グループおよびオブジェクトは、ローカルまたは分散のいずれかです。デフォルトでは、オブジェクト、リージョン、サブリージョンおよびグループはローカルとして定義されます。デフォルトのローカルの値を変更するには、オブジェクト、リージョンまたはグループの定義時に DISTRIBUTE 属性を設定します。

分散キャッシュには、ローカル・オブジェクトと分散オブジェクトの両方が含まれる場合があります。

Java Object Cache のいくつかの属性およびメソッドを使用すると、分散オブジェクトを処理でき、キャッシュ間のオブジェクト・データの整合性レベルを制御できます。7-52 ページの「リモート・キャッシュのオブジェクトへのアクセス」も参照してください。

分散オブジェクトでの REPLY 属性の使用方法

複数のキャッシュにわたるオブジェクトを更新、無効化または破棄するときは、関係するすべてのサイトで操作がいつ完了したかを認識することが役立つ場合があります。REPLY 属性を設定すると、関係するすべてのキャッシュが、オブジェクトに対してリクエストした操作が完了したときに、送信者に応答を送信します。CacheAccess.waitForResponse() メソッドを使用すると、ユーザーはすべてのリモート操作が完了するまでブロックできます。

分散操作が複数のキャッシュ全体で完了するのを待機するには、CacheAccess.waitForResponse() を使用します。レスポンスを無視するには、レスポンスの収集に使用されるキャッシュ・リソースを解放する CacheAccess.cancelResponse() メソッドを使用します。

CacheAccess.waitForResponse() および CacheAccess.cancelResponse() は、いずれも CacheAccess オブジェクトがアクセスするすべてのオブジェクトに適用されます。この機能により、アプリケーションは、複数のオブジェクトを更新した後で、そのすべての応答を待機できます。

例 7-23「応答を使用した分散キャッシング」は、オブジェクトを分散として設定する方法、および REPLY 属性が設定されているときの応答の処理方法を示しています。この例では、属性はリージョン全体にも設定できます。また属性は、アプリケーションにあわせて、グループまたは個々のオブジェクトに設定できます。

例 7-23 応答を使用した分散キャッシング

```
import oracle.ias.cache.*;

CacheAccess cacc;
String     obj;
Attributes attr = new Attributes ();
MyLoader   loader = new MyLoader();

// mark the object for distribution and have a reply generated
// by the remote caches when the change is completed

attr.setFlags(Attributes.DISTRIBUTE|Attributes.REPLY);
attr.setLoader(loader);

CacheAccess.defineRegion("testRegion",attr);
cacc = CacheAccess.getAccess("testRegion"); // create region with
  //distributed attributes

obj = (String)cacc.get("testObject");
cacc.replace("testObject", obj + "new version"); // change will be
  // propagated to other caches

cacc.invalidate("invalidObject"); // invalidation is propagated to other caches
```

```
try
{
// wait for up to a second,1000 milliseconds, for both the update
// and the invalidate to complete
    cacc.waitForResponse(1000);

catch (TimeoutException ex)
{
   // tired of waiting so cancel the response
   cacc.cancelResponse();
}
cacc.close();
}
```

SYNCHRONIZE および SYNCHRONIZE_DEFAULT の使用方法

複数のキャッシュにわたるオブジェクトを更新するとき、あるいは複数スレッドで単一のオブジェクトにアクセスするときは、更新操作を調整する場合があります。SYNCHRONIZE 属性を設定すると、同期化された更新が可能になり、アプリケーションは、オブジェクトをロードまたは更新する前にオブジェクトの所有権を取得する必要があります。

SYNCHRONIZE 属性は、リージョン、サブリージョンおよびグループにも適用されます。SYNCHRONIZE 属性がリージョン、サブリージョンまたはグループに適用される場合、そのリージョン、サブリージョンまたはグループ内のオブジェクトをロードまたは置換するには、リージョン、サブリージョンまたはグループの所有権を取得する必要があります。

リージョン、サブリージョンまたはグループに SYNCHRONIZE_DEFAULT 属性を設定すると、そのリージョン、サブリージョンまたはグループ内のすべてのオブジェクトに SYNCHRONIZE 属性が適用されます。リージョン、サブリージョンまたはグループ内の個々のオブジェクトをロードまたは置換するには、そのオブジェクトの所有権を取得する必要があります。

オブジェクトの所有権を取得するには、CacheAccess.getOwnership() を使用します。所有権を取得すると、他の CacheAccess インスタンスはそのオブジェクトのロードまたは置換を許可されません。オブジェクトの読取りおよび無効化は同期による影響を受けません。

所有権を取得し、オブジェクトの変更が完了した後、CacheAccess.releaseOwnership() をコールしてオブジェクトを解放してください。CacheAccess.releaseOwnership() は、リモート・キャッシュでの更新の完了を、指定した時間まで待機します。指定した時間内に更新が完了した場合は所有権が解放されます。指定した時間内に完了しない場合は TimeoutException がスローされます。メソッドがタイムアウトになった場合は、CacheAccess.releaseOwnership() を再度コールしてください。所有権が解放されるように、CacheAccess.releaseOwnership() は必ず正常に戻される必要があります。タイムアウト値が -1 の場合、所有権は他のキャッシュからのレスポンスを待機せずにただちに解放されます。

例 7-24「SYNCHRONIZE および SYNCHRONIZE_DEFAULT を使用した分散キャッシング」に、SYNCHRONIZE および SYNCHRONIZE_DEFAULT を使用した分散キャッシングを示します。

例 7-24 SYNCHRONIZE および SYNCHRONIZE_DEFAULT を使用した分散キャッシング

```
import oracle.ias.cache.*;

CacheAccess cacc;
String     obj;
Attributes attr = new Attributes ();
MyLoader   loader = new MyLoader();

// mark the object for distribution and set synchronize attribute
attr.setFlags(Attributes.DISTRIBUTE|Attributes.SYNCHRONIZE);
attr.setLoader(loader);

//create region
CacheAccess.defineRegion("testRegion");
cacc = CacheAccess.getAccess("testRegion");
```

```
cacc.defineGroup("syncGroup", attr); //define a distributed synchronized group
cacc.defineObject("syncObject", attr); // define a distributed synchronized object
attr.setFlagsToDefaults()  // reset attribute flags

// define a group where SYNCHRONIZE is the default for all objects in the group
attr.setFlags(Attributes.DISTRIBUTE|Attributes.SYNCHRONIZE_DEFAULT);
cacc.defineGroup("syncGroup2", attr);
try
{
// try to get the ownership for the group don't wait more than 5 seconds
   cacc.getOwnership("syncGroup", 5000);
   obj = (String)cacc.get("testObject", "syncGroup"); // get latest object
   // replace the object with a new version
   cacc.replace("testObject", "syncGroup", obj + "new version");
   obj = (String)cacc.get("testObject2", "syncGroup"); // get a second object
   // replace the object with a new version
   cacc.replace("testObject2", "syncGroup", obj + "new version");
}

catch (TimeoutException ex)
{
   System.out.println("unable to acquire ownership for group");
   cacc.close();
   return;
}
try
{
   cacc.releaseOwnership("syncGroup",5000);
}
catch (TimeoutException ex)
{
   // tired of waiting so just release ownership
   cacc.releaseOwnership("syncGroup", -1));
}
try
{
   cacc.getOwnership("syncObject", 5000); // try to get the ownership for the object
   // don't wait more than 5 seconds
   obj = (String)cacc.get("syncObject");  // get latest object
   cacc.replace("syncObject", obj + "new version"); // replace the object with a new version
}
catch (TimeoutException ex)
{
   System.out.println("unable to acquire ownership for object");
   cacc.close();
   return;
}
try
{
   cacc.releaseOwnership("syncObject", 5000);
}
catch (TimeoutException ex)
{
   cacc.releaseOwnership("syncObject", -1)); // tired of waiting so just release ownership
}
try
{
   cacc.getOwnership("Object2", "syncGroup2", 5000); // try to get the ownership for the object
   // where the ownership is defined as the default for the group don't wait more than 5 seconds
   obj = (String)cacc.get("Object2", "syncGroup2"); // get latest object
   // replace the object with new version
   cacc.replace("Object2", "syncGroup2", obj + "new version");
}
```

```
catch (TimeoutException ex)
{
   System.out.println("unable to acquire ownership for object");
   cacc.close();
   return;
}
try
{
   cacc.releaseOwnership("Object2", 5000);
}
catch (TimeoutException ex)
{
   cacc.releaseOwnership("Object2", -1)); // tired of waiting so just release ownership
}
   cacc.close();
}
```

リモート・キャッシュのオブジェクトへのアクセス

Cache Service では、リモート・キャッシュの内容にアクセスするための 4 つのメソッドを提供しています。protected メソッドの CacheLoader.getFromRemote() および CacheLoader.netSearch() は、CacheLoader.load() からのみアクセスできます。getFromRemote は、特定のオブジェクトをローカル・キャッシュにロードするために指定のリモート・キャッシュを検索します。また、netSearch は、特定のオブジェクトをローカル・キャッシュにロードするためにシステム内のすべてのキャッシュを検索します。CacheAccess.replaceRemote() は、指定のリモート・キャッシュ内の特定のオブジェクトを置換します。CacheAccess.getAllCached() は、キャッシュの内容は変更しませんが、分散システムのすべてのキャッシュ内にある指定オブジェクトのすべてのインスタンスのベクターを戻します。

オブジェクトがリモート・キャッシュに存在する場合、getFromRemote は、そのオブジェクトをローカル・キャッシュにロードし、その参照を戻します。useRemoteTtl ブール・パラメータを設定すると、リモート・オブジェクトの値が TimeToLive や IdleTime などのローカル・オブジェクトの属性に設定されます。netSearch も同じように使用できますが、このメソッドではすべてのリモート・キャッシュが検索されるため、特定のリモート・キャッシュを指定する必要はありません。

次の例に、CacheLoader.load() 実装での getFromRemote と netSearch の使用方法を示します。

```
import oracle.ias.cache.*;
   class MyLoader extends CacheLoader
   {
      public Object load (Object handle, Object args) throws CacheException
      {
         Object obj = null;
         try
         {
            obj = getFromRemote(handle, (CacheAddress)args, 10000, true); // use remote ttl
            }
            catch (CacheException ex)
            {
               throw ex;
            }
            return obj;
         }
   }
**************************************
import oracle.ias.cache.*;
   class MyLoader extends CacheLoader
   {
      public Object load (Object handle, Object args) throws CacheException
      {
```

```
        Object obj = null;
        try
        {
           obj = netSearch(handle, 10000, true);  // use remote ttl
        }
        catch (CacheException ex)
        {
           throw ex;
        }
        return obj;
     }
  }
```

getFromRemote は、メモリー、ディスクおよびストリーム・オブジェクトで使用できますが、netSearch はメモリー・オブジェクトでのみ使用できます。

replaceRemote がコールされる場合、リモート・ターゲット・キャッシュの内容が優先されます。置換対象のオブジェクトがリモート・キャッシュ内でローカル・オブジェクトとしてマーク付けされている場合、またはリモート・キャッシュがいっぱいの場合、操作は失敗して例外が発生します。ターゲット・アドレスがローカル・キャッシュ・アドレスの場合、操作はローカル置換として処理されます。指定されたオブジェクトがリモート・キャッシュに存在しない場合、操作はリモート配置（put）として処理されます。replaceRemote の対象も、メモリー・オブジェクトに限定されます。

次の例に、replaceRemote の使用方法を示します。この例の実行後、Cache1.java のオブジェクト obj に対して cacc.get() を実行すると、文字列 object 1 replacement が戻されます。

```
Cache1.java
-----------
import oracle.ias.cache.*;
   Attributes attr = new Attributes();
   attr.setFlags(Attributes.DISTRIBUTE);
   CacheAccess.defineRegion("region", attr);
   CacheAccess cacc = CacheAccess.getAccess("region");
   cacc.put("obj", "object 1");


Cache2.java
-----------
import oracle.ias.cache.*;
   Attributes attr = new Attributes();
   attr.setFlags(Attributes.DISTRIBUTE);
   CacheAccess.defineRegion("region", attr);
   CacheAccess cacc = CacheAccess.getAccess("region");
   CacheAddress cache1 = getCache1CacheAddress();
   cacc.replaceRemote("obj", "object 1 replacement", cache1);
```

getAllCached は、getFromRemote/netSearch および replaceRemote とは異なり、キャッシュの内容を変更しません。このメソッドでは、メモリー、ストリームまたはディスク・オブジェクトを処理できますが、1 回のコールで処理できるのは 1 つのタイプのみという制限があります。つまり、オブジェクトで getAllCached をコールする場合、そのオブジェクトの基本タイプが分散システムのすべてのキャッシュ間で一致している必要があります。現行のキャッシュ内に指定の名前と一致するオブジェクトが存在しない場合、空のベクターが戻されます。オブジェクトの検索時にリモート・キャッシュでスローされる可能性のある例外を考慮するかどうかに応じて、オプションの ignoreRemoteEx ブール・パラメータを設定できます。デフォルトでは、このパラメータは true に設定されるため、すべてのリモート例外は無視されます。

次の例に、getAllCached の使用方法を示します。

```
Cache1.java
-----------
import oracle.ias.cache.*;
   Attributes attr = new Attributes();
   attr.setFlags(Attributes.DISTRIBUTE);
   CacheAccess.defineRegion("region", attr);
   CacheAccess cacc = CacheAccess.getAccess("region");
   MemObjType1 obj = new MemObjType1();
   cacc.put("obj", obj);

Cache2.java
-----------
import oracle.ias.cache.*;Attributes attr = new Attributes();
   attr.setFlags(Attributes.DISTRIBUTE);
   CacheAccess.defineRegion("region", attr);
   CacheAccess cacc = CacheAccess.getAccess("region");
   MemObjType2 obj = new MemObjType2();
   cacc.put("obj", obj);

...

Cache9.java
-----------
import oracle.ias.cache.*;Attributes attr = new Attributes();
   attr.setFlags(Attributes.DISTRIBUTE);
   CacheAccess.defineRegion("region", attr);
   CacheAccess cacc = CacheAccess.getAccess("region");
   MemObjType9 obj = new MemObjType9();
   cacc.put("obj", obj);

Cache10.java
------------
import oracle.ias.cache.*;Attributes attr = new Attributes();
   attr.setFlags(Attributes.DISTRIBUTE);
   CacheAccess.defineRegion("region", attr);
   CacheAccess cacc = CacheAccess.getAccess("region");
   Vector objects = cacc.getAllCached("obj");
```

キャッシュ間でやり取りされるすべてのオブジェクトは、Serializable オブジェクトである必要があります。

キャッシュされたオブジェクトの整合性レベル

Java Object Cache 内で、各キャッシュはその JVM プロセス内で所有オブジェクトをローカルで管理します。分散モードでは、複数プロセスを使用しているとき、あるいはシステムが複数のサイトで実行されているときは、オブジェクトのコピーが複数のキャッシュに存在する可能性があります。

Java Object Cache では、複数のキャッシュで使用可能なオブジェクトのコピー間で必要な整合性レベルを指定できます。指定する整合性レベルは、アプリケーションおよびキャッシュされるオブジェクトによって決まります。サポートされる整合性レベルは、指定しない場合から、オブジェクトのすべてのコピーの整合性をすべての通信キャッシュにわたって保つように指定する場合まで多数あります。

オブジェクト属性を設定すると、整合性のレベルが指定されます。異なるキャッシュに存在するオブジェクト間の整合性は、次の 4 つのレベルに分類されます。

- ローカル・オブジェクトの使用（整合性要件なし）
- 応答待機なしの変更の伝播
- 変更の伝播および応答の待機
- 複数のキャッシュ間にわたる変更のシリアライズ

ローカル・オブジェクトの使用

分散キャッシュ内のオブジェクト間の整合性が必要でない場合、オブジェクトはローカル・オブジェクトとして定義してください（`Attributes.DISTRIBUTE` が未設定の場合は、ローカル・オブジェクトであることを示しています）。ローカルは、オブジェクトのデフォルト設定です。ローカル・オブジェクトの場合は、すべての更新および無効化がローカル・キャッシュでのみ参照できます。

応答待機なしの変更の伝播

分散キャッシュ間でオブジェクトの更新を配布するには、`DISTRIBUTE` 属性を設定してオブジェクトを分散として定義します。分散オブジェクトに対するすべての変更が、システム内の他のキャッシュに配布されます。このレベルの整合性を使用した場合、オブジェクトがキャッシュにロードされたり更新されるタイミングは制御または指定されず、すべてのキャッシュにおける変更の完了に関する通知は行われません。

変更の伝播および応答の待機

分散キャッシュ間でオブジェクトの更新を配布し、次の処理に進む前に変更の完了を待機するには、オブジェクトの `DISTRIBUTE` 属性および `REPLY` 属性を設定します。これらの属性を設定すると、すべてのキャッシュで変更が完了したときに通知が行われます。オブジェクトに対して `Attributes.REPLY` を設定すると、リモート・サイトで変更が完了したときに、変更元のキャッシュに応答が戻されます。これらの応答は非同期に戻されるため、`CacheAccess.replace()` メソッドおよび `CacheAccess.invalidate()` メソッドはブロックしません。応答を待機してブロックするには、`CacheAccess.waitForResponse()` を使用します。

複数のキャッシュ間にわたる変更のシリアライズ

Java Object Cache の最も高いレベルの整合性を使用するには、リージョン、サブリージョン、グループまたはオブジェクトに適切な属性を設定して、オブジェクトが同期化オブジェクトとして動作するようにします。

リージョン、サブリージョンまたはグループに Attributes.SYNCHRONIZE_DEFAULT を設定すると、そのリージョン、サブリージョンまたはグループ内のすべてのオブジェクトに SYNCHRONIZE 属性が設定されます。

オブジェクトに Attributes.SYNCHRONIZE を設定すると、アプリケーションは、オブジェクトをロードまたは変更する前にそのオブジェクトの所有権を取得する必要があります。この属性を設定すると、オブジェクトへの書込みアクセスが効果的にシリアライズされます。オブジェクトの所有権を取得するには、CacheAccess.getOwnership() メソッドを使用します。Attributes.SYNCHRONIZE 属性を設定すると、更新が完了したときに、所有者に通知が送信されます。未処理の更新が完了し、応答が受信されるまでブロックするには、CacheAccess.releaseOwnership() を使用します。このメソッドは、他のキャッシュがオブジェクトを更新またはロードできるようにオブジェクトの所有権を解放します。

注意： オブジェクトに対して Attributes.SYNCHRONIZE を設定することによる効果は、Java メソッドで synchronized を設定した場合とは異なります。Attributes.SYNCHRONIZE が設定されていると、Java Object Cache は、キャッシュでオブジェクトの作成と更新がシリアライズされるようにしますが、Java プログラマは、オブジェクトの参照を取得し、そのオブジェクトを変更できます。

Attributes.SYNCHRONIZE でこのレベルの整合性を使用すると、オブジェクトを外部ソースからロードする前に、CacheLoader.load() メソッドで CacheLoader.netSearch() をコールします。ロード・メソッドで CacheLoader.netSearch() をコールすると、Java Object Cache は、他のすべてのキャッシュを検索してオブジェクトのコピーを探します。このプロセスにより、異なるバージョンのオブジェクトが外部ソースからキャッシュにロードされることがなくなります。SYNCHRONIZE 属性を REPLY 属性および無効化メソッドとともに正しく使用することで、キャッシュ・システム全体でオブジェクトの整合性が保証されます。

OC4J サーブレットでのキャッシュ・オブジェクトの共有

Java Object Cache の配布機能を利用したり、キャッシュ・オブジェクトをサーブレット間で共有するには、アプリケーションのデプロイを多少変更する必要があります。サーブレット間で共有される、または JVM 間に配布されるユーザー定義オブジェクトは、システム・クラス・ローダーによってロードされる必要があります。デフォルトでは、サーブレットによってロードされたオブジェクトは、コンテキスト・クラス・ローダーによってロードされます。これらのオブジェクトは、ロードしたコンテキスト内のサーブレットでのみ参照可能です。オブジェクト定義は、他のサーブレットまたは別の JVM のキャッシュでは使用できません。オブジェクトがシステム・クラス・ローダーによってロードされた場合、そのオブジェクト定義は、他のサーブレットや他の JVM のキャッシュで使用できるようになります。

OC4J では、システムの classpath は、oc4j.jar ファイルと関連 JAR ファイル（cache.jar など）のマニフェストから導出されます。環境内の classpath は無視されます。キャッシュ・オブジェクトを OC4J の classpath に組み込むには、クラス・ファイルを *OLE_HOME*/javacache/sharedobjects/classes にコピーするか、JAR ファイル *OLE_HOME*/javacache/cachedobjects/share.jar に追加します。classes ディレクトリと share.jar ファイルは両方とも、cache.jar のマニフェストに組み込まれています。

ユーザー定義のクラス・ローダーの使用

オブジェクトは、ユーザー定義のクラス・ローダーを必要とするキャッシュに配置できます。Cache Service では、2 つのメソッド Attributes.setClassLoader() および Attributes.getClassLoader() を提供して、ユーザー定義のクラス・ローダーをサポートします。ユーザー定義のクラス・ローダーを使用するようにリージョンまたはグループを設定すると、リージョンまたはグループ下のすべてのオブジェクトは、そのクラス・ローダーを使用して（属性を継承して）ロードされます。この属性は、リージョンまたはグループに属さないオブジェクトには適用されず、設定されている場合は無視されます。

次の例は、ユーザー定義のクラス・ローダーの設定方法を示しています。オブジェクト A は、MyClassLoader を使用してロードされます。

```
import oracle.ias.cache.*;
import java.lang.ClassLoader;
Classloader loader = this.getClass().getClassLoader();
CacheAttributes cAttr = new CacheAttributes();
Attributes attr = new Attributes();
CacheAccess cacc;
Cache.init(cAttr);
attr.setClassLoader(loader);
CacheAccess.defineRegion("region A", attr);
cacc = CacheAccess.getAccess("region A");
cacc.get("object A")
```

分散キャッシュの HTTP およびセキュリティ

この項では、分散キャッシュの HTTP およびセキュリティについて説明します。

HTTP

デフォルトでは、Cache Service は、キャッシュ間通信に HTTP プロトコルを使用します。ただし、下位互換性のために、独自の TCP プロトコルを使用したキャッシュ間通信も引き続きサポートされます。互換性の理由により独自のプロトコルは保持されますが、より新しい機能のいくつかは、HTTP のためだけに実装されます。特に分散キャッシュ・システムでは、リモート・キャッシュからディスク・オブジェクトまたはストリーム・オブジェクトを取得する際に、HTTP モードが必要です。たとえば、ディスク・オブジェクトおよびストリーム・オブジェクトの処理に HTTP モードが必要とされる操作には CacheAccess.getAllCached()、CacheLoader.getFromRemote() および CacheLoader.netSearch() の 3 つがあります。独自モードを有効化するには、CacheAttributes.setTransport() メソッドの CacheAttributes.setTransport(NamedCacheAttributes.TCP) を使用する必要があります。

分散システムのすべてのキャッシュで、同じ通信プロトコルを使用する必要があります。

次の例に、TCP モードを有効化する 2 つの方法を示します。

例 1: cache_attributes.xml での TCP の有効化

```
--------------------
<?xml version="1.0" encoding="UTF-8"?>
<cache-configuration xmlns="http://www.oracle.com/oracle/ias/cache/configuration"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <communication>
      <isDistributed>true</isDistributed>
      <transport>TCP</transport>
   </communication>
</cache-configuration>
```

```
import oracle.ias.cache.*;
Cache.open("cache_attributes.xml");
```

例 2: コードでの TCP の有効化

```
import oracle.ias.cache.*;
CacheAttributes cAttr = new CacheAttributes();
cAttr.distribute = true;
cAttr.setTransport(NamedCacheAttributes.TCP;
Cache.init(cAttr);
```

SSL

キャッシュ間の通信を保護するため、Cache Service では SSL プロトコルをサポートします。SSL は、必ず HTTP プロトコルと組み合せて使用します。JDK keytool プログラムを使用し、証明書を生成してキーストアを設定できます。詳細は、Sun 社の J2SE 1.4.2 Key and Certificate Management Tool の Web ページを参照してください。OC4J に使用されるのと同じ鍵のペアおよび証明書を Cache Service に使用することができます。

SSL を使用するには、分散システム内のすべてのキャッシュで有効化する必要があります。

キーストアの設定後、次のコマンドを使用してキーストアの場所をキャッシュに指定する必要があります。

```
java –jar $OLE_HOME/javacache/lib/cache.jar sslconfig <cache_attributes.xml>
<keystore_file> <password>
```

指定する内容は次のとおりです。

- `$OLE_HOME` は、Oracle *i*AS インスタンスのホーム・ディレクトリです。
- `cache_attributes.xml` は、キャッシュ構成ファイルです。
- `keystore_file` は、keytool によって生成されるキーストア・ファイルへのフルパスです。
- `password` は、鍵のペアの生成に keytool で使用されるパスワードです。

これにより、キャッシュで使用する SSL 構成ファイルが生成されます。ファイルの名前は、cache_attributes.xml に指定された名前です。また、CacheAttributes.isSSLEnabled を true に設定する必要があります。

次の例に、SSL を有効化する 2 つの方法を示します。

例 1: `cache_attributes.xml` での SSL の有効化

```
<?xml version="1.0" encoding="UTF-8"?>
<cache-configuration
 xmlns="http://www.oracle.com/oracle/ias/cache/configuration"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <communication>
      <isDistributed>true</isDistributed>
      <useSSL>true</useSSL>
      <keyStore>.keyStore</keyStore>
      <sslConfigFile>.sslConfig</sslConfigFile>
   </communication>
</cache-configuration>


import oracle.ias.cache.*;
Cache.open("cache_attributes.xml");
```

例 2: コードでの SSL の有効化

```
import oracle.ias.cache.*;
CacheAttributes cAttr = new CacheAttributes();
cAttr.distribute = true;
cAttr.isSSLEnabled = true;
cAttr.keyStoreLocation = ".keyStore";
cAttr.sslConfigFilePath = ".sslConfig";
Cache.init(cAttr);
```

2 つのキャッシュは、相互に通信するために同じ鍵のセットを使用する必要があります。システム内のキャッシュが複数のマシン上にある場合は、キーストア・ファイルをすべてのマシンにコピーし、システム内の各キャッシュ構成ファイルに対して java -jar ... コマンドを実行する必要があります。

ファイアウォール

ファイアウォールを越えて分散キャッシュ・システムを動作させるための現在の対応策は、一連のアウトバウンドの TCP ポートをファイアウォールで有効化し、それらを cache_attributes.xml に定義することです。

たとえば、ポートの範囲が 7100 ～ 7199 の場合、cache_attributes.xml は次のようになります。

```
cache_attributes.xml
--------------------
<?xml version = '1.0' encoding = 'UTF-8'?>
<cache-configuration xmlns="http://www.oracle.com/oracle/ias/cache/configuration"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <communication>
      <isDistributed>true</isDistributed>
      <useSSL>false</useSSL>
      <sslConfigFile>.sslConfig</sslConfigFile>
      <port lower="7100" upper="7199"/>
      <discoverer discovery-port="7100" original="true" xmlns=""/>
   </communication>
</cache-configuration>
```

discovery-port が指定した範囲内にあることを確認してください。

着信接続要求の制限

複数のアドレスを指定して構成されているシステムで、複数のネットワーク・サブネット（プライベートおよびパブリックなど）をサポートするために、構成要素 localAddress を cache_attributes.xml に指定し、着信接続要求を、指定したローカル・アドレスに制限できます。デフォルトでは、分散キャッシュ・システムはリスナー・ソケットをオペレーティング・システムから戻されるプライマリ・ホスト・アドレスにバインドします。ただし、localAddress が指定されている場合、キャッシュはリスナー・ソケットを指定されたアドレスにバインドします。localAddress に指定された値は、完全修飾されたホスト名または IP アドレスであることが必要です。次に例を示します。

```
cache_attributes.xml
--------------------
<?xml version = '1.0' encoding = 'UTF-8'?>
<cache-configuration xmlns="http://www.oracle.com/oracle/ias/cache/configuration"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <communication>
      <isDistributed>true</isDistributed>
      <localAddress>123.456.78.90</localAddress>
      <discoverer discovery-port="7100" original="true" xmlns=""/>
   </communication>
</cache-configuration>
```

または

```
cache_attributes.xml
--------------------
<?xml version = '1.0' encoding = 'UTF-8'?>
<cache-configuration xmlns="http://www.oracle.com/oracle/ias/cache/configuration"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <communication>
      <isDistributed>true</isDistributed>
      <localAddress>computer.oracle.com</localAddress>
      <discoverer discovery-port="7100" original="true" xmlns=""/>
```

```
   </communication>
</cache-configuration>
```

後者の例では、仮想ホスト名の下の IP が変わっても、JOC は影響を受けません。

監視およびデバッグ

Cache.listCacheObjects() および Cache.dump() の他に、Cache Service には、キャッシュの現在ステータスと、リージョン、グループおよびキャッシュ内の個々のオブジェクトの現在ステータスを示すための追加メソッドが用意されています。これらのメソッドは、クラス CacheAccess、Cache、ObjectStatus および AggregateStatus にあります。

Cache の各メソッドは、キャッシュ独自のステータスを示します。getActiveHostInfo は、キャッシュ・システム内のすべてのアクティブなキャッシュに対して、CacheHostInfo オブジェクトの配列を戻します。getCacheSize は、キャッシュでメモリー・オブジェクトによって使用される総領域（バイト数）を見積ります。getDistributedDiskCacheSize および getLocalDiskCacheSize はそれぞれ、分散ディスク・キャッシュおよびローカル・ディスク・キャッシュでオブジェクトによって使用される総領域（バイト数）を見積ります。getObjectCount は、キャッシュ内の現在のオブジェクトの総数を戻します。

CacheAccess の各メソッドは、リージョン、グループおよび個々のオブジェクトのステータスを示します。listNames は、リージョン下のすべてのオブジェクトに名前を付けます。listObjects は、リージョン下のすべてのオブジェクトに名前を付け、そのオブジェクトへのアクセスも提供します。listRegions は、リージョン下のすべてのサブリージョンに名前を付けます。これら 3 つのメソッドは再帰的ではありません。たとえば、listRegions は、リージョンのサブリージョン下のサブリージョンをリストしません。

CacheAccess.getStatus() は、ObjectStatus オブジェクト形式のリージョン下の個々の名前付きオブジェクトまたはグループについて、より詳細なステータス情報を示します。この情報には、キャッシュ済オブジェクトのアクセス数、作成時間、ディスク上のサイズ（ディスクに格納されている場合）、最後にアクセスされた時間、ロード時間（分）、優先順位（オブジェクトの作成者により設定）およびキャッシュ内のサイズ（バイト）が含まれます。オブジェクトまたはグループの名前が指定されていない場合、getStatus はリージョンのステータスを戻します。

一方、CacheAccess.getAggregateStatus() は、AggregateStatus オブジェクト形式の名前付きグループまたはリージョン（サブリージョン）について、全体的な統計情報を戻します。AggregateStatus は、リージョンまたはグループ下のオブジェクトの属性について、下限、平均および上限の値を示します。これらの属性には、アクセス数、作成時間、最後にアクセスされた時間、ロード時間、優先順位およびキャッシュ内のサイズが含まれます。また、AggregateStatus オブジェクトには、リージョンまたはグループのオブジェクト合計数も含まれています。AggregateStatus クラスのリフレクション・メソッドを使用すると、これらの数値すべてに個別にアクセスできます。

Cashe Service では、クリーン間隔ごとに、getAggregateStatus によって示された情報が自動的に集計されます。最新の情報を取得するには、getAggregateStatus をコールする前に Cache.updateStats() をコールする必要があります。

次の例は、getAggregateStatus の使用方法を示しています。

```
import oracle.ias.cache.*;
import java.util.Date;
import java.io.*;

CacheAccess cacc;

// create objects, load objects, and so on.
...

AggregateStatus aggStats;
long            avgCreateTime;
Date            avg;
```

```
Cache.updateStats();
aggStats = cacc.getAggregateStatus();
avgCreateTime = aggStats.getCreateTime(AggregateStatus.AVG);
avg = new Date(avgCreateTime);

System.out.println("average creation time: " + avg);
```

JOC では、DMS をサポートするために、プロセス全体を対象とするキャッシュ・レベルで次のメトリックを提供しています。

- メモリー・サイズ : キャッシュ内のオブジェクトにより使用されるメモリーの合計バイト数。このバイト数の正確さは、Attributes.MEASURE がオブジェクトに設定されているかどうかに応じて変化します。設定されていない場合は、ユーザーが指定したオブジェクト・サイズが正確であるかどうかに応じて変化します。MEASURE が設定されておらず、明示的に指定されたサイズも不正確である場合、メモリー・サイズは正確ではなくなります。
- メモリー・オブジェクト数 : キャッシュ内のオブジェクトの合計数。
- ディスク・サイズ : キャッシュ内のオブジェクトにより使用されるディスク領域の合計バイト数。
- ワーカー・スレッド数 : バックグラウンド・ワーカー・スレッドの合計数。キャッシュでは、これらのスレッドを生成してバックグラウンドのルーチン・タスクを実行し、リモート・キャッシュ・リクエストに応答します。
- タスク数 : バックグラウンド非同期タスクの現在の数。これには、ディスクにメモリー・オブジェクトをスプールするタスクや、定期的にキャッシュをクリーニングするタスクなどが含まれます。
- レスポンス・キュー・サイズ : レスポンス・キューの現在のサイズ。レスポンス・キューでは、レスポンス・オブジェクト（システム内の他のキャッシュに対するリクエストへのレスポンスを管理するために使用されるオブジェクト）を追跡します。
- 時間キュー・サイズ : 時間キューの現在のサイズ。時間キューは、グループ・オブジェクトまたはリージョン・オブジェクトの有効期限を追跡するために使用されるソート済のリストです。

リージョン・レベルでは、JOC は、DMS 用に次のメトリックを追跡します。ディスク・メトリックは、diskcache が有効化されている場合にのみ追跡できます。

- メモリー・サイズ : リージョン内のオブジェクトにより使用されるメモリーの合計バイト数。このバイト数の正確さは、Attributes.MEASURE がオブジェクトに設定されているかどうかに応じて変化します。設定されていない場合は、ユーザーが指定したオブジェクト・サイズが正確であるかどうかに応じて変化します。MEASURE が設定されておらず、明示的に指定されたサイズも不正確である場合、メモリー・サイズは正確ではなくなります。
- メモリー・オブジェクト数 : リージョン内のオブジェクトの合計数。
- メモリー平均ロード時間 : リージョン内のオブジェクトの平均ロード時間。
- メモリー・オブジェクト・アクセス数 : リージョン内のオブジェクトに対するアクセスの合計数。
- ディスク・サイズ : リージョン内のオブジェクトにより使用されるディスク領域の合計バイト数。
- ディスク数 : リージョン内のディスク・オブジェクトの合計数。
- ディスク平均ロード時間 : リージョン内のディスク・オブジェクトの平均ロード時間。

CacheWatchUtil

デフォルトでは、Cache Service には CacheWatchUtil キャッシュ監視ユーティリティが用意されています。このユーティリティにより、システム内の現在キャッシュの表示、キャッシュ済オブジェクトのリストの表示、キャッシュの属性の表示、キャッシュ・ログ出力の重大度のリセット、キャッシュの内容のログへのダンプなどが可能になります。

CacheWatchUtil を起動するには、キャッシュの稼働時に、次のコマンドのいずれかを入力します。

```
java oracle.ias.cache.CacheWatchUtil [-config=cache_config.xml] [-help]
```

または

```
java –jar $OLE_HOME/javacache/lib/cache.jar watch [-config=cache_config.xml] [-help]
```

-config= および -help は、オプションのパラメータです。cache_config.xml は、キャッシュ構成ファイルです。

- help で、キャッシュ・ウォッチャで起動できるコマンドのリストが表示されます。次のようなコマンドがあります。
- set severity=<level> [CacheId] は、特定のキャッシュに対してログ出力の重大度レベルを設定します。次のレベルがあります。
  - -1: オフ
  - 0: 致命的
  - 3: エラー
  - 4: デフォルト
  - 6: 警告
  - 7: トレース
  - 10: 情報
  - 15: デバッグ
- set timeout=<value> は、キャッシュ・システムのグループ通信のタイムアウトに値を設定します。
- dump [CacheId] は、特定のキャッシュの内容をログ・ファイルにダンプします。
- invalidate は、キャッシュ・システム内のすべてのオブジェクトを無効化します。
- destroy は、キャッシュ・システム内のすべてのオブジェクト（メモリー、ストリーム、ディスクなど）を破棄します。

get config [CacheId] と入力すると、特定のキャッシュについて、キャッシュ構成情報が戻されます。次の例に示すように、検証のためにリモート・キャッシュ構成を取得できます。

```
cache> get config 3
ache 3 at localhost:53977
distribute = true
version = 10.1.3
max objects = 200
max cache size = 48
diskSize = 32
diskPath = <disk_path>
clean interval = 3
LogFileName = <log_file_name>
Logger = MyCacheLogger
Log severity = 3
cache address list = [127.0.0.1:22222, pos=-1, uid=0, orig, name=, pri=0
]
```

list caches または lc と入力すると、システム内のすべてのアクティブなキャッシュが出力されます。次の例に示すように、キャッシュ・ウォッチャもそのリストの一部を占めます。

UID 列には、各キャッシュの ID が表示されます。キャッシュ・ウォッチャは、構成されていてもアクティブでないキャッシュを検出できません。

```
cache> lc
Current coordinator: [127.0.0.1:53957, pos=0, uid=0, tag=27979955, pri=0]
#       UID     CacheAddress
-       ---     --------------------
1       0       localhost:53957
2       1       localhost:53965
3       2       localhost:53974
4       3       localhost:53977
5       4       localhost:53980
6       5       localhost:53997 <-- this cache watcher
```

次のように入力します。

```
"list objects [CacheId] [region=<region>] [sort=<0...7>]"
```

または

```
"lo [CacheId] [region=<region>] [sort=<0...7>]"
```

これにより、指定されたキャッシュ内、指定されたリージョン下のすべてのオブジェクトが、ソート・オプションによって指定された順序で出力されます。次のソート・オプションがあります。

- 0: リージョン名の順
- 1: オブジェクト名の順
- 2: グループ名の順
- 3: オブジェクト・タイプの順
- 4: 有効なステータスの順
- 5: 参照数の順
- 6: アクセス数の順
- 7: 有効期限の順

オプションを一切指定しない場合、lo は、すべてのキャッシュ内のすべてのオブジェクトをソートせずに出力します。次の例は、キャッシュ 3、A リージョンについてオブジェクト名順に出力する lo を示しています。例の各列は、読みやすいように調整されています。

```
cache> lo 3 region=A-Region sort=1
Cache 3 at localhost:53977
 REGION     OBJNAME    GROUP      TYPE   REFCNT   ACCCNT   EXPIRE      VALID   LOCK
--------   ---------  -------    ------ -------- -------- --------    ------- ------
[A-Region] [A-Group]  [A-Region] Group  0        1        None        true    null
[A-Region] [A-Region] [null]     Region 0        4        295 Seconds true    null
[A-Region] [B-Group]  [A-Region] Group  0        3        None        true    null
[A-Region] [bar]      [B-Group]  Loader 0        1        None        true    null
```

最後に、groupdump と入力すると、すべてのキャッシュについて、すべてのグループ通信情報がログ・ファイルにダンプされます。このコマンドを使用したり、その出力が役に立つ可能性は少ないですが、グループ通信エラーの場合には、技術サポートで問題を診断するために情報の提供を要求されることがあります。

キャッシュ構成用の XML スキーマ

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.oracle.com/oracle/ias/cache/configuration"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.oracle.com/oracle/ias/cache/configuration"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:element name="cache-configuration" type="CacheConfigurationType">
      <xs:annotation>
         <xs:documentation>Oracle JavaCache implementation</xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:complexType name="CacheConfigurationType">
      <xs:sequence>
         <xs:element name="logging" type="loggingType" minOccurs="0"/>
         <xs:element name="communication" type="communicationType" minOccurs="0"/>
         <xs:element name="persistence" type="persistenceType" minOccurs="0"/>
         <xs:element name="preload-file" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="max-objects" type="xs:positiveInteger" default="1000" minOccurs="0"/>
         <xs:element name="max-size" type="xs:positiveInteger" default="1000" minOccurs="0"/>
         <xs:element name="clean-interval" type="xs:positiveInteger" default="60" minOccurs="0"/>
         <xs:element name="ping-interval" type="xs:positiveInteger" default="60" minOccurs="0"/>
         <xs:element name="cacheName" type="xs:string" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="loggingType">
      <xs:sequence>
         <xs:element name="location" type="xs:string" minOccurs="0"/>
         <xs:element name="level" type="loglevelType" minOccurs="0"/>
         <xs:element name="logger" type="xs:string" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="communicationType">
      <xs:sequence>
         <xs:element name="isDistributed" type="xs:boolean" default="false" minOccurs="0"/>
         <xs:element name="transport" type="transportType" minOccurs="0"/>
         <xs:element name="useSSL" type="xs:boolean" minOccurs="0"/>
         <xs:element name="sslConfigFile" type="xs:string" minOccurs="0"/>
         <xs:element name="keyStore" type="xs:string" minOccurs="0"/>
         <xs:element name="port" minOccurs="0">
            <xs:complexType>
               <xs:attribute name="lower" type="xs:nonNegativeInteger" use="optional" default="0"/>
               <xs:attribute name="upper" type="xs:nonNegativeInteger" use="optional" default="0"/>
            </xs:complexType>
         </xs:element>
         <xs:element name="localAddress" type="xs:string" minOccurs="0"/>
         <xs:element name="discoverer" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
               <xs:complexContent>
                  <xs:extension base="discovererType">
                     <xs:attribute name="order" type="xs:nonNegativeInteger"/>
                     <xs:attribute name="original" type="xs:boolean"/>
                  </xs:extension>
               </xs:complexContent>
            </xs:complexType>
         </xs:element>
         <xs:element name="discovererElection" type="electionType" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="discovererType">
      <xs:attribute name="ip" type="xs:string"/>
      <xs:attribute name="discovery-port" type="xs:positiveInteger" use="required"/>
   </xs:complexType>
```

```
   <xs:complexType name="persistenceType">
      <xs:sequence>
         <xs:element name="location" type="xs:string"/>
         <xs:element name="disksize" type="xs:positiveInteger" default="30" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:simpleType name="loglevelType">
      <xs:restriction base="xs:token">
         <xs:enumeration value="OFF"/>
         <xs:enumeration value="FATAL"/>
         <xs:enumeration value="ERROR"/>
         <xs:enumeration value="DEFAULT"/>
         <xs:enumeration value="WARNING"/>
         <xs:enumeration value="TE"/>
         <xs:enumeration value="INFO"/>
         <xs:enumeration value="DEBUG"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="transportType">
      <xs:restriction base="xs:token">
         <xs:enumeration value="TCP"/>
         <xs:enumeration value="HTTP"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:complexType name="electionType">
      <xs:sequence>
         <xs:element name="useMulticast" type="xs:boolean" minOccurs="0"/>
         <xs:element name="updateInterval" type="xs:positiveInteger" minOccurs="0"/>
         <xs:element name="resolutionInterval" type="xs:positiveInteger" minOccurs="0"/>
         <xs:element name="multicastAddress" minOccurs="0">
            <xs:complexType>
               <xs:attribute name="ip" type="xs:string" use="optional"/>
               <xs:attribute name="port" type="xs:string" use="optional"/>
               <xs:attribute name="TTL" type="xs:nonNegativeInteger" use="optional"/>
            </xs:complexType>
         </xs:element>
         <xs:element name="usePriorityOrder" type="xs:boolean" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

属性の宣言用の XML スキーマ

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.oracle.com/oracle/ias/cache/configuration/declarative"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.oracle.com/oracle/ias/cache/configuration/declarative"
 elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:complexType name="regionType">
      <xs:sequence>
         <xs:element name="attributes" type="attributesType" minOccurs="0"/>
         <xs:element name="region" type="regionType" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="group" type="groupType" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="cached-object" type="cached-objectType" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
   </xs:complexType>
   <xs:complexType name="attributesType">
      <xs:sequence>
         <xs:element name="time-to-live" type="xs:positiveInteger" minOccurs="0"/>
         <xs:element name="default-ttl" type="xs:positiveInteger" minOccurs="0"/>
         <xs:element name="idle-time" type="xs:positiveInteger" minOccurs="0"/>
         <xs:element name="version" type="xs:string" minOccurs="0"/>
         <xs:element name="max-count" type="xs:positiveInteger" minOccurs="0"/>
         <xs:element name="priority" type="xs:positiveInteger" minOccurs="0"/>
         <xs:element name="size" type="xs:positiveInteger" minOccurs="0"/>
         <xs:element name="flag" minOccurs="0" maxOccurs="unbounded">
            <xs:simpleType>
               <xs:restriction base="flagType">
                  <xs:enumeration value="distribute"/>
                  <xs:enumeration value="reply"/>
                  <xs:enumeration value="synchronize"/>
                  <xs:enumeration value="spool"/>
                  <xs:enumeration value="group_ttl_destroy"/>
                  <xs:enumeration value="original"/>
                  <xs:enumeration value="synchronize-default"/>
                  <xs:enumeration value="allownull"/>
                  <xs:enumeration value="measure"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
         <xs:element name="event-listener" type="event-listenerType" minOccurs="0"/>
         <xs:element name="cache-loader" type="userDefinedObjectType" minOccurs="0"/>
         <xs:element name="capacity-policy" type="userDefinedObjectType" minOccurs="0"/>
         <xs:element name="user-defined" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="key" type="xs:string"/>
                  <xs:element name="value" type="xs:string"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
   <xs:simpleType name="flagType">
      <xs:list itemType="xs:token"/>
   </xs:simpleType>
   <xs:complexType name="userDefinedObjectType">
      <xs:sequence>
         <xs:element name="classname" type="xs:string"/>
         <xs:element name="parameter" type="propertyType" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="propertyType">
```

```
      <xs:simpleContent>
         <xs:extension base="xs:string">
            <xs:attribute name="name" type="xs:string" use="required"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
   <xs:complexType name="event-listenerType">
      <xs:sequence>
         <xs:element name="classname" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="handle-event" type="handle-eventType" use="required"/>
      <xs:attribute name="default" type="xs:boolean"/>
   </xs:complexType>
   <xs:simpleType name="handle-eventType">
      <xs:restriction>
         <xs:simpleType>
            <xs:list itemType="xs:token"/>
         </xs:simpleType>
         <xs:enumeration value="object-invalidated"/>
         <xs:enumeration value="object-updated"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:complexType name="groupType">
      <xs:sequence>
         <xs:element name="attributes" type="attributesType" minOccurs="0"/>
         <xs:element name="group" type="groupType" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="cached-object" type="cached-objectType" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
   </xs:complexType>
   <xs:complexType name="cached-objectType">
      <xs:sequence>
         <xs:element name="attributes" type="attributesType" minOccurs="0"/>
         <xs:element name="name" type="nameType" minOccurs="0"/>
         <xs:element name="object" type="userDefinedObjectType" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="nameType">
      <xs:choice>
         <xs:element name="string-name" type="xs:string"/>
         <xs:element name="object-name" type="userDefinedObjectType"/>
      </xs:choice>
   </xs:complexType>
   <xs:element name="cache">
      <xs:complexType>
         <xs:sequence maxOccurs="unbounded">
            <xs:element name="region" type="regionType"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

8

Oracle XML Query Service

この章では、Oracle XML Query Service（XQS）の使用方法について説明します。XQS サービスはエンタープライズ・データの取得、分析、統合および変換に役立つユーザー・モデルを提供します。この章には、次の項目が含まれます。

- Oracle XML Query Service の概要
- XQS の機能の概要
- データソースを使用するための準備
- XQS 関数の構成方法
- 問合せの設計方法
- アプリケーション・コードの開発方法 : XQS クライアント・インタフェースを使用した場合
- OC4JPackager を使用した XQS アプリケーションのパッケージ方法
- XQS パフォーマンス機能の使用方法
- XQS エラー処理モードおよび API の使用方法
- BPEL 環境での XQS の使用
- XQS クライアント API リファレンス
- XQS 構成ファイルのリファレンス
- OC4JPackager のリファレンス
- XQS のトラブルシューティング
- XQS の例

Oracle XML Query Service の概要

この項では、XQS および XQS の基盤となる関連テクノロジの概略を提供し、Oracle XML Query Service について説明します。内容は次のとおりです。

- XQS の概要
- XQS 関連のテクノロジ
- XQS 使用の利点
- 要件、制限事項および特記事項

XQS の概要

XQS とは、OC4J サービスの 1 つで、XQuery（XML 問合せ言語）で作成されており、エンタープライズ・データの統合ビューを作成するための簡単な宣言方式を提供します。

通常、XQS などのサービスがなければ、XQuery は XML 文書にしかアクセスできません。XQS を使用することで、SOAP や JDBC などのアクセス・メカニズムを介して、XML 以外のドキュメント、リレーショナル・データベース、および XML 以外のその他のエンタープライズ情報システムからもデータを取得できます。

XQS を使用すると、プログラミング作業が簡略化されます。計画の段階では、XQuery 式の設計（問合せ、組込みファンクション・コールまたはユーザー定義ファンクション・コール）およびデータ構造のみに集中して作業できます。データソースへのアクセスの詳細は、構成設定によって後で決定されます。XQuery 式とアクセスするデータ間のバインドは、この構成設定で発生します。その後、XQS によって、XQS 関数と呼ばれる外部 XQuery 関数が作成されます。この関数は、目的のソースからデータを取得するために使用します。

XQS は、非定型問合せ（実行時に渡される XQuery 式）または XQS ビュー（以前作成して保存した問合せ）をサポートします。ビューは XQuery 関数になり、後で名前でアクセスできます。（XQuery 自体は、ビューの概念をサポートしていません。）ビューを使用することで、XQuery 式を再利用し、強力で保守しやすい階層問合せを作成できます。

XQS は、多様なクライアント・インタフェースを提供します。そのため、EJB、JSP タグ・ライブラリ、Java クラスまたは Web サービスを介して XQuery 式を実行できます。

XQS 関連のテクノロジ

この項では、XQS の基盤となるテクノロジの概要を説明します。

- XQuery の概略
- Oracle XQuery の実装
- XQS と XQuery API for Java の比較

注意： これらのテクノロジおよび関連テクノロジの入門チュートリアルは、次のサイトで探すことができます（サイトにはその他のチュートリアルも多数含まれています）。

`http://www.w3schools.com/`

XQuery の概略

XQuery は、XML データの問合せと変換に使用できる宣言型言語です。目的の条件を使用して XML ソースをインテリジェントに問い合せ、要素と属性、およびそれらに含まれるデータを取得し、解析できます。これは、XML データベースや XML 文書などの多種多様な XML ソースで幅広く適用される、柔軟性の高い標準テクノロジです。XQuery は、SQL と類似した機能を持ち、リレーショナル・データベースの問合せにおける SQL のような役割を果します。

XQuery では、データの問合せだけでなく、XML データの変換も行えます。そのため、XSLT の補足機能または代替として使用できます。

XPath 2.0 に基づく式のデータ・モデルでは、XQuery の基本的な構成メンバーは式です。それぞれの XQuery の結果は、XML 順序として戻されます。XML 順序は、0（ゼロ）個以上の連続したデータ項目の列で、各項目はスカラー値、XML ノードまたは XML 文書です。項目のタイプの定義は、XML Schema 標準に基づきます。

各問合せは、1 つ以上の、モジュールと呼ばれる部分で構成され、各モジュールは、1 つのプロローグおよびそれに続く 1 つの問合せボディで構成されます。**プロローグ**は、モジュールの処理環境を定義する一連の宣言およびインポートです。

XQuery に含まれる問合せの一般的なモデルは、for-let-where-order-return（FLWOR）式です。次の疑似コードは、FLWOR 式で可能な処理の例を示しています。

```
For each department in depts.xml
Let [variable] represent each employee in each department
Where the headcount of employees in a department is at least 10
Order by average salary of department
Return headcount and average salary of each department
```

XQuery では、定義および事前定義済関数を使用して問合せを実行することもできます。複雑な問合せの場合は、外部関数を別途提供し、XQuery 定義から宣言および起動できます。外部関数は、Java や SQL などの様々な言語で実装できます。

外部関数の使用は、XQS 機能の重要な部分です。独自のデータに対して目的の問合せが実行できるように、XQS 構成に従って外部関数が作成されます。

注意：

- XQuery 1.0 は XPath 2.0 の拡張機能です。そのため、同じデータ・モデル、関数および構文を共有します。データ・モデルは、XSLT 2.0 にも共通しています。
- XQuery の詳細は、次の Web サイトにある「XQuery 1.0: An XML Query Language」の仕様を参照してください。

  `http://www.w3.org/TR/xquery/`

Oracle XQuery の実装

Java での Oracle XQuery の実装は、XQS の基礎となるエンジンであり、Oracle XML Query Service の一部として提供されます。この Oracle XQuery 実装は、Oracle データベース製品の Oracle XML DB の一部である Oracle XQuery 実装と起点や機能が共通していますが、この 2 つを混同しないようにしてください。

XML 項目タイプ

Oracle の XQuery 実装では、タイプ `oracle.xml.xqxp.datamodel.XMLItem` を使用して、問合せ結果の XML 順序に含まれる個々の項目が表されます。

事前定義済の名前空間および接頭辞

次の表に、XQuery および XQS で使用できる、事前定義済の名前空間および接頭辞を示します。

表 8-1 事前定義済の名前空間および接頭辞

接頭辞	名前空間	説明
ora	http://xmlns.oracle.com/xdb	Oracle XML DB の名前空間
ユーザーが選択した任意の接頭辞	urn:oracle.bpel.xq	Oracle BPEL 環境における XQS ビューのデフォルトの名前空間
local	http://www.w3.org/2003/11/xpath-local-functions	XPath ローカル関数宣言の名前空間
fn	http://www.w3.org/2003/11/xpath-functions	XPath 関数の名前空間
xdt	http://www.w3.org/2003/11/xpath-datatypes	XPath データ型の名前空間
xml	http://www.w3.org/XML/1998/namespace	XML の名前空間
xs	http://www.w3.org/2001/XMLSchema	XML Schema の名前空間
xsi	http://www.w3.org/2001/XMLSchema-instance	XML Schema インスタンスの名前空間

これらの接頭辞は、XQuery 式のプロローグで宣言せずに XQuery 式で使用できます。また、xml 以外は、すべてプロローグ内で再定義できます。ora および urn 以外のすべての接頭辞は、XQuery 標準で事前定義済です。

Oracle XQuery の拡張関数

Java バージョンの Oracle XQuery には、このデータベース・バージョンで最初に導入された ora 拡張関数のサポートが含まれています。サポート対象は次のとおりです。

- ora:contains を使用すると、フルテキスト述語を伴う構成検索を制限できます。
- ora:matches を使用すると、正規表現を使用して文字列内のテキストを一致させることができます。
- ora:replace を使用すると、正規表現を使用して文字列内のテキストを置換できます。
- ora:sqrt を使用すると、数値の平方根を戻すことができます。
- ora:view を使用すると、XML 文書のように、XQuery 式に含まれる既存のデータベース表またはデータベース・ビューを問い合せることができます。実際には、この関数によって、リレーショナル・データに関連する XML ビューが作成されます。

これらの関数の詳細は、『Oracle XML DB 開発者ガイド』を参照してください。

注意： 今回のリリースの時点では、まだ、W3C XQuery ワーキング・グループによって XQuery の勧告が公開されていませんでした。オラクル社は、XQuery 標準が勧告になるまで、その進化を継続して追跡します。その間、オラクル社では、XQuery 標準の進化に対応するために、以前のリリースまたはパッチ・セットとの下位互換性がない XQuery 実装の更新をリリースすることを余儀なくされる可能性があります。オラクル社は、XQuery 勧告が公開されるまでの間、XQuery 実装にかかわるデータベース・リリース間またはパッチ・セット間のすべての下位互換性を保証しません。XQuery 標準が勧告になった後、オラクル社は、XQuery 勧告の実装を組み込んだリリースまたはパッチ・セットを作成します。それ以後は、下位互換性にかかわるオラクル社の標準ポリシーが Oracle XQuery 実装に適用されます。XQuery のステータスに関する最新の情報は、`http://www.w3.org` を参照してください。

XQuery 標準で指定された実装の選択肢

暗黙的タイムゾーンのサポート : XQS の XQuery では、暗黙的タイムゾーンは常に `Z` とみなされます。

実装の XQuery 標準からの逸脱

+0 と -0 の境界条件の相違 : XQuery 標準では正のゼロと負のゼロが区別されますが、XQS の XQuery では区別されません。両方とも 0 と表示され、同様に処理されます。

XQuery のオプション機能

W3C によって定義されている XQuery のオプション機能を次に示します。ただし、これらの機能は現時点ではサポートされていません。

- スキーマ・インポート機能
- スキーマ検証機能
- モジュール機能

これらの定義済のオプション機能に加えて、W3C 仕様では、実装によって、実装定義のプラグマおよび拡張機能が提供されます。提供されるものは次のとおりです。

- プラグマ
- Must-understand 拡張機能
- Static-typing 拡張機能

Oracle 実装では、このようなプラグマや拡張機能は要求されません。

XQuery の関数および演算子のサポート

XQS では、最新の仕様に含まれるほとんどの XQuery 関数と XQuery 演算子がサポートされていますが、次のものはサポートされていません。

- XQuery regular-expression 関数

  正規表現（regular-expression）の操作には、かわりに Oracle 拡張機能を使用してください。

- 関数 `fn: trace`、`fn:id`、`fn:idref`、`fn:codepoint-equal`、`fn:prefix-from-QName`、`fn:doc-available` および `fn:collection()`（引数を取らない `fn:collection()`）

XQuery 関数 doc および collection

XQuery 組込み関数 fn:doc および fn:collection は、本質的に実装定義済です。XQS では、これらの関数はサポートされています。関数 doc は、URI 引数によって対象に指定されたローカル・ファイル・システムからファイルを取得します。ファイルは、整形式の XML データである必要があります。関数 collection も同様ですが、ファイルではなくフォルダを取得します（フォルダ内の各ファイルは整形式の XML データである必要があります)。

fn:doc および fn:collection の XQS 実装では、次に示すファンクション・コールのように、引数はプロトコルが file で、ローカル・ファイル・システムにあるファイルへのパスを指定する URL である必要があります。

```
fn:doc("C:/MyDocuments/XQS/myView.xq")
```

URL のプロトコル部分は省略できます。プロトコルは、デフォルトでは file です。

fn:doc() 関数のかわりに、XQS ドキュメント関数を使用し、Oracle Application Server がサポートする任意の URL を介してドキュメントにアクセスできます。たとえば、HTTP プロトコルを使用して XML ドキュメントにアクセスするには、次の構成で <document-source> を定義します。

```
<document-source>
      <function-name prefix="ns"> genericFile </function-name>
</document-source>
```

次に示すように、XQS 関数 genericFile() は問合せ式の一部にすることも可能です。

```
declare namespace ns = "…";
declare function ns:genericFile() external;
for $i in ns:genericFile("http://www.acme.com/repository/item137560.xml") return ...
```

XQS と XQuery API for Java の比較

XQuery API for Java（XQJ）は、Java アプリケーションから XQuery 式を実行するための、進化中の標準です。(JSR-225 を参照してください。）また、JDBC に相当する低水準プログラム・インタフェースです。ただし、XQJ はまだ標準ではないため、Oracle Application Server では、サポートされていません。

XQS では、同様の基礎機能のための高水準 API が提供されます。データソースに接続するためのコードを記述せずに Java から XQuery 式を実行したり、XQuery 式のオブジェクトを作成するなどの処理が可能です。

XQS 使用の利点

XQS は様々な用途に使用できます。Web サービスおよび Java アプリケーションだけでなく、XML データや XML 以外のデータへのアクセスにも役立ちます。

また、XQS を使用すると、低水準の XQJ インタフェースを使用して XQuery 式を実行するなどの、コーディング手順を行う必要がありません。プログラム的でない構成を介したデータソース・アクセスの XQS モデルにより、より迅速に、より便利に開発が行えます。さらに、XQS によって作成される外部関数のライブラリの登録は自動的に処理され、名前空間の割当ては自由に制御できます。

データを取得するだけでなく、XQS は、XML データを変換したり、複数のソースからのデータを結合するための便利なメカニズムとしての機能を果します。

これ以外にも、XQS には次の利点と利便性があります。

- XQS 構成モデルでは、アクセスする特定のデータソースまたは XML 文書とは関係なく問合せを作成できるため、アプリケーションをより移植しやすくなります。必要な手順は、XQS 構成のソースの場所またはドキュメントの場所を変更するだけです。
- XQS ビュー機能を使用すると、階層問合せをより簡単に行うことができ、問合せの共有、再利用および集中管理が可能になります。

- その他の Oracle Application Server コンポーネントや Oracle Database コンポーネントと連携してパフォーマンスの最適化が行われるため、XQS を介して実行した問合せでは、少なくともハンドコードされた XQJ と同等、通常はそれ以上のパフォーマンスが得られます。XQS 自体でも、追加のパフォーマンス拡張が提供されます。これにより、ソースデータおよび結果の構成可能なキャッシングを使用でき、ラージ・データセットを効率的に処理できます。
- XQS によって、中間層とデータベース層間の移行が単純化されます。データソースが中間層の容量を超えた場合、通常は、中間層アプリケーションを変更または再デプロイせずに、Oracle XDB にデータを移行できます。データベースにデータを移動し、必要に応じて、そのデータソースの構成を更新します。中間層のそれ以外のデータソースは、中間層に残ります。

8-13 ページの「XQS のパフォーマンス機能および最適化機能の概要」も参照してください。

要件、制限事項および特記事項

XQS 実装には、次の要件、制限事項および特記事項があります。

- XQS では、OC4J Java Object Cache を使用してキャッシングが行われます。そのため、キャッシュが実行中で（必要であれば自動的に起動させる）、Java Object Cache の構成ファイルが存在することが要求されます。キャッシングに関する追加情報は、「XQS キャッシングの構成」を参照してください。
- 主に現在の XQuery 仕様の制限により、XQS では、データソースからの読取りはできますが、データソースへの書込みはできません。データの問合せはできますが、データの更新、挿入または削除は行えません。
- ファンクション・コールによって戻された（XML データ・モデルで定義されている）一連のノード内では、各ノードに一意の ID が指定されています。ただし、1 つの問合せで同じ関数が 2 回以上コールされた場合、異なる結果セット間でのノード ID の一意性は、XQS によって保証されません。そのため、ノード ID に基づく問合せの中には、結果が非決定的になるものもあります。たとえば、次のような問合せ（Q1 および Q2）があるとします。

```
(: Q1 :)
declare namespace xqs="http://xmlns.oracle.com/ias/xqs";
declare function xqs:a() external;
let $x := xqs:a()
return $x/b is $x/b
```

ノード ID テストの対象が、同じファンクション・コールから取得されたノードであれば、Q1 では、常に `true` が戻されます。

```
(: Q2 :)
declare namespace xqs="http://xmlns.oracle.com/ias/xqs";
declare function xqs:a() external;
xqs:a()/b is xqs:a()/b
```

Q2 では、パフォーマンスの最適化の設定によって、`true` または `false` が戻されます。

XQS の機能の概要

この項では、XQS の重要な機能の概要を説明します。詳細は、この章で後述しています。

- XQS データソースのサポート
- XQS の構成および構成ファイルの概要
- XQS クライアント・インタフェースの概要
- OC4JPackager の概要
- XQS アプリケーションのセキュリティ
- XQS のパフォーマンス機能および最適化機能の概要
- XQS エラー処理の概要
- XQS の主な使用手順の要約

XQS データソースのサポート

前述のように、XQS では、数種類のデータソースがサポートされています。この項には、次の関連項目が含まれます。

- データソースのサポート対象カテゴリ
- XQuery 関数を介したデータソース・アクセス
- データソース関数オブジェクトの役割
- データソースの準備の概要

データソースのサポート対象カテゴリ

XQS では、多様なアクセス・プロトコルが要求されるデータソースは、WSDL ドキュメントを介して記述されます。これは、XQS のサポートにより、一般に普及している Apache Web Services Invocation Framework（WSIF）（SOAP や HTTP の背後にある Web サービス・メカニズムの拡張機能）で、すべてのカスタム起動プロトコルが組み込まれるためです。このメカニズムにより、XQS は、SOAP ベースの Web サービスではなく、WSDL ベースのソースをサポートします。XQS では、SQL で記述された XQS ビューのメカニズムを使用し、JDBC を介してリレーショナル・データベースにアクセスすることもできます。

XQS によってサポートされるすべてのデータソースのカテゴリは、次のとおりです。これらのデータソースには、SQL ベースのビューまたは WSDL ドキュメントを介して多様なソースにアクセスするための特別な機能があります。

- ドキュメント : ファイル、つまり XML 文書を含む URL ベースの入力ストリームにアクセスします。これは、組込み XQuery 関数 `fn:doc()` の機能に似ています。XQS では、XML 以外のファイルもサポートしているため、XML 以外のデータを XML に変換する Oracle Data Definition Description Language（D3L）トランスレータ・プラグインをサポートしています。
- XQS ビュー : 前に格納された問合せを実行します（データベースのストアド・プロシージャと概念が類似しています）。XQS により、問合せの検出、外部変数のバインドおよび問合せの実行が行われます。
- SQL ベースの XQS ビュー : リレーショナル・データベースの表、ビュー、PL/SQL ストアド・プロシージャまたは Java ストアド・プロシージャにアクセスします。これは、SQL ベースの XQS ビューの実行時に、JDBC を介してデータベースに接続することで実行します。Oracle XML データベースを使用している場合は、SQL ベースの XQS ビューを使用して、データベースから直接 XML 型の値を選択できます。詳細は、8-18 ページの「データベース・ソースを使用するための準備（SQL ベースの XQS ビュー）」を参照してください。
- SOAP バインディングを伴う WSDL ソース : Web サービスを介してリソースにアクセスします。Web サービス操作およびデータソースを記述するために、SOAP バインディングを伴う WSDL ドキュメントを指定する必要があります。

- Java または EJB のバインディングを伴う WSDL ソース : XML データを戻すカスタム Java クラスまたは EJB を介して、リソースにアクセスします。その手段として、Java 用 WSIF プロバイダまたは EJB 用 WSIF プロバイダが使用されます。Java クラスまたは EJB を実装し、Java クラス名または EJB 名、および引数の型を指定して、Java または EJB のバインディングを伴う WSDL ドキュメントを指定する必要があります。

WSDL ソース用に指定した WSDL ドキュメントでは、適切なバインディングを使用して、一連のコール可能な操作が記述され、ソースへの接続に関する詳細が提供されます。XQS 構成では、各操作の名前を参照し、WSDL をポイントして操作の説明を参照します。WSDL には作業 URL が必要です。作業 URL は、XQS 構成で指定します。XQS では、WSDL をフェッチする際にこの URL に移動します。また、XQS では、WSDL URL を XQuery 名前空間として使用することや、WSDL ポートの個々の操作をその名前空間のローカル名として処理することが可能です。

XQuery 関数を介したデータソース・アクセス

データソースへのアクセスと操作の実行には、前述の、XQS 関数と呼ばれる外部 XQuery 関数が使用されます。XQS は、XQS 構成に基づいて、自動的に作成および起動されます。構成内で、関数の名前と名前空間、およびその他の関連項目（入力パラメータなど）を指定します。問合せでデータソースを使用する際には、XQuery プロローグで、関連する XQS 関数を外部関数として宣言する必要があります。

名前空間は、関数オブジェクトを含む 1 つの関数ライブラリに対応します。各関数オブジェクトは、1 つの操作に対応します。たとえば、SOAP バインディングを伴う WSDL ソースの場合、WSDL ドキュメント内の各操作に対して 1 つの関数オブジェクトがあります。XQS ビュー・ソースでは、各関数オブジェクトは個々のストアド・クエリーに対応しています。

データソース関数オブジェクトの役割

XQS によって作成されたそれぞれの関数オブジェクトは、XQS 構成に従ってインスタンス化され、次の基本タスクを実行します。

1. XQuery エンジンから渡された入力引数を受け入れます。許容される入力タイプは、XML 順序で許容されるものに準じます。すなわち、Java プリミティブ型および XML ノードです。XQS では、Java オブジェクトと XML のマッピングを実行できませんが、Oracle TopLink を使用するなどして、問合せの前にマッピングを実行できます。
2. 基礎となるデータソースに対して問合せを起動します。XQS により、関数名が対応する XQS 構成要素にマップされ、データソースの接続情報が検出されて、接続が作成されます。
3. 結果を受け取り、パッケージ化します。XQS は、同期的にデータソースから XML 書式で結果を受け取り、受け取った結果を XML 順序にパッケージ化します。
4. エラー（データソース・アクセスの問題や型の不一致など）を処理し、エラー処理の構成に従ってエラー情報を戻します。

データソースの準備の概要

XQS で使用できるほとんどのデータソースには、必要な準備手順があります。次に例を示します。

- XML 以外のドキュメント・ソースを使用するには、XQS で提供される変換ツール Oracle D3L を使用するための準備が必要です。準備には、データ書式の指示を含む D3L スキーマ・ファイルを指定することが含まれます。
- XQS ビューを使用するには、データの入力型と戻り型を考慮し、問合せを設計して `.xq` ファイルとして保存し、その配置場所を決定する必要があります。
- サポートされているいずれかのバインディングを伴う WSDL ソースを使用するには、適切な WSDL ドキュメントを指定（場合によっては作成）する必要があります。

詳細は、8-14 ページの「データソースを使用するための準備」を参照してください。

XQS の構成および構成ファイルの概要

XQS 構成ファイルは xqs-config.xml です。XQS は、アプリケーション EAR ファイルのトップレベルにある xqs-resources.jar ファイル（OC4J パッケージ化ユーティリティで作成）で、この構成ファイルを検索します。この構成ファイルを使用して、XQS 関数およびデータソースの情報を指定します。

xqs-config.xml のトップレベルの要素は次のとおりです。

- <document-source> を使用すると、XML ファイルまたは XML 以外のファイルを含むドキュメントのデータにアクセスできます。
- <xqsview-source> を使用すると、XQS ビューを使用できます。
- <wsdl-source> は、ユーザーが指定した WSDL ドキュメントに関連するデータソースに使用できます。

XQS は、構成ファイルから情報を読み取り、その情報を使用して、データソースにアクセスする XQS 関数オブジェクトを移入します。

構成の更新時には、OC4J の再起動は不要ですが、アプリケーションを再デプロイする必要があります。（詳細は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。）

追加情報は、8-21 ページの「XQS 関数の構成方法」を参照してください。

注意： Oracle Application Server では、Oracle Enterprise Manager 10*g* Application Server Control コンソールに、XQS 構成のページはまだ含まれていません。XQS 構成は、xqs-config.xml ファイルで直接管理できます。

XQS クライアント・インタフェースの概要

XQS では、XQuery 機能の実装に使用できる、次のクライアント・インタフェースがサポートされています。各インタフェースの使用方法および例は、8-37 ページの「アプリケーション・コードの開発方法 : XQS クライアント・インタフェースを使用した場合」を参照してください。

注意： 問合せにいずれかの XQS クライアント API を使用する場合、問合せの実行によって結果が定義されますが、すべての結果がただちにメモリーに格納されるとはかぎりません。メモリーにただちに、および完全に格納される一連の結果は、「マテリアライズされている」（実体がある）と表現されます。これに対して、暗黙カーソルを介し、次の項目を処理するための（next item）機能を使用して 1 回（バッチの場合は複数回）にアクセスされる一連の結果は、「マテリアライズされていない」（実体がない）と表現されます。マテリアライズされていない結果は、いつ取得されてメモリーに書き込まれるのかが保証されません。次の項目を処理する（next item）コールによって、次の結果項目を作成する XQuery 式の評価がトリガーされます。

後続の説明のように、XQS クライアント API では、マテリアライズされた結果に対してステートレス・クライアント・オブジェクトを使用するか、マテリアライズされていない結果に対してステートフル・クライアント・オブジェクトを使用するかを選択できます。

これらの概念の詳細および重要な考慮事項は、8-40 ページの「ステートフル・クライアントとステートレス・クライアント」を参照してください。

- クライアント API: XQS では、汎用 Java インタフェース XQSFacadeI が提供されます。この Java インタフェースは、目的の XQuery 機能の Java 実装に使用できます。このインタフェースは、ファサードというデザイン・パターンに基づいており、表面上は、基礎となる XQS 機能の詳細や複雑さが見えないようになっています。適切な実行（execute）メソッドを使用して、非定型問合せまたは XQS ビュー名、およびバインド・パラメータを渡します。どちらの場合も、次の項目を取得する（get next item）メソッドを使用して、結果を処理します。問合せの実行後、XQSFacadeI インスタンスにより XML の結果順序が

ラップされ、現在の状態が維持されます。順序の項目を段階的に取得して処理するか（ステートフル・アプローチ）、順序の全項目を一度に取得して処理し、その後基礎となるデータソース・リソースを解放するか（ステートレス・アプローチ）は、クライアントによって決まります。

注意： `XQSFacadeI` インタフェースも、EJB クライアント API および JSP タグ・ライブラリでは背後で使用されます。

- Oracle BPEL XQuery 関数 : Oracle BPEL PM 環境では、XQueriy を実行する Oracle BPEL XPath 関数を使用して、XQuery および XQS を使用できます。Oracle BPEL PM には、BPEL ユーザー（`com.collaxa.cube.xml.xpath.functions.xml.GetElementFromXQueryFunction`）用の Java クラスが用意されています。このクラスは BPEL プロセス付きの XPath 関数として登録し、実行用に BPEL プロセスとともにパッケージ化した XQuery のいずれかの名前を渡すことができます。
- EJB クライアント API: EJB クライアント API は、リモート（RMI 経由）またはローカルで、セッション Bean を介してアプリケーションにアクセスする方法を提供します。XQS では、ステートフル・セッション Bean とステートレス・セッション Bean のどちらかの使用がサポートされます。ステートレス・セッション Bean を使用すると、EJB へのコールが最小限に抑えられます。一方、ステートフル・セッション Bean を使用すると、メモリー使用量が少なくて済みます。どちらの場合も、適切な実行（execute）メソッドで、非定型問合せまたは XQS ビュー名、およびバインド・パラメータを指定します。ステートフル・セッション Bean の場合は、次の項目を取得する（get next item）メソッドを使用して結果を処理します。ステートレス・セッション Bean では、順序は常にマテリアライズされ、実行（execute）メソッドによって戻されます。
- JSP タグ・ライブラリ : JSP タグ・ライブラリでは、HTTP を介してアプリケーションにアクセスする方法が提供されます。XQS では、ステートフル・アクセスとステートレス・アクセスのどちらかの JSP タグが提供されます。execute タグまたは executeCursor タグを使用して、非定型問合せまたは XQS ビュー名を指定します。どちらのタグを使用するかは、ステートフル・アクセスとステートレス・アクセスのどちらを使用するかによります。バインド・パラメータを指定するには、ネストされた parameter タグを使用します。ステートフル・アクセスの場合は、executeCursor タグと関連付けられた next タグを使用して、結果を処理します。ステートレス・アクセスでは、順序は常にマテリアライズされ、execute タグによって戻されます。結果は、execute タグまたは executeCursor タグの属性設定に基づいて、JSP 出力ストリーム、DOM ドキュメントまたは Java `Object` インスタンスの配列として戻されます。
- XQS ビュー Web サービス : XQS ビューの使用時には、オプションで、ビューを Web サービス操作として公開するために、XQS で作成される操作を WSDL ドキュメントに追加できます。（これは、XQS ビューの構成の一部として処理されます。）その後、その他の Web サービスと同様に、この Web サービスのクライアントを実装できます。

注意： XQS `QueryParameterI` インタフェースにも注意する必要があります。Java クライアント API または EJB クライアント API の外部バインドを伴う問合せを使用するには、そのバインド値の `QueryParameterI` オブジェクトの配列を作成する必要があります。8-81 ページの「XQS QueryParameterI リファレンス」を参照してください。（このインタフェースも、JSP クライアント・インタフェースでは背後で使用されます。）

OC4JPackager の概要

XQS で提供される OC4JPackager は、XQS 関連のファイルを J2EE アプリケーションまたは Web アプリケーションでパッケージ化し、そのアプリケーションで XQS 機能を使用できるようにするコマンドライン Java ツールです。

通常は、XQS クライアント API（`XQSFacadeI` インタフェース、EJB クライアント API または JSP タグ・ライブラリ）の 1 つを介して XQS をコールする Web アプリケーションを書き込みます。また、関連するデータソースおよび XQS ビューをポイントする XQS 構成ファイル `xqs-config.xml` を作成し、ビューが配置されているディレクトリである XQS リポジトリを選択します。その後、アプリケーション、XQS 構成ファイルおよび XQS リポジトリを EAR ファイルにバンドルします。このファイルは後でデプロイします。

コマンドライン・パラメータを介して OC4JPackager に命令が出されると、すべてのバンドルの手順が自動的に完了します。具体的には、次の処理が行われます。

1. `xqs-config.xml` およびすべての XQS リポジトリ・ファイルを、`xqs-resources.jar` というファイルにバンドルします。
2. アプリケーションに関連付けられている各アーカイブ（WAR ファイルや EJB JAR ファイルなど）を開き、マニフェスト（`MANIFEST.MF`）内の `Class-Path` 属性を変更して、`xqs-resources.jar` を含めます。
3. すべてのアプリケーション・アーカイブ・ファイルおよび `xqs-resources.jar` で構成される EAR アーカイブをバンドルします。
4. EAR アーカイブ内の `orion-application.xml` ファイルを（必要に応じて）作成または変更し、EAR のすべてのコンポーネントがアクセスできるライブラリのリストに `xqs-resources.jar` を追加します。

OC4JPackager を実行する際には、次のいずれかを含むディレクトリを指定します。

- 既存の EAR ファイル。OC4JPackager によってバンドルが解除され、再作成されます。
- WAR ファイルや EJB JAR ファイルなどの既存の J2EE モジュール・セット。OC4JPackager によって新規 EAR にバンドルされます。

Web サービス操作として公開する XQS ビューに対し、OC4JPackager により、Web サービス操作の起動が XQS ビューに委譲され、追加のタスクが実行されます。操作は、自動的に生成される WSDL ドキュメントに追加されます。詳細は、8-60 ページの「XQS ビューを Web サービス操作として公開するための OC4JPackager の追加の出力」を参照してください。

一般的な OC4JPackager の詳細は、8-57 ページの「OC4JPackager を使用した XQS アプリケーションのパッケージ方法」および 8-115 ページの「OC4JPackager のリファレンス」を参照してください。

XQS アプリケーションのセキュリティ

XQS 自体では、セキュリティ層はまったく追加されません。この点で、XQS は本質的に J2EE アプリケーションと似ています。セキュリティは、使用する XQS クライアントおよびアクセスするデータソースのタイプに応じて、必須の Web および J2EE の標準セキュリティ機能を介して、OC4J によって提供されます。

たとえば、SQL バインディングを伴う SQL ベースの XQS ビューを使用する場合は、データソース・セキュリティ機能を使用します。XQS EJB クライアント・インタフェースを使用する場合は、標準の EJB セキュリティを使用します。XQS JSP タグ・ライブラリを使用する場合は、標準の HTTP 接続認証を使用します。その他を使用する場合も、それぞれのセキュリティ機能を使用します。

XQS のパフォーマンス機能および最適化機能の概要

XQS では、パフォーマンスを向上させる次の機能が提供されます。

- キャッシング : XQS では、将来の問合せで使用できるように、XQS ビューの実行結果、またはデータソースの XML データをキャッシュできます。これにより、特に、データソース（外部 Web サービスなど）へのアクセスに時間がかかる状況で、パフォーマンスが向上します。キャッシングは、非同期ソースへのアクセス時にも重要です。XQS 構成を介して、キャッシュ・データの有効期限および無効化の設定を指定するか、オプションでキャッシング全体を無効にできます。
- ラージ・データセットの処理 : XQS には、数十 MB 以上もあるようなラージ・データセットへのアクセス時に、システム・メモリーが不足する危険性を最小限に抑える機能があります。このような状況になると、XQS では、データの必要性に応じて結果が部分的にマテリアライズされ、メモリー使用量が一定の範囲内にとどめられます（作業単位方式）。オプションで、データ項目を 1 つずつ戻すこともできます。
- スケーラビリティ : XQS では、入力が効率的に XQuery エンジンに配信され、キャッシュ・リソース（解析済 WSDL ドキュメントなど）を共有および再利用できます。また、スケーラビリティに応じて、マテリアライズされていない結果順序を使用できます。これらの動作は、XQS 構成を介して制御できます。

詳細は、8-64 ページの「XQS パフォーマンス機能の使用方法」を参照してください。

注意 : XQS では、Oracle XQuery エンジンおよび XDK と密接に連携してデータ・フローが最適化されますが、問合せの最適化は行われません。

XQS エラー処理の概要

XQS アプリケーションの実行時には、入力パラメータをデータソースで要求されるタイプに変換する際や、データソースによって戻されたデータを XML に変換する際などに、XQS 外部関数および基礎となるデータソースに関連する問題が発生する場合があります。また、データソースが使用できない場合や、エラーが戻されることもあります。XQS のデフォルトの動作では、このような状況で XQuery 動的エラーが発生します。それによって問合せの実行が停止し、結果が戻されなくなります。

XQS 構成属性（`onError`）を使用して、より影響の少ない動作を選択すると、XQS を続行して、発生したエラーに関する追加情報を取得できます。使用できるエラー処理モードは次のとおりです。

- `dynamicError`（デフォルト）
- `emptySequence`
- `errorMessage`

`emptySequence` モードまたは `errorMessage` モードを使用した場合、そのモードの設定対象となった XQS 外部関数の実行時にエラーが発生しても、問合せの実行は停止せず、必要に応じて後で取得できるように XQS によって保存されます。XQS により、空の XML 順序（`emptySequence` モードの場合）または 1 つのエラー・メッセージで構成される単一項目の XML 順序（`errorMessage` モードの場合）が戻されます。`errorMessage` モードでは、XQS 構成で固定エラー・メッセージを事前に構成しておくか、データソースから戻されたエラー・メッセージをそのまま使用することができます。

XQS 外部関数の抑止されたエラーの情報は、XQS によって、`XQSErrorI` オブジェクトのイテレータ内に戻されます。

XQS では、XQS 関数内部のエラーにのみ、特別な処理が適用されます。構文エラーやタイプの不一致などの通常の XQuery エラーの場合は、いずれの XQS エラー・モードでも、問合せの実行が停止します。

XQS エラー・モードは、XQS 関数ごとに個別に構成します。1 つの関数のエラー・モードが、同じ問合せ内の他の XQS 関数の動作に影響することはありません。

詳細は、8-68 ページの「XQS エラー処理モードおよび API の使用方法」を参照してください。

XQS の主な使用手順の要約

XQS を使用する場合の主な手順を次に示します。各手順の詳細を記述した項へのリンクも提供しています。

1. 必要に応じてデータソースを準備します。たとえば、ドキュメント・ソースの場合は、XML 以外を XML に変換するための設定などを行います。XQS ビューの場合は、問合せの定義や保存などを行います。データベースなどのソースへのアクセスにも、特別な設定が必要になる場合があります。8-14 ページの「データソースを使用するための準備」を参照してください。
2. XQS を構成します。`xqs-config.xml` ファイルを使用して、アクセスするデータソースおよび実行する問合せの指定と構成を行い、問合せのデータソースを表す XQS 関数へのマッピングを作成します。8-21 ページの「XQS 関数の構成方法」を参照してください。
3. 問合せを設計します。8-34 ページの「問合せの設計方法」を参照してください。
4. アプリケーションを開発します。開発では主に、XQS によって提供されるいずれかの API を使用して、XQuery 式を実行し、戻された XML 結果を処理することが要求されます。8-37 ページの「アプリケーション・コードの開発方法 : XQS クライアント・インタフェースを使用した場合」を参照してください。
5. アプリケーションをパッケージ化し、デプロイします。XQS に付属のパッケージャを使用します。8-57 ページの「OC4JPackager を使用した XQS アプリケーションのパッケージ方法」を参照してください。

注意 : ここに示した手順は単純化されたものです。通常、特に XQS ビューが含まれている複数のソースでは、これらの手順をモジュール的に順番に行って完了させることはできません。実際には、処理の繰り返しが必要になります。たとえば、このようになります。最初にデータベース・ソースの準備を行い、次にそのソースの問合せを設計します。その後、問合せを XQS ビューとして保持します。つまり、XQS ビュー・ソースを準備します。さらに、そのビューを使用する別の問合せを設計します。

データソースを使用するための準備

XQS で使用するほとんどのデータソースには、必要な準備手順があります。この項には、次の項目が含まれます。

- XML 以外のドキュメント・ソースを使用するための準備
- XQS ビューを使用するための準備
- データベース・ソースを使用するための準備（SQL ベースの XQS ビュー）
- SOAP バインディングを伴う WSDL ソースを使用するための準備
- カスタム・クラスまたは EJB を使用するための準備（Java または EJB のバインディングを伴う WSDL ソース）

XML 以外のドキュメント・ソースを使用するための準備

ドキュメントの中には、ネイティブのメッセージ書式として XML を使用せず、バイトおよび文字の構造化された記録などのネイティブ書式を使用するものがあります。この例として、Excel のカンマ区切り（CSV）ファイルがあります。XQS および XQuery で XML 以外のデータが理解されるには、XQS で適切な変換ツールを使用できることが必要です。そのためには、事前定義済の、構造化された一連の規則にデータが従っている必要があります。このような構造化された書式のデータは処理が可能なため、取得されてから XML 書式に変換され、アプリケーションで使用できるようになります。

XML 以外のドキュメントを使用できるように、XQS 実装では、Oracle Data Definition Description Language（D3L）をサポートしています。XQS で XML 以外のドキュメントを使用するための準備として、次の手順を行います。

1. XML 以外のデータに、D3L 変換メカニズムとの互換性があることを確認します。
2. D3L によってデータが解析および変換されるように、D3L にデータ書式の指示を出すスキーマ・ファイルを指定します。
3. スキーマ・ファイルの名前および場所を指定して、D3L を使用できるように XQS を構成します。

次の各項では、D3L およびその使用方法の概要を示します。

- D3L とは
- D3L スキーマ・ファイル
- D3L を使用するための XQS の構成

> **注意：** D3L の機能の詳細は、『Oracle Application Server Integration InterConnect ユーザーズ・ガイド』を参照してください。

D3L とは

D3L には、ネイティブの、XML 書式以外のドキュメントが従う必要がある構造が記述されます。これにより、特定の Oracle の中間層コンポーネント（OracleAS Integration InterConnect が一般的ですが、ここでは XQS）で処理されるようになります。

D3L によって提供されるものは、次のとおりです。

- バイナリ・データ、文字列データ、構造化データおよび配列データのレコード・レイアウトなど、ネイティブ・ファイルの書式を示す、XML ベースのデータ記述言語
- D3L スキーマ・ファイルからの指示を使用してネイティブ書式のファイルの内容を変換する、変換エンジン

D3L スキーマ・ファイルは、D3L Document Type Definition（DTD）によって定義された構文に準拠している必要があります。XQS 構成を介して、使用する D3L スキーマ・ファイルを指定します。

> **重要：** D3L を使用するには、基礎となるネイティブ書式データ内のフィールド数が固定され、認識されていることが必要です。D3L は、任意の構造を持つデータ（標準の XML など）、名前 / 値ペアのデータ、または解析にトークンの先読みを必要とする条件付きデータ構造には適しません。

D3L スキーマ・ファイル

D3L スキーマ・ファイルには、D3L 変換エンジンがデータを解析して XML に変換するために必要な、データ型、書式およびデリミタが記述されています。このスキーマ・ファイルは、d3l.dtd の仕様に準拠する必要があります。

D3L スキーマ・ファイルの詳細および作成方法は、『Oracle Application Server Integration InterConnect ユーザーズ・ガイド』を参照してください。ここでは、このファイルの概要がわかるように、ファイルの一部分を示します。

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE message SYSTEM "d3l.dtd">
<message name="replyFlight" type="BookingReplyType" object="Booking"
         header="D3L-Header" value="replyOptions">
   <unsigned4 id="u4" />
   <unsigned2 id="u2" />
   <struct id="DateTimeRecord">
      <field name="DateInfo">
         <date format="MMDDYY">
            <pfxstring id="datstr" length="u4" />
         </date>
      </field>
      <field name="TimeHour"><limstring delimiter="*" /></field>
      <field name="TimeMinute"><limstring delimiter="*" /></field>
   </struct>
...
</message>
```

D3L を使用するための XQS の構成

XML 以外のドキュメント・ソースに対し、XQS で D3L の変換メカニズムを使用するには、次の手順を行います。

1. <document-source> の <XMLTranslate> サブ要素を使用して、D3L を使用するように XQS に対して指定します。（後で、その他のツールもサポートして、同様に指定することができます。）
2. <XMLTranslate> の <schema-file> サブ要素を使用して、データ解析に使用する D3L のスキーマ・ファイルを指定します。

次に、PersonalInfoD3L.xml という名前の D3L スキーマ・ファイルを使用するための XQS 構成を示します。

```
<document-source ... >
   ...
   <XMLTranslate method="D3L">
      <schema-file>http://host:port/xqs/PersonalInfoD3L.xml</schema-file>
   </XMLTranslate>
   ...
</document-source>
```

注意：

- これらの要素のリファレンス情報は、8-93 ページの「XQS 構成ファイルのリファレンス」にあります。
- 8-22 ページの「ドキュメント・ソースにアクセスする XQS 関数の構成」も参照してください。

XQS ビューを使用するための準備

前述のように、XQS ビューとは、将来使用できるように格納された問合せです。XQS では、ビュー自体がソースとして扱われるため、ビューの構成は、<xqsview-source> 要素を介して行います。問合せを XQS ビューとして使用するための準備手順は、次のとおりです。

1. 問合せで要求される入力パラメータを考慮します。XQS ビューのパラメータは、XQuery のプロローグでそのパラメータを外部変数として宣言することで指定できます。
2. 問合せを設計します。これには、入力パラメータに対して外部変数を宣言することも含まれます。設計した問合せを .xq ファイルとして保存します。（8-34 ページの「問合せの設計方法」も参照してください。）
3. .xq ファイルの保存場所を考慮します。.xq ファイルの保存場所は、XQS リポジトリと呼ばれます。この場所は、XQS 構成を介して（<repository> 要素を使用して）指定できるほか、OC4JPackager の実行時に -repository オプションを介して指定した場所を使用することもできます。
4. Web サービス操作として XQS ビューを公開するかどうかを考慮します。この処理は、XQS 構成ファイルにある <xqsview-source> 要素の WSDLvisibility 属性を介して行います。これにより、1 つの操作として Web サービスにビューが含められ、XQS によってアプリケーションに追加されます。Web サービス操作としてビューを公開する場合は、XML の出力タイプに注意してください。この出力タイプを、構成に含まれる <xqsview-source> の <output-element> サブ要素に反映することをお薦めします。（追加情報として、8-56 ページの「Web サービス操作として公開された XQS ビューの使用方法」も参照してください。）

次に示すのは、XQS ビューに使用する XQuery 式です。この式では、外部変数 loc が受け入れられ、ファイルから注文（purchase order）データを読み取る関数 readFile に渡されます。.xq ファイルの保存時には目的の名前を付け、XQS 構成でその名前を指定できます。ビューを実行するために XQS が実装する XQS 関数にも、目的の名前を指定する必要があります。ファイル名と関数名は別個のものですが、必要に応じて、デフォルトで関数名をファイルのベース名として使用できます。

```
declare variable $loc external;
declare namespace xqs = "http://xmlns.oracle.com/ias/xqs";
declare function xqs:readFile($l as xs:string) external;
for $po in xqs:readFile($loc)//po return $po//total
```

通常、XQS 構成の <xqsview-source> 要素の中では、少なくとも次のものを使用します。

- Web サービス操作としてビューを公開する予定の場合は WSDLvisibility 属性、また、実際にビューを公開する場合は <output-element> サブ要素
- ビューの XQS 関数に目的の名前を指定するための <function-name> サブ要素
- XQS 関数の入力パラメータに対応する外部変数（前の例では loc）を指定するための、<input-parameters> サブ要素およびそのサブ要素
- ビューが含まれている .xq ファイルの名前を指定するための <queryName> サブ要素（XQS 関数名と同じファイル・ベース名を使用する場合を除く）
- .xq ファイルの場所を指定するための <repository> サブ要素（OC4JPackager の実行時に -repository オプションを介して指定した場所にファイルがある場合は不要）

注意：

- これらの要素のリファレンス情報は、8-93 ページの「XQS 構成ファイルのリファレンス」にあります。
- 上の例の続きは、8-24 ページの「XQS ビューを使用する XQS 関数の構成」を参照してください。

データベース・ソースを使用するための準備（SQL ベースの XQS ビュー）

SQL ベースの XQS ビューを使用すると、リレーショナル・データベースから取得したデータに（XML のサポートも含め）XQuery を使用できます。データは JDBC 経由で取得されます。SQL ベースの XQS ビューは、SQL 言語で記述されたストアド・クエリーです。XQS では、SQL ベースのビュー自体がソースとして扱われます。ビューは、<xqsview-source> 要素を介して構成されます。

SQL 問合せを XQS ビューとして使用するには、次の手順を実行します。

1. SQL 問合せに使用するデータベースとスキーマを決定します。接続を表す OC4J data-sources.xml 構成ファイルで、適切な <managed-data-source> または <data-source> 要素を探すか定義します。そのデータソースの JNDI 名を、<xqsview-source> 構成要素の参照として使用します。データソースの JNDI 名は、<xqsview-source> 要素の <dataSource> サブ要素に明示的に指定します。XQS では、JDBC 接続の確立にデータソースを使用します。SQL 問合せに基づいた XQS ビューの場合、<dataSource> 構成要素は必須です。

2. 問合せに必要な入力パラメータを検討します。SQL 言語で記述された問合せの場合は、問合せパラメータとして、:1、:2、:3 などの位置指定のバインド変数を使用します。

3. 必要なデータベース・オブジェクト（表、ビュー、ストアド・プロシージャまたは PL/SQL パッケージ）を決定します。その後、問合せを記述して、入力パラメータの位置指定のバインド変数を指定します。XQS では、SQL 問合せ文（SELECT 文）のみが実行されます。問合せでは、Java または PL/SQL ストアド・プロシージャを起動できます。INSERT や UPDATE の SQL 文、またはデータ定義言語（DDL）文は使用できません。問合せを .sql ファイルとして保存します。

4. データベース問合せで、データをリレーショナル（表）または XML 形式のいずれで戻すかを決定します。Oracle XML データベースを使用していて、問合せ結果が XMLType 型の場合、XQS では結果はそのまま使用され、変換は行われません。結果がすでに XML 形式であることを指定するには、XMLType に SQLResultType 属性を設定します（データベースへの追加のラウンドトリップが必要で、パフォーマンスが低下するため、XQS では問合せの独自の分析は行われません）。

   データベース問合せでデータをリレーショナル形式で戻す場合は、SQLResultType 属性が relational に設定されている必要があります（これはデフォルトでもあります）。この場合、XQS は Oracle XML-SQL ユーティリティ（XSU）を使用して、リレーショナルの結果を XML に変換します。XSU では、結果セットのすべての行が XML 要素に変換されます。デフォルトの要素名は ROW です。デフォルトの要素は、<xqsview-source> 構成要素の rowTag 属性を設定することでカスタマイズできます。行要素は、ユーザーが選択した属性で番号付けできます。XSU では、デフォルトでは行要素には番号は割り当てられません。行を番号付けする場合は、<xqsview-source> 構成要素の rowIdAttr 属性を指定します。rowIdAttr 属性の値は、結果の行要素の番号付け属性の名前です。行は 1 から開始して、1、2、3 のように番号付けされます。

   XSU では、リレーショナルの結果の各列を、行要素にネストする要素に変換します。要素タグは、列名から導出されます。列要素名はカスタマイズできません。ただし、SQL 問合せでリレーショナルの結果を作成する場合には、列に別名を割り当てられます。

   たとえば、デフォルトの Oracle Order Entry スキーマ OE で実行される次のような SQL 問合せがあるとします。

   ```
   select PRODUCT_ID,PRODUCT_NAME,PRODUCT_STATUS,LIST_PRICE from
   PRODUCT_INFORMATION where product_id < :1
   ```

   rowTag="product" および rowIdAttr="record_num" 属性が設定されている場合、この SQL ベースのビューの実行結果は次のようになります。

   ```
   <product record_num="1">
      <product_id>1726<product_id>
      <product_name>LCD Monitor 11/PM<product_name>
      <product_status>under development<product_status>
      <list_price>259<list_price>
   </product>
   <product record_num="2">
   ```

```
    <product_id>1729<product_id>
    <product_name>Chemicals - RCP<product_name>
    <product_status>orderable<product_status>
    <list_price>80<list_price>
</product>
```

SQL 問合せを記述する際には、前述の XML 変換も考慮してください。

5. `.sql` ファイルを保存する場所を決定します。`.sql` ファイルの保存場所は、XQS リポジトリと呼ばれます。場所はビュー・ソースの XQS 構成を介して（`<repository>` 要素を使用して）指定します。または、`OC4JPackager` ユーティリティの実行時に `-repository` オプションを使用して指定します。同じリポジトリに複数の `.xq` および `.sql` ファイルを保存できます。

   前の例の SELECT 文は、SQL ベースの XQS ビューで使用される式です。SQL バインド変数 `:1` を受け入れて SQL 問合せを実行し、結果を要素の XML 順序の形式で戻します。`.sql` ファイルは任意の名前で保存し、XQS 構成に指定できます。ビューを実行する XQS ビュー関数の名前は、`getProductInfo` です。`.sql` ファイルの名前と関数名は別個のものですが、関数名はファイルのデフォルトのベース名と同一にできます。

次に示す要素および属性は、通常、SQL ベースの XQS ビュー用の XQS 構成で使用されます。要素は、`<xqsview-source>` 要素のサブ要素です。

- `<function-name>` サブ要素は、ビューに XQS 関数の目的の名前を指定するために使用されます。`<dataSource>` サブ要素は、data-sources.xml 構成ファイルに定義されているデータソースの JNDI 名を指定するために使用されます。データソースは、JDBC 接続の確立に使用されます。このサブ要素は、SQL ベースのビューには必須です。
  `<input-parameters>` サブ要素およびその `<part>` サブ要素は、XQS 関数の入力パラメータに対応するバインド変数を指定するために使用されます。SQL バインド変数は位置によってバインドされるため、この場合、`<part>` サブ要素の `name` 属性は無視されますが、位置属性は重要です（前の例で説明されているように、`:1` は `position="1"` と等価です）。`<xqsview-source>` 要素の `fetchSize` 属性は、一度のラウンドトリップでデータベースから複数の行が転送されるよう設定するために使用されます。デフォルトは `1` です。この属性は、`java.sql.PreparedStatement` の `setFetchSize` メソッドへのコールに変換されます。JDBC の `setFetchSize` パラメータは（つまり、XQS の `fetchSize` 属性も）単なるヒントであり、バインディングではありません。`<queryName>` サブ要素は、SQL 問合せが含まれている `.sql` ファイルの名前を指定するために使用されます（XQS 関数名と同じベース・ファイル名を使用する場合を除く）。`<repository>` サブ要素は、`.sql` ファイルの場所を指定するために使用されます（`OC4JPackager` ユーティリティの使用時に `-repository` オプションを介して指定した場所にファイルがある場合は不要）。

次に示す `<xqsview-source>` 要素の属性はオプションです。属性が設定されていない場合は、デフォルト値が使用されます。

- `fetchSize="1"`
- `XMLFormat="XSU"`
- `rowTag="ROW"`
- `rowIdAttr=""`
- `SQLResultType="relational"`

注意： XQS 構成要素および属性の詳細は、8-93 ページの「XQS 構成ファイルのリファレンス」を参照してください。また、8-24 ページの「XQS ビューを使用する XQS 関数の構成」も参照してください。

SOAP バインディングを伴う WSDL ソースを使用するための準備

Web サービス操作を介してデータソースにアクセスするには、Web サービスの WSDL ドキュメントを指定し、XQS 構成で WSDL の場所を指定する必要があります。1 つの XQS ビュー関数は、XQS によって実装される 1 つの Web サービス操作に対応します。

WSDL を調査します。XQS 構成ファイルに含まれる <wsdl-source> 要素およびそのサブ要素を使用して、XQS 関数を構成します。その際、次に示すように、WSDL エントリに対応する構成設定（一部はオプション）を使用します。

- WSDL 操作は、XQS 構成の <operation> 要素に対応します。
- WSDL 入力メッセージの場合、入力メッセージの各パートは、<input-parameters> の <part> サブ要素に対応します。
- 適用可能な WSDL サービスは、<service> 要素に対応します。
- 適用可能な WSDL ポートは、<port> 要素に対応します。
- 適用可能な WSDL ポート・タイプは、<portType> 要素に対応します。

また、XQS から WSDL ドキュメントへのアクセス方法を考慮し、<wsdlURL> 要素を使用してドキュメントの場所を指定します。

注意：

- これらの要素のリファレンス情報は、8-93 ページの「XQS 構成ファイルのリファレンス」にあります。
- WSDL ドキュメントおよび対応する XQS 構成の部分の例は、8-30 ページの「WSDL ソースにアクセスする XQS 関数の構成」を参照してください。

カスタム・クラスまたは EJB を使用するための準備（Java または EJB のバインディングを伴う WSDL ソース）

Java 用または EJB 用の WSIF プロバイダを使用すると、カスタム Java クラスまたは EJB を介してデータソースにアクセスできます。これは、Apache Foundation の公開テクノロジであるため、XQS による特別な動作は行われません。必要な手順は次のとおりです。

- 後で問い合せる XML データを戻す、カスタム Java クラスまたは EJB を実装します。
- 必要に応じて Java または EJB のバインディングを伴う WSDL ドキュメントを作成し、目的の操作を定義します。
- 必要になる XML-Java 型マッピングを考慮します。

XQS 構成に関する WSDL の一般的な考慮事項は、8-20 ページの「SOAP バインディングを伴う WSDL ソースを使用するための準備」を参照してください。

クラスまたは Bean の実装方法は、『Oracle Application Server Web Services 開発者ガイド』、また、Java 用および EJB 用の WSIF プロバイダの情報は、『Oracle Application Server Web Services アドバンスト開発者ガイド』を参照してください。

注意： 8-30 ページの「WSDL ソースにアクセスする XQS 関数の構成」も参照してください。

Java バインディングを伴う WSDL ドキュメントの一部を例として示します。これを使用すると、操作名に基づいて、Java メソッド readEntry() などを起動できます。WSDL の <format:typeMapping> の仕様は、XQS 構成の <typeMap> 要素に対応することに注意してください。

```
< definitions>
  ...
  <binding name="JavaBinding" type="tns:AddressBook">
    <java:binding />
    <format:typeMapping encoding="Java" style="Java">
       <format:typeMap typeName="typens:address"
             formatType = "localjava.client.stub.addressbook.wsiftypes.Address" />
       <format:typeMap typeName="xsd:string"
             formatType="java.lang.String" />
    </format:typeMapping>
    <operation name="readEntry">
        <java:operation methodName="readEntry"
              parameterOrder = "name"
              methodType = "instance" />
        <input name="ReadEntryWholeNameRequest" />
    <operation name="readAllMatchingEntries">
     ...
    </operation>
  </binding>
  <service name="AddressBookService">
     <port name="JavaPort" binding="tns:JavaBinding">
        <java:address
          className = "localjava.service.AddressBookImpl" />
     </port>
  </service>
  ...
</definitions>
```

XQS 関数の構成方法

すべての XQuery 式で使用されるそれぞれの XQuery 外部関数、つまり、XQS によって実装されるそれぞれの XQS 関数に対して、アプリケーションの XQS 構成内に、問合せ、アクセスするデータソースおよび入力パラメータに使用される XQS 関数の詳細（名前など）を指定する 1 つのエントリが必要になります。構成には、ドキュメント・ソース、XQS ビュー、WSDL ソースの 3 つの基本カテゴリがあります。また、これらに対応する `<document-source>`、`<xqsview-source>`、`<wsdl-source>` という 3 つの高レベルの構成要素があります。

この項には、次の項目が含まれます。

- ドキュメント・ソースにアクセスする XQS 関数の構成
- XQS ビューを使用する XQS 関数の構成
- WSDL ソースにアクセスする XQS 関数の構成

XQS 構成ファイルの関連する概要は、8-10 ページの「XQS の構成および構成ファイルの概要」を参照してください。

この項で説明した構成要素のリファレンス情報は、8-93 ページの「XQS 構成ファイルのリファレンス」を参照してください。

ドキュメント・ソースにアクセスする XQS 関数の構成

この項では、ドキュメント・ソースにアクセスする XQS 関数の構成方法を説明し、最後に例を示します。この項に出てくる要素は、<document-source> 要素のサブ要素です。追加情報は、8-96 ページの「<document-source>」を参照してください。すべてのサブ要素の情報へのリンクも含まれています。

必須

ドキュメント・ソースに対して、少なくとも次の要素を構成する必要があります。

- <function-name>: この要素を介して、ドキュメント・ソースへのアクセス用に XQS によって実装される XQS 関数の修飾名を指定します。この要素の値はローカル名で、名前空間の属性（namespace または prefix。詳細は 8-98 ページの「<function-name>」を参照）があります。XQS 関数を XQuery 外部関数として宣言する際には、XQuery のプロローグでも同じ名前を使用する必要があります。

  構成の例を示します。

```
<document-source ... >
   <function-name namespace="http://xmlns.oracle.com/ias/xqs">
      myFileSource
   </function-name>
   ...
</document-source>
```

対応する XQuery の宣言と使用方法は次のとおりです。

```
declare namespace xqs = "http://xmlns.oracle.com/ias/xqs";
declare function xqs:myFileSource() external;
for $po in xqs:myFileSource()//po
return $po
```

オプション（場合によっては必須）

場合により必須または適切であるドキュメント・ソースの要素は、次のとおりです。

- <documentURL>: この要素を使用して、ドキュメント・ソースの URL を指定します。次に例を示します。

```
<document-source ... >
   ...
   <documentURL>
      http://host:port/xqsdemos/Repository/pos-2KB.xml
   </documentURL>
   ...
</document-source>
```

> **重要：** ドキュメント URL を指定する際には、次のことに注意してください。ドキュメントがローカル・ファイル・システムにある場合は、http:// プロトコルを使用するよりも、file:// プロトコルを使用してファイルの絶対パスを指定した方が、データの取得速度が速くなります。

<documentURL> 要素を指定すると、XQS 関数が引数を取らないことを暗黙的に指定することになります。<documentURL> を省略することもできます。その場合は、次の例で示すように、実行時に URL を XQS 関数に渡す必要があります。また、この関数を XQuery で宣言して、URL 用のパラメータを 1 つ定義する必要があります。

```
declare namespace xqs = "http://xmlns.oracle.com/ias/xqs";
declare function xqs:myFileSource($bind as xs:string) external;
for $po in xqs:myFileSource
          ("http://host:port/xqsdemos/Repository/pos-2KB.xml")//po
return  $po
```

- <output-element>: このサブ要素を使用すると、XML 要素の修飾名または戻されたデータの出力型（単純型または複合型）を指定できます。このサブ要素は常に必須ではありませんが、この情報は、XQS によるタイプ・チェックに使用できます。次に例を示します。

```
<document-source ... >
   ...
   <output-element namespace="http://xmlns.oracle.com/ias/xqs/CustomerDemo"
                location="http://host:port/xqsdemos/Customers.xsd"> type=Customers
   </output-element>
   ...
</document-source>
```

  <output-element> を定義する <xqsview-source> 要素については、基になる問合せ、または別の <xqsview-source> 関数を使用してネストされている問合せの errorMessage エラー処理オプションを定義する XQS 関数ごとに、<output-element> サブ要素を使用する必要があります。この要件の詳細は、8-68 ページの「XQS エラー処理モードおよび API の使用方法」を参照してください。

- <XMLTranslate>: XML 以外のドキュメント・ソースについては、この要素を使用して、使用する変換ツールを指定し、そのツールに必要なすべての情報を指定します。現在サポートされているのは D3L のみです。D3L を使用するには、<schema-file> サブ要素を介して D3L スキーマ・ファイルを指定する必要があります。D3L の詳細は、8-15 ページの「XML 以外のドキュメント・ソースを使用するための準備」を参照してください。

  次に例を示します。

```
<document-source ... >
   ...
   <XMLTranslate method="D3L">
      <schema-file>
         http://host:port/xqsdemos/paymentInfoD3L.xml
      </schema-file>
   </XMLTranslate>
   ...
</document-source>
```

パフォーマンスとエラー処理

エラー処理の設定を行い、キャッシングやラージ・データセットの特別な処理などのパフォーマンス機能の使用を指定できます。これらの設定には、<document-source> の属性 isCached、largeData、onError、およびサブ要素の <cache-properties> と <error-message> を使用します。これらの機能の構成は、ここでは省略します。8-68 ページの「XQS エラー処理モードおよび API の使用方法」および 8-64 ページの「XQS パフォーマンス機能の使用方法」を参照してください。

例

それぞれの例では、キャッシング、ラージ・データおよびエラー処理のデフォルト設定が使用されています。これは、<document-source> 属性が isCached="false"、largeData="false" および onError="dynamicError" と設定されていることと同じです。

固定の場所にある XML 文書のソースの例を示します。

```
<document-source>
   <function-name namespace="http://xmlns.oracle.com/ias/xqs">
      myFileSource
   </function-name>
   <documentURL>
      http://host:port/xqsdemos/Repository/pos-2KB.xml
   </documentURL>
</document-source>
```

D3L 変換ツールおよび paymentInfoD3L.xml D3L スキーマ・ファイルを使用する、XML 書式以外のドキュメント・ソースの例を示します。

```
<document-source>
   <function-name prefix="xqs">
      paymentStatusInfo
   </function-name>
   <documentURL>http://host:port/xqsdemos/paymentInfo.csv</documentURL>
   <XMLTranslate method="D3L">
      <schema-file>
         http://host:port/xqsdemos/paymentInfoD3L.xml
      </schema-file>
   </XMLTranslate>
</document-source>
```

XQS ビューを使用する XQS 関数の構成

この項では、XQS ビューを使用する XQS 関数の構成方法を説明し、最後に例を示します。この項に出てくる要素は、<xqsview-source> 要素のサブ要素です。追加情報は、8-112 ページの「<xqsview-source>」を参照してください。すべてのサブ要素の情報へのリンクも含まれています。

使用する XQS ビューの作成方法などの準備手順は、8-17 ページの「XQS ビューを使用するための準備」を参照してください。

WSDLvisibility の設定

<xqsview-source> の属性設定 WSDLvisibility="true" を使用すると、XQS ビューを Web サービス操作として公開できます。XQS によって生成される WSDL に、その操作が追加されます。Web サービス操作は、そのアプリケーションの Web サービスの一部になります。WSDLvisibility 属性はオプションで、デフォルト値は false です。

> **注意：** バインド変数を使用する SQL インジェクションのリスクを最低限に抑えるため、XQS では SQL ベースの XQS ビューを Web サービス操作として公開することはできません。SQL ベースの XQS ビュー機能、つまり SQL 問合せに渡す前に、すべてのユーザー入力をアプリケーションでフィルタ処理および検証する必要があります。SQL ベースのビューで WSDLVisibility 属性が true に設定されている場合、XQS により警告が発行され、その設定は無視されます。

XQS ビューを Web サービスとして公開する場合の詳細情報は、8-60 ページの「XQS ビューを Web サービス操作として公開するための OC4JPackager の追加の出力」を参照してください。

必須

後述の説明は、次に示す例が XQS ビューとして mytotals.xq に保存されていることを前提としています。この例では、ファイルからデータを読み取る関数が使用されており、入力パラメータ loc（ファイルの場所を示す文字列変数）を取得して関数に渡し、整数の合計を戻します。

```
declare variable $loc external;
declare namespace xqs = "http://xmlns.oracle.com/ias/xqs";
declare function xqs:readFile($l as xs:string) as xs:int external;
for $po in xqs:readFile($loc)//po return $po//total
```

XQS ビューを使用するには、少なくとも次の要素を構成する必要があります。

- <function-name>: この要素を介して、ビューの実行用に XQS によって実装される XQS 関数の修飾名を指定します。この要素の値はローカル名で、名前空間の属性（namespace または prefix。詳細は 8-98 ページの「<function-name>」を参照）があります。（executeView() コールで直接ビューを使用するのではなく）問合せでこの関数を使用する場合には、<function-name> 要素で指定した関数名を使用して、この XQS 関数を XQuery 外部関数として宣言する必要があります。

構成の例を示します。

```
<xqsview-source ... >
   <function-name namespace="http://xmlns.oracle.com/ias/xqs">
      totals
   </function-name>
   ...
</xqsview-source>
```

- <input-parameters>: XQS ビューで外部変数（SQL ベースのビューの場合はバインド変数）が使用される場合は、XQS によって実装されるこの関数で、ビュー内の外部変数またはバインド変数ごとに、入力パラメータを使用する必要があります。問合せの実行前に、XQS によって、外部またはバインド変数に関数の引数の値が割り当てられます。<input-parameters> 要素を使用して、入力パラメータを構成します。その際には、パラメータごとに 1 つの <part> サブ要素を使用します。XQS ビューでは、この要素は常に必須です。入力パラメータがない場合は、空の要素を使用してください。

  ```
  <xqsview-source ... >
     ...
     <input-parameters/>
     ...
  </xqsview-source>
  ```

  外部文字列変数 loc にバインドされる 1 つの入力パラメータを使用した例を次に示します。XQS ビューの入力パラメータには、<schema-type> または <xquery-sequence> 要素を使用して、オプションでタイプ情報を指定することができます。この例は、xs 接頭辞が XML Schema の名前空間に対応するように設定されていることを前提としています。（<input-parameters>、<part>、<schema-type> および <xquery-sequence> の各要素の詳細は、この章の後半部分に含まれているリファレンス・ドキュメントを参照してください。）

  ```
  <xqsview-source ... >
     ...
     <input-parameters type-match="none" >
        <part position="1" name="loc">
           <schema-type prefix="xs">string</schema-type>
        </part>
     </input-parameters>
     ...
  </xqsview-source>
  ```

SQL ベースのビューのバインド変数は、:1、:2 などの位置で示されます。同一の <input-parameters> および <part> 構成要素を使用します。<part> には名前属性を指定する必要がありますが、名前の値は無視されます。SQL バインド変数は、1 つ目の位置の引数は :1 に、2 つ目の位置の引数は :2 などのように、実際の引数の位置によってそれぞれの値にバインドされます。

次の例では、問合せでの XQS ビューの使用方法を示します。XQuery の宣言および使用方法は、前述の <function-name> および <input-parameters> の説明に対応しています。

```
declare namespace xqs = "http://xmlns.oracle.com/ias/xqs";
declare function xqs:totals($loc as xs:string) as xs:int external;
for $t in xqs:totals("C:¥MyPurchaseOrders.xml") return
<outstanding_balance> fn:sum($t) </outstanding_balance>
```

オプション（場合によっては必須）

場合により必須または適切である XQS ビューの要素は、次のとおりです。

- <output-element>: この要素を使用すると、XML 要素の修飾名または戻されたデータの出力型（単純型または複合型）を指定できます。WSDLvisibility="true" を使用する XQS ビューでは、特に <output-element> 要素の使用が適しており、その属性である location は、要素定義またはタイプ定義を含む XML スキーマ・ファイルをポイントする必要があります。次に例を示します。

```
<xqsview-source WSDLvisibility="true">
   ...
   <output-element namespace="http://xmlns.oracle.com/ias/xqs/CustomerDemo"
                location="http://host:port/xqsdemos/Customers.xsd">
   </output-element>
   ...
</xqsview-source>
```

  WSDLvisibility="false" を使用する XQS ビューでも、<output-element> 要素を使用するとタイプ・チェックに役立ちます。接頭辞 xs は、XML Schema の名前空間に対応するように設定されているとします。

```
<xqsview-source WSDLvisibility="false">
   ...
   <output-element prefix="xs">integer</output-element>
   ...
</xqsview-source>
```

- <queryName>: この要素を使用して、XQS ビューを保存した .xq または .sql ファイルの名前を指定します。.xq または .sql 拡張子は指定してもしなくても構いません（必要に応じて自動的に追加されます）。この <queryName> 要素はオプションです。この要素を省略した場合、.xq または .sql ファイルのベース名は <function-name> 要素で指定された XQS 関数名と同じであるとみなされます。

  注意： XQuery ベースの XQS ビュー・ファイルの拡張子は .xq である必要があります。SQL ベースのビュー・ファイルの拡張子は .sql である必要があります。<xqsview-source> 構成要素に <dataSource> サブ要素が含まれている場合のみ、XQS ではビューが SQL ベースであるとみなされます。詳細は、「SQL ベースの XQS ビューに固有の考慮事項（必須）」を参照してください。

  mytotals.xq に保存された XQS ビューの例を示します。

```
<xqsview-source ... >
   ...
   <queryName>mytotals</queryName>
   ...
</xqsview-source>
```

  <function-name> 要素に関する前述の例では、<queryName> を省略すると、XQS によって totals.xq という名前のファイルが検索されます。

- <repository>: この要素を使用して、XQS ビュー・リポジトリの場所（.xq または .sql ファイルの場所）を指定します。この要素を省略した場合は、OC4JPackager の -repository オプションを使用して指定したリポジトリとみなされます。

  次に例を示します。

```
<xqsview-source ... >
   ...
   <repository>META-INF/xqs/mydir</repository>
   ...
</xqsview-source>
```

- SQLResultType: <xqsview-source> 要素のオプションの属性で、SQL ベースのビューにのみ適用できます。この属性では、データベース問合せで、データをリレーショナル（表）形式または XML のいずれで戻すかを設定します。SQLResultType のデフォルト値は、relational です。Oracle XML データベースを使用している場合は、SQLResultType 属性を XMLType に設定することで、SQL ベースの XQS ビューを使用して（それ以上変換せずに）データベースから XML 型の値を選択できます。XQS では、リレーショナルから XML への変換をデータベースの結果に適用するかどうかの決定は、SQLResultType 属性のみに依存します。データベースへの追加のラウンドトリップが必要で、パフォーマンスが低下するため、XQS では問合せの独自の分析は行われません。

  データベース問合せでデータをリレーショナル形式で戻す場合は、SQLResultType 属性が、明示的にまたはデフォルトで relational に設定されている必要があります。この場合、XQS は Oracle XML-SQL ユーティリティ（XSU）を使用して、リレーショナルの結果を XML に変換します。

- rowTag および rowIdAttr: <xqsview-source> 要素のオプションの属性で、SQL ベースのビューにのみ適用できます。これらの属性が指定されている場合、データベースの結果のリレーショナルから XML への変換で使用された要素および属性のデフォルト名は、これらの値でそれぞれ上書きされます。これらの属性は、SQLResultType 属性が relational に設定されている場合にのみ適用可能です。

  XSU では、結果セットのすべての行が XML 要素に変換されます。デフォルトの要素名は ROW です。デフォルトの要素は、<xqsview-source> 構成要素の rowTag 属性を設定することでカスタマイズできます。行要素は、ユーザーが選択した属性で番号付けできます。XSU では、デフォルトでは行要素には番号は割り当てられません。行を番号付けする場合は、<xqsview-source> 構成要素の rowIdAttr 属性を指定します。rowIdAttr 属性の値は、結果の行要素の番号付け属性の名前です。行は 1 から開始して、1、2、3 のように番号付けされます。設定 rowIdAttr="" はデフォルトの動作と同じで、行の番号付け属性を抑止します。

  rowTag および rowIdAttr 属性は、それぞれ別々に使用されます。つまり、指定できる属性は 1 つのみで、もう一方にはデフォルトを使用します。

  注意： XSU では、リレーショナルの結果の各列を、行要素にネストする要素に変換します。要素タグは、列名から導出されます。列要素名はカスタマイズできません。ただし、SQL 問合せでリレーショナルの結果を作成する場合には、列に別名を割り当てられます。

- fetchSize: <xqsview-source> 要素のオプションの属性で、SQL ベースのビューにのみ適用できます。fetchSize は、JDBC を介した 1 回のラウンドトリップでデータベースから取得する行数を推奨します。この属性は、java.sql.PreparedStatement の setFetchSize メソッドへのコールに変換されます。JDBC の setFetchSize パラメータは（つまり、XQS の fetchSize 属性も）単なるヒントであり、実装で無視される場合もあります。fetchSize 属性のデフォルト値は 1 です。

- XMLFormat: <xqsview-source> 要素のオプションの属性で、現在使用できる値はデフォルトの XSU のみです。その他の値は今後のリリースで使用できるようになり、WebRowset 形式の結果がサポートされます。現行リリースの制限は、JDK 1.5 機能への依存性が原因です。

<output-element> を使用する際の考慮事項

WSDLvisibility 属性を true に設定すると、ビューの Web サービス操作が生成され、WSDL ドキュメントにこの操作の定義が含められます。次に示すように、<xqsview-source> の <output-element> の内容により、WSDL 操作の出力メッセージのタイプが決定されます。

- <output-element> を省略した場合、WSDL 操作の出力メッセージの結果タイプは、次のように宣言されます。

  ```
  <sequence> <any/> </sequence>
  ```

- <output-element> が存在する場合は、その中で、名前空間属性または接頭辞属性の書式で出力要素の名前空間が指定されている必要があります。この名前空間は、WSDL では <import> 要素になります。次に例を示します。

  ```
  <output-element namespace="urn:PurchaseOrders"

  location="http://myhost:80/myapp/PurchaseOrders.xsd" />
  ```

  このコードにより、WSDL で次のインポート要素が生成されます。

  ```
  <types>
          <schema...>
             <import namespace="urn:PurchaseOrders"
  schemaLocation="http://myhost:80/myapp/PurchaseOrders.xsd" />
  ...
  </schema>
  </types>
  ```

  ロケーション属性を使用することで、WSDL の <import> 要素がより完全になります。そのため、ロケーション属性はできるかぎり指定してください。

- <output-element> で属性タイプが指定される場合は、この属性が namespace 属性または prefix 属性とともに使用され、結果要素タイプの修飾名が作成されます。<output-element> のテキスト値は、結果要素自体の名前として使用されます。

  次に例を示します。

  ```
  <output-element namespace="urn:PurchaseOrders"

  location="http://myhost:80/myapp/PurchaseOrders.xsd"
                                       type="POType" >
                              po
  </output-element>
  ```

  このコードは、前述の <import> 要素と、次に示す WSDL 操作の結果タイプの定義になります。

  ```
  <complexType name="POswithRetTypeResultType">
  ...
  <element name="result" nillable="true">
  <complexType >
  <sequence>
            <element name="po" xnlns:_ns1="urn:PurchaseOrders"  type="_ns1:POType"
  minOccurs="0" maxOccurs="unbounded" />
   </sequence>
  ...
  ```

- <output-element> によって type 属性が指定されない場合は、WSDL 操作の出力メッセージの結果要素は、インポートされたスキーマの要素への参照として宣言されます。たとえば、次の構成要素があるとします。

  ```
  <output-element namespace="urn:PurchaseOrders"

  location="http://myhost:80/myapp/PurchaseOrders.xsd">
                              polist                          </output-element>
  ```

このような要素は、WSDL で次のような結果定義を生成します。

```
<complexType name="POListResultType">
...
<element name="result" nillable="true">
<complexType >
<sequence>
            <element ref="_ns1:polist"   minOccurs="0" maxOccurs="unbounded" />

</sequence>
...
```

- <output-element> によって type 属性が指定されていても、それが空の（text 要素値がない）場合、XQS では、結果要素の固定名 item が使用されますが、指定されたタイプ名が使用されます。たとえば、次の構成があるとします。

```
<output-element namespace="urn:PurchaseOrders"

location="http://myhost:80/myapp/PurchaseOrders.xsd"
                                        type="POType" />
```

このような構成では、次のような結果要素の定義が生成されます。

```
<complexType name="POsResultType">
...
<element name="result" nillable="true">
<complexType >
<sequence>
         <element name="item" xnlns:_ns1="urn:PurchaseOrders"  type="_ns1:POType"
minOccurs="0" maxOccurs="unbounded" />
 </sequence>
```

重要：（WSDL の type 属性を介して）結果要素を定義するタイプ、または（WSDL の ref 属性を介して）参照によって結果要素を定義する要素は、インポートされたスキーマのトップレベルのタイプまたは要素の定義を参照する必要があります。前述の例では、POType が、名前空間 urn:PurchaseOrders に対してインポートされたスキーマでトップレベルのタイプ定義であるか、（参照による要素定義に使用する）polist が、次の場所で、インポートされたスキーマのトップレベルで定義されている必要があります。

http://myhost:80/myapp/PurchaseOrders.xsd

SQL ベースの XQS ビューに固有の考慮事項（必須）

<dataSource>: XQS にデータベース接続情報を提供します。SQL 問合せに基づいた XQS ビューには、<dataSource> 構成要素が必須です。OC4J data-sources.xml 構成ファイルの <managed-data-source> または <data-source> 要素に定義されているように、<dataSource> 要素にはデータソースの JNDI 名が含まれている必要があります。

パフォーマンスとエラー処理

エラー処理の設定を行い、キャッシングやラージ・データセットの特別な処理などのパフォーマンス機能の使用を指定できます。これらの設定には、<xqsview-source> の属性 isCached、largeData、onError、およびサブ要素の <cache-properties> と <error-message> を使用します。これらの機能の構成は、ここでは省略します。8-68 ページの「XQS エラー処理モードおよび API の使用方法」および 8-64 ページの「XQS パフォーマンス機能の使用方法」を参照してください。

例

次の例では、前に示したそれぞれの部分を 1 つにまとめて XQS 関数 `totals` が構成されています。また、文字列変数 `loc`（ファイルの場所用）を使用して整数の結果が出力されます。便宜上、ここでも XQS ビュー `mytotals.xq` を使用します。

```
declare variable $loc external;
declare namespace xqs = "http://xmlns.oracle.com/ias/xqs";
declare function xqs:readFile($l as xs:string) external;
for $po in xqs:readFile($loc)//po return $po//total
```

次の例では、キャッシング、ラージ・データおよびエラー処理のデフォルト設定が使用されています。これは、`<xqsview-source>` 属性が `isCached="false"`、`largeData="false"` および `onError="dynamicError"` と設定されていることと同じです。XQS は、Web サービスとして公開されません。

```
<xqsview-source WSDLvisibility="false">
   <function-name namespace="http://xmlns.oracle.com/ias/xqs">
      totals
   </function-name>
   <input-parameters type-match="none" >
      <part position="1" name="loc">
         <schema-type prefix="xs">string</schema-type>
      </part>
   </input-parameters>
   <repository>META-INF/xqs/mydir</repository>
   <queryName>mytotals</queryName>
   <output-element prefix="xs">int</output-element>
</xqsview-source>
```

WSDL ソースにアクセスする XQS 関数の構成

この項では、WSDL ベースのソースにアクセスする XQS 関数の構成方法について説明し、最後に WSDL ドキュメントの一部分の例および対応する構成を示します。この項に出てくる要素は、`<wsdl-source>` 要素のサブ要素です。追加情報は、8-110 ページの「<wsdl-source>」を参照してください。すべてのサブ要素の情報へのリンクも含まれています。

使用する WSDL ソースのタイプにより、8-20 ページの「SOAP バインディングを伴う WSDL ソースを使用するための準備」または 8-20 ページの「カスタム・クラスまたは EJB を使用するための準備（Java または EJB のバインディングを伴う WSDL ソース）」も参照してください。

必須

WSDL ソースに対して、少なくとも次の要素を構成する必要があります。

- `<function-name>`: この要素を介して、WSDL ソースへのアクセス用に XQS によって実装される XQS 関数の修飾名を指定します。この要素の値はローカル名で、名前空間の属性（`namespace` または `prefix`。詳細は 8-98 ページの「<function-name>」を参照）があります。XQS 関数を XQuery 外部関数として宣言する際には、XQuery のプロローグでも同じ名前を使用する必要があります。

  構成の例を示します。

  ```
  <wsdl-source ... >
     <function-name namespace="http://xmlns.oracle.com/ias/xqs">
        getmySearchCachedPage
     </function-name>
     ...
  </wsdl-source>
  ```

- `<input-parameters>`: それぞれの WSDL ソースで、XQS によって実装される関数は、WSDL で指定された入力パートごとに 1 つの入力パラメータを使用する必要があります。`<input-parameters>` 要素を使用して、入力パラメータを構成します。その際には、パ

ラメータごとに 1 つの <part> サブ要素を使用します。WSDL ソースでは、この要素は常に必須です。入力パラメータがない場合は、空の要素を使用してください。

```
<wsdl-source ... >
   ...
   <input-parameters/>
   ...
</wsdl-source>
```

次に、2 つの入力パラメータを使用した例を示します。ここでは、外部文字列変数 key および url を使用しています。<schema-type> 要素を介して指定するタイプ情報は、WSDL ソースの入力パラメータでは必須ではありませんが、使用すると、Web サービスの起動時にタイプ・チェックを実行できます。次の例は、xs 接頭辞が XML Schema の名前空間に対応するように設定されていることを前提としています。
(<input-parameters>、<part> および <schema-type> の各要素の詳細は、この章の後半部分に含まれているリファレンス・ドキュメントを参照してください。)

```
<wsdl-source ... >
   ...
   <input-parameters>
      <part position="1" name="key" >
         <schema-type prefix="xs">string</schema-type>
      </part>
      <part position="2" name="url" >
         <schema-type prefix="xs">string</schema-type>
      </part>
   </input-parameters>
   ...
</wsdl-source>
```

- <wsdlURL>: URL を指定して、XQS に対し、WSDL ドキュメントの検索場所を指示します。次に例を示します。

```
<wsdl-source ... >
   ...
   <wsdlURL>http://api.mySearch.com/mySearch.wsdl</wsdlURL>
   ...
</wsdl-source>
```

- <operation>: WSDL 内の操作名に対応する、実行対象の Web サービス操作を指定します。次に例を示します。

```
<wsdl-source ... >
   ...
   <operation>doGetCachedPage</operation>
   ...
</wsdl-source>
```

- <port>: WSDL 内のポート名に対応する、適切な Web サービス・ポートを指定します。次に例を示します。

```
<wsdl-source ... >
   ...
   <port namespace="urn:mySearch">mySearchPort</port>
   ...
</wsdl-source>
```

前述の例で使用された <function-name> および <input-parameters> の例に対応する、関数の宣言の例を示します。

```
declare namespace xqs = "http://xmlns.oracle.com/ias/xqs";
declare function xqs:getmySearchCachedPage ($key as xs:string, $url as xs:string)
       as xs:base64Binary external;
```

オプション（場合によっては必須）

場合により必須または適切である WSDL ソースの要素は、次のとおりです。

- <service>: この要素を使用して、WSDL 内のサービス名に対応する、適切なサービスを指定します。この要素は、WSDL によって 1 つのサービスしか定義されない場合のみ、オプションです。サービスの構成の例を示します。

```
<wsdl-source ... >
   ...
   <service namespace="urn:mySearch">mySearchService</service>
   ...
</wsdl-source>
```

- <portType>: この要素を使用して、WSDL 内のポート・タイプ名に対応する、適切なポート・タイプを指定します。この要素は、WSDL で 1 つのバインディングのみ（つまり、1 つのポート・タイプのみ）がサポートされている場合にかぎり、オプションです。ポート・タイプの構成の例を示します。

```
<wsdl-source ... >
   ...
   <portType namespace="urn:mySearch">mySearchPort</portType>
   ...
</wsdl-source>
```

- <output-element>: オプションでこの要素を使用すると、XML 要素の修飾名または戻されたデータの出力型（単純型または複合型）を指定できます。この要素は必須ではありませんが、この情報は、XQS によるタイプ・チェックに使用されます。次に例を示します。

```
<wsdl-source ... >
   ...
   <output-element namespace="http://xmlns.oracle.com/ias/xqs/CustomerDemo"
                location="http://host:port/xqsdemos/Customers.xsd">
type="Customers>
   </output-element>
   ...
</wsdl-source>
```

- <typeMap>: Java または EJB のバインディングを伴う WSDL ソースでは、この要素を使用して XML 型を Java 型にマップできます。その際には、次に示すように、サブ要素 <mapping> およびそのサブ要素 <xmlType> を使用します。（<typeMap>、<mapping> および <xmlType> の各要素の詳細は、この章の後半部分に含まれているリファレンス・ドキュメントを参照してください。）

```
<wsdl-source ... >
   ...
   <typeMap>
      <mapping typeClass="org.w3c.dom.Node">
         <xmlType prefix="myeis">Customer</xmlType>
      </mapping>
      ...
   </typeMap>
   ...
</wsdl-source>
```

（<typeMap> 要素は、SOAP のバインディングを伴う WSDL ソースでは無視されます。）

パフォーマンスとエラー処理

エラー処理の設定を行い、キャッシングの使用を指定できます。これらの設定には、<wsdl-source> の属性 isCached と onError、およびサブ要素の <cache-properties> と <error-message> を使用します。これらの機能の構成は、ここでは省略します。8-68 ページの「XQS エラー処理モードおよび API の使用方法」および 8-65 ページの「XQS キャッシングの構成」を参照してください。

例

この例では、前に示したそれぞれの部分のいくつかを 1 つにまとめて、（この後に説明する）mySearch.wsdl で定義される Web サービス操作の XQS 関数が構成されます。

次の例では、キャッシングおよびエラー処理のデフォルト設定が使用されています。これは、<wsdl-source> 属性が isCached="false" および onError="dynamicError" と設定されていることと同じです。

```
<wsdl-source>
   <function-name namespace="http://xmlns.oracle.com/ias/xqs">
      getmySearchCachedPage
   </function-name>
   <wsdlURL>http://api.mySearch.com/mySearch.wsdl</wsdlURL>
   <operation>doGetCachedPage</operation>
   <service namespace="urn:mySearch">mySearchService</service>
   <port namespace="urn:mySearch">mySearchPort</port>
   <input-parameters>
      <part position="1" name="key" >
         <schema-type prefix="xs">string</schema-type>
      </part>
      <part position="2" name="url" >
         <schema-type prefix="xs">string</schema-type>
      </part>
   </input-parameters>
</wsdl-source>
```

mySearch.wsdl

次に示すのは、前述の構成に関連する mySearch.wsdl の各部分です。WSDL の太字で示した部分は、構成要素に対応します。

```
<?xml version="1.0" ?>

<definitions name="mySearch" targetNamespace="urn:mySearch"
   xmlns:typens="urn:mySearch"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns="http://schemas.xmlsoap.org/wsdl/">
   ...
   <message name="doGetCachedPage">
      <part name="key" type="xs:string" />
      <part name="url" type="xs:string" />
   </message>
   <message name="doGetCachedPageResponse">
      <part name="return" type="xs:base64Binary" />
   </message>
   ...
   <portType name="mySearchPort">
      <operation name="doGetCachedPage">
         <input message="typens:doGetCachedPage" />
         <output message="typens:doGetCachedPageResponse" />
      </operation>
      ...
   </portType>
```

```
   ...
   <binding name="mySearchBinding" type="typens:mySearchPort">
      <soap:binding style="rpc"
                    transport="http://schemas.xmlsoap.org/soap/http" />
      <operation name="doGetCachedPage">
         <soap:operation soapAction="urn:mySearchAction" />
         <input>
            <soap:body use="encoded" namespace="urn:mySearch"
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
         </input>
         <output>
            <soap:body use="encoded" namespace="urn:mySearch"
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
         </output>
      </operation>
      ...
   </binding>
   ...
   <service name="mySearchService">
      <port name="mySearchPort" binding="typens:mySearchBinding">
         <soap:address location="http://api.mySearch.com/search/beta2" />
      </port>
   </service>
</definitions>
```

問合せの設計方法

この章の内容は、開発者がすでに XQuery 言語および効果的で効率的な問合せの設計方法の基礎を理解していることを前提としていますが、この項では、XQS を使用する際の特別な考慮事項など、重要なポイントを要約し、問合せの例を示します。この項には、次の項目が含まれます。

- 問合せの考慮事項
- 問合せの例
- 入力パラメータのタイプ・チェック

注意：

- Oracle では、XQuery 1.0 を完全にサポートしているため、XQuery によってサポートされるすべての構文を使用できます。
- 問合せの開発には、Oracle JDeveloper のクエリー・ビルダーや任意のサード・パーティのツールを使用できますが、JDeveloper を含むすべてのツールで、現時点では XQS 用の特別なサポートは提供されていません。

問合せの考慮事項

最初の手順として、当然ながらデータソースを考慮する必要があります。これには、アクセス方法や含まれるデータのタイプも含まれます。一部のデータソースに対して行う必要がある準備手順は、8-14 ページの「データソースを使用するための準備」で前述しています。

次に、問合せ自体の様々な側面を考慮する必要があります。これには、次のことが含まれます。

- 問合せによって入力パラメータが要求されるかどうか。要求される場合は、タイプ・チェックを実行するかどうか。(8-36 ページの「入力パラメータのタイプ・チェック」を参照してください。)
- 入力パラメータのデータ型および問合せの戻り値のデータ型。
- データを変換し、別の XML 構造として出力するかどうか。
- 非定型問合せを使用するか、または問合せを XQS ビュー（.xq ファイル）として保存するか。
- チューニングおよびパフォーマンスに関する考慮事項、および XQS のパフォーマンス機能を使用するかどうか。XQS では、ソースデータをキャッシュできるほか、大量のデータを処理するための特別なモードも装備しています。(詳細は、8-64 ページの「XQS パフォーマンス機能の使用方法」を参照してください。)

前述のように、XQS では、データソースにアクセスするための外部 XQuery 関数が実装されます。XQS によって作成されるこの関数の宣言を、選択した関数名を使用して XQuery のプロローグに含める必要があります。また、このプロローグは、適切な名前空間を参照する必要があります。宣言する関数名は、データソースの構成時に <function-name> 要素で指定した名前と一致する必要があります。(8-21 ページの「XQS 関数の構成方法」を参照してください)。

問合せの例

XML 文書からデータを取得する、単純な問合せを次に示します。このドキュメントの場所は、実行時に XQS 関数に渡されます。

```
declare namespace xqs = "http://xmlns.oracle.com/ias/xqs";
declare function xqs:get_poSQ($bind as xs:string) external;
for $po in xqs:get_poSQ("http://host:port/xqsdemos/Repository/pos-2KB.xml")
return  $po
```

XQS では、XQS 構成に基づいて関数が実装されます。関数名（この例では get_poSQ）は、ユーザーが選択する名前です。この関数は、ドキュメント・ソース pos-2KB.xml の名前と場所を示す文字列を入力として使用し、このファイルから注文（purchase order）データを戻します。

次に、XQuery コードを介してパラメータ値を渡す問合せの例を示します。この XQS 関数は example で、構成に基づいて XQS によって実装されます。

```
declare namespace xqs = "http://xmlns.oracle.com/ias/xqs";
declare function xqs:example($i as xs:int, $d as xs:duration,
                             $h as xs:hexBinary, $bin as xs:base64Binary,
                             $t as xs:boolean) external;

for $result in xqs:example(xs:int(1),
                           xs:duration("P1Y2MT2H"),
                           xs:hexBinary("0FB7"),
                           xs:base64Binary("vYrfOJ39673//-BDiIIGHSPM=+"),
                           xs:boolean("true"))
return $result;
```

次に、より複雑な問合せの例を示します。この問合せでは、顧客の名前を指定すると、その顧客の注文に関する支払記録が検索されます。そして、顧客の情報および注文が戻されます。この例には、2 つのデータソースが関係しています。1）顧客情報は WSIF のカスタム拡張機能 myeis を介してアクセスされます。2）支払情報は Excel シート（XML 以外のドキュメント・ソース）に保存されています。顧客情報のデータソースへのアクセスには、XQS 関数 customerInfo が使用され、顧客名を示す文字列が入力として使用されます。この関数は、顧

客情報のデータソースにアクセスし、指定された顧客の情報（顧客キーを含む）を生成します。その後、XQS 関数 paymentStatusInfo を使用して支払情報の Excel ファイルにアクセスし、顧客情報のデータソースと同じキーを持つ顧客の注文情報を戻します。この XQuery コードには、結果の XML 変換が含まれています。

```
declare namespace xqs = "http://xmlns.oracle.com/ias/xqs";
(: returns payment info for all customers:)
declare function xqs:paymentStatusInfo() external;

(: returns customer info given customer name :)
declare function xqs:customerInfo ($name as xs:string) external;

(: customer name passed in to query :)
declare variable $custName external;

let $custInfo := xqs:customerInfo($custName)
for $custOrderInfo in xqs:paymentStatusInfo()/excel/Row[CustomerKey eq $custInfo/key]
return
   <result>
      <MYEIS_RESULT>
         <Row>
            <Name> { $custInfo/name } </Name>
            <Company> { $custInfo/company} </Company>
            <Address> { $custInfo/address} </Address>
            <City> { $custInfo/city} </City>
            <State> { $custInfo/state} </State>
            <Zip> { $custInfo/zip} </Zip>
         </Row>
      </MYEIS_RESULT>
      <EXCEL_RESULT>
         <Row>
            <OrderID>{$custOrderInfo/OrderId}</OrderID>
            <Amount>{$custOrderInfo/Amount}</Amount>
            <PaymentStatus>{$custOrderInfo/PaymentStatus}</PaymentStatus>
         </Row>
      </EXCEL_RESULT>
   </result>
```

入力パラメータのタイプ・チェック

WSDL ソースまたは XQS ビューで、基礎となるデータソースに入力引数を送る前に、XQuery による追加のタイプ・チェックの手順を実行する場合は、次の手順を行います。

1. <input-parameters> 要素の下にある <part> サブ要素を介して、構成内で適切な XQS 関数に対して指定した入力パラメータごとに、パラメータ・タイプを指定します。XQS ビュー・ソースの場合は、必要に応じて、次のいずれかを介して指定します。

   - <part> の <schema-type> サブ要素
   - 入力順序の場合は、<part> の <xquery-sequence> サブ要素、および <xquery-sequence> の <itemType> サブ要素

   WSDL ソースの場合は、順序は入力としてサポートされないため、関係があるのは <schema-type> のみです。

   （これらの各要素の詳細は、この章の後半部分に含まれているリファレンス・ドキュメントを参照してください。）

2. XQuery のプロローグで関数を宣言する際に、パラメータ・タイプを含めます。
3. 入力パラメータがユーザー定義タイプの場合は、タイプ定義を含むスキーマまたはファイルをインポートします。

次に例を示します。最初は XQS 構成です。

```
<wsdl-source isCached="false">
   <function-name namespace="http://xmlns.oracle.com/ias/xqs">
      SplitRatio
   </function-name>
   ...
   <input-parameters>
      <part position="1" name="parameters">
         <schema-type namespace="http://www.xignite.com/services/">
            GetSplitRatio
         </schema-type>
      </part>
   </input-parameters>
</wsdl-source>
```

次は、対応する XQuery 式です。

```
import schema namespace xignite="http://www.xignite.com/services/"
       at "http://www.xignite.com/xSecurity.asmx?wsdl"
declare namespace xqs="http://xmlns.oracle.com/ias/xqs";
declare function xqs:SplitRatio($params as xignite:GetSplitRatio)
        as xignite:GetSplitRatioResponse external;

let $in := <xignite:GetSplitRatio> .....</xignite:GetSplitRatio>

let $y := xqs:SplitRatio($in) return <split>$y//xignite:Ratio</split>
```

実行時に、`xqs:SplitRatio` に渡された引数のタイプが、インポートされたスキーマのタイプ `xignite:GetSplitRatio` であることが XQuery によって確認されます。

アプリケーション・コードの開発方法 : XQS クライアント・インタフェースを使用した場合

この項では、(8-10 ページの「XQS クライアント・インタフェースの概要」で最初に説明した) XQS で提供されるクライアント API を使用する場合の手順について説明し、例を示します。ここでは、Java クライアント API、EJB クライアント API および JSP タグ・ライブラリについて説明します。Web サービス・クライアントに関して、XQS ビューを Web サービス操作として公開する XQS の詳細は、パッケージ化の項で後述しています (8-60 ページの「XQS ビューを Web サービス操作として公開するための OC4JPackager の追加の出力」を参照)。それ以外では、XQS によって SOAP 1.1 バインディングを伴う標準の WSDL が生成されます。ここに示す内容は、Web サービス・クライアントの作成に関する知識があることを前提としています。

> **注意 :** XQS では、Web サービス・クライアント用の Java コードは生成されません。各ユーザーの責任において、Web サービスを動的に起動するか、WSDL に基づいてクライアント起動コードを生成してください。

この項には、次の項目が含まれます。

- サポートされている問合せパラメータのタイプ
- XQS クライアント API を使用するための一般的なコーディング手順
- ステートフル・クライアントとステートレス・クライアント
- Java インタフェース・クライアント API の使用方法
- EJB クライアント API の使用方法
- JSP タグ・ライブラリの使用方法
- Web サービス操作として公開された XQS ビューの使用方法

注意 :

- XQS クライアント・インタフェースを使用する前に、8-14 ページの「データソースを使用するための準備」および 8-21 ページの「XQS 関数の構成方法」の説明に従って、データソースの準備と構成を完了しておく必要があります。また、EJB を使用する場合は `ejb-jar.xml` ファイル、Web モジュールを使用する場合は `web.xml` ファイルなど、追加の構成もすべて完了しておく必要があります。
- どのクライアント API を使用するかは、使用するデータソースのタイプとは関係しません。これらは、完全に区別して考慮してください。

サポートされている問合せパラメータのタイプ

XQS の `oracle.xqs.client.QueryParameterI` インタフェースは、XQS の Java クライアント API および EJB クライアント API で、問合せ用のバインド・パラメータの配列に使用します。このクラスは、XQS の JSP タグ・ライブラリでも、背後で使用されます。(追加情報は、8-81 ページの「XQS QueryParameterI リファレンス」を参照してください。) 表 8-2 は、XQS でサポートされている XML 型、および `QueryParameterI` インタフェースのインスタンスを介して入力値を渡すために使用する Java 型の対応を示しています。

表 8-2 バインド・パラメータ用に XQS でサポートされている型

サポートされている XML 型	QueryParameterI インタフェースの対応する Java 型
boolean	boolean
string	java.lang.String
int	int
integer	int、long、java.math.BigInteger
long	long
float	float
double	double
decimal	java.math.BigDecimal
base64Binary	java.lang.String
hexBinary	java.lang.String
anyURI	java.net.URI
dateTime	java.util.GregorianCalendar (前提条件 : カレンダ値のタイムゾーンが適切に初期化されていること) java.lang.String (dateTime の字句表現。説明は http://www.w3.org/TR/xmlschema-2/#dateTime を参照。)
duration	java.lang.String (期間の字句表現。説明は http://www.w3.org/TR/xmlschema-2/#duration を参照。)
dayTimeDuration	double (dayTime 期間の秒単位の表現)
yearMonthDuration	int (yearMonth 期間の月単位の表現)
anyType、ユーザー定義の XML 型	org.w3c.dom.Node

XQS クライアント API を使用するための一般的なコーディング手順

XQS クライアント API を使用するためには、次に示す、基本的なコーディング手順を行います。

1. 問合せを作成します。非定型問合せを使用する場合は、問合せをハードコードするか、ユーザー入力からの問合せを受け入れるコードを記述します。XQS ビューを使用する場合は、`.xq` ファイルに問合せを保存します。8-34 ページの「問合せの設計方法」も参照してください。
2. 静的メソッド `getXQSFacade()` を使用して、または JSP ページに XQS JSP タグ・ライブラリを含めることで、`XQSFactory` ファクトリから `XQSFacadeI` インタフェースのインスタンスを取得します。
3. 入力パラメータを指定します。Java クライアント API または EJB API の場合は、タイプ `oracle.xqs.client.QueryParameterI` の配列を作成し、静的メソッド `getQueryParameter()` を使用して `XQSFactory` ファクトリから `QueryParameterI` インタフェースのインスタンスを取得して、その配列を移入します。（8-81 ページの「XQS QueryParameterI リファレンス」を参照してください。）JSP タグ・ライブラリの場合は、`execute` または `executeCursor` の `param` サブタグを使用します。（8-88 ページの「XQS param タグ」を参照してください。）前項の「サポートされている問合せパラメータのタイプ」も参照してください。
4. 問合せを実行します。
5. 結果を読み取り、処理します。ステートレス・クライアントの場合は、XQS によって、`java.util.Vector` インスタンス内に一度にすべての結果が戻されるか、（JSP タグを使用している場合は）`java.util.ArrayList` インスタンスと XML 文書が戻されます。ステートフル・クライアントの場合は、Oracle XQuery タイプ `oracle.xml.xqxp.datamodel.XMLItem` を使用するか、（JSP タグを使用している場合は）`java.lang.Object` インスタンスと XML 文書のノードを使用して、アイテムごとに結果を取得します。XQS EJB API および JSP タグ・ライブラリの場合は、ステートレスとステートフルのいずれかのアクセス・パターンを選択して使用できます。

   `XQSFacadeI` インタフェースでは、ステートフル・アクセス・パターンを使用します。`XQSFacadeI` インタフェースの使用に関する情報は、8-41 ページの「Java インタフェース・クライアント API の使用方法」を参照してください。
6. 必要に応じて、エラーを取得し処理します。問合せの XQS 関数の実行中にエラーが発生した場合でも処理が続行されるように XQS を構成した場合は、エラーに関する情報を取得できます。8-68 ページの「XQS エラー処理モードおよび API の使用方法」を参照してください。
7. ステートフル・クライアントの使用時には、クライアント・オブジェクトをクローズして、問合せに関連付けられているすべてのリソースを解放します。

注意 : ここでは、セキュリティまたはパフォーマンスの考慮事項については言及しません。これらの項目の情報は、8-12 ページの「XQS アプリケーションのセキュリティ」および 8-64 ページの「XQS パフォーマンス機能の使用方法」を参照してください。

ステートフル・クライアントとステートレス・クライアント

XQS クライアント API は、問合せ結果を XML 順序として戻します。この順序には 1 つ以上の XML 項目が含まれ、各項目は、XML 文書、XML ノードまたはプリミティブ値になります。問合せを実行することで、結果は定義されますが、すべての結果項目が即座にマテリアライズされてプロセス・メモリーに格納されるわけではないことを認識しておくことが重要です。リレーショナル問合せの結果セットのように、XML 順序でのカーソルの位置は、最初の項目の前になるように暗黙的に維持されており、次へ移動するための（next）操作で移動できます。暗黙カーソルを使用して次へ（next）の操作のコールを繰り返すことで XML 順序にアクセスした場合は、次への操作をコールするたびに、要求された次の項目を作成するための XQuery 式の評価が、実際にトリガーされる可能性があります。XQuery によって式が評価される正確なタイミング、また、評価が段階的に行われるか一度に全部行われるかは、その XQuery 式および XQuery 実装の最適化レベルによって異なります。XQS クライアント API では、このようにカーソルでアクセスするかわりに、シングル・ステップですべての問合せ結果を読み取り、プロセス・メモリーに格納することもできます。

シングル・ステップ・モードおよびカーソル・モードの実行には、それぞれ長所と短所があります。ほとんどの状況では、シングル・ステップによる実行が適しています。これは、関連するリソース（基礎となるデータソースへの接続、およびその式専用の内部 XQuery リソース）が、実行の完了後にただちに解放されるためです。また、データソースへ繰り返しアクセスすることによるパフォーマンスへの潜在的な影響を受けることもありません。ただし、結果順序に含まれる項目の全体のサイズが大きすぎて、順序全体を一度にマテリアライズするとプロセス・メモリーが不足する場合には、シングル・ステップによる実行を使用できません。このような場合は、問合せが結果を項目ごとに提供するのを妨げるもの（たとえば集計式など）がないかぎり、カーソル・モードが唯一の手段になります。

シングル・ステップ・モードまたはカーソル・モードを選択できるように、XQS では 2 種類のクライアント API を用意しています。シングル・ステップ・モード用のステートレスおよびカーソル・モード用のステートフルです。（この場合の「ステート」は、XQuery エンジンの内部状態を指します。）これらのクライアント API は、次のように動作します。

- ステートレス API を介して XQuery 式を実行した場合は、結果順序全体が一度にマテリアライズされ、すべての内部状態およびリソースが、直後に解放されます。ステートレス実行の唯一の要件は、結果順序全体がプロセス・メモリーに収まることです。
- 対照的に、ステートフル API を介して XQuery 式を実行した場合は、現在の位置が結果順序の最初の項目の前になります。最初の項目から順番に各項目を取得するには、クライアントが次へ（next）の操作を起動する必要があります。問合せの状態は、現在の位置および次の項目を評価するのに必要なすべての内部リソースによって構成されます。この状態は、順序の最後の項目が戻されるか、問合せが明示的にクローズされるまで、解放されません。ただし、注意しなければならないのは、ステートフル実行の少ないメモリー要件を生かすには、クライアント・コードによって、各結果項目の処理後にその項目を解放する必要があるということです。

XQS EJB クライアント API および JSP クライアント API には、それぞれステートレス実行用の機能とステートフル実行用の機能があります。XQS Java インタフェース（`XQSFacadeI`）クライアント API は、ステートフル・アクセスを提供します。すべての結果項目をただちに取得し、クライアント・オブジェクトを解放することで、ステートレス・アクセスをシミュレートできます。

Java インタフェース・クライアント API の使用方法

8-39 ページの「XQS クライアント API を使用するための一般的なコーディング手順」を参照してください。（問合せの作成後に）XQS Java クライアント API を使用してこれらの手順を適用するには、次の手順に従います。

1. 静的メソッド `getXQSFacade()` を使用して、`XQSFactory` ファクトリから `XQSFacadeI` インタフェースのインスタンスを取得します。次の各手順のメソッド・コールは、このインスタンスからコールされます。
2. 入力パラメータ用の `QueryParameterI` 配列を作成し、静的メソッド `getQueryParameter()` を使用して `XQSFactory` ファクトリから `QueryParameterI` インタフェースのインスタンスを取得して、その配列を移入します。
3. `execute()` メソッドを使用して非定型問合せを実行するか、`executeView()` メソッドを使用して XQS ビューを実行し、問合せと `QueryParameterI` 配列（入力パラメータがない場合は `null`）を渡します。`executeView()` を使用する場合は、そのビューの XQS 関数の名前空間も渡します。
4. 問合せ内で使用したデータソースの構成に `emptySequence` エラー・モードまたは `errorMessage` エラー・モードを使用した場合は、必要に応じて `getErrors()` メソッドを使用して、問合せの実行時に発生したエラーを取得できます。これにより、`XQSErrorI` オブジェクトのコレクションに対するイテレータが戻されます。エラーの構成と処理の詳細は、8-68 ページの「XQS エラー処理モードおよび API の使用方法」を参照してください。
5. `getNextItem()` メソッドを繰り返し使用して、項目ごとに結果を取得します。各項目は、`XMLItem` インスタンス内に戻されます。必要に応じてこれらの項目を処理します。
6. `close()` メソッドを使用して、問合せに関連付けられているすべてのリソースを解放します。

これらの手順は、次の例の中で示されています。8-84 ページの「XQSFacadeI リファレンス」も参照してください。

例 1: XQSFacadeI API

この例は、`XQSFacadeI` API の一般的な使用方法を示しています。この問合せでは、バインド・パラメータが順序内に戻され、バインド方法とデータの取得方法が示されます。関数が使用されないため、構成は不要です。

問合せ結果を処理するために、コードには（`oracle.xml.xqxp.datamodel` パッケージ内で）`XMLItem` クラスが使用されています。

```
import oracle.xml.xqxp.datamodel.XMLItem;
import oracle.xml.parser.v2.XMLDocument;
import oracle.xml.parser.v2.XMLElement;
import oracle.xqs.client.XQSFacadeI;
import oracle.xqs.client.QueryParameterI;
import oracle.xqs.client.XQSErrorI;
import java.util.Vector;

public class XQSFacadeITest {
public static void main(String[] args) throws Exception {
 try{
          //get XQSFacadeI
          XQSFacadeI facade = XQSFactory.getXQSFacade();
          String xquery = "declare variable $bind1 external;\n"+
             "declare variable $bind2 external;\n"+
             "declare variable $bind3 external;\n"+
             "declare variable $bind4 external;\n"+
             "declare variable $bind5 external;\n"+
             "declare variable $bind6 external;\n"+
             "declare variable $bind7 external;\n"+
             "declare variable $bind8 external;\n"+
             "declare variable $bind9 external;\n"+
```

```
        "declare variable $bind10 external;\n"+
        "declare variable $bind11 external;\n"+
        "declare variable $bind12 external;\n"+
        "declare variable $bind13 external;\n"+
        "declare variable $bind14 external;\n"+
        "declare variable $bind15 external;\n"+
        "declare variable $bind16 external;\n"+
        "("+"$bind1,$bind2,$bind3,$bind4,$bind5,$bind6,$bind7,\n"+
        "$bind8,$bind9,$bind10,$bind11,$bind12,$bind13,$bind14,$bind15,
           $bind16)";

    QueryParameterI params[] = new QueryParameterI[16];
    params[0]=XQSFactory.getQueryParameter ("bind1");
    params[0].setString("test");
    params[1]=XQSFactory.getQueryParameter ("bind2");
    params[1].setInteger(new BigInteger("100"));
    params[2]=XQSFactory.getQueryParameter ("bind3");
    params[2].setBoolean(true);
    params[3]=XQSFactory.getQueryParameter ("bind4");
    params[3].setFloat(-1);
    params[4]=XQSFactory.getQueryParameter ("bind5");
    params[4].setDuration("P1Y2M3DT10H30M");
    params[5]=XQSFactory.getQueryParameter ("bind6");
    params[5].setDouble(-11.0);

    TimeZone LAtz = new SimpleTimeZone(-28800000,
    "America/Los_Angeles",
          Calendar.APRIL, 1, -Calendar.SUNDAY,
          7200000,
          Calendar.OCTOBER, -1, Calendar.SUNDAY,
          7200000,
          3600000);
    GregorianCalendar cal = new GregorianCalendar(LAtz);

params[6]=XQSFactory.getQueryParameter ("bind7");
params[6].setDateTime(cal,true);

URI uri = new URI("http://www.test.com");

params[7]=XQSFactory.getQueryParameter ("bind8");
params[7].setAnyURI(uri);
params[8]=XQSFactory.getQueryParameter ("bind9");
params[8].setInt(-7);
params[9]=XQSFactory.getQueryParameter ("bind10");
params[9].setLong(50l);
params[10]=XQSFactory.getQueryParameter ("bind11");
params[10].setDecimal(new BigDecimal(999999));
params[11]=XQSFactory.getQueryParameter ("bind12");
XMLDocument document = new XMLDocument();

    Element elem = document.createElementNS("http://client.xqs.oracle/",
       "tns:result");
    document.appendChild(elem);

params[11].setNode(elem);
params[12]=XQSFactory.getQueryParameter ("bind13");
params[12].setBase64Binary("vYrfOJ39673//-BDiIIGHSPM=+");
params[13]=XQSFactory.getQueryParameter ("bind14");
params[13].setHexBinary("0FB7");
params[14]=XQSFactory.getQueryParameter ("bind15");
params[14].setDayTimeDuration(60.5);
params[15]=XQSFactory.getQueryParameter ("bind16");
params[15].setYearMonthDuration(13);
```

```
facade.execute(xquery, params);
// lookup functions
XMLItem item = facade.getNextItem();

while(item != null) {
  if (item.instanceOfType(XMLItem.XMLITEM_STRING)) {
     System.out.println("string item value: expected: \"test\",
     actual:", item.getString());
  }
  else if (item.instanceOfType(XMLItem.XMLITEM_INT)) {
     System.out.println("int item value: expected: \"-7\", actual: ",
     item.getInt());
  }
  else if (item.instanceOfType(XMLItem.XMLITEM_LONG)) {
     System.out.println("long item value: expected: \"501\", actual: ",
     item.getInt());
  }
  else if(item.instanceOfType(XMLItem.XMLITEM_INTEGER)) {
     if(item.intFormat())
        System.out.println("integer item value: expected: \"100\",
        actual:", item.getInt());
     else
       System.out.println("integer item value: expected: \"100\",
       actual: ", item.getDecimal().intValue());
     }
     else if(item.instanceOfType(XMLItem.XMLITEM_BOOLEAN)) {
        System.out.println("boolean item value: expected: \"true\",
        actual: ", item.getBoolean());
     }
     else if(item.instanceOfType(XMLItem.XMLITEM_FLOAT)) {
        System.out.println("float item value: expected: \"-1\",
        actual: ",(float)item.getDouble());
     }
     else if (item.instanceOfType(XMLItem.XMLITEM_XDT_DAYTIMEDURATION)) {
        System.out.println("dayTimeDuration item value: expected: \"60.5
        seconds\", actual: ", item.getDayTimeDuration());
     }
     else if (item.instanceOfType(XMLItem.XMLITEM_XDT_YEARMONTHDURATION)) {
        System.out.println("yearMonthDuration item value: expected: \"13
        months\", actual: ",  item.getYearMonthDuration());
     }
     else if (item.instanceOfType(XMLItem.XMLITEM_DURATION)) {
        System.out.println("duration item value: expected: \"
        P1Y2M3DT10H30M \", actual: ", item.getLexicalValue());
     }
     else if(item.instanceOfType(XMLItem.XMLITEM_DOUBLE)) {
        System.out.println("double item value: expected: \" -11.0\",
        actual: ", item.getDouble());
     }
     else if(item.instanceOfType(XMLItem.XMLITEM_DATETIME)) {
        System.out.println("dateTime item value: expected Timezone:
        \"Pacific (Latz) \", actual: ", item.getCalendar().getTimeZone());
     }
     else if (item.instanceOfType(XMLItem.XMLITEM_ANYURI)) {
        System.out.println("anyURI item value: expected:
        \"http://www.test.com\", actual: ", item.getString());
     }
     else if (item.instanceOfType(XMLItem.XMLITEM_DECIMAL)) {
        System.out.println("decimal item value: expected: \"999999 \",
        actual: ", item.getDecimal().intValue());
  }
  else if(item.instanceOfType(XMLItem.XMLITEM_NODE))
  {
```

```
            XMLElement node = (XMLElement)item.getNode();

            //when obtained from document function, it returns XMLDocument
            if(node instanceof XMLDocument)
               node = (XMLElement)node.getFirstChild();
            System.out.println("node item value: expected namespace: \"
               http://client.xqs.oracle \", actual: ",node.getNamespaceURI());
            System.out.println("node item value: expected local name: \"result\",
               actual: ",node.getLocalName());
         }
         else if (item.instanceOfType(XMLItem.XMLITEM_HEXBINARY)) {
            System.out.println("hexBinary item value: expected: \"0FB7\",
               actual: ", item.getString());
         }
         else if (item.instanceOfType(XMLItem.XMLITEM_BASE64BINARY)) {
            System.out.println("base64Binary item value: expected: \"
               vYrfOJ39673//-BDiIIGHSPM=+ \", actual: ", item.getString());
         }
         else
            {
            System.out.println("item type not supported:"+item.getLexicalValue());
            }
            item = facade.getNextItem();
         }
      }catch(Exception ex){
         ex.printStackTrace();
         fail(ex.getMessage());
         assertTrue("xquery execution failed", false);
    }
    facade.close();
  }
}
```

例 2: XQSFacadeI API を使用した非定型問合せ

この項では、Java クラスを使用して、クライアント・コード内の XQSFacadeI API および関連する XQS 構成の使用の手順を示します。この例の非定型問合せでは、XQS ソースが使用されます。また、この XQS ソースでは、ドキュメント・ソースが使用されます。

問合せ　この例では、次の非定型問合せが使用されます。

```
declare namespace xqs="http://xmlns.oracle.com/ias/xqs";
declare function xqs:get_poSQ($bind as xs:string) external;
for $po in xqs:get_poSQ("http://localhost:8888/myrepository/pos-2KB.xml")
    return $po
```

構成　次に、この問合せで、次の例に示すように構成された XQS ビュー・ソースに対応する XQS 関数 get_poSQ が使用されます。.xq ファイルの名前と場所を（<queryName> 要素および <repository> 要素を介して）構成内で指定しなかった場合、デフォルトでは、ファイル名は関数名、ファイルの場所は OC4JPackager の実行時に -repository オプションを介して指定した場所であるとみなされます。

```
<xqsview-source>
   <function-name prefix="xqs">get_poSQ</function-name>
   <input-parameters>
      <part position="1" name="bind" >
         <schema-type prefix="xs">string</schema-type>
      </part>
   </input-parameters>
</xqsview-source>
```

ファイル get_poSQ.xq で定義された XQS ビュー get_poSQ では、ドキュメント・ソースが使用されます。このドキュメント・ソースは、次のように定義されます。

```
declare namespace xqs = "http://xmlns.oracle.com/ias/xqs";
declare variable $bind external;
declare function xqs:genericFile($bind as xs:string) external;
for $po in xqs:genericFile($bind)//po
   return $po
```

また、ドキュメント・ソース genericFile は、次のように構成されます。

```
<document-source>
   <function-name namespace="http://xmlns.oracle.com/ias/xqs">
      genericFile
   </function-name>
</document-source>
```

genericFile 関数では、ドキュメント URL が入力として使用されるため、<documentURL> 要素は使用されていません。

Java コード　次はコードです。このコードは、次のことを行う完全なクラスです。

- 静的メソッド getXQSFacade() を使用して、XQSFactory ファクトリから XQSFacadeI インタフェースのインスタンスを取得します。
- XQS ビュー get_poSQ を使用する非定型問合せを定義します。問合せは文字列バッファ xqueryBuf に統合され、その後、文字列 xqueryStr に格納されます。バックエンド・ドキュメント・ソースの URL は、get_poSQ 関数によって入力として使用されます。
- 問合せを実行します。この問合せには入力パラメータがないため、execute() メソッドに、QueryParameterI 配列のかわりに null が渡されます。（次の、8-46 ページの「例 3: XQSFacadeI API を使用した XQS ビュー」の例では、入力パラメータが使用されています。）
- ここではエラー処理については示していませんが、ソースの構成が適切であれば（エラー・モード errorMessage または emptySequence を使用）、エラー・オブジェクトを取得して処理できます。（次の例では、エラー処理についても示しています。もしくは、追加情報として、8-68 ページの「XQS エラー処理モードおよび API の使用方法」を参照してください。）
- while ループは、項目ごとに結果順序を経由します。そのため、ノード内の各項目は、XML 文書 doc に格納されます。そこから、結果を表示したり、必要に応じてさらに処理できます（この時点からの処理は、XQS 固有ではありません）。
- close() コールにより、カーソルがクローズされ、関連付けられているリソースが解放されます。

```
import oracle.xml.xqxp.datamodel.XMLItem;
import oracle.xml.parser.v2.XMLDocument;
import oracle.xml.parser.v2.XMLElement;
import oracle.xml.parser.v2.XMLNode;
import oracle.xqs.client.XQSFacadeI;

public class XQSFacadeITest {

    public static void main(String[] args) throws Exception{
        XQSFacadeI bean = XQSFactory.getXQSFacade();
        StringBuffer xqueryBuf = new StringBuffer();
        xqueryBuf.append
             ("declare namespace xqs=\"http://xmlns.oracle.com/ias/xqs\";\n");
        xqueryBuf.append
             ("declare function xqs:get_poSQ($bind as xs:string) external;\n");
        xqueryBuf.append("for $po in xqs:get_poSQ
                    (\"http://localhost:8888/myrepository/pos-2KB.xml\") \n");
        xqueryBuf.append("return $po\n");
```

```
        String xqueryStr = xqueryBuf.toString();

        //no parameters to be bound, so pass null
        bean.execute(xqueryStr, null);

        XMLItem item = bean.getNextItem();

        XMLDocument doc = new XMLDocument();
        XMLElement rootElem = (XMLElement)doc.createElement("QueryResult");
        doc.appendChild(rootElem);

        while(item != null) {
            /* the test is actually superfluous - all items are expected to nodes
               for Purchase Order, here we show a standard treatment of items that
               are XML nodes, including entire documents
               (where node type is DOCUMENT_NODE)
            */
            if(item.getItemType().isNode()) {
                   rootElem.appendChild(doc.importNode(
                      item.getNode().getNodeType()==
                                          XMLNode.DOCUMENT_NODE?(XMLNode)
                      ((XMLDocument)item.getNode()).getDocumentElement():
                      (XMLNode)item.getNode(), true));
            }
            item = bean.getNextItem();
        }
        bean.close();
    }
}
```

例 3: XQSFacadel API を使用した XQS ビュー

この項では、XQSFacadeI API を使用し、executeView() コールを介して XQS ビューを直接実行する場合の例を示します。

構成 BasicFileWS という名前の XQS ビューの構成を示します。

```
<xqsview-source WSDLvisibility="true" isCached="false" onError="errorMessage">
   <function-name  prefix="xqs">BasicFileWS</function-name>
   <input-parameters type-match="none" >
      <part position="1" name="custName">
         <schema-type prefix="xs">string</schema-type>
      </part>
   </input-parameters>
   <output-element namespace="http://xmlns.oracle.com/ias/xqs/CustomerDemo"
               location="http://host:port/xqsdemos/MyTypes.xsd"
type="CustomerOrdersType">
   </output-element>
</xqsview-source>
```

<queryName> 要素がないため、.xq ファイルの名前は、(指定された関数名と同じ) BasicFileWS.xq とみなされます。このビューでは、文字列のパラメータ custName が入力として使用されます。

次に、ビュー BasicFileWS.xq (8-35 ページの「問合せの例」で前述した問合せ) の定義を示します。顧客名を入力として使用するための変数も含まれます。このビューでは、データソースに対して 2 つの XQS 関数が使用されます。このデータソースの構成は、この例の後に示しています。

```
declare namespace xqs = "http://xmlns.oracle.com/ias/xqs" ;
(: returns payment info for all customers:)
declare function xqs:paymentStatusInfo () external;

(: returns customer info given customer name :)
```

```
declare function xqs:customerInfo ($name as xs:string) external;

(: customer name passed in to query :)
declare variable $custName external;

let $custInfo := xqs:customerInfo($custName)
for $custOrderInfo in xqs:paymentStatusInfo()/excel/Row[CustomerKey eq $custInfo/key]
return
   <result>
      <MYEIS_RESULT>
         <Row>
            <Name> { $custInfo/name } </Name>
            <Company> { $custInfo/company} </Company>
            <Address> { $custInfo/address} </Address>
            <City> { $custInfo/city} </City>
            <State> { $custInfo/state} </State>
            <Zip> { $custInfo/zip} </Zip>
         </Row>
      </MYEIS_RESULT>
      <EXCEL_RESULT>
         <Row>
            <OrderID>{$custOrderInfo/OrderId}</OrderID>
            <Amount>{$custOrderInfo/Amount}</Amount>
            <PaymentStatus>{$custOrderInfo/PaymentStatus}</PaymentStatus>
         </Row>
      </EXCEL_RESULT>
   </result>
```

次に、関数 paymentStatusInfo に関連付けられているドキュメント・ソースの構成を示します。このドキュメントは XML ではないため、変換に D3L が使用されます。(8-15 ページの「XML 以外のドキュメント・ソースを使用するための準備」を参照してください。)

```
<document-source isCached="false">
   <function-name prefix="xqs">paymentStatusInfo</function-name>
   <documentURL>http://host:port/xqsdemos/paymentInfo.csv</documentURL>
   <XMLTranslate method="D3L">
      <schema-file>
         http://host:port/xqsdemos/paymentInfoD3L.xml
      </schema-file>
   </XMLTranslate>
</document-source>
```

次に、関数 customerInfo に関連付けられている WSDL ソースの構成を示します。これには、Java と XML の型マッピングに関する情報も含まれます。

```
<wsdl-source isCached="false">
   <function-name  namespace="http://xmlns.oracle.com/ias/xqs">
      customerInfo
   </function-name>
   <wsdlURL>http://host:port/xqsdemos/CustomerInfo.wsdl</wsdlURL>
   <operation>getCustomer</operation>
   <service prefix="myeis">CustomerInfoMYEISService</service>
   <port prefix="myeis">CustomerInfo</port>
   <input-parameters>
      <part position="1" name="name">
         <schema-type prefix="xs">string</schema-type>
      </part>
   </input-parameters>
   <typeMap>
      <mapping typeClass="org.w3c.dom.Node">
         <xmlType prefix="myeis">
            Customer
         </xmlType>
      </mapping>
```

```
   </typeMap>
</wsdl-source>
```

Java コード　この項では、ビュー BasicFileWS を実行するための Java コードを使用し、次のことを行う完全なクラスを示します。

- 静的メソッド getXQSFacade() を使用して、XQSFactory ファクトリから XQSFacadeI インタフェースのインスタンスを取得します。
- 顧客名を入力するための QueryParameterI 配列を設定します。静的メソッド getQueryParameter() を使用して、XQSFactory ファクトリから QueryParameterI インタフェースのインスタンスを取得します。顧客名を設定するには、QueryParameterI インタフェースの setString() メソッドを使用します。
- 問合せを実行します。executeView() メソッドに渡されるビュー名は、構成内でそのビューに関連付けられている関数名と一致する必要があります。このメソッドでは、関数の名前空間（同じく構成内で指定済）も使用されます。この例では、ビューは XQS 名前空間で宣言されています。
- エラーを処理します。XQSErrorI インタフェースのメソッドは、問合せ時に発生したエラーの情報を印刷するために使用します。その前提として、適切な XQS のエラー構成で emptySequence モードまたは errorMessage モードが使用されている必要があります。そうでない場合は、エラーが発生すると問合せが停止し、エラーのイテレータには有用な情報が含まれません。BasicFileWS の構成で示したように、この例では、errorMessage モードが使用されています。（追加情報は、8-68 ページの「XQS エラー処理モードおよび API の使用方法」および 8-92 ページの「XQSErrorI リファレンス」を参照してください。）
- 結果を処理します。問合せ結果は、（oracle.xml.xqxp.datamodel パッケージ内の）クラス XMLItem のインスタンスとして取得されます。XMLItem クラスの型定数および型固有のアクセッサを使用して、値が取得されます。
- カーソルをクローズして、問合せに関連付けられているリソースを解放します。

注意 : XQS 関数 BasicFileWS を executeView() コールで直接実行した場合は、この関数を宣言する必要はありません。

```
import oracle.xml.xqxp.datamodel.XMLItem;
import oracle.xml.parser.v2.XMLDocument;
import oracle.xml.parser.v2.XMLElement;
import oracle.xml.parser.v2.XMLNode;
import oracle.xqs.client.XQSFacadeI;
import oracle.xqs.client.XQSErrorI;
import oracle.xqs.client.QueryParameterI;

public class XQSFacadeIViewTest {

    public static void main(String[] args) throws Exception{
        XQSFacadeI facade = XQSFactory.getXQSFacade();
        String xqueryView = "BasicFileWS";
        QueryParameterI param = XQSFactory.getQueryParameter("custName");
        param.setString("John Ford");
        QueryParameterI[] params = new QueryParameterI[1];
        params[0] = param;

        facade.executeView(xqueryView, "http://xmlns.oracle.com/ias/xqs", params);

        java.util.ListIterator errors = facade.getErrors();
        while(errors.hasNext()) {
            XQSErrorI error = (XQSErrorI)errors.next();
            System.out.println
               ("The function which gives error is: " + error.getFunctionQName());
```

```
    System.out.println("The error type is: "+error.getErrorType()+ " which
       is - "+XQSErrorI.typeNames(error.getErrorType()));
    System.out.println("The error code is:" + error.getErrorCode());
    System.out.println("The error message is:" + error.getErrorMessage());
}

XMLItem item = facade.getNextItem();
StringBuffer resultBuf = new StringBuffer();

XMLDocument doc = new XMLDocument();
XMLElement rootElem = (XMLElement)doc.createElement("QueryResult");
doc.appendChild(rootElem);

while(item != null) {
if(item.instanceOfType(XMLItem.XMLITEM_STRING)) {
   resultBuf.append(item.getString());
}
else if(item.instanceOfType(XMLItem.XMLITEM_INT)) {
   resultBuf.append(item.getInt());
}
else if(item.instanceOfType(XMLItem.XMLITEM_LONG)) {
   resultBuf.append(item.getInt());
}
else if(item.instanceOfType(XMLItem.XMLITEM INTEGER)) {
   if(item.intFormat())
       resultBuf.append(item.getInt());
   else
       resultBuf.append(item.getDecimal().intValue());
}
else if(item.instanceOfType(XMLItem.XMLITEM_BOOLEAN)) {
   resultBuf.append(item.getBoolean());
}
else if(item.instanceOfType(XMLItem.XMLITEM_FLOAT)) {
   resultBuf.append((float)item.getDouble());
}
else if(item.instanceOfType(XMLItem.XMLITEM_DOUBLE)) {
   resultBuf.append(item.getDouble());
}
else if(item.instanceOfType(XMLItem.XMLITEM_XDT_DAYTIMEDURATION)) {
   resultBuf.append(item.getDayTimeDuration());
}
else if(item.instanceOfType(XMLItem.XMLITEM_YEARMONTHDURATION)) {
   resultBuf.append(item.getYearMonthDuration());
}
else if(item.instanceOfType(XMLItem.XMLITEM_DURATION)) {
   resultBuf.append(item.getLexicalValue());
}
else if(item.instanceOfType(XMLItem.XMLITEM_DATETIME)) {
   resultBuf.append(item.getCalendar().getTime());
}
else if(item.instanceOfType(XMLItem.XMLITEM_ANYURI)) {
   resultBuf.append(item.getString());
}
else if(item.instanceOfType(XMLItem.XMLITEM_DECIMAL)) {
   resultBuf.append(item.getDecimal().intValue());
}
else if(item.instanceOfType(XMLItem.XMLITEM_NODE)) {
   resultBuf.append(item.getNode().getNodeName());
}
else if(item.instanceOfType(XMLItem.XMLITEM_HEXBINARY)) {
   resultBuf.append(item.getString());
}
```

```
            else if(item.instanceOfType(XMLItem.XMLITEM_BASE64BINARY)) {
               resultBuf.append(item.getString());
            }

            item = facade.getNextItem();
            }
            facade.close();
        }
}
```

EJB クライアント API の使用方法

8-39 ページの「XQS クライアント API を使用するための一般的なコーディング手順」を参照してください。XQS EJB クライアント API の場合は、（問合せの作成後に）ステートフル・セッションとステートレス・セッションのどちらを選択するかによって、次に示すように手順の詳細が異なります。

1. 検索の実行、EJB ローカル・ホーム・インタフェースの取得、ローカル・インタフェースの作成などを行って、Bean インスタンスの適切な作成手順を完了します。次の各手順のメソッド・コールは、このインスタンスからコールされます。ステートフル Bean とステートレス Bean の作成方法は、次項「ステートフル・セッション用およびステートレス・セッション用の EJB クライアント」を参照してください。
2. 入力パラメータ用の `QueryParameterI` 配列を作成し、静的メソッド `getQueryParameter()` を使用して `XQSFactory` ファクトリから `QueryParameterI` インタフェースのインスタンスを取得して、その配列を移入します。
3. `execute()` メソッドを使用して非定型問合せを実行するか、`executeView()` メソッドを使用して XQS ビューを実行し、問合せと `QueryParameterI` 配列（入力パラメータがない場合は `null`）を渡します。`executeView()` を使用する場合は、そのビューの XQS 関数の名前空間も渡します。
4. ステートフル Bean の場合は、`getNextItem()` メソッドを繰返し使用して、項目ごとに結果を取得します。各項目は、`XMLItem` インスタンス内に戻されます。必要に応じてこれらの項目を処理します。

   ステートレス Bean の場合は、問合せ結果がベクター `java.util.Vector` として完全にマテリアライズされ、`XMLItem` インスタンスが含まれます。必要に応じてこれらの項目のベクターを処理します。
5. 問合せ内で使用したデータソースの構成に `emptySequence` エラー・モードまたは `errorMessage` エラー・モードを使用した場合は、必要に応じて `getErrors()` メソッドを使用し、問合せの実行時にこれらのデータソース内で発生したエラーを取得できます。これにより、`XQSErrorI` オブジェクトのコレクションに対するイテレータが戻されます。エラーの構成と処理の詳細は、8-68 ページの「XQS エラー処理モードおよび API の使用方法」を参照してください。
6. ステートフル Bean の場合は、`close()` メソッドを使用して、問合せに関連付けられているすべてのリソースを解放します。この手順は、ステートレス Bean には適用されません。

これらの手順は、次の例の中で示されています。8-85 ページの「XQS EJB クライアント API リファレンス」も参照してください。

ステートフル・セッション用およびステートレス・セッション用の EJB クライアント

ステートフル・セッションとステートレス・セッションの使い分けの詳細は、8-40 ページの「ステートフル・クライアントとステートレス・クライアント」を参照してください。

ステートフル EJB とステートレス EJB のどちらを取得するかは、JNDI 検索および使用する XQS パッケージで決まります。次の例では、ステートフル Bean が取得されます。

```
//Look up and create the EJB to execute the query.
InitialContext ic = new InitialContext();
//Use Local client.
oracle.xqs.client.ejb.stateful.XQSClientLocalHome home =
   (oracle.xqs.client.ejb.stateful.XQSClientLocalHome)ic.lookup
                                        ("java:comp/env/XQSClientStatefulLocal");
oracle.xqs.client.ejb.stateful.XQSClientLocal bean = home.create();
```

次の例では、ステートレス Bean が取得されます。

```
//Look up and create the EJB to execute the query.
InitialContext ic = new InitialContext();
//Use Local client.
oracle.xqs.client.ejb.stateless.XQSClientLocalHome home =
   (oracle.xqs.client.ejb.stateless.XQSClientLocalHome)ic.lookup
                                        ("java:comp/env/XQSClientStatelessLocal");
oracle.xqs.client.ejb.stateless.XQSClientLocal bean = home.create();
```

前述のように、ステートレス EJB セッションで問合せ（非定型またはビュー）を実行すると、`XMLItem` インスタンスを含むベクターとして、結果が完全にマテリアライズされます。また、このベクターは、必要に応じて後で処理します。ステートフル・セッションの場合は、問合せの実行後に、`getNextItem()` メソッドを使用して項目を 1 つずつ取得します。これにより、タイプ `XMLItem` の項目が戻されます。

ステートフル・セッションでの EJB クライアント API の使用

XQS EJB ステートフル・クライアント API は、`XQSFacadeI` API と同じです。前の項で示した Bean 作成用のコード以外にも、ステートフル EJB クライアントのコード例は、8-44 ページの「例 2: XQSFacadeI API を使用した非定型問合せ」および 8-46 ページの「例 3: XQSFacadeI API を使用した XQS ビュー」で示した `XQSFacadeI` の例と類似します。`getNextItem()` メソッドを繰返し使用して、項目を 1 つずつ、`XMLItem` インスタンスとして取得し、処理します。

制限事項

ステートフル EJB と関連付けられている XQS の問合せ状態は、非アクティブ化および後続の再アクティブ化では維持されません。問合せ状態は、`execute()` または `executeView()` メソッドの起動後に確立されます。問合せ状態には問合せの実行計画を管理する内部構造が含まれ、項目の取得後には、結果順序の現在の項目の位置も含まれます。ステートフル Bean インタフェースを使用すると、毎回サーバー・セッションに接続することで、結果項目を一度に 1 つずつフェッチできます。

このオプションは、各項目のサイズまたは結果順序の合計サイズが大きすぎて、一度のラウンドトリップでは送信できない場合に重要です。また、最初の項目をすぐに戻すことによる迅速なレスポンスが必要な場合にも便利です。このオプションのデメリットは、サーバーによって Bean が非アクティブ化され、処理中の問合せの状態が失われる可能性があることです。

問合せ状態が失われ、クライアントにより後続の `getNextItem()` のコールが試行されると、XQS によりエラー・コードが `XQS 0088` の XQSException がスローされます。この制限を緩和する方法の 1 つは、キャッシュ済の XQS ビューとしてステートフル Bean に起動される問合せを構成することです。キャッシュ済の XQS ビューを使用すると、`getNextItem()` のコールを複数回実行して結果順序の目的の位置まで移動することにより、問合せを即座に再実行し、中断する前に現在の位置であった場所からフェッチを再開できます。

例 : ステートレス・セッションで EJB クライアント API を使用した XQS ビュー

このステートレス EJB の例では、8-46 ページの「例 3: XQSFacadeI API を使用した XQS ビュー」で使用した BasicFileWS XQS ビューを再使用しており、そのコードの一部が繰り返されています。ビューとその構成は、この項を参照してください。

JNDI 参照およびステートレス Bean の作成に加えて、このステートレス EJB の例と XQSFacadeI の例の重要な違いは、結果の処理にあります。ステートレス・セッションの結果は、XMLItem インスタンスを含むベクターとして完全にマテリアライズされます。XMLItem インスタンスを取得するための getNextItem() メソッドが Bean 内にあるのとは対照的に、その後、そこから項目を 1 つずつ取得し、XMLItem としてキャストします。ただし、処理の基本的な性質は変わりません。

```
...
//lookup and create the EJB to execute the xquery
InitialContext ic = new InitialContext();

oracle.xqs.client.ejb.stateless.XQSClientLocalHome home =
                (oracle.xqs.client.ejb.stateless.XQSClientLocalHome)
                ic.lookup("java:comp/env/XQSClientStatelessLocal");
oracle.xqs.client.ejb.stateless.XQSClientLocal bean = home.create();

String xqueryView = "BasicFileWS";
QueryParameterI param = XQSFactory.getQueryParameter("custName");
param.setString("John Ford");
QueryParameterI[] params = new QueryParameterI[1];
params[0] = param;

java.util.Vector  result =
     bean.executeView(xqueryView,"http://xmlns.oracle.com/ias/xqs",params);

java.util.Enumeration resultEnum = result.elements();
while(resultEnum!=null && resultEnum.hasMoreElements()) {
    XMLItem item = (XMLItem)resultEnum.nextElement();
    if(item.instanceOfType(XMLItem.XMLITEM_STRING)) {
       resultBuf.append(item.getString());
    }
    else if(item.instanceOfType(XMLItem.XMLITEM_INT)) {
       resultBuf.append(item.getInt());
    }
    else if(item.instanceOfType(XMLItem.XMLITEM_LONG)) {
       resultBuf.append(item.getInt());
    }
    else if(item.instanceOfType(XMLItem.XMLITEM_INTEGER)) {
       if(item.intFormat())
           resultBuf.append(item.getInt());
       else
           resultBuf.append(item.getDecimal().intValue());
    }
    else if(item.instanceOfType(XMLItem.XMLITEM_BOOLEAN)) {
       resultBuf.append(item.getBoolean());
    }
    else if(item.instanceOfType(XMLItem.XMLITEM_FLOAT)) {
       resultBuf.append((float)item.getDouble());
    }
    else if(item.instanceOfType(XMLItem.XMLITEM_DOUBLE)) {
       resultBuf.append(item.getDouble());
    }
    else if(item.instanceOfType(XMLItem.XMLITEM_XDT_DAYTIMEDURATION)) {
       resultBuf.append(item.getDayTimeDuration());
    }
    else if(item.instanceOfType(XMLItem.XMLITEM_YEARMONTHDURATION)) {
       resultBuf.append(item.getYearMonthDuration());
```

```
    }
    else if(item.instanceOfType(XMLItem.XMLITEM_DURATION)) {
       resultBuf.append(item.getLexicalValue());
    }
    else if(item.instanceOfType(XMLItem.XMLITEM_DATETIME)) {
       resultBuf.append(item.getCalendar().getTime());
    }
    else if(item.instanceOfType(XMLItem.XMLITEM_ANYURI)) {
       resultBuf.append(item.getString());
    }
    else if(item.instanceOfType(XMLItem.XMLITEM_DECIMAL)) {
       resultBuf.append(item.getDecimal().intValue());
    }
    else if(item.instanceOfType(XMLItem.XMLITEM_NODE)) {
       resultBuf.append(item.getNode().getNodeName());
    }
    else if(item.instanceOfType(XMLItem.XMLITEM_HEXBINARY)) {
       resultBuf.append(item.getString());
    }
    else if(item.instanceOfType(XMLItem.XMLITEM_BASE64BINARY)) {
       resultBuf.append(item.getString());
    }
// Here is where you might use a bean.getErrors() call and process errors.
// (Assuming appropriate configuration.)
}
...
```

JSP タグ・ライブラリの使用方法

8-39 ページの「XQS クライアント API を使用するための一般的なコーディング手順」を参照してください。XQS JSP タグ・ライブラリの場合は、（問合せの作成後に）ステートフル・アクセス・モードとステートレス・アクセス・モードのどちらを選択するかによって、次に示すように手順の詳細が異なります。

1. `taglib` を介して、TLD URI および目的の接頭辞を指定して XQS JSP タグ・ライブラリをインポートします。
2. `executeCursor` タグを使用してステートフル・モードで問合せを実行するか、`execute` タグを使用してステートレス・モードで問合せを実行します。`executeCursor` タグまたは `execute` タグで次の処理を行います。
   - `xqueryString` 属性を使用して非定型問合せを指定するか、`xqsViewName` 属性および `namespace` 属性を使用して XQS ビューを指定します。
   - 問合せ結果の出力変数を定義します。出力先は、XML 文書（`toXMLDoc` 属性を使用）、および結果の `java.util.ArrayList` インスタンス（`resultItems` 属性を使用）です。必要に応じて、JSP 出力ストリーム（`toWriter` 属性を使用）に出力することもできます。結果は、8-38 ページの表 8-2「バインド・パラメータ用に XQS でサポートされている型」に基づいて、Java オブジェクト型で表されます。
   - 問合せ内で使用したデータソースの構成に `emptySequence` エラー・モードまたは `errorMessage` エラー・モードを使用した場合は、必要に応じて `errors` 属性を使用し、問合せの実行時にこれらのデータソース内で発生したエラーを処理できます。これにより、`XQSErrorI` オブジェクトのコレクションに対するイテレータが戻されます。エラーの構成と処理の詳細は、8-68 ページの「XQS エラー処理モードおよび API の使用方法」を参照してください。
3. `executeCursor` タグまたは `execute` タグでは、各入力パラメータに対して `param` サブタグを使用します。必要に応じて、`param` タグの `localName`、`namespace`、`value` および `type` の各属性を使用します。
4. `executeCursor` タグの場合は、関連する `next` タグを使用して結果を項目ごとに取得するか、結果をバッチで取得して出力媒体に送ります。

   `execute` タグの場合は、結果が完全にマテリアライズされて、出力媒体に送られます。

5. 1 つ以上の出力媒体で繰返し処理を行い、結果を処理します。使用できる出力媒体（JSP 出力ストリーム、XML 文書、Java のリスト）は標準であるため、これは XQS 固有の手順ではありません。next を 1 回実行するたびに、その結果を処理する必要があることに注意してください。処理せずに再び next を実行すると、結果が上書きされます。
6. executeCursor タグの場合は、close タグを使用して、問合せに関連付けられているすべてのリソースを解放します。この手順は、execute タグには適用されません。

これらの手順は、次の例の中で示されています。8-86 ページの「XQS JSP タグ・ライブラリのリファレンス」も参照してください。

注意 : 必要に応じて、JSP ページで JSTL タグを使用して結果を変換することもできます。

ステートフル・アクセス用およびステートレス・アクセス用の JSP タグ

ステートフル・アクセスとステートレス・アクセスの使い分けの詳細は、8-40 ページの「ステートフル・クライアントとステートレス・クライアント」を参照してください。

XQS を使用する JSP ページでは、ステートフル・アクセスとステートレス・アクセスのどちらを使用するかは、使用するタグによって決まります。executeCursor タグの場合はステートフル・アクセスを使用しますが、execute タグの場合はステートレス・アクセスを使用します。

前述のように、問合せ（非定型またはビュー）をステートレス JSP アクセス・パターンで実行すると、結果が完全にマテリアライズされて、属性の設定（XML DOM ドキュメント、Object[] 配列、およびオプションの JSP 出力ストリーム）を介して選択した出力媒体に送られます。その後、これらの結果を必要に応じて処理します。ステートフル・アクセス・パターンの場合は、問合せの実行後に、next タグを使用して、1 つずつ項目を取得して処理します。

例 : ステートフル・アクセス・パターンで JSP タグを使用した XQS ビュー

このステートフル JSP の例では、8-46 ページの「例 3: XQSFacadeI API を使用した XQS ビュー」の BasicFileWS XQS ビューを再使用しています。ビューとその構成は、この項を参照してください。

この項のコード例では、次のことが実行されます。

- page ディレクティブによって、要求されたクラスがインポートされます。
- taglib ディレクティブによって、タグ・ライブラリの TLD URI および使用するタグ接頭辞が指定されます。
- executeCursor タグによってビューが実行され、オープン・カーソルがフェッチ可能な状態になります。namespace 属性によって、基礎となる XQSView 関数が定義される名前空間が指定されます。
- executeCursor の param サブタグによって、入力パラメータ（顧客名）が指定されます。
- cursorId の設定を介して executeCursor タグに関連付けられている next タグにより、結果ごとに処理が繰り返され、出力媒体（XML 文書および結果項目の ArrayList インスタンス）が移入されます。その際には、処理対象のデータが残っているかどうかを調べるために、繰返し処理のたびに itemsFetched 属性が使用されます。next タグは、executeCursor タグで指定されたカーソル ID（mycursor）を参照する必要があります。
- close タグにより、カーソルがクローズされ、関連付けられているリソースが解放されます。このタグも、executeCursor タグで指定されたカーソル ID（mycursor）を参照する必要があります。

注意 :

- ここでは、エラー処理について言及しません。適切なエラー構成（データソース構成の emptySequence モードまたは errorMessage モード）を使用していれば、myerrors イテレータを処理できます。8-68 ページの「XQS エラー処理モードおよび API の使用方法」を参照してください。
- ここでは、出力媒体の処理について言及しません。XML 文書または結果オブジェクトの配列の処理に関しては、XQS 固有の処理はありません。

```
<%@ page import="oracle.xml.parser.v2.XMLElement" %>
<%@ page import="oracle.xml.parser.v2.XMLDocument" %>

<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/xquerytag.tld"
           prefix="xq"%>
<%
String xqueryView = "BasicFileWS";
int items;
%>
<xq:executeCursor
                xqsViewName="<%=xqueryView%>"
                namespace="http://xmlns.oracle.com/ias/xqs"
                resultItems="myresults"
                toXMLDoc="mydoc"
                cursorId="mycursor"
                errors="myerrors" >
   <xq:param
              localName="custName"
              value="John Ford"
              type="String"/>
</xq:executeCursor>
<%
do{
// Populate the XML document and result items array.
%>
   <xq:next
                cursorId="mycursor"
                itemsFetched="fetched" />
<%
  items=fetched.intValue();
 }while(items>0)
// Here is where you might process the "myerrors" error iterator.
// (Assuming appropriate configuration.)

// Retrieve results from the XML document mydoc and the result items array myresults.
// (This processing is not XQS-specific.)
%>
<xq:close cursorId="mycursor" />
```

例 : ステートレス・アクセス・パターンで JSP タグを使用した非定型問合せ

この例では、XQS JSP execute タグを使用して非定型問合せが実行され、ステートレス・アクセス・パターンが使用されます。この例の主な目的は、前述のステートフル・アクセスの使用例と比較することで、次の事項を比較および対比させることにあります。

- ビューではなく非定型問合せを使用します。関連する構成で同じ問合せを使用している、8-44 ページの「例 2: XQSFacadeI API を使用した非定型問合せ」を参照してください。問合せは 1 つの文字列バッファに統合され、その後、バッファから文字列に書き込まれます。
- ステートフル・アクセスでは executeCursor タグを使用しますが、ステートレス・アクセスでは、かわりに execute タグを使用します。execute タグに next サブタグが使用されていないため、close タグは不要です。

```
<%@ page import="oracle.xml.xqxp.datamodel.XMLItem" %>
<%@ page import="oracle.xml.parser.v2.XMLElement" %>
<%@ page import="oracle.xml.parser.v2.XMLDocument" %>

<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/xquerytag.tld"
           prefix="xq"%>
<%
StringBuffer xqueryBuf = new StringBuffer();
xqueryBuf.append("declare namespace xqs=\"http://xmlns.oracle.com/ias/xqs\";\n");
xqueryBuf.append("declare function xqs:get_poSQ($bind as xs:string) external;\n");
xqueryBuf.append("for $po in
xqs:get_poSQ(\"http://localhost:8888/myrepository/pos-2KB.xml\") \n");
xqueryBuf.append("return $po\n");

String xqueryStr = xqueryBuf.toString();
%>
<xq:execute
                xqueryString="<%=xqueryStr%>"
                toXMLDoc="mydoc"
                resultItems="myresults"
                errors="myerrors" />
<%
// Here is where you might process the "myerrors" error iterator.
// (Assuming appropriate configuration.)
%>
<%
// Retrieve the results from either the XML document mydoc or result items array
// myresults. (This processing is not XQS-specific.)
%>
```

Web サービス操作として公開された XQS ビューの使用方法

XQS では、自動生成されたサーブレットを介して実装された Web サービスを使用して、XQS ビューを Web サービス操作として公開できます。詳細および関連する詳細は、8-60 ページの「XQS ビューを Web サービス操作として公開するための OC4JPackager の追加の出力」を参照してください。

Web サービス・クライアントは、独自に提供する必要があります。

OC4JPackager を使用した XQS アプリケーションのパッケージ方法

この項では、OC4JPackager を使用して XQS 関連ファイルをアプリケーションにバンドルする方法について説明します。概要は、8-12 ページの「OC4JPackager の概要」を参照してください。

OC4JPackager では、他の EAR のように OC4J にデプロイできる、EAR ファイルが作成されます。一般情報は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。

この項には、次の項目が含まれます。

OC4JPackager の使用手順

この項では、J2EE アプリケーションで OC4JPackager を実行し、XQS 機能を使用できるようにするための基本的な準備および実行の手順を示します。ここに出てくる OC4JPackager のパラメータおよび Java のプロパティに関する一般情報は、8-115 ページの「OC4JPackager のリファレンス」を参照してください。

注意： OC4JPackager では、ディレクトリ名またはファイル名でのスペースの使用はサポートされていません。

OC4JPackager を実行するための準備

OC4JPackager を実行するには、次の準備手順を完了してください。

1. 通常どおりに、アプリケーション・コンポーネントの JAR ファイル（Web モジュール用 WAR ファイル、EJB モジュール用 EJB JAR ファイルなど）を作成します。その際、すべてのファイルに、適切な標準 J2EE 構成ファイル（`web.xml` および `ejb-jar.xml` など）を使用します。その後、必要に応じてこれらのファイルを OC4JPackager の EAR ファイルにバンドルするか、他の関連ファイル（`application.xml` など）とともに、EAR ファイルの構造と内容を反映するディレクトリ構造に置いておきます。

2. アプリケーションの目的のディレクトリを選択し、（必要に応じて）EAR ファイルまたは構造化された EAR の内容をそのディレクトリに配置します。

   `-appArchives` パラメータ（必須）を介して、OC4JPackager にそのディレクトリを指定します。

3. すべての XQS ビュー（`.xq` または `.sql`）ファイルが配置される XQS リポジトリとして目的のディレクトリを選択し、XQS ビュー（`.xq` または `.sql`）ファイルをそこに配置します。

   `-repository` パラメータを介して、OC4JPackager にそのディレクトリを指定します。

4. `xqs-config.xml` ファイルの目的のディレクトリを選択し、ファイルをそのディレクトリに配置します。ただし、XQS 構成が完了していることが前提条件になります（8-21 ページの「XQS 関数の構成方法」を参照）。

   `-xqsConfig` パラメータ（必須）を介して、OC4JPackager にそのディレクトリを指定します。

OC4JPackager の実行 : 必須およびオプションのパラメータとプロパティ

OC4JPackager の実行時には、必要に応じて次を指定します。例は、次項「コマンドラインでの OC4JPackager の実行」を参照してください。

プログラム・パラメータ

1. 必須の `-appArchives` パラメータを使用して、アプリケーションが配置されている場所（EAR ファイルの場所、つまり EAR 構造のトップレベル・ディレクトリ）を指定します。
2. オプションの `-repository` パラメータを使用して、XQS リポジトリのディレクトリを指定します。
3. 必須の `-xqsConfig` パラメータを使用して、アプリケーションの `xqs-config.xml` ファイルへのフルパス（ファイル名を含む）を指定します。
4. 必須の `-name` パラメータおよび `-output` パラメータを使用して、OC4JPackager によって作成される EAR ファイルの名前と出力ディレクトリを指定します。
5. アプリケーションによって使用される XQS 機能用ファイルが OC4JPackager に含まれるように、またはファイルが除外されるように、該当するフラグを使用して指定します。
   - XQS JSP タグ・ライブラリを使用する場合は、`-jsp` フラグを使用します。
   - XQS ステートフル EJB クライアント API を使用する場合は、`-sf` フラグを使用します。
   - XQS ステートレス EJB クライアント API を使用する場合は、`-sl` フラグを使用します。
   - OC4JPackager が Web サービスを生成しないようにする場合は、`-no_ws` フラグを使用します。

   （`XQSFacadeI` インタフェースは常に含まれます。）

Java VM プロパティ

1. 必須の Java プロパティ `java.home` を使用して、Java のホーム・ディレクトリを指定します。OC4JPackager によって、このディレクトリから、タスクに必要な `java` コマンドが実行されます。
2. OC4JPackager に（必要に応じて EAR ファイルのバンドル解除およびバンドルを行う）特定の作業ディレクトリを使用させる場合は、オプションの Java プロパティ `xds.packager.work.dir` を使用します。デフォルトは、`user.home` Java プロパティで指定されたディレクトリです。
3. オプションの Java プロパティ `java.util.logging.properties.file` を使用して、Java のロギング設定（OC4JPackager のロギング・レベルを含む）を指定するロギング・プロパティ・ファイルを指定します。ロギング・プロパティは、次の場所で説明されているように、標準の J2SE ロギングによって定義されます。

   `http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/`
4. Oracle ホーム・ディレクトリを示す場合は、オプションの Java プロパティ `oracle.home` を使用します。この処理は、OC4JPackager フラグ `-jsp`、`-sf`、`-sl`（8-116 ページの「OC4JPackager のパラメータ」を参照）、および関連する JAR ファイルを使用する場合には必須です。

コマンドラインでの OC4JPackager の実行

OC4JPackager は、OC4JPackager.jar を起動する java コマンド（コマンド・プロンプトは %）を使用して、コマンドラインから実行できます。

```
% java -jar OC4JPackager.jar ...
```

OC4JPackager.jar は、XQS 配布の xds/tools ディレクトリにあり、OC4J コンテナ内のインストール場所から使用する必要があります。

次に例を示します（$ORACLE_HOME は Oracle インストールのホーム・ディレクトリとして設定されており、\ は行折返しを示します）。この例は、XQS JSP タグ・ライブラリおよびステートフル EJB クライアント API を使用するアプリケーション用です。現在のディレクトリからの相対パス、または絶対パスを指定できます。

```
% java -jar OC4JPackager.jar \
       -Djava.home=/home/myjavainstall \
       -Dxds.packager.work.dir=$ORACLE_HOME/temp \
       -Doracle.home=$ORACLE_HOME \
       -Djava.util.logging.properties.file=myfile.properties \
       -appArchives $ORACLE_HOME/myapp \
       -xqsConfig $ORACLE_HOME/xds/myconfig/xqs-config.xml \
       -repository $ORACLE_HOME/xds/repository \
       -output $ORACLE_HOME/xds/mystaging -name myxqsapp.ear \
       -jsp -sf
```

もしくは、次の項で説明するように、Ant を介して OC4JPackager を実行できます。

8-60 ページの「OC4JPackager の基本的な出力」も参照してください。

Ant を介した OC4JPackager の実行

前の項で説明したように、コマンドラインから OC4JPackager を実行するかわりに、標準の Ant java タスクを介して、Ant を使用したビルド処理の一部として OC4JPackager を実行できます。Ant の一般情報は、Apache の Web サイト http://ant.apache.org/manual/ を参照してください。（また、OC4J 固有の Ant タスクに関する情報は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。）

例として次に示す Ant ビルド・ファイルの一部では、OC4JPackager が起動されます。この例は、前項で示した例のコマンドラインに対応しています。

```
<java jar="OC4JPackager.jar" fork="true" failonerror="true">
   <jvmarg value="-Doracle.home=${ORACLE_HOME}"/>
   <jvmarg value="-Djava.home=/home/myjavainstall"/>
   <jvmarg value="-Dxds.packager.work.dir=${ORACLE_HOME}/temp"/>
   <jvmarg value="-Djava.util.logging.properties.file=myfile.properties"/>
   <arg line="-appArchives ${ORACLE_HOME}/myapp"/>
   <arg line="-xqsConfig ${ORACLE_HOME}/xds/myconfig/xqs-config.xml"/>
   <arg line="-repository ${ORACLE_HOME}/xds/repository"/>
   <arg line="-output ${ORACLE_HOME}/xds/mystaging" />
   <arg line="-name myxqsapp.ear"/>
   <arg line="-jsp"/>
   <arg line="-sf"/>
</java>
```

重要： 上に示したように、OC4JPackager パラメータを設定する <arg> 要素では、value 属性のかわりに line 属性を使用します。

OC4JPackager の基本的な出力

OC4JPackager で作成された EAR ファイルは標準的な構造を持ち、少なくとも 1 つの追加コンポーネントがあります。このコンポーネントは xqs-resources.jar という名前のファイルで、XQS 構成ファイルおよび XQS リポジトリ・ファイル（XQS ビューの .xq ファイル）のディレクトリが含まれます。

注意： Web サービス操作として公開する XQS ビューが存在する場合は、EAR ファイルに追加のコンポーネントが含まれます。次項「XQS ビューを Web サービス操作として公開するための OC4JPackager の追加の出力」を参照してください。

Web モジュール mywebapp および EJB モジュール myejb で構成されるアプリケーションでは、OC4JPackager によって次のような EAR ファイルの内容が作成されます。

```
META-INF/
   application.xml
   orion-application.xml
   data-sources.xml
mywebapp.war
myejb.jar
xqs-resources.jar
```

xqs-resources.jar 以外にも、アプリケーションには、application.xml ファイル、必要に応じてオプションの orion-application.xml ファイルと data-sources.xml ファイル、Web モジュール用 WAR ファイル、EJB モジュール用 EJB JAR ファイルなどがすでに含まれている必要があります。

xqs-resources.jar の構造は次のとおりです。

```
   APP-REPOSITORY/
      filename.xq
      filename.xq
      filename.xq
      ...
   xqs-config.xml
   xds-application.properties
```

ファイル xds-application.properties は、XQS によって内部的に使用されます。

XQS ビューを Web サービス操作として公開するための OC4JPackager の追加の出力

XQS 構成で WSDLvisibility="true" と設定された XQS ビュー（SQL ベースのビューを除く）では、XQS によって、ビューが Web サービス操作として公開されます。いずれの Web サービスでも、操作を起動する Web サービス・クライアントが必要です。

注意： WSDLvisibility プロパティが true に設定された状態で XQS ビューが Web サービス操作として公開されてしまうため、OC4JPackager によってアプリケーションに Web サービス・モジュールが作成および追加されないようにする場合は、次のようにして、OC4JPackager の実行時に -no_ws フラグを指定します。

```
% java -jar OC4JPackager.jar ... -no_ws
```

このフラグはオプションです。デフォルトでは、Web サービス・モジュールが追加されます。

OC4JPackager によって、Web サービスの実装、および公開される各 XQS ビューの操作を定義する WSDL ドキュメントが作成されます。その後、OC4JPackager によって、次の内容を含む WAR ファイルが作成されます。

- WSDL ドキュメント
- Web サービス・サーブレットを構成するための Web モジュール構成ファイル `web.xml`
- Web サービス構成ファイル `oracle-webservices.xml`

また、OC4JPackager によって、アプリケーションの `application.xml` ファイルが更新され、追加の Web モジュールのエントリが作成されます。

XQS ビュー Web サービスを表す追加の Web モジュールの名前、コンテキスト・ルートおよび WAE ファイルは、アプリケーションの名前に対応します。アプリケーション `myapp` の場合は次のようになります。

- `/myapp-WS`—Web サービス・モジュールのコンテキスト・ルート
- `myapp-WS-web.war`

生成された WSDL ファイルは、常に `xqsview-WS.wsdl` になります。

Web サービスを表し、Oracle Web サービスのテスト・ページにつながるサーブレットの URL パターンは、常に `/xqs-ws` です。

つまり、XQS ビュー Web サービスの WSDL の URL は、`http://`*`host`*`:`*`port`*`/myapp-WS/xqs-ws?WSDL` となります。

注意： 追加の WAR ファイルには、Web サービスのクライアント起動コードは含まれません。このコードは、サーバー上のアプリケーション・デプロイメントの一部になる予定です。

WSDL ドキュメントでは、`wsdl:operation` の名前は、XQS 構成（`<xqsview-source>` の `<function-name>` サブ要素）で定義された対応する XQS 関数の名前と同じになります。各 `wsdl:input` メッセージ・タイプは、XQS 構成（`<xqsview-source>` の `<input-parameters>` サブ要素）で定義された入力パラメータ・タイプに対応する XML 型にマップされます。各 `wsdl:output` メッセージ・タイプは、XQS 構成（`<xqsview-source>` の `<output-element>` サブ要素）で定義された出力要素に対応する XML 要素または型にマップされます。これらは、`string` や `int` などの単純型、またはユーザーが定義した複合型のどちらかになります。バインディングは、SOAP ドキュメント / ラップ・バインディングになります。

次の例は、次のすべての項の内容を合せたものです。

- 例 : Web サービス操作としてビューを公開するための構成
- 例 : Web サービス操作として公開されるビューの EAR ファイル
- 例 : Web サービス操作として公開されるビューの WAR ファイル
- 例 : Web サービス操作として公開されるビューの WSDL ドキュメント

例 : Web サービス操作としてビューを公開するための構成

この章で前述した次の構成では、WSDLvisibility="true" の設定を介して、ビュー BasicFileWS が Web サービス操作として公開されます。

```
<xqsview-source WSDLvisibility="true"  isCached="false">
   <function-name  prefix="xqs">BasicFileWS</function-name>
   <input-parameters type-match="none" >
      <part position="1" name="custName">
         <schema-type prefix="xs">string</schema-type>
      </part>
   </input-parameters>
   <output-element namespace="http://xmlns.oracle.com/ias/xqs/CustomerDemo"
               location="http://host:port/xqsdemos/MyTypes.xsd"
type=CustomerOrdersType>customerOrders </output-element>
</xqsview-source>
```

後述の説明は、この例が myapp というアプリケーションの一部であることを前提としています。

例 : Web サービス操作として公開されるビューの EAR ファイル

アプリケーション myapp が mywebapp.war および myejb.jar で構成されており、1 つ以上の XQS ビューが Web サービス操作として公開されると仮定します。OC4JPackager 実行後のアプリケーション EAR ファイルの関連する内容は、次のようになります。

```
META-INF/
   application.xml
mywebapp.war
myejb.jar
myapp-WS-web.war
xqs-resources.jar
```

WAR ファイル myapp-WS-web.war は、公開されるビューの Web サービス操作の自動作成済サーブレット実装用です。この内容は、次の項を参照してください。

例 : Web サービス操作として公開されるビューの WAR ファイル

公開されるビューの Web サービス操作のサーブレット実装用に作成された myapp-WS-web.war の内容は、次のとおりです。

```
WEB-INF/
   web.xml
   oracle-webservices.xml
   wsdl/
      xqsview-WS.wsdl
```

特に、WSDL ドキュメントに注意してください。WSDL の部分的な例は、次項「例 : Web サービス操作として公開されるビューの WSDL ドキュメント」を参照してください。

例 : Web サービス操作として公開されるビューの WSDL ドキュメント

この項では、Web サービスとして公開される XQS ビュー用の OC4JPackager によって生成される WSDL ドキュメントの一部を示します。この例では、BasicFileWS という名前のビューに関連する部分を示します。

```
...
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xqscwsNsPref="xqs-client-ws"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xqs0="http://xmlns.oracle.com/ias/xqs" xmlns:tns="xqs-
    client-ws_myapp-WS" name="myapp-WS" targetNamespace="xqs-
    client-ws_myapp-WS">
<types>
...
<schema targetNamespace="http://xmlns.oracle.com/ias/xqs"
xmlns:xqs1="http://xmlns.oracle.com/ias/xqs/CustomerDemo"
xmlns:tns="http://xmlns.oracle.com/ias/xqs"
...
>
<import namespace="xqs-client-ws"/>
<import namespace="http://xmlns.oracle.com/ias/xqs/CustomerDemo"
schemaLocation="http://host:port/xqsdemos/MyTypes.xsd"/>

<complexType name="BasicFileWSType">
      <sequence>
          <element name="custName" type="string"
              nillable="true" />
      </sequence>
   </complexType>
   <complexType name="BasicFileWSResultType">
       <sequence>
         <element name="return" nillable="true">
           <complexType>
             <sequence>
               <element name="result" nillable="true">
                 <complexType>
                   <sequence>
                     <element name="customerOrders"
                        type="xqs1:CustomerOrdersType"
                        minOccurs="0" maxOccurs="unbounded" />
                   </sequence>
                 </complexType>
               </element>
              <element name="errors" nillable="true" minOccurs="0">
                <complexType>
                  <sequence>
                     <element name="xqserror"
                       type="xqscwsNsPref:XQSErrorType"
                       maxOccurs="unbounded" />
                   </sequence>
                 </complexType>
               </element>
             </sequence>
           </complexType>
         </element>
      </sequence>
    </complexType>
<element name="BasicFileWS" type="tns:BasicFileWSType" />
<element name="BasicFileWSResult" type="tns:BasicFileWSResultType" />

…
</schema>
…
```

```
</types>
…
<message name="BasicFileWSRequest">
 <part name="parameters" element="xqs0:BasicFileWS" />
</message>
<message name="BasicFileWSResponse">
 <part name="return" element="xqs0:BasicFileWSResult" />
</message>
…
<portType name="XQSViewWebServices">
   <operation name="BasicFileWS">
      <input message="tns:BasicFileWSRequest" />
      <output message="tns:BasicFileWSResponse" />
    </operation>
    …
    </portType>
      <binding name="HttpSoap11Binding"  type="tns:XQSViewWebServices">
          <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="BasicFileWS">
           <soap:operation />
          <input>
              <soap:body use="literal" parts="parameters" />
           </input>
          <output>
              <soap:body use="literal" parts="return" />
           </output>
         </operation>
          …
    </binding>
   <service name="XQSView-WS">
       <port name="HttpSoap11" binding="tns:HttpSoap11Binding">
           <soap:address xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
             location="http://host:port/myapp-WS/xqs-ws" />
       </port>
   </service>
</definitions>
```

XQS パフォーマンス機能の使用方法

この項では、XQS で提供されるパフォーマンス機能について説明します。この項には、次の項目が含まれます。

- ステートレスまたはステートフルの XQS クライアント API を使用する際のパフォーマンスの考慮事項
- XQS キャッシングの構成
- ラージ・データ用の XQS ドキュメント・ソースの構成

ステートレスまたはステートフルの XQS クライアント API を使用する際のパフォーマンスの考慮事項

前述のように、XQS クライアント API では、結果が完全にマテリアライズされてプロセス・メモリーに格納されるステートレス問合せの実行、またはカーソルを介して結果が段階的にアクセスされるステートフル問合せの実行を選択できます。どちらの方法にも、長所と短所があります。

ほとんどの状況では、ステートレス実行が適しています。これは、評価が完了した直後に、関連付けられているリソースが解放されるためです。ただし、結果順序に含まれる項目の全体のサイズが大きすぎて、順序全体を一度にマテリアライズするとプロセス・メモリーが不足する場合には、ステートレス実行を使用できません。このような場合は、問合せが結果を項目ごとに提供するのを妨げるもの（たとえば集計式など）がないかぎり、カーソル・モードが唯一の手段になります。

追加情報は、8-40 ページの「ステートフル・クライアントとステートレス・クライアント」を参照してください。

XQS キャッシングの構成

この項では、XQS 使用時のキャッシュの構成方法を詳しく説明し、関連する考慮事項を示します。

- XQS キャッシュ設定
- XQS キャッシングの対策
- キャッシングと非決定的な結果

重要： XQS 実装では、OC4J Java Object Cache を使用してキャッシングが行われます。Java Object Cache が実行中で（また、必要に応じて自動的に起動され）、その構成ファイル *ORACLE_HOME*`/j2ee/home/config/javacache.xml` が存在する必要があります。通常は、このファイル・ディレクトリを更新する必要はありませんが、更新してもかまいません。第 7 章「Java Object Cache」を参照してください。

Java Object Cache のすべての構成は、XQS アプリケーションだけでなく、OC4J インスタンス全体に適用されることに注意してください。

XQS キャッシュ設定

XQS キャッシングは、XQS 関数ごとに（本質的には、アクセスするデータソースごとに）個別に構成されます。特定の XQS 関数に対してキャッシングを使用するには、次の手順を行います。

考慮すべき対策は、次項「XQS キャッシングの対策」を参照してください。すべての要素と属性に関する完全な説明は、8-93 ページの「XQS 構成ファイルのリファレンス」を参照してください。

1. `<document-source>` 要素、`<xqsview-source>` 要素または `<wsdl-source>` 要素の `isCached` 属性を介して、キャッシングを有効化します。次に例を示します。

```
<document-source isCached="true">
   ...
</document-source>
```

重要： `isCached` の値が `false`（デフォルト）に設定されている場合は、XQS 関数のその他のキャッシュ設定がすべて無視されます。

---

2. <document-source> 要素、<xqsview-source> 要素または <wsdl-source> 要素の <cache-properties> サブ要素を使用します。time-to-live 属性には、キャッシュ・エントリの期限が切れるまでの保持時間（秒単位）を決定する整数値が使用されます。次に例を示します。

```
<cache-properties time-to-live="600">
   ...
</cache-properties>
```

3. <cache-properties> の <in-memory> サブ要素を使用し、useSpool 属性および useDiskCache 属性を介して、使用するキャッシュ・モードを指定します。単純なメモリーベースのキャッシングの場合は、useSpool と useDiskCache の両方を false に設定します。メモリー不足になると必要に応じてキャッシュ・データをディスクにスプールできるメモリーベースのキャッシングの場合は、useSpool を true に設定し、useDiskCache を false に設定します。ディスクベースのキャッシングの場合は、useDiskCache を true に設定し、useSpool を false に設定します。（関連する考慮事項は、次項「XQS キャッシングの対策」を参照してください。）次の例では、必要に応じてスプーリングを使用する、メモリーベースのキャッシングが使用されています。

```
<in-memory useSpool="true" useDiskCache="false"/>
```

次に、完全なキャッシュ構成の例を示します。

```
<document-source isCached="true">
   ...
   <cache-properties time-to-live="600">
      <in-memory useSpool="true" useDiskCache="false"/>
   </cache-properties>
   ...
</document-source>
```

XQS キャッシングの対策

XQS キャッシングに関して、次の考慮事項に注意してください。

- 時間、アクセスのコスト、アプリケーション・オーバーヘッドなど、なんらかの形でデータにコストがかかっている場合には、ソースをキャッシュすることをお薦めします。キャッシングが OC4J インスタンスのシステム・メモリーに与える影響も考慮する必要がありますが、メモリーへの影響は、ディスクベースのキャッシング、またはスプーリングを伴うメモリーベースのキャッシングを使用することで軽減できます。（後続の項を参照してください。）
- 問合せ結果をキャッシュするために、問合せを XQS ビューとして定義します。ビューは、ある種の XQS ソースであり、キャッシュ可能です（つまり、その結果もキャッシュされます）。
- キャッシュ・オブジェクトの有効期限、つまり TTL は、ユーザーが判断します。データの性質および有益な期間に基づいて、データが期限切れになるまでのキャッシュ期間を決定します。
- <in-memory> 設定の定義では、単純なメモリーベースのキャッシング（useSpool="false" および useDiskCache="false"）、メモリー不足により必要になった場合にディスクにスプールできるメモリーベースのキャッシング（useSpool="true" および useDiskCache="false"）、ディスクベースのキャッシング（useSpool="false" および useDiskCache="true"）のどれを使用するかを考慮します。追加情報は、8-99 ページの「<in-memory>」を参照してください。注意事項は次のとおりです。
  - 単純なメモリーベースのキャッシングでは、パフォーマンスが最も速くなりますが、メモリー不足またはサーバーでの障害の発生により、キャッシュ・データが失われる危険性も最も高くなります。
  - スプーリングを伴うメモリーベースのキャッシングでは、（メモリーの問題によりキャッシュ・データが常にスプールされないかぎり）ディスクベースのキャッシングよりパフォーマンスが速くなりますが、サーバーで障害が発生した場合にキャッシュ・データが失われる危険性があります。

– ディスクベースのキャッシングでは、パフォーマンスは最も遅くなりますが、キャッシュ・データの安全性は最も高くなります。メモリーを考慮する必要がなく、サーバーで障害が発生してもキャッシュ・データはリカバリされます。

キャッシュ・データの危険度と実行速度の重要度を比較検討します。

キャッシングと非決定的な結果

8-7 ページの「要件、制限事項および特記事項」で説明している XQS 関数とノード ID の問題は、一部の例外的または特殊な状況で発生し、キャッシングを使用しているかどうかによって、問合せ結果が異なる場合があることに注意してください。ただし、ほとんどの問合せが（ノードではなく）値が等しいことに依存するため、通常はこの問題は発生しません。

ラージ・データ用の XQS ドキュメント・ソースの構成

システム容量や一般的なメモリー節約オプションの使用に関係なく、データセットが大きいためにメモリーによる制限を受ける場合があります。この問題を解決するには、背後で作業単位方式を使用します。この方法では、取得されるデータの合計量に関係なく、一定量のメモリーが使用されます。XQS は、Oracle XDK 内の依存するテクノロジおよび中間層の XQuery と連携しているため、XQS の実行中は常に、このスケーラブルな解決方法が提供されます。

ただし、この作業単位方式でも解決できないほどデータセットが大きい場合もあります。ドキュメント・ソースについてこのような状況が発生した場合は、データソースによって大量のメモリー・リソースが消費されるため、メモリー節約の内部最適化オプションを使用するように、XQS に指示することができます。次の例のように、`<document-source>` 要素の属性である `largeData` フラグを使用します。

```
<document-source largeData="true">
   ...
</document-source>
```

サイズの大きいドキュメント・ソースをメモリーにロードできないときは、このフラグを有効化することで可能になる場合があります。

重要：

- `largeData` フラグは XQS ビューおよび WSDL ソースには適用されません。
- `largeData="true"` を使用すると、処理速度が低下します。この設定は、必要な場合のみ使用してください。このような理由から、デフォルト値は `false` に設定されています。

XQS エラー処理モードおよび API の使用方法

8-13 ページの「XQS エラー処理の概要」で示したように、XQS 関数を構成して、（データソースを使用できない、またはデータ変換の問題などの）エラーが発生しても処理を続行することができます。処理を続行するように XQS 関数を構成した場合は、後で `oracle.xqs.client.XQSErrorI` オブジェクトのイテレータとして戻されるエラーを取得し、エラーの情報を入手できます。

XQS では、XQS 関数内部のエラーにのみ、特別な処理が適用されます。構文エラーやタイプの不一致などの通常の XQuery エラーの場合は、いずれの XQS エラー・モードでも、問合せの実行が停止します。

XQS エラー・モードは、XQS 関数ごとに個別に構成します。1 つの関数のエラー・モードが、同じ問合せ内の他の XQS 関数の動作に影響することはありません。

次の各項では、処理を続行してエラー情報を戻すための XQS 関数の構成方法、およびエラー情報の取得に使用できるメソッドについて説明します。

- XQS 関数のエラー処理の構成
- XQS エラー・オブジェクトの取得
- XQS エラー・オブジェクトからの情報の取得
- 例 : エラーの取得および処理

XQS 関数のエラー処理の構成

`xqs-config.xml` で構成する各データソースについて、ソース要素 `<document-source>`、`<xqsview-source>` または `<wsdl-source>` の `onError` 属性を介して、XQS のエラー処理を指定できます。サポートされている設定は次のとおりです。

- `dynamicError`（デフォルト）: この設定では、XQS 関数の評価時に問題が発生すると、XQuery 動的エラーが発生し、問合せが停止します。このような状況ではエラー・オブジェクトは戻されず、例外メッセージ以外の情報を取得できません。
- `emptySequence`: この設定では、XQS 関数によって生成されたエラーが発生しても、空の XML 順序が関数の結果として戻され、問合せの実行は続行されます。次の各項で示すように、エラー・オブジェクトを取得して、そのオブジェクトから情報を入手できます。
- `errorMessage`: この設定では、XQS 関数によって生成されたエラーが発生すると、関数の結果に対して単一項目の XML 順序が戻されます。順序内の項目はエラー・メッセージを表します。このメッセージは、事前構成済のメッセージまたはデータソースが戻すなんらかのメッセージで、構成によって異なります。また、次の各項で示すように、エラー・オブジェクトを取得して、そのオブジェクトから情報を入手できます。

特定のデータソースに対する XQS エラー処理の構成手順は、場合によって 2 つになります。

1. `<document-source>` 要素、`<xqsview-source>` 要素または `<wsdl-source>` 要素の `onError` 属性の設定を指定します。
2. `errorMessage` モードを使用する場合は、必要に応じて `<document-source>`、`<xqsview-source>` または `<wsdl-source>` の `<error-message>` サブ要素を使用します。これにより、問合せ結果に対して戻される単一項目の XML 順序に、XQS のエラー・メッセージが含まれるように、エラー・メッセージが事前構成されます。

次に、`emptySequence` モードを使用するソースの例を示します。

```
<document-source onError="emptySequence">
   ...
</document-source>
```

次に、事前構成済のメッセージを使用する errorMessage モードの使用例を示します。

```
<wsdl-source onError="errorMessage">
   ...
   <error-message>No information available</error-message>
   ...
</wsdl-source>
```

errorMessage モードの場合、事前構成済のメッセージを指定しないと（つまり、<error-message> 要素を使用しないと）、データソースによって戻されたメッセージがそのまま XQS によって使用され、そのメッセージが、関数の結果として戻された単一項目の XML 順序に含められます。データソースのエラー・レポート機能が不十分な場合には、意味のあるエラー・メッセージを取得することが不可能になることがあります。この場合は、事前構成済のメッセージが指定されていないと、エラー・メッセージが空の状態で戻されます。

戻される順序に含まれる単一項目のタイプは、xs:string または xs:node のどちらかになります。

- 項目が文字列の場合、その文字列は単純に、事前定義されたか実際に発生したエラーから取得されたエラー・メッセージです。
- 項目が XML ノードの場合、その項目は xqserror で、そのテキスト値がエラー・メッセージになります。また、エラー・ノードには、functionName、type および code という 3 つの属性があります。XQS エラー・ノードは次のようになります。

```
<xqserror functionName="XQS function name" type="XQSERR_XXX" code="XQS 0NNN" >
   error message
   </xqsrror>
```

XQS により、関数のタイプ情報に基づいて、そのエラー項目により適したタイプが選択されます。

- XQS 関数がドキュメント関数の場合、エラー項目のタイプは xs:node になります。これは、ドキュメント関数が常に XML ノードを戻すことになっているためです。
- XQS 関数が <output-element> を使用して構成されており、タイプ名または要素名がプリミティブ XML Schema タイプでない場合、この関数は XQS によって XML 要素タイプであるとみなされ、エラー項目が XML ノードとして作成されます。
- （xs:string ではないにもかかわらず）<output-element> によってプリミティブ XML Schema タイプが指定されている場合、または <output-element> が指定されていない場合は、XQS によって、エラー項目が文字列として作成されます。

注意事項および考慮事項

- errorMessage エラー・モードを使用して XQS 関数を構成し、<output-element> を指定しない場合、または <output-type> を指定したがユーザーが定義した複合型もしくは xs:string でない場合は、その関数を使用する問合せの XQuery プロローグでその関数の戻り型を指定しないでください。外部 XQuery 関数の戻り型の宣言は、オプションです。前述の状況で戻り型を指定すると、実行時に、この戻り型が XQuery によって強制的に使用されます。XQS 関数の実行時にエラーが発生すると、XQS によって、エラー・メッセージが文字列として戻されます。これが、問合せのプロローグで宣言された戻り型と競合します。戻り型を宣言しない場合は、強制的な型の一致は実行されません。
- XQS ビューでその他の XQS 関数およびソースが使用される状況では、ビューのエラー処理構成と基礎となる関数のエラー処理構成は関連性を持ちます。基礎となる関数の構成により、エラー発生時の関数の動作が決まります。動作は、動的エラーの発生時に停止するか、または続行して事前決定済の値（空の順序またはエラー・メッセージ）を戻すかのどちらかです。ビュー自体の構成では、エラー発生時にビューに関連付けられている問合せが停止するか続行するかが決まります。この場合のエラーには、基礎となる関数によって発生したものが含まれますが、これに限定されません。
- <output-element> を定義する <xqsview-source> 要素について、基礎となる問合せまたは別の <xqsview-source> 関数を使用してネストされた問合せの中の XQS 関数によって、errorMessage エラー処理オプションが定義されているかどうかを確認してください。このような、onError="errorMessage" オプションを使用するネストされた <xqsview-source> 関数では、それぞれ適切な <output-element> 定義が指定される必要

があります。この要件が満たされていない場合に、ネストされた XQS ビュー関数でエラーがすると、XQS によって、その関数に対して xs:string タイプの戻り値が作成されます。これは、宣言された最外部の XQS ビュー関数のタイプに違反するため、タイプの不一致による追加の XQuery エラーが作成されます。

その他のエラー処理オプションは、この要件の対象ではありません。

- デフォルトの `dynamicError` モードでエラーが発生した場合は、XQS クライアントによって、例外が次のように伝播されます。まず、EJB によって EJB 例外が戻されます。次に、`XQSFacadeI` インスタンスによって XQS 例外をラップする XQS 例外がスローされます。そして、JSP タグによって、標準の JSP `errorPage` 属性を介して処理できる JSP タグ例外がスローされます。
- XQS では、エラー・オブジェクトが作成されるだけでなく（次項「XQS エラー・オブジェクトの取得」を参照）、全体的な OC4J のロギング構成に基づいて、OC4J ログ・ファイルにエラーがレポートされます。XQS のエラー、警告または情報の各メッセージは、関連ログ・ファイル内で検索できます（*`ORACLE_HOME`*`/j2ee/home/log/oc4j/log.xml` など）。OC4J ロギングの一般情報は、『Oracle Containers for J2EE 構成および管理ガイド』を参照してください。

XQS エラー・オブジェクトの取得

XQS 関数が `emptySequence` または `errorMessage` のエラー処理モードを使用するように構成されていれば、XQS クライアント API の機能を使用して、エラー・オブジェクトのイテレータを取得でき、このイテレータから、問合せ中に発生した問題に関する情報を入手できます。概要は次のとおりです。

- `XQSFacadeI` インタフェースまたは XQS EJB API のどちらか一方（ステートフルまたはステートレス）を使用する場合は、次のメソッドを使用して、エラー・オブジェクトを含むイテレータを取得します。

  ```
  java.util.ListIterator getErrors()
  ```

- XQS JSP タグ・ライブラリ（ステートフルまたはステートレス）を使用する場合は、`execute` タグまたは `executeCursor` タグの `errors` 属性の値として目的の変数名を指定します。JSP タグの実装により、`ListIterator` 変数が作成され、この変数に、エラー・オブジェクトを含むイテレータが移入されます。

どちらの場合も、`oracle.xqs.client.XQSErrorI` オブジェクトを含むイテレータを取得できます。次に例を示します。

```
...
XQSFacadeI bean = XQSFactory.getXQSFacade();
...
bean.executeView(viewname, namespace, queryparams);
...
ListIterator errorIt = bean.getErrors();
XQSErrorI error = null;
while(errorIt.hasNext()) {
   error = (XQSErrorI)errorIt.next();
   (Process errors)
}
...
bean.close();
...
```

次に、JSP の例を示します。

```
XQSErrorI error=null;
...
<xq:executeCursor ... errors="errorIt" ... >
   ...
</xq:executeCursor>
...
<xq:next ... />
while(errorIt.hasNext()) {
```

```
    error = (XQSErrorI)errorIt.next();
    (Process errors)
}
```

詳細およびエラー情報取得の例は、次の「XQS エラー・オブジェクトからの情報の取得」および「例 : エラーの取得および処理」の各項を参照してください。

XQS エラー・オブジェクトからの情報の取得

前項の「XQS エラー・オブジェクトの取得」で、oracle.xqs.client.XQSErrorI オブジェクトを含む java.util.ListIterator インスタンスの取得方法について説明しました。

このイテレータ・インスタンスを処理すると、個々の XQSErrorI オブジェクトを取得できます。ListIterator タイプには、次のメソッドが含まれます。

- Object next(): このメソッドを使用すると、イテレータ内に次の要素が戻されます。各要素は、XQSErrorI タイプにキャストできます。
- boolean hasNext(): このメソッドを使用すると、要素が残っている場合に true が戻されます。

その後、個々の XQSErrorI インスタンスを使用して、問合せ中に発生した各エラーの情報を取得できます。

注意 :

- XQS によって XQSErrorI オブジェクトが戻されるのは、問合せで使用する 1 つ以上の XQS 関数を、デフォルト以外の emptySequence モードまたは errorMessage モードを使用して構成した場合のみです。エラーが発生しなかった場合は、ListIterator は空になります。
- XQSErrorI Java オブジェクトをタイプ xs:node のエラー項目と混同しないでください。8-68 ページの「XQS 関数のエラー処理の構成」で説明しているように、後者は、場合によって、エラー発生時に XQS 関数の戻り値として XQS によって構成されます。XQSErrorI は、XML ノードではなく Java オブジェクトです。XQSErrorI オブジェクトは、XQS のクライアントに戻されます。一方、XML エラー・ノードは、XQuery 実行に戻されます。また、エラー・モードが emptySequence であっても、XQSErrorI オブジェクトが戻されます。これに対して、XML エラー・ノードは、エラー・モードが errorMessage の場合のみ構成されます。

この項では、エラー情報を取得するための XQSErrorI メソッドを要約し、XQS エラーのタイプを要約します。完全な情報は、8-92 ページの「XQSErrorI リファレンス」を参照してください。

各 XQSErrorI インスタンスでは、次の処理を行えます。ここに示す例では、xqserr という XQSErrorI インスタンスを使用しています。

1. エラーを戻した XQS 関数の修飾名を取得します。

   ```
   String funcname = xqserr.getFunctionQName();
   ```

2. エラー・タイプを取得します（タイプの詳細は次の項を参照してください）。

   ```
   int errortype = xqserr.getErrorType();
   ```

3. エラー・タイプの文字列表現を取得します。

   ```
   String typeStr= XQSErrorI.typeNames(xqserr.getErrorType()
   ```

4. エラー・メッセージを取得します。

   ```
   String errormessage = xqserr.getErrorMessage();
   ```

5. エラー・メッセージとは別にエラー・コードを提供するデータソース（Oracle データベースの ORA-xxxxx コードなど）について、エラー・コードを取得します。

```
String errorcode = xqserr.getErrorCode();
```

6. 必要に応じて、ユーティリティ・メソッドを使用して関数名、エラー・タイプ、エラー・メッセージおよびエラー・コードを 1 つの文字列に一体化します。この文字列は、XML 文字列（適切な XML マークアップを含む文字列）または XML ノードです。

```
String errorstring = xqserr.toString();
...
String errorxmlstring = xqserr.toXMLString();
...
org.w3c.dom.node errornode = xqserr.toXMLNode();
```

エラー・タイプの整定数の要約は、次のとおりです。

- XQSERR_MISSING_SRC: データソースが見つかりません。
- XQSERR_SRC_ACCESS: データソースは見つかりましたがアクセス・エラー（無効なログインなど）が戻されました。
- XQSERR_SRC_ERROR: データソースが見つかりアクセスしましたが、その他のエラーが戻されました。
- XQSERR_PARAM: バインド・パラメータをソースに渡すとき、もしくはソースから出力パラメータを受け取るときに、問題が発生しました。
- XQSERR_SRC_CONFIG: XQS 構成の処理中に問題が発生しました。
- XQSERR_INTERNAL: XQS で予期しない内部エラーが発生しました。

これらの定数の追加情報は、8-92 ページの「XQSErrorI リファレンス」を参照してください。

例 : エラーの取得および処理

この項では、単純なエラー処理の例および対応する構成を示します。

構成 （8-68 ページの「XQS 関数のエラー処理の構成」で説明したように）XQS アプリケーションで XQSErrorI オブジェクトを使用するには、データソースがエラー・モード emptySequence または errorMessage 用に構成されている必要があります。次の例では、errorMessage モードが使用されています。

```
<xqsview-source ... onError="errorMessage">
   <function-name prefix="xqs">BasicFileWS</function-name>
<output-element namespace="myNS">myElement</output-element>
   ...
</xqsview-source>
```

<error-message> 要素が使用されていないため、戻される単一項目の XML 結果順序には、事前定義済メッセージが含まれず、かわりにデータソースから取得されるメッセージがそのまま転送されます。XQS ビュー関数で実際にエラーが発生すると、XQS によって <xqserror> XML ノードが構成され、そのノードが関数の結果として戻されます。エラー・メッセージは、<xqserror> ノードのテキスト値になります。XQS ビュー構成によって <output-element> 構成内のユーザー定義の要素が指定されるため、XML のノード表現が選択されます。

Java コード　次の例は、ビューを実行してから次のエラー処理を行うコードの一部分です。

- XQSFacadeI インタフェースの getErrors() メソッドをコールして、XQSErrorI オブジェクトの java.util.ListIterator インスタンスを取得します。
- ListIterator クラスの hasNext() メソッドを使用して、反復するエラー・オブジェクトの数を指定します。
- エラーごとに、エラーの原因となった XQS 関数の修飾名を出力してから、エラー・タイプ、エラー・メッセージおよびエラー・コードを出力します。

```
...
XQSFacadeI bean = XQSFactory.getXQSFacade();
String xqueryView = "BasicFileWS";
QueryParameterI param = XQSFactory.getQueryParameter("custName");
param.setString("John Ford");
QueryParameterI[] params = new QueryParameterI[1];
params[0] = param;
bean.executeView(xqueryView, null, params);
while(bean.getNextItem() !=null) {
}
ListIterator errors = bean.getErrors();
while(errors.hasNext()) {
   XQSErrorI error = (XQSErrorI)errors.next();
   System.out.println("The function which gives error is: " +
                      error.getFunctionQName());
   System.out.println("The error type is:"  + error.getErrorType()) +
", which means " +XQSErrorI.typeNames(error.getErrorType()));

   System.out.println("The error message is:" + error.getErrorMessage());
   System.out.println("The error code is:" + error.getErrorCode());
}
...
```

BPEL 環境での XQS の使用

XQuery および XQS は、Oracle Application Server 上で実行されている Oracle BPEL Process Manager から使用できます。

> **注意：** この項では、Oracle BPEL Process Manager に採用されている概念について記述します。この項では、Oracle BPEL における XQuery および XQS の使用に固有の概念のみを説明します。ここに記載される一般的な BPEL および Oracle BPEL の概念の詳細は、『Oracle BPEL Process Manager 開発者ガイド』を参照してください。

Oracle BPEL には、Oracle BPEL Process Manager による XQuery ベースの XML 問合せを可能にする XPath 機能があります。問合せは XQS ビューに保存できます。XQS ビューはフォルダ（2 階層まで）および名前空間に編成できます。XQS ビューを使用すると、ある XQS ビューを他の XQS ビューから呼び出すことで複雑な問合せを作成する機能が改善されるだけでなく、アプリケーションのモジュール性も向上します。

> **注意：** BPEL 環境から起動される問合せは、XQS ビューと同じように動作します。ただし、BPEL 環境で使用可能な機能は、Oracle Application Server で実行されている汎用 J2EE アプリケーションの XQS ビューと比べて制限されています。

この項には、次の項目が含まれます。

- XQuery 用の BPEL XPath 機能の登録
- BPEL XQuery 関数のパラメータの使用
- XQuery を使用したコンテキスト項目へのアクセス
- BPEL における XQS ビュー機能の使用
- 外部 XQuery 変数の使用
- XQS ビューにおける BPEL プロセス変数の使用
- XQS ビュー用の BPEL 名前空間およびフォルダの使用
- ビューの名前としての URL の使用

XQuery 用の BPEL XPath 機能の登録

Oracle BPEL には、次に示すように、`orabpel¥domains¥default¥config¥xpath-functions.xml` ファイルに登録されている Java クラス（`com.collaxa.cube.xml.xpath.functions.xml.GetElementFromXQueryFunction`）が用意されています。

```
<function id="processXQuery">
   <classname>
      com.collaxa.cube.xml.xpath.functions.xml.GetElementFromXQueryFunction
   </classname>
   <comment>
      <![CDATA[This function returns result of XQuery execution. The
         signature of this function is
         <i>ora:processXQuery('view_name','context_node'?)</i>.]]>
   </comment>
   <property id="namespace-uri">
      <value>http://schemas.oracle.com/xpath/extension</value>
      <comment>Namespace URI for this function</comment>
   </property>
   <property id="namespace-prefix">
      <value>ora</value>
      <comment>Namespace prefix for this function</comment>
   </property>
</function>
```

この例では、`GetElementFromXQueryFunction.class` ファイルを `orabpel¥system¥classes¥com¥collaxa¥cube¥xml¥xpath¥functions¥xml` ディレクトリに配置したことを前提としています。`ora:processXQuery` 関数は、BPEL プロセス定義のその他の XPath 関数と同じように使用されています。次に例を示します。

```
<assign name="runXQuery">
   <copy>
      <from expression="'Associated Press'"/>
      <to variable="publisher"/>
   </copy>
   <copy>
      <from>
      <category xmlns="http://samples.otn.com">Computing</category>
      </from>
      <to variable="category"/>
   </copy>
   <copy>
      <from expression="ora:processXQuery
         ('books.xq',bpws:getVariableData('input','payload'))"/>
      <to variable="output" part="payload"/>
   </copy>
</assign>
```

BPEL XQuery 関数のパラメータの使用

前の例（ora:processXQuery）で示された Oracle BPEL XQuery 関数には 2 つのパラメータがあります。

- view_name
- context_node

view_name パラメータは必須で、文字列である必要があります。このパラメータでは、実行する保存済の XQuery ファイル（XQS ビューとも呼ばれる）の名前を指定します。オプションで、ファイル名にはファイル拡張子 .xq を指定することもできます。1 つ目の引数が .xq で終わらない場合は、ファイルを検索する前に XQS によってファイル拡張子が追加されます。

context_node パラメータはオプションです。指定する場合、値は XML ノードである必要があります。この XML ノードは、実行される問合せのコンテキスト項目になります。詳細は、「XQuery を使用したコンテキスト項目へのアクセス」を参照してください。

XQuery を使用したコンテキスト項目へのアクセス

外部変数にバインドせずに、XML ドキュメントを XQuery 式に渡すことができます。これは、ドキュメントのコンテキスト項目へのバインドと呼ばれます。問合せ式内では、コンテキスト項目には特別な構文（.）を使用してアクセスします。

たとえば、コンテキスト項目がルート要素 <books> および一連の <book> 子要素を持つドキュメントにバインドされている場合は、次の式が有効です。

```
for $i in ./book/title return $i
```

XML ドキュメントを XQuery コンテキスト項目としてバインドするには、BPEL XQuery 関数の context_node パラメータを使用します。

注意： コンテキスト項目は、Oracle BPEL 環境でのみ XQS によって実行された XQuery に提供されます。この機能は、XQS を使用する汎用の J2EE アプリケーションでは使用できません。ただし、XML ノードは、通常の問合せパラメータおよび XQS 関数の引数として XQuery に渡すことができます。

BPEL における XQS ビュー機能の使用

XQS を使用すると、XQS ビューと呼ばれるファイルに保存された問合せという形で、標準の XQuery 言語を拡張できます。XQS には名前および関連付けられた XQuery 式があります。

ビューの実装は、そのビューと関連付けられた XQuery 式です。次に示す問合せ式を、ファイル myView.xq と関連付けることができます。

```
for $i in  fn:doc("/dir/file.xml") return $i/line-item[@id=10]
```

トップレベルのビュー

トップレベルのビューは、BPEL プロセスから直接起動されます。次の例では、myTopView という名前のビューが、processXQuery XPath 関数を使用して BPEL プロセスから起動されています。

```
<copy>
   <from expression="ora:processXQuery('myTopView.xq'))"/>
   <to variable="output" part="payload"/>
</copy>
```

トップレベルのビューには次の規則が適用されます。

- トップレベルのビューの XQuery ファイルは、BPEL プロセスのルートのスーツケース・フォルダに直接配置する必要があります。サブフォルダに配置することはできません。
- 暗黙的に宣言された外部変数（次の項を参照）のみが、トップレベルのビューで使用されます。また、外部変数の名前は、ビューが呼び出されているプロセスのアクティブな範囲内にある BPEL プロセスの変数と一致している必要があります。

- コンテキスト項目のドキュメントは、トップレベルのビューとのみ使用できます。

例として、myTopView の内容を次に示します。

```
for $i in fn:doc("/dir/file.xml") return $i/line-item[@id=$bpelVar]
```

外部変数 $bpelVar は宣言されておらず、BPEL プロセス変数とみなされています。

ネストされたビュー

ネストされたビューは、別の XQS ビューから呼び出された XQS ビューです。XQuery 言語の構文に合せるため、呼出し元のビューはネストされたビューを外部関数として参照します。ネストされたビューでは、ビューの名前は外部関数の名前でもあり、そのビューを使用する別の問合せ式で宣言されます。次の例では、myView がネストされたビューの名前です。

```
declare namespace bpel="urn:oracle.bpel.xq";
declare function bpel:myView($arg as xs:integer) as node()* external;
```

外部関数 myView() が問合せ式で呼び出されるたびに、XQS が関数の整数パラメータを myView() 内の外部関数にバインドし、前の問合せを実行します。結果順序は、外部関数呼出しの結果です。

XQS ビューは、基本的にストアド・クエリーであり、ある XQuery を別の XQuery 内に埋め込むことができます。同時に、ネストされた問合せを他から分離し、モジュール化して個別に変更できるようにします。たとえば、外側（埋込み先）の問合せは次のようになります。

```
declare namespace bpel="urn:oracle.bpel.xq";
declare function bpel:myView($arg as xs:integer) as node()* external;
for $i in bpel:myView(2301) return $i/shipping-status
```

外部の XQS ビュー関数は、関数と同じ名前のテキスト・ファイルを介して問合せと関連付けられます。ファイル拡張子は .xq です。たとえば、myView.xq という名前のファイルには次のテキストが含まれます。

```
declare variable $x as xs:integer external;
for $i in  fn:doc("/dir/file.xml") return $i/line-item[@id=$x]
```

これまでの例からわかるように、ネストされたすべてのビューはいずれかの名前空間に属します。この名前空間は、デフォルトの Oracle BPEL の名前空間 urn:oracle.bpel.xq、またはユーザー定義の名前空間です。ネストされたビューをユーザー定義の名前空間にグループ化する方法は、「XQS ビュー用の BPEL 名前空間およびフォルダの使用」で説明します。

外部 XQuery 変数の使用

XQuery の外部変数を使用すると、XQuery 式を別の XQuery 式に埋め込む環境からの入力値を渡すことができます。この概念は、SQL のバインド変数に似ています。次に、3 つの外部変数（$x、$y および $z）を宣言する単純な問合せの例を示します。問合せでは、変数の値が順番に戻されます。

```
declare variable $x as xs:string external;
declare variable $y as xs:integer external;
declare variable $z external;
($x, $y, $z)
```

外部変数のタイプの宣言は、変数 $z で示されているようにオプションです。問合せに対する追加の静的および動的タイプ・チェックが実行され、アプリケーションがより堅牢になるため、あらかじめタイプがわかる場合にはタイプを指定して外部変数を宣言することをお薦めします。

また、XQuery では外部変数を暗黙的に宣言できます。前もって宣言せずに XQuery 式で変数表記（$）が使用された場合、変数は暗黙的に宣言された外部変数とみなされます。次の XQuery 式は、3 つの外部変数が暗黙的に宣言されていること以外は前の例と同じです。

```
($x, $y, $z)
```

注意： 暗黙的に宣言された外部変数は、BPEL 環境でのみ使用可能です。XQS を使用する汎用の J2EE アプリケーションでは、問合せの中で明示的に宣言された外部変数のみを使用します。

XQuery 評価では、すべての外部変数が実際の値にバインドされている必要があります。暗黙的に宣言された外部変数と明示的に宣言された外部変数では処理が異なります。

外部変数が暗黙的に宣言されている場合、その名前は現在対象範囲内にある BPEL プロセス変数の名前と一致する必要があります。BPEL Process Manager および XQS により、プロセス変数の値が同じ名前の外部変数にバインドされます。

注意： 暗黙的に宣言された外部変数を使用できるのは、BPEL XQuery 関数から直接呼び出されたビューでのみで、ネストされた XQS ビューでは使用できません。

XQS ビューで BPEL プロセス変数を使用するその他の例は、「XQS ビューにおける BPEL プロセス変数の使用」を参照してください。

XQuery の外部変数が問合せ内で明示的に宣言されている場合、そのビューで外部変数が宣言されている順序に従って、変数の値は XQS ビュー関数に対する引数から取得されます。別のビューから XQS 関数として起動された XQS ビューは、ネストされたビューと呼ばれます。

注意： BPEL 環境では、明示的に宣言された外部変数は、ネストされたビューでのみ使用されます。

まとめると、XQS ビューが BPEL XQuery 関数から直接起動された場合、そのビューで使用するのは暗黙的に宣言された外部変数のみの必要があります。また、その値は、XQuery 関数が呼び出された際に対象範囲であった同じ名前の BPEL プロセス変数から取得されます。次に例を示します。

`topLevelView.xq` XQS ビュー・ファイル内の問合せ

```
for $b in $bpelVar1/a/b return $b[@id = $bpelVar2]
```

この問合せは、変数 `$bpelVar1` および `$bpelVar2` が対象範囲である BPEL プロセスで起動されます。

```
<copy>
   <from expression="ora:processXQuery('topLevel.xq')"/>
   <to variable="output" part="payload"/>
</copy>
```

XQS の外部変数を使用して XQS ビューが別の XQS ビューから起動された場合、そのビューで使用されるのは明示的に宣言された外部変数のみである必要があります。XQS ビュー関数の引数の値は引数が渡された順序で取得され、そのビューで変数が宣言された順序で外部変数に割り当てられます。`nestedView.xq` XQS ビュー・ファイルの例を示します。

```
declare variable $x as element() external;
declare variable $y as xs:integer external;
for $b in $x/a/b return $b[@id = $y]
```

`nestedView` 問合せは、`topView` という別のビューから起動されます。`topView.xq` の内容は次のとおりです。

```
declare namespace ns1="folder1";
declare function ns1:nestedView($arg1 as element(), $arg2 as xs:integer) external;
let $el := <x>
              <a>
                 <b id="1">b-content</b>
              </a>
           </x>
```

```
return ns1:nestedView($el, 1)
```

nestedView を実行する前に、XQS では外部変数 $x を要素 <x> に、外部変数 $y を整数 1 にバインドします。

XQS ビューにおける BPEL プロセス変数の使用

XQuery XPath 関数から直接呼び出された XQS ビューでは、そのビュー内で追加の宣言を行わずに、BPEL プロセスで宣言された変数を使用できます。

BPEL プロセスに変数 $bpPurchaseAmt および $state が含まれ、これらの変数には BPEL プロセス内で値が割り当てられているとします。

また、XQS ビュー・ファイル（purchaseAmtAddTax.xq）は BPEL プロセスとともにパッケージ化されており、次のような内容だとします。

```
declare namespace bpl = "urn:oracle.bpel.xq";
declare function bpl:stateTax($st as xs:string) as xs:decimal external;
$bpPurchaseAmt *(1+ bpl:stateTax($state))
```

この例では、別の XQS ビュー・ファイル（stateTax.xq）も BPEL プロセスとともにパッケージ化されていると仮定されています。

例では、2 つの外部変数（$bpPurchaseAmt および $state）が使用されています。これらの外部変数は問合せ内では宣言されていません。検証を成功させるためには、BPEL プロセス内の XQuery XPath 関数から XQS ビューを呼び出す必要があります。また、プロセスには、対象範囲内の変数 $bpPurchaseAmt および $state が含まれていて、その値が処理する問合せに対して適切である必要があります。

XQS ビュー用の BPEL 名前空間およびフォルダの使用

BPEL 環境内での XQS ビュー用の名前空間およびフォルダの使用には、2 つのオプションがあります。

- デフォルトの名前空間およびフォルダを使用
- カスタムの名前空間およびサブフォルダを使用

デフォルトの名前空間およびフォルダ

XQuery ファイルのデフォルトの場所は、BPEL プロセス・スーツケースのルート・フォルダです。XQS は、名前空間とフォルダを関連付けます。デフォルトの XQuery フォルダは、デフォルトの Oracle BPEL 名前空間 urn:oracle.bpel.xq と関連付けられます。この名前空間を指定するには、dflt、bpel、myroot など、任意の接頭辞を使用できます。

BPEL プロセスから直接起動されたビューであるすべてのトップレベルの XQS ビューは、デフォルトのフォルダに配置する必要があります。つまり、トップレベルのすべてのビューはデフォルトの名前空間 urn:oracle.bpel.xq に属します。そのため、XQuery processXQuery 関数にビューの名前が渡される際には、デフォルトの名前空間は渡されません。

デフォルトのフォルダに配置されたビューも、デフォルトの名前空間の接頭辞を使用してネストされたビューとして起動されます。次のフォルダ構造があるとします。

```
<process suitcase root>
   /view1.xq
   /view2.xq
```

view2.xq を起動できる view1.xq の内容は次のとおりです。

```
declare namespace dflt = "urn:oracle.bpel.xq";
declare function dflt:view2($st as xs:string) as xs:decimal;
for $i in dflt:view2("Oracle") return $i
```

カスタムの名前空間およびサブフォルダ

Oracle BPEL 環境の XQS には、問合せをデフォルト・フォルダのサブフォルダに編成するオプションもあります。サブフォルダは 1 階層のみです。たとえば、次のようなフォルダ構造では、XQS はビュー view1.xq、view2.xq、view3.xq、view4.xq および view5.xq を検出できます。一方、invisible.xq はスーツケースのルートから 2 階層下のフォルダにあるため、XQS で検出できません。

```
<process suitcase root>
   /view1.xq
   /view2.xq
   /subfolder1
      /view3.xq
      /view4.xq
      /subfolder12
         /invisible.xq
   /subfolder2
      /view5.xq
```

サブフォルダに配置されたビューは、ネストされたビューとしてのみ使用されます。ネストされたすべてのビューは、呼出し元の問合せで外部関数として宣言する必要があるため、名前空間に属している必要があります。サブフォルダに配置されたビューの名前空間が、サブフォルダの名前になります。ただし、その名前空間に任意の接頭辞を 1 つ以上付けることができます。

フォルダ構造の例では、subfolder1/ および subfolder2/ のビューを起動する前に、次の宣言が使用されています。

```
declare namespace dflt="urn:oracle.bpel.xq";
declare namespace sub1="subfolder1";
declare namespace sub2="subfolder2";
declare function dflt:view1() as external;
declare function sub1:view4() as external;
declare function sub2:view5() as external;
```

> **注意：** XQuery では、あるフォルダの別のフォルダに対する相対位置は計算されません。呼出し元の問合せがどこに配置されているかにかかわらず、ネストされたビューが配置されているフォルダの名前から導出される名前空間によって、ネストされたビューを参照してください。

フォルダ構造の前の例で説明すると、view1.xq および view3.xq は次のようにして view5.xq から起動されます。

```
declare namespace dflt="urn:oracle.bpel.xq";
declare namespace sub1="subfolder1";
declare function dflt:view1() as external;
declare function sub1:view3() as external;

dflt:view1()//price - sub1:view3()//discount
```

ビューの名前としての URL の使用

XQS では、ローカル・ファイル・システムのファイルだけでなく、(`http:`、`ftp:` または `file:` プロトコルを使用している) 有効な Web URL を使用してアクセス可能なビューを起動することもできます。ビューの名前が URL の場合、アクセスするファイルの正確な名前を反映している必要があります (XQS では、`.xq` ファイル拡張子は追加されません)。また、ビューのテキスト・ファイルが、起動時にその URL からアクセスできる必要があります。次に例を示します。

```
<copy>
   <from expression="ora:processXQuery
      ('http://myhost/bpeldemos/views/myView.xq')"/>
   <to variable="output" part="payload"/>
</copy>
```

> **注意:** URL をビューの名前として使用できるのはトップレベルのビューにおいてのみで、ネストされたビューでは使用できません。

XQS クライアント API リファレンス

この項では、XQS public クラスおよびユーザー・インタフェースに関するリファレンス・ドキュメントを提供します。

- XQSFactory リファレンス
- XQS QueryParameterI リファレンス
- XQSFacadeI リファレンス
- XQS EJB クライアント API リファレンス
- XQS JSP タグ・ライブラリのリファレンス
- XQSErrorI リファレンス

XQSFactory リファレンス

この項では、`oracle.xqs.client.XQSFactory` クラスのメソッドを示します。ファクトリ・クラスは、メインの XQS クライアント・インタフェース `XQSFacadeI` および `QueryParameterI` のインスタンスの作成に使用されます。これらのメソッドは、ファクトリによって戻される特定のインタフェースが記載されている後続の項で詳細に説明されています。`oracle.xqs.client.XQSFactory` クラスには次のメソッドがあります。

- `XQSFacadeI getXQSFacade()`
- `QueryParameterI getQueryParameter()`
- `QueryParameterI getQueryParameter(String namespace, String localName)`
- `QueryParameterI getQueryParameter(String localName)`

XQS QueryParameterI リファレンス

この項では、oracle.xqs.client.QueryParameterI インタフェースおよびそのメソッドの取得に使用されるファクトリ・メソッドについて説明します。QueryParameterI 配列は、XQS の汎用 Java API または EJB API の使用時に、問合せの外部変数をバインドするために使用します。

8-46 ページの「例 3: XQSFacadeI API を使用した XQS ビュー」には、入力パラメータの QueryParameterI の使用例が含まれています。

QueryParameterI を戻すファクトリ・メソッド

oracle.xqs.client.XQSFactory クラスには、QueryParameterI インタフェースのインスタンスを作成する次のメソッドがあります。

- getQueryParameter()
- getQueryParameter(String namespace, String localName)
- getQueryParameter(String localName)

QueryParameterI インスタンスを XQSFactory ファクトリから取得する際には、バインドする外部 XQuery 変数の名前を指定します。この名前は、XQuery プロローグ内の対応する外部変数宣言の名前と一致する必要があります。また、名前を付けずに QueryParameterI インスタンスを作成し、QueryParameterI インタフェースの setName() メソッドを使用して後からパラメータ名を設定できます。この名前も、XQuery プロローグの外部変数の名前と一致する必要があります。XQuery 変数では XML 名が使用されるため、QueryParameterI コンストラクタでは、名前空間用とローカル名用の 2 つの文字列値を使用できます。名前空間を指定しない場合は、修飾名内の名前空間に、null が割り当てられます。

QueryParameterI のメソッド

QueryParameterI インタフェースでは、外部 XQuery 変数の値を設定するための、次のメソッドが提供されます。これらのメソッドは、XQS でサポートされている XML 型、およびそれに対応する Java 型を反映しています。8-38 ページの「サポートされている問合せパラメータのタイプ」も参照してください。パラメータ名を設定および取得するメソッドや、パラメータが NULL かどうかを確認するメソッドもあります。

- javax.xml.namespace.QName getName()

  このメソッドは、パラメータ名の特定に使用します。

- void setName(String namespace, String localPart)

  このメソッドは、名前空間および QName のローカル部分を指定してパラメータ名を設定するために使用します。

- boolean isNull()

  このメソッドは、パラメータ値が Null に設定されているかどうかを特定するために使用します。パラメータを Null に設定するには、この項にリストされている set*Null() メソッドのいずれかを呼び出します。初期化されていないパラメータは、Null に設定されているとはみなされません。

- void setBase64Binary(String value)

  このメソッドは、Java 文字列（たとえば、vYrfOJ39673//-BDiIIGHSPM=+）で表された、XML base64Binary 型に対応する Base64Binary 値のバインドに使用します。

- void setBase64BinaryNull()

  このメソッドは、Base64Binary 外部変数を Null にバインドするために使用します。

- void setBoolean(boolean value)

  このメソッドは、XML boolean 型に対応する boolean 値のバインドに使用します。

- void setBooleanNull()

  このメソッドは、boolean 外部変数を Null にバインドするために使用します。

- `void setDateTime(java.util.GregorianCalendar value boolean isTimeZoneSet)`

  このメソッドは、XML dateTime 型に対応する、ある特定の時間を表す GregorianCalendar 値のバインドに使用します。カレンダ・オブジェクトの TimeZone プロパティが設定されている場合は、isTimeZone を true に設定します。（GregorianCalendar オブジェクトの使用に関する情報は、http://java.sun.com/j2se/1.4.2/docs/api/ を参照してください。）

- `void setDateTimeNull()`

  このメソッドは、dateTime 外部変数を Null にバインドするために使用します。

- `void setDateTime(String strValue)`

  このメソッドは、XML dateTime 型に対応する、ある特定の時間を表す String 値のバインドに使用します。

- `void setDecimal(java.math.BigDecimal value)`

  このメソッドは、XML decimal 型に対応する BigDecimal 値のバインドに使用します。

- `void setDecimalNull()`

  このメソッドは、decimal 外部変数を Null にバインドするために使用します。

- `void setDouble(double value)`

  このメソッドは、XML double 型に対応する double 値のバインドに使用します。

- `void setDoubleNull()`

  このメソッドは、double 外部変数を Null にバインドするために使用します。

- `void setDuration(String value)`

  このメソッドは、XML duration 型に対応する、期間を表す String 値のバインドに使用します。期間の字句（文字列）表現の情報は、http://www.w3.org/TR/xmlschema-2/#duration を参照してください。

- `void setDurationNull()`

  このメソッドは、duration 外部変数を Null にバインドするために使用します。

- `void setDayTimeDuration(double seconds)`

  このメソッドは、XML dayTimeDuration 型に対応する、期間の値（秒単位）の dayTime 部分を表す double 値のバインドに使用します。dayTimeDuration の表現の情報は、http://www.w3.org/TR/xmlschema-2/#dayTimeDuration を参照してください。

- `void setDayTimeDurationNull()`

  このメソッドは、DayTimeDuration 外部変数を Null にバインドするために使用します。

- `void setYearMonthDuration(int months)`

  このメソッドは、XML yearMonthDuration 型に対応する、期間の値（月単位）の yearMonth 部分を表す int 値のバインドに使用します。yearMonthDuration の表現の情報は、http://www.w3.org/TR/xmlschema-2/#yearMonthDuration を参照してください。

- `void setYearMonthDurationNull()`

  このメソッドは、yearMonthDuration 外部変数を Null にバインドするために使用します。

- `void setFloat(float value)`

  このメソッドは、XML float 型に対応する浮動小数点値のバインドに使用します。

- `void setFloatNull()`

  このメソッドは、`float` 外部変数を `Null` にバインドするために使用します。

- `void setHexBinary(String value)`

  このメソッドは、Java 文字列（たとえば、`0FB7`）で表された、XML `hexBinary` 型に対応する 16 進バイナリ値のバインドに使用します。

- `void setHexBinaryNull()`

  このメソッドは、`hexBinary` 外部変数を `Null` にバインドするために使用します。

- `void setInt(int value)`

  このメソッドは、XML `int` 型に対応する `integer` 値のバインドに使用します。

- `void setIntNull()`

  このメソッドは、`int` 外部変数を `Null` にバインドするために使用します。

- `void setInteger(int value)`
- `void setInteger(long value)`
- `void setInteger(java.math.BigInteger value)`

  いずれかの `setInteger()` メソッドを使用して、XML `integer` 型に対応する整数値をバインドします。値の表現または大きさに基づいて、適切なシグネチャを選択します。

- `void setIntegerNull()`

  このメソッドは、`integer` 外部変数を `Null` にバインドするために使用します。

- `void setLong(long value)`

  このメソッドは、XML `long` 型に対応する `long` 整数値のバインドに使用します。

- `void setLongNull()`

  このメソッドは、`long external` 変数を `Null` にバインドするために使用します。

- `void setString(String value)`

  このメソッドは、XML `string` 型に対応する Java `string` 値のバインドに使用します。

- `void setStringNull()`

  このメソッドは、`string` 外部変数を `Null` にバインドするために使用します。

- `void setAnyURI(java.net.URI value)`

  このメソッドは、XML `anyURI` 型に対応する `URI` 値のバインドに使用します。

- `void setAnyURINull()`

  このメソッドは、`anyURI` 外部変数を `Null` にバインドするために使用します。

- `void setNode(org.w3c.dom.Node value)`

  このメソッドは、XML `anyType` 型またはユーザー定義の XML 型に対応する XML ノードのバインドに使用します。

- `void setNodeNull()`

  このメソッドは、外部変数 `anyType` またはユーザー定義の XML 型を `Null` にバインドするために使用します。

注意： `QueryParameterI` クラスには、これらの setter メソッドに対応する getter メソッドはありません。

XQSFacadeI リファレンス

この項では、oracle.xqs.client.XQSFacadeI インタフェースおよびそのメソッドの取得に使用されるファクトリ・メソッドについて説明します。このインタフェースは、XQS が提供する汎用目的の Java クライアント API です。使用方法および例は、8-41 ページの「Java インタフェース・クライアント API の使用方法」を参照してください。

XQSFacadeI を戻すファクトリ・メソッド

oracle.xqs.client.XQSFactory クラスには、XQSFacadeI インタフェースのインスタンスを作成する次のメソッドがあります。

- getXQSFacade()

XQSFacadeI のメソッド

XQSFacadeI インタフェースでは、次のメソッドが提供されます。

- void execute(String xquery, QueryParameterI[] params)

  このメソッドを使用して、非定型問合せを実行します。問合せを含む文字列、およびバインド・パラメータの QueryParameterI 配列を渡します。（8-81 ページの「XQS QueryParameterI リファレンス」を参照してください。）渡すパラメータがない場合、配列は null になります。

- void executeView(String viewName, String namespace, QueryParameterI[] params)

  このメソッドを使用して、XQS ビューを直接実行します。ローカル名および名前空間を（文字列として）指定することでビュー名を渡し、バインド・パラメータの QueryParameterI 配列（または null）を渡します。ビュー名および名前空間は、ともに修飾名の定義に使用されるため、<xqsview-source> の <function-name> サブ要素、およびその namespace 属性または prefix 属性に基づいて、構成内の対応する関数名および名前空間と一致する必要があります。

- oracle.xml.xqxp.datamodel.XMLItem getNextItem()

  execute() または executeView() を使用した後、getNextItem() を使用して結果順序内の次の項目を Oracle XMLItem インスタンスとして取得します。最初の getNextItem() コールでは、順序内の最初の項目が戻されます。最後の項目を取得した後にこのメソッドをコールすると、null が戻されます。

  重要： getNextItem() メソッドをコールすると、XQuery によって項目の評価がトリガーされます。問合せの実行時に要求されるリソース（データベース接続やファイル・ハンドラなど）は、最後の getNextItem() コールが終わるまで解放されません。

- java.util.ListIterator getErrors()

  問合せで使用するいずれかの XQS 関数を、エラーの発生後も続行するように構成した場合は（8-68 ページの「XQS 関数のエラー処理の構成」を参照）、getErrors() を使用して、XQS 関数の実行時に発生した可能性があるエラーの情報を取得できます。このメソッドでは、XQSErrorI オブジェクトのコレクションに対するイテレータが戻されます。このイテレータから、各エラーの詳細を取得できます。（8-92 ページの「XQSErrorI リファレンス」を参照してください。）エラーが発生しなかった場合は、イテレータによって空のコレクションがポイントされます。

- void close()

  このメソッドを使用して、問合せに関連付けられているすべてのリソースを解放します。close() は、問合せ結果をすべて取得した後でコールするのが適正です。

XQS EJB クライアント API リファレンス

この項では、リモート EJB 用の XQSClientRemote インタフェースと XQSClientHome インタフェース、およびローカル EJB 用の XQSClientLocal インタフェースと XQSClientLocalHome インタフェースを介して XQS でサポートされる、EJB クライアント・メソッドについて説明します。ステートフル・セッション Bean またはステートレス・セッション Bean 用に、これらのインタフェースには複数のバージョンがあります。

使用方法および例は、8-50 ページの「EJB クライアント API の使用方法」を参照してください。

ステートフル EJB クライアント・メソッド

ステートフル・セッション Bean では、次に示す Oracle 固有のメソッドを、パッケージ oracle.xqs.client.ejb.stateful に含まれる EJB インタフェースから使用できます。これらのメソッドの機能は、8-84 ページの「XQSFacadeI リファレンス」に含まれる同じ名前のメソッドと同じです。

- `void execute(String xquery, QueryParameterI[] params) throws EJBException`
- `void executeView(String viewName, String namespace, QueryParameterI[] params) throws EJBException`
- `oracle.xml.xqxp.datamodel.XMLItem getNextItem() throws EJBException`
- `java.util.ListIterator getErrors()`
- `void close()`

ステートレス EJB クライアント・メソッド

ステートレス・セッション Bean では、パッケージ oracle.xds.client.ejb.stateless に含まれる EJB インタフェースを介して、次に示す Oracle 固有のメソッドを使用できます。

- `java.util.Vector execute(String xquery, QueryParameterI[] params) throws EJBException`

  このメソッドを使用して、非定型問合せを実行します。問合せを含む文字列、およびバインド・パラメータの QueryParameterI 配列を渡します。（8-81 ページの「XQS QueryParameterI リファレンス」を参照してください。）渡すパラメータがない場合、配列は null になります。結果は完全にマテリアライズされ、Vector インスタンス内に戻されます。

- `java.util.Vector executeView(String viewName, String namespace, QueryParameterI[] params) throws EJBException`

  このメソッドを使用して、XQS ビューを実行します。ローカル名および名前空間を（文字列として）指定することでビュー名を渡し、バインド・パラメータの QueryParameterI 配列（または null）を渡します。ビュー名および名前空間は、ともに修飾名の定義に使用されるため、<xqsview-source> の <function-name> サブ要素、およびその namespace 属性または prefix 属性に基づいて、構成内の対応する関数名および名前空間と一致する必要があります。結果は完全にマテリアライズされ、Vector インスタンス内に戻されます。

- `java.util.ListIterator getErrors()`

  このメソッドには、8-84 ページの「XQSFacadeI リファレンス」に含まれる同じ名前のメソッドと同じ機能があります。

XQS JSP タグ・ライブラリのリファレンス

XQS では、XQuery へのステートフル・アクセスまたはステートレス・アクセス用のカスタム JSP タグが提供されます。

使用方法および例は、8-53 ページの「JSP タグ・ライブラリの使用方法」を参照してください。

JSP タグ・ライブラリの使用に関する一般情報は、『Oracle Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

注意： ここで示すタグ構文には、接頭辞 xq: が使用されています。これは慣例によるもので、必須ではありません。`taglib` ディレクティブでは、接頭辞を必要に応じて指定できます。

タグ構文に関して、次のことに注意してください。

- イタリックは、値または文字列を指定する場所を示します。
- オプションの属性は、大カッコ `[...]` で囲まれます。
- オプション属性のデフォルト値は、**太字**で示されます。
- 属性値の選択肢は、縦線 | で区切られます。
- 注釈がないかぎり、JSP 実行時式を使用して、タグの属性値 `<%=` *jspExpression* `%>` を設定できます。

ステートフル・アクセス用 JSP タグ

これらのタグは、XQuery へのステートフル・アクセスを使用する場合に使用します。

XQS executeCursor タグ このタグを使用して、問合せの準備および実行を行います。構文は次のとおりです。(`param` サブタグは、すべてのバインド・パラメータに使用されます。後続の説明を参照してください。)

```
<xq:executeCursor
  [ xqueryString = "query" ]
  [ xqsViewName = "viewname" ]
  [ namespace = "namespace" ]
    cursorId = "cursorname"
  [ maxItems = "maxnumber" ]
  [ toWriter = "true" | "false" ]
    toXMLDoc = "docname"
    resultItems = "arrayname"
    errors = "errorvarname" >

   <xq:param ... />
   <xq:param ... />
   ...

</xq:executeCursor>
```

重要：

- 問合せ入力には、`xqueryString` 属性を使用するか、`xqsViewName` 属性と `namespace` 属性を使用する必要があります。
- 出力属性 `toXMLDoc`、`resultItems` および `errors` の変数名を指定する必要があります。

---

executeCursor タグの属性は、次のように使用します。

- xqueryString: 非定型問合せを使用する場合は、この属性を使用して、問合せの完全な XQuery 構文を指定します。通常は、文字列変数で問合せを定義し、その変数の名前を JSP 式で指定するのが最も簡単です。

```
String myquerystring = "...";
...
<xq:executeCursor ... xqueryString = "<%=myquerystring%>" ... >
   ...
</xq:executeCursor>
```

- xqsViewName: XQS ビューを使用する場合は、この属性を使用してビューを指定します。厳密には、これは、構成内の対応する XQS 関数の名前です。
- namespace: xqsViewName を使用してビューを指定する場合は、namespace も併用して、XQS 構成に基づいてビューの XQS 関数の名前空間を指定する必要があります。
- cursorId（必須）: このタグを使用して、カーソルの変数の名前を指定し、後で使用できるようにします。（この変数は、JSP コンテナによって宣言されます。）指定した変数名は、next タグを使用してカーソルから結果を取得するとき、および close タグを使用してカーソルをクローズするときに参照します。次に例を示します。

```
<xq:executeCursor ... cursorId="mycursor" ... >
   ...
</xq:executeCursor>
...
<xq:next cursorId="mycursor" />
...
<xq:close cursorId="mycursor" />
```

- maxItems: maxItems="10000" のように、問合せから受け取る項目の最大数を指定する場合に使用します。超過分の項目は廃棄され、maxItem の項目を受け取った後は、next タグによって値が戻されなくなります。
- toWriter: true に設定すると、JSP 出力ストリームに問合せ結果が出力されます。書込みは、next タグが実行されるたびに行われます。
- toXMLDoc（必須）: XML DOM ドキュメントに問合せ結果を出力するために、関連する next タグで使用される変数の名前を指定します。ドキュメントは、executeCursor タグの後に使用できます。タイプ org.w3c.dom.Document の変数は、JSP コンテナによって宣言されます。次に例を示します。

```
<xq:executeCursor cursorId="mycursor" toXMLDoc="mydoc" ... >
   ...
</xq:executeCursor>

<xq:next cursorId="mycursor" />

Element root = mydoc.getDocumentElement();
...
```

executeCursor タグの使用後は、next タグを実行するたびに、ドキュメント mydoc 内で収集された結果ノードのバッチを受け取ります。

重要:

- toXMLDoc ドキュメントでは、XML ノードの結果のみが収集されます。プリミティブ型の結果は、XML 文書では無視されますが、resultItems 配列を介して収集されます。
- toXMLDoc ドキュメントでは、next タグの 1 回の実行で取得された項目のバッチのみが保持されます。その後、next を実行するたびに、保持されていた項目が新しい項目のセットに上書きされます。
- toXMLDoc 属性に対して実行式を使用することはできません。

注意： 指定した名前は、ドキュメントのルート要素名としても使用されます。

- resultItems（必須）：配列リストに問合せ結果を出力するために、関連する next タグで使用される変数の名前を指定します。タイプ java.util.ArrayList の変数は、JSP コンテナによって宣言されます。ArrayList インスタンスは、executeCursor タグの後に使用できます。次に例を示します。

```
<xq:executeCursor ... resultItems="myobjects" ... >
   ...
</xq:executeCursor>

Node node = (Node)myobjects.get(0);
...
```

executeCursor タグの使用後は、next タグを実行するたびに、myobjects 変数から結果項目にアクセスできます。

重要：

- resultItems 変数では、next タグの 1 回の実行で取得された項目のバッチのみが保持されます。その後、next を実行するたびに、保持されていた項目が新しい項目のセットに上書きされます。
- resultItems 属性に対して実行式を使用することはできません。

- errors（必須）：XQS 問合せの実行により発生したエラーの取得に使用する変数の名前を指定します。タイプ java.util.ListIterator の変数は、XQSErrorI オブジェクト（エラーごとに 1 つのオブジェクト）のコレクションに対するイテレータの JSP コンテナによって宣言されます。イテレータは、executeCursor タグの後に使用できます。次に例を示します。

```
<xq:executeCursor cursorId="mycursor" errors="myerroriter" ... >
   ...
</xq:executeCursor>
<xq:next cursorId="mycursor" />

XQSErrorI error = (XQSErrorI)myerroriterator.next();
...
```

XQSErrorI オブジェクトの使用方法は、8-92 ページの「XQSErrorI リファレンス」を参照してください。

重要： errors 属性に対して実行式を使用することはできません。

XQS param タグ executeCursor および execute のサブタグです。このタグを使用して、問合せの外部バインド・パラメータ（1 つの param タグ・パラメータ）を指定します。構文は次のとおりです。

```
<xq:param
    localName = "localvarname"
  [ namespace = "namespace" ]
    value = "bindvalue"
    type = "bindparamtype"
/>
```

param タグの属性は、次のように使用します。

- localName（必須）：外部変数名のローカル部分の指定に使用します。
- namespace: 外部変数名の名前空間部分の指定に使用します。名前空間を伴わないローカル変数名も使用できるため、この属性はオプションです。

- `value`（必須）：バインドされる値に使用します。推定上、この値は JSP 実行時式を使用したランタイム値になります。

  ```
  <xq:param ... value = "<%=jspExpression%>"... />
  ```

  注意： バインド値のタイプは、表 8-3 で示すように、`type` 属性の設定に対応している必要があります。

- `type`（必須）：XQS `QueryParameterI` インタフェースによって対応する Java 型がサポートされている XML 型を示す、バインド・パラメータのデータ型の指定に使用します。サポートされている設定および使用に適した型は、表 8-3 を参照してください。（8-38 ページの「サポートされている問合せパラメータのタイプ」も参照してください。）

表 8-3 type 属性の設定および対応する Java 型

type 属性の設定	対応する Java 型
boolean	java.lang.Boolean
string	java.lang.String
int	java.lang.Integer
integer	java.lang.Integer, java.lang.Long, java.math.BigInteger
long	java.lang.Long
float	java.lang.Float
double	java.lang.Double
decimal	java.math.BigDecimal
base64Binary	java.lang.String （値の表現。）
hexBinary	java.lang.String （値の表現。）
anyURI	java.net.URI
dateTime	java.util.GregorianCalendar （前提条件：カレンダ値のタイムゾーンが適切に初期化されていること。）
duration	java.lang.String （期間の字句表現。説明は `http://www.w3.org/TR/xmlschema-2/#duration` を参照。）
node	java.lang.String （ノードのテキスト表現。XML パーサーによって解析される。）

XQS next タグ このタグを使用して、項目またはバッチごとに、executeCursor タグから結果を処理します。その際には、そのタグで指定したカーソル ID 名が参照されます。next タグの構文は次のとおりです。

```
<xq:next
    cursorId = "cursorname"
  [ batchSize= "numitems" ]
  [ itemsFetched = "intvarname" ]
/>
```

next タグの属性は、次のように使用します。

- cursorId（必須）: この属性を使用して、executeCursor タグでの指定に従って、問合せのカーソル名を参照します。カーソルは、まだオープンである（同じカーソル名を指定する close タグがまだ存在していない）ことが必要です。
- batchSize: XQS での問合せにバッチ・モードを使用する場合は、この属性を使用します。next タグを実行するたびにカーソルからフェッチする項目数を、整数値で指定します。デフォルトは 1（バッチ不使用）です。
- itemsFetched: 現在の next タグの実行でカーソルからフェッチされた項目数を示す値の java.lang.Integer 変数の名前を指定します。この変数は、JSP コンテナによって宣言されます。

次に例を示します。

```
...
<xq:executeCursor cursorId="mycursor"
                  resultItems="myresults"
                  toXMLDoc="mydoc"
                  errors="myerrors" ... >
   <xq:param ... />
   ...
</xq:executeCursor>
<%
  int items;
  do {
  // Populate output vehicles (myresults array and mydoc document)
%>
   <xq:next
               cursorId="mycursor"
               itemsFetched="fetched" />
<%
  items = fetched.intValue();
  } while(items>0)
  // ... Here is where you would retrieve results from the output vehicles ...
%>
...
```

XQS close タグ（executeCursor タグを使用する）ステートフル JSP の場合は、close タグを使用して、問合せに関連付けられているすべてのリソースを解放します。構文は次のとおりです。

```
<xq:close
    cursorId = "cursorname"
/>
```

close タグの属性は、次のように使用します。

- cursorId（必須）: この属性を使用して、executeCursor タグでの指定に従って、クローズする問合せに対応するカーソル名を参照します。

ステートレス・アクセス用 JSP タグ

XQuery へのステートレス・アクセスを使用する場合は、execute タグを使用します。

XQS execute タグ このタグを使用して、問合せの準備および実行を行います。構文は次のとおりです。

```
<xq:execute
  [ xqueryString = "query" ]
  [ xqsViewName = "viewname" ]
  [ namespace = "namespace" ]
  [ maxItems = "maxnumber" ]
  [ toWriter = "true" | "false" ]
    toXMLDoc = "docname"
    resultItems = "arrayname"
    errors = "errorvarname" >

   <xq:param ... />
   <xq:param ... />
   ...

</xq:execute>
```

重要:

- 問合せ入力には、xqueryString 属性を使用するか、xqsViewName 属性と namespace 属性を使用する必要があります。
- 必須の属性 toXMLDoc、resultItems および errors を使用して、出力媒体の変数名を指定する必要があります。

---

cursorId を除き、このタグでは、(ステートフル・アクセス用の) executeCursor タグと同じ属性が使用されます。executeCursor と同様に、バインド・パラメータに対して XQS param サブタグも使用されます。8-86 ページの「XQS executeCursor タグ」および 8-88 ページの「XQS param タグ」を参照してください。

XQSErrorI リファレンス

この項では、インタフェース oracle.xqs.client.XQSErrorI のリファレンス・ドキュメントを提供します。

問合せで使用するいずれかの XQS 関数を、エラーの発生後も続行するように構成した場合は（8-68 ページの「XQS 関数のエラー処理の構成」を参照）、問合せの XQS 関数の実行時に発生した可能性があるエラーの情報を取得できます。エラーは、XQSErrorI オブジェクトのイテレータ内に戻されます。

XQSErrorI インタフェースでは、エラーが発生した XQS 関数、エラーのタイプ（XQS 固有の指定）、エラー・メッセージおよびエラー・コード（適用される場合）などの主な情報を様々な書式で戻すメソッドが多数提供されます。使用方法および例は、8-71 ページの「XQS エラー・オブジェクトからの情報の取得」を参照してください。

- `String getFunctionQName()`

  このメソッドは、エラーが発生した XQS 関数の完全修飾名を取得する場合に、次のように使用します。

  {namespace_URI}/local_name

- `int getErrorType()`

  このメソッドは、次のような XQS エラー・タイプの指定を示す整定数を取得するために使用します。

  - XQSERR_MISSING_SRC: HTTP 404 エラーが戻された、ドキュメント・ソースのファイルが見つからない、ソースが停止している、目的の表がデータベース・ソースに存在しないなどの理由で、データソースが見つかりません。
  - XQSERR_SRC_ACCESS: データソースは見つかりましたがアクセス・エラーが戻されました。これには、Oracle データベースの ORA-xxxxx エラー、Web サイトのログイン・エラー、ドキュメント・ソース読取り時のパーサー・エラーなどが含まれます。
  - XQSERR_SRC_ERROR: データソースは見つかりましたが、アクセス・エラー以外のなんらかのエラーが戻されました。これには、Web サービスのフォルト・メッセージ、データベース・ソースの SQL エラーなどが含まれます。
  - XQSERR_PARAM: バインド・パラメータをデータソースに渡すとき、もしくはデータソースから出力パラメータを受け取るときに、XQS で問題が発生しました。これが戻されるのは、たとえば、WSDL ドキュメントに従って要求される対応データ型に入力文字列を変換できなかった場合や、使用する特定の変換機能で XML 以外のファイルを XML 文書に変換できなかった場合などです。
  - XQSERR_SRC_CONFIG: XQS 関数の構成の処理中に、XQS で問題が発生しました。
  - XQSERR_INTERNAL: XQS で予期しない内部エラーが発生しました。

- `String getErrorMessage()`

  このメソッドは、必要に応じて、データソースまたは XQS から、エラー・メッセージを取得する場合に使用します。このメソッドでは、null が戻されることはありません。データソースによってエラーが生成され、メッセージが戻されなかった場合は、関数でエラーが生成されたことを示す汎用メッセージが、XQS によって戻されます。

- `String getErrorCode()`

  このメソッドは、データソースによって生成されたエラーについて、XQS エラー・コードの文字列を戻すために使用します。

- `String toString()`

  このメソッドは、関数名、エラー・タイプ、エラー・コードおよびエラー・メッセージを組み合せて単一の文字列にするために使用します。

- `String toXMLString()`

  このメソッドは、関数名、エラー・タイプ、エラー・コードおよびエラー・メッセージを組み合せて XML エラー要素の 1 つの文字列表現にするために使用します。

- `public static String[] typenames`

  この静的配列には、次に示す XQS エラー・タイプの文字列表現が含まれます。

  - `XQSERR_MISSING_SRC`
  - `XQSERR_SRC_ACCESS`
  - `XQSERR_PARAM`
  - `XQSERR_SRC_ERROR`
  - `XQSERR_SRC_CONFIG`
  - `XQSERR_INTERNAL`

  Java オブジェクトのエラー・タイプが `XQSErrorI` の場合は、式 `XQSErrorI.typNames[error.getErrorType()]` によって、オブジェクトのエラー・タイプに適した文字列が戻されます。

- `org.w3c.dom.Node toXMLNode()`

  このメソッドは、関数名、エラー・タイプ、エラー・コードおよびエラー・メッセージを DOM ノード書式で取得するために使用します。

XQS 構成ファイルのリファレンス

この項では、XQS 構成ファイル `xqs-config.xml` の要素の説明をアルファベット順に示します。この XQS 構成ファイルの XML スキーマは、`http://www.oracle.com/technology/tech/xml/xqs/xqs-config.xsd` にあります。また、このファイルは、`xds/samples/XQSDemo/XDS/xqs-config.xsd` の XLSDemo アプリケーションでも使用可能です。データソースの各カテゴリの主要な要素の使用方法の詳細は、8-21 ページの「XQS 関数の構成方法」を参照してください。

XQS により、アプリケーション EAR ファイルのトップレベルにある `xqs-resources.jar` ファイルの `xqs-config.xml` が参照されます。XQuery 式に対して宣言された各外部関数では、XQS により `xqs-config.xml` ファイルの関数の構成が参照されます。

XQS 構成ファイルの要素の階層を要約します。構成するそれぞれの XQS 関数について、関数がアクセスするデータソースの種類に応じて、`<document-source>`、`<wsdl-source>` または `<xqsview-source>` 要素を使用します。

```
<xqs-config>
   <bind-prefix>
      <use-prefix>
   <xqs-sources>
      <wsdl-source>
         <function-name>
         <cache-properties>
            <in-memory>
         <output-element>
         <error-message>
         <wsdlURL>
         <operation>
         <service>
         <portType>
         <port>
         <input-parameters>
            <part>
               <schema-type>
         <typeMap>
            <mapping>
               <xmlType>
      <xqsview-source>
         <function-name>
         <cache-properties>
            <in-memory>
         <output-element>
         <error-message>
         <repository>
         <queryName>
         <input-parameters>
            <part>
               <schema-type>
               <xquery-sequence>
                  <itemType>
         <dataSource>
      <document-source>
         <function-name>
         <cache-properties>
            <in-memory>
         <output-element>
         <error-message>
         <documentURL>
         <XMLTranslate>
            <schema-file>
```

<bind-prefix>

親要素： <xqs-config>

子要素： <use-prefix>

必須 / オプション オプション。使用数 : 0 または 1 つ。

XQS 構成要素内の特定の XML 名前空間を表す接頭辞を指定する場合は、<use-prefix> サブ要素とともにこの要素を使用します。1 つの <use-prefix> サブ要素によって、1 つの接頭辞が指定されます。

<bind-prefix> 要素には、属性はありません。

<cache-properties>

親要素： <document-source>、<wsdl-source> または <xqsview-source>

子要素： <in-memory>

必須 / オプション オプション。使用数 : 0 または 1。使用場所 : 出現する各親要素の内部。

<document-source>、<wsdl-source> または <xqsview-source> の isCached 属性を介して XQS キャッシングを有効化する場合は、<cache-properties> 要素およびその必須のサブ要素 <in-memory> を使用して、特定の XQS 関数で使用するキャッシング・プロパティを設定します。関連する考慮事項は、8-66 ページの「XQS キャッシングの対策」を参照してください。

isCached="false"（デフォルト）と設定されている場合、<cache-properties> 要素は無視されます。

表 8-4 <cache-properties> 属性

名前	説明
time-to-live	値 : 整数（秒単位） デフォルト : 該当なし（必須） 項目が期限切れになるまでの、キャッシュ内での保持期間を指定します。

<dataSource>

親要素： <xqsview-source>

子要素： なし

必須 / オプション オプション。使用数 : 0 または 1。使用場所 : 出現する各親要素の内部。

この要素の値を使用して、data-sources.xml 構成ファイルに定義されているデータソースの JNDI 名を指定します。指定されたデータソースは、次の例に示すように、XQS によるデータベースへの接続に使用されます。

```
<dataSource>jdbc/OracleCoreDS</dataSource>
```

<dataSource> 要素には、属性はありません。

<document-source>

親要素： <xqs-sources>

子要素： <cache-properties>、<documentURL>、<error-message>、<function-name>、<output-element>、<XMLTranslate>

必須 / オプション オプション。使用数 : 0 以上。

構成する XQS 関数のうち、ドキュメント・ソースを使用するそれぞれの関数に、<document-source> 要素および（必要に応じて）そのサブ要素を使用します。

実行時に URL を使用する汎用ドキュメント・ソースの使用を構成できます。もしくは、<documentURL> サブ要素を使用して、構成内で固定 URL を指定できます。（実行時に URL を使用する例は、8-22 ページの「ドキュメント・ソースにアクセスする XQS 関数の構成」を参照してください。）

XML 以外のドキュメントの場合は、そのドキュメントが D3L ツールでサポートされている書式に準拠している場合は、<XMLTranslate> 要素を使用して XML への変換を構成できます。（D3L の詳細は、8-15 ページの「XML 以外のドキュメント・ソースを使用するための準備」を参照してください。）

表 8-5 <document-source> 属性

名前	説明
isCached	値 : ブール デフォルト : false データソースから結果をキャッシュするには、true に設定します。<cache-properties> 要素の詳細は、8-65 ページの「XQS キャッシングの構成」も参照してください。
largeData	値 : ブール デフォルト : false 大量のデータを処理するために、メモリー節約内部最適化オプションを使用するように XQS に指示する場合は、true に設定します。（必要な場合のみ使用してください。）8-67 ページの「ラージ・データ用の XQS ドキュメント・ソースの構成」も参照してください。
onError	値 : dynamicError | emptySequence | errorMessage デフォルト : dynamicError XQS で使用されるエラー処理モードを決定します。8-68 ページの「XQS 関数のエラー処理の構成」を参照してください。

<documentURL>

親要素： <document-source>

子要素： なし

必須 / オプション オプション。使用数 : 0 または 1。使用場所 : 出現する各親要素の内部。

次の例のように、この要素の値を使用して、データソースとして使用するドキュメントを指定します。

```
<documentURL>http://host:port/xqsdemos/Repository/pos-2KB.xml</documentURL>
```

XML 以外のドキュメントを指定する場合は、<document-source> の <XMLTranslate> サブ要素を使用して、XQS で使用する変換ツールを指定します。

<documentURL> 要素には、属性はありません。

注意 :

- ファイアウォールを使用しており、指定した URL で外部インターネット・アクセスが要求される場合は、適切なプロキシ設定を使用して OC4J も構成する必要があります。
- <documentURL> 要素を介してドキュメント URL を事前に定義するかわりに、この要素を省略して（構成する XQS 関数に対応する）XQuery 外部関数を宣言し、ドキュメント URL を実行時引数として使用できます。
- ドキュメント URL を指定する際には、次のことに注意してください。ドキュメントがローカル・ファイル・システムにある場合は、http:// プロトコルを使用するよりも、file:// プロトコルを使用した方が、データの取得速度が速くなります。

<error-message>

親要素： <document-source>、<wsdl-source> または <xqsview-source>

子要素： なし

必須 / オプション オプション。使用数 : 0 または 1。使用場所 : 出現する各親要素の内部。

XQS errorMessage エラー・モード（<document-source>、<wsdl-source> または <xqsview-source> の onError 属性で指定）では、データソースから受け取ったメッセージをそのまま転送するかわりに固定エラー・メッセージを事前に定義する場合は、<error-message> 要素を使用します。次の例のように、要素値によってメッセージが構成されます。

```
<error-message>No information available</error-message>
```

8-13 ページの「XQS エラー処理の概要」も参照してください。

<error-message> 要素には、属性はありません。

<function-name>

親要素： <document-source>、<wsdl-source> または <xqsview-source>

子要素： なし

必須 / オプション 必須。使用数 : 1 つのみ。使用場所 : 出現する各親要素の内部。

この要素を使用して、データソースへのアクセス用に XQS によって実装される XQuery 関数の目的の名前を宣言します。要素値は関数名で、属性の設定によって関数の名前空間が指定されます。（それぞれの XQuery 関数は、いずれかの名前空間に属する必要があります。）XQuery 式のデフォルトとして特定の名前空間を宣言していないかぎり、XQuery 式内で関数を起動するたびに、関数の名前空間を参照する必要があります。

namespace 属性を使用して名前空間を直接指定します。すでに、<bind-prefix> 要素を介して名前空間の接頭辞を定義している場合は、prefix 属性をショートカットとして使用します。namespace および prefix を同時に使用することはできません。必ずどちらか一方のみを使用してください。<output-element> 要素の使用に関する情報は、8-28 ページの「<output-element> を使用する際の考慮事項」を参照してください。

ここで指定する関数名は、対応するデータソースにアクセスするために XQuery 外部関数宣言で宣言する関数名です。

次に例を示します。

```
<function-name  namespace="http://xmlns.oracle.com/ias/xqs">
   customerInfo
</function-name>
```

一方、xqs が XQS 名前空間の接頭辞として定義されている場合は、次のようになります。

```
<function-name prefix="xqs">customerInfo</function-name>
```

次に、前述の構成に対応する、XQuery 関数の宣言の例を示します。

```
declare namespace xqs="http://xmlns.oracle.com/ias/xqs";
declare function xqs:customerInfo() external;
```

表 8-6 <function-name> 属性

名前	説明
namespace	値 : URI デフォルト : なし（prefix が使用されていない場合は必須） 名前空間を直接指定します。
prefix	値 : 文字列 デフォルト : なし（namespace が使用されていない場合は必須） 事前定義済の接頭辞を介して名前空間を指定します。

<in-memory>

親要素： <cache-properties>

子要素： なし

必須 / オプション 親要素が使用されている場合は必須。使用数 : 1 つのみ。

<in-memory> 要素およびそのサブ要素 <in-memory> を使用して、XQS キャッシュのプロパティを設定します。<in-memory> 要素の属性により、キャッシング・モードが決定されます。単純なメモリーベースのキャッシングの場合は、useSpool と useDiskCache の両方を false に設定します。メモリー不足になると必要に応じてキャッシュ・データをディスクにスプールできるメモリーベースのキャッシングの場合は、useSpool を true に設定し、useDiskCache を false に設定します。ディスクベースのキャッシングの場合は、useDiskCache を true に設定し、useSpool を false に設定します。ディスクベースのキャッシングを使用すると、サーバーに障害が発生しても、キャッシュ・データは失われません。関連する考慮事項は、8-66 ページの「XQS キャッシングの対策」を参照してください。

必要に応じてスプーリングを使用する、メモリーベースのキャッシングの例を示します。

```
<cache-properties time-to-live="600">
   <in-memory useSpool="true" useDiskCache="false"/>
</cache-properties>
```

追加情報は、8-65 ページの「XQS キャッシングの構成」を参照してください。

表 8-7 <in-memory> 属性

名前	説明
useSpool	値 : ブール デフォルト : false スプーリングを有効化する場合は、メモリーベースのデータ・キャッシングに対して useSpool="true" を設定します。これにより、キャッシュ・データがメモリーからディスクにスプールされるため、メモリー空間の再利用のためにキャッシュ・オブジェクトが削除されても、データは失われません。
useDiskCache	値 : ブール デフォルト : false ディスクベースのデータ・キャッシングに対して、useDiskCache="true" を設定します。これにより、サーバーに障害が発生しても、キャッシュ・データがリカバリされます。

<input-parameters>

親要素： <wsdl-source> または <xqsview-source>

子要素： <part>

必須 / オプション 必須。使用数 : 1 つのみ。使用場所 : 出現する各親要素の内部。

この要素を使用して、関連するデータソースの XQS 関数への入力パラメータを指定します。各パラメータには、<part> サブ要素を使用します。タイプ・マッチングは、WSDL ソースについてのみ、type-match 属性に基づきます。

この要素は、入力パラメータがない場合でも必須です。その場合は、空の要素を使用します。

表 8-8 <input-parameter> 属性

名前	説明
type-match	値 : strict | none デフォルト : strict WSDL ソースについて、入力パラメータに対して XQS によるタイプ・マッチングが実行されるかどうかを決定します。type-match="strict"（デフォルト）の設定を使用すると、（<part> の <schema-type> サブ要素を介して）指定した入力タイプが、WSDL ドキュメント内の対応する部分のタイプと比較されます。現在サポートされている唯一のレベルである strict（厳格な）マッチングでは、タイプ名の完全一致が要求されます。実際の入力タイプが <schema-type> 要素でサポートされていない場合は、サポートされている中で最も近いタイプを選択し、type-match="none" の設定を使用します。（同様に、<schema-type> 要素を使用しない場合は、タイプ・マッチングを回避できます。）

<itemType>

親要素： <xquery-sequence>

子要素： なし

必須 / オプション オプション。使用数 : 0 または 1。使用場所 : 出現する各親要素の内部。

<xquery-sequence> 要素を使用して、順序のタイプを（順序の各メンバーのタイプが同じ）XQS ビューの入力タイプとして使用する場合は、<itemType> サブ要素を使用してメンバーのタイプを指定できます。XQS によるタイプ・チェックの実行を許可する場合に便利です。

namespace 属性を使用して、型を定義する名前空間を直接指定します。すでに、<bind-prefix> 要素を介して名前空間の接頭辞を定義している場合は、prefix 属性をショートカットとして使用します。namespace および prefix を同時に使用することはできません。必ずどちらか一方のみを使用してください。

表 8-9 <itemType> 属性

名前	説明
namespace	値 : URI デフォルト : なし（prefix が使用されていない場合は必須） 名前空間を直接指定します。

表 8-9 <itemType> 属性（続き）

名前	説明
prefix	値 : 文字列 デフォルト : なし（namespace が使用されていない場合は必須） 事前定義済の接頭辞を介して名前空間を指定します。
location	値 : URI デフォルト : なし ユーザー定義の型の場合は、その型を定義するスキーマがある場所の URI を指定できます。対応する、パラメータの詳細な宣言が XQuery プロローグに存在する場合は、XQS によるタイプ・チェックに役立ちます。

<mapping>

親要素 : <typeMap>

子要素 : <xmlType>

必須 / オプション 親要素が使用されている場合は必須。使用数 : 1 つ以上。使用場所 : 親要素の内部。

Java または EJB のバインディングを伴う WSDL ソースでは、<xmlType> サブ要素と併用され、Java 型と XML 型をマップします。typeClass 属性を使用して、Java 型を指定します。

表 8-10 <mapping> 属性

名前	説明
typeClass	値 : 文字列 デフォルト : 該当なし（必須） この属性によって、完全修飾された、XML-Java 型マッピングの Java クラス名が指定されます。このクラスは、OC4J クラス・パスに含める必要があります。

<operation>

親要素 : <wsdl-source>

子要素 : なし

必須 / オプション 必須。使用数 : 1 つのみ。使用場所 : 出現する各親要素の内部。

実行する Web サービス操作の指定に使用します。指定する操作は、<wsdlURL> 要素がポイントする WSDL ドキュメント内で定義された操作になります。定義には、WSDL に含まれる <operation> 要素の name 属性に対応する XQS <operation> 要素の値が使用されます。次に例を示します。

```
<operation>getCustomerByKey</operation>
```

<operation> 要素には、属性はありません。

<output-element>

親要素： <document-source>、<wsdl-source> または <xqsview-source>

子要素： なし

必須 / オプション オプション。使用数 : 0 または 1。使用場所 : 出現する各親要素の内部。

この要素の値は、XQS 関数の結果である順序の項目タイプを定義します。この要素は必須ではありませんが、この情報は、XQS によるタイプ・チェックおよび最適化に使用されます。この要素は、WSDLvisibility="true" と設定されている XQS ビュー・ソースに使用することを特にお薦めします。タイプ情報がなくても、XQS では有効な Web サービス操作が生成されますが、タイプ情報を持たない Web サービスを本番システムで定義することは好ましくありません。

結果項目の定義方法は 2 つあります。型による定義、および前に別のスキーマで定義された要素の参照による定義です。このようなインポート済要素の名前は、要素 <output-element> のテキスト値として渡されます。たとえば、結果順序内の項目が、スキーマ urn:PurchaseOrders 名前空間の <po> 要素になる場合、<output-element> は次のようになります。

```
<output-element namespace="urn:purchaseOrders" >po</output-element>
```

結果順序内の項目が、http://myHost:/mySchemas/PurchaseOrders.xsd にあるスキーマ urn:PurchaseOrders 名前空間で定義されたタイプ POType の要素になる場合、<output-element> は次のようになります。

```
<output-element prefix="po_ns"
   location="http://myHost:/mySchemas/PurchaseOrders.xsd"
   type="POType"/>
```

namespace 属性を使用して名前空間を直接指定します。すでに、<bind-prefix> 要素を介して名前空間の接頭辞を定義している場合は、prefix 属性をショートカットとして使用します。namespace および prefix を同時に使用することはできません。必ずどちらか一方のみを使用してください。

次に、複合型の例を示します。

```
<output-element namespace="http://xmlns.oracle.com/ias/xqs/CustomerDemo"
   location="http://host:port/xqsdemos/Customers.xsd"
   type="CustomersType">
   customers
</output-element>
```

次に、単純型の例を示します。

```
<output-element prefix="xs">float</output-element>
```

表 8-11 <output-element> 属性

名前	説明
namespace	値 : URI デフォルト : なし（prefix が使用されていない場合は必須） 名前空間を直接指定します。
prefix	値 : 文字列 デフォルト : なし（namespace が使用されていない場合は必須） 事前定義済の接頭辞を介して名前空間を指定します。

表 8-11 <output-element> 属性（続き）

名前	説明
location	値 : URI デフォルト : なし 要素または型を定義するスキーマの場所の URI を指定します。WSDLvisibility="true" と設定されている <xqsview-source> 要素内の <output-element> 要素を使用する場合は、この属性は必須です。それ以外の場合は必須ではありませんが、対応する、パラメータの詳細な宣言が XQuery プロローグに存在する場合は、XQS によるタイプ・チェックに役立ちます。
type	値 : 文字列 デフォルト : なし XQS 関数の結果である順序の項目タイプになる、XML 型のローカル名を指定します。タイプ名の名前空間は、namespace 属性または prefix 属性によって決まります。結果項目のタイプが、<output-element> 要素の次の値として指定された要素名を参照して定義されている場合は、type 属性は必須ではありません。

<part>

親要素 : <input-parameters>

子要素 : <schema-type>、<xquery-sequence>

必須 / オプション 親要素が空の場合は必須。使用数 : 入力引数ごとに 1 つ。

WSDL ソースまたは XQS ビューについて、入力パラメータごとに 1 つの <part> 要素を使用します。属性を介して、パラメータの名前および位置（順序）を指定します。必要に応じて <schema-type> サブ要素または <xquery-sequence> サブ要素（XQS ビューのみ）を使用して、タイプを指定します。タイプの指定は必須ではありませんが、指定すると、XQS によるタイプ・チェックを実行できます。

WSDL ソースの場合は、ソースへのアクセスに使用する WSDL 操作の入力メッセージの一部分に、各 <part> 要素が対応している必要があります。XQS の場合は、各 <part> 要素が、ビューを定義する XQuery の外部変数に対応しているか、ビューを定義する SQL 問合せのバインド変数に対応している必要があります。

表 8-12 <part> 属性

名前	説明
name	値 : 文字列 デフォルト : 該当なし（必須） 入力パラメータの名前を指定します。WSDL ソースの場合は、WSDL ドキュメントに含まれているものと同じ名前にする必要があります。XQS ビューの場合は、外部 XQuery 変数の名前または SQL バインド変数と同じにする必要があります。
position	値 : 正の整数 デフォルト : 該当なし（必須） <input-parameters> 要素に含まれるすべての <part> 要素の位置属性を使用して、メッセージ部分に割り当てられる入力パラメータ（WSDL ソースの場合）や、外部またはバインド変数に割り当てられる入力パラメータ（XQS ビューの場合）の順序が決定されます。position="1" で始まる、最も低い位置が設定されているパラメータから順に使用されます。各 <part> 要素には、一意の位置設定を指定する必要があります。

<port>

親要素： <wsdl-source>

子要素： なし

必須 / オプション 必須。使用数 : 1 つのみ。使用場所 : 出現する各親要素の内部。

この要素の値は、<wsdlURL> 要素がポイントする WSDL ドキュメント内で定義された Web サービス・ポートの名前を指定します。この値は、WSDL 内の <port> 要素の name 属性に対応します。

属性の設定によって名前空間が指定されますが、適用可能なサービスに 1 つしかポートがない場合は、名前空間を指定する必要はありません。

namespace 属性を使用して名前空間を直接指定します。すでに、<bind-prefix> 要素を介して名前空間の接頭辞を定義している場合は、prefix 属性をショートカットとして使用します。namespace および prefix を同時に使用することはできません。

次に例を示します。

```
<port namespace="http://customer.myeis.com/">CustomerInfo</port>
```

別の方法の例を示します。

```
<bind-prefix>
   <use-prefix prefix="myeis">http://customer.myeis.com/</use-prefix>
</bind-prefix>
...
<port prefix="myeis">CustomerInfo</port>
```

表 8-13 <port> 属性

名前	説明
namespace	値 : URI デフォルト : なし 名前空間を直接指定します。
prefix	値 : 文字列 デフォルト : なし 事前定義済の接頭辞を介して名前空間を指定します。

<portType>

親要素： <wsdl-source>

子要素： なし

必須 / オプション 必須の場合あり。使用数 : 0 または 1。使用場所 : 出現する各親要素の内部。

この要素の値は、<wsdlURL> 要素がポイントする WSDL ドキュメント内で定義された Web サービス・ポート・タイプの名前を指定します。この値は、WSDL 内の <portType> 要素の name 属性に対応します。この要素は、操作に使用するポートのバインディングが WSDL に複数含まれている場合には必須です。

属性の設定により、名前空間が指定されますが、WSDL に含まれるすべてのポート・タイプが同じ名前空間に属する場合は、名前空間を指定する必要はありません。

namespace 属性を使用して名前空間を直接指定します。すでに、<bind-prefix> 要素を介して名前空間の接頭辞を定義している場合は、prefix 属性をショートカットとして使用します。namespace および prefix を同時に使用することはできません。

次に例を示します。

```
<portType namespace="http://customer.myeis.com/">CustomerInfoType</portType>
```

表 8-14 <portType> 属性

名前	説明
namespace	値 : URI デフォルト : なし 名前空間を直接指定します。
prefix	値 : 文字列 デフォルト : なし 事前定義済の接頭辞を介して名前空間を指定します。

<queryName>

親要素： <xqsview-source>

子要素： なし

必須 / オプション オプション。使用数 : 0 または 1。使用場所 : 出現する各親要素の内部。

次の例で示すように、この要素の値を使用して、XQS ビューが定義されている XQuery 式のテキスト・ファイル（.xq または .sql ファイル）の名前を指定します。

```
<queryName>UserByOrderNum.xq</queryName>
```

ファイル名の拡張子 .xq または .sql は、XQS によって自動的に推定されるため、指定しなくてもかまいません。XQS がファイルを検索するディレクトリは、<xqsview-source> の <repository> サブ要素（あるいはデフォルトの場所）に基づきます。

<queryName> 要素を使用しない場合、XQS では、XQS ビュー・ファイルの名前は、<xqsview-source> の <function-name> サブ要素で指定した関数名と同じで、ビューのタイプに応じて拡張子は .xq または .sql であるとみなされます。

<queryName> 要素には、属性はありません。

<repository>

親要素： <xqsview-source>

子要素： なし

必須 / オプション オプション。使用数 : 0 または 1。使用場所 : 出現する各親要素の内部。

この要素の値は、XQS ビューが定義されている XQuery 式のテキスト・ファイル（.xq ファイル）または SQL 問合せファイル（.sql ファイル）を含むディレクトリへの絶対パスです。<xqsview-source> の <queryName> サブ要素で名前を指定します。

<repository> 要素を使用しない場合は、OC4JPackager の -repository オプションを介して指定したディレクトリ内が XQS によって検索されます。XQS のデモ・アプリケーションのデフォルト・ディレクトリを明示的に指定する場合の例を示します。

```
<repository>/META-INF/xqs/mydir/</repository>
```

<repository> 要素には、属性はありません。

<schema-file>

親要素： <XMLTranslate>

子要素： なし

必須 / オプション 親要素が使用されている場合は必須。使用数 : 1 つのみ。

ドキュメント・ソースが、XML 以外の、D3L ツールによってサポートされている書式である場合は、<XMLTranslate> 要素および <schema-file> サブ要素を使用して、そのドキュメントを XML に変換するための情報を XQS に渡します。（D3L の詳細は、8-15 ページの「XML 以外のドキュメント・ソースを使用するための準備」を参照してください。）

<XMLTranslate> 要素は、使用する変換ツール（D3L）を指定し、<schema-file> 要素は、変換に使用するスキーマ・ファイルを指定します。

次に例を示します。

```
<documentURL>http://host:port/xqsdemos/paymentInfo.csv</documentURL>
<XMLTranslate method="D3L">
   <schema-file>http://host:port/xqsdemos/paymentInfoD3L.xml</schema-file>
</XMLTranslate>
```

<schema-file> 要素には、属性はありません。

<schema-type>

親要素： <part>

子要素： なし

必須 / オプション 説明を参照。使用数 : 1 つのみ。

XQS ビューの各入力パラメータについて、この要素または <xquery-sequence> 要素（入力順序の場合）を使用して、パラメータ・タイプを指定する必要があります。

WSDL ソースの各入力パラメータでは、この要素を <part> 親要素内で使用することはオプションですが、この要素は、タイプ・チェックに役立つほか、すべてのタイプのリストを XQS 構成ファイルに含めるという編成上の目的にも役立ちます。（WSDL ソースでは、入力順序は適用されません。）

WSDL ソースでタイプ・チェックを実行するには、<input-parameters> 要素の属性が type-match="strict" と設定されている必要があります。

<schema-type> では、要素値によってタイプが指定され、属性設定によって名前空間が指定されます。namespace 属性を使用して名前空間を直接指定します。すでに、<bind-prefix> 要素を介して名前空間の接頭辞を定義している場合は、prefix 属性をショートカットとして使用します。namespace および prefix を同時に使用することはできません。必ずどちらか一方のみを使用してください。

次に例を示します。

```
<input-parameters>
  <part position="1" name="empname">
    <schema-type namespace="http://www.w3.org/2001/XMLSchema">string</schema-type>
  </part>
</input-parameters>
```

別の方法として、xs を XML Schema の名前空間として定義する場合の例を示します。

```
<bind-prefix>
   <use-prefix prefix="xs">http://www.w3.org/2001/XMLSchema</use-prefix>
</bind-prefix>
...
<input-parameters>
  <part position="1" name="empname">
    <schema-type prefix="xs">string</schema-type>
  </part>
</input-parameters>
```

ユーザー定義の型の場合は、次のようになります。

```
<input-parameters>
   <part position="1" name="DBServiceSelect_cust_id_inparameters">
      <schema-type namespace=
         "http://xmlns.oracle.com/pcbpel/adapter/db/top/TestDBAdapter">
         DBServiceSelect_cust_id
      </schema-type>
   </part>
</input-parameters>
```

ユーザー定義の XML 型に加えて、現在、XQS では、XQuery 1.0 および XPath 2.0 データ・モデルで定義されている型のサブセット、すなわち xs:boolean、xs:string、xs:int、xs:long、xs:float、xs:double、xs:decimal、xs:base64Binary、xs:hexBinary、xs:anyURI、xs:dateTime、xs:duration および xs:anyType をサポートしています。（8-38 ページの「サポートされている問合せパラメータのタイプ」も参照してください。）サポート対象外の XQuery/XPath 入力タイプを使用する場合は、サポートされている中で、データに最も適した Java 表現のタイプを選択し、<input-parameters> 要素内で type-match="none" の設定を使用します。（Java または EJB のバインディングを伴う WSDL ソースの場合、このような状況では、<wsdl-source> の <typeMap> サブ要素も使用します。）

表 8-15 <schema-type> 属性

名前	説明
namespace	値 : URI デフォルト : なし（prefix が使用されていない場合は必須） 名前空間を直接指定します。
prefix	値 : 文字列 デフォルト : なし（namespace が使用されていない場合は必須） 事前定義済の接頭辞を介して名前空間を指定します。

<service>

親要素 : <wsdl-source>

子要素 : なし

必須 / オプション 必須の場合あり。使用数 : 0 または 1。使用場所 : 出現する各親要素の内部。

この要素の値は、サービスの名前を指定します。指定するサービスは、<wsdlURL> 要素がポイントする WSDL ドキュメント内で定義されたサービスになります。定義には、WSDL に含まれる <service> 要素の name 属性に対応する XQS <service> 要素の値が使用されます。XQS <service> 要素は、WSDL ドキュメントで複数のサービスが定義されている場合には必須です。

属性の設定により、名前空間が指定されますが、WSDL に含まれるすべてのサービス要素が同じ名前空間に属する場合は、名前空間を指定する必要はありません。

namespace 属性を使用して名前空間を直接指定します。すでに、<bind-prefix> 要素を介して名前空間の接頭辞を定義している場合は、prefix 属性をショートカットとして使用します。namespace および prefix を同時に使用することはできません。

次に例を示します。

```
<service namespace="http://customer.myeis.com/">CustomerInfoMYEISService</service>
```

表 8-16 <service> 属性

名前	説明
namespace	値 : URI デフォルト : なし 名前空間を直接指定します。
prefix	値 : 文字列 デフォルト : なし 事前定義済の接頭辞を介して名前空間を指定します。

<typeMap>

親要素： <wsdl-source>

子要素： <mapping>

必須 / オプション オプション。使用数 : 0 または 1。使用場所 : 出現する各親要素の内部。

Java または EJB のバインディングを伴う WSDL ソースでは、必要に応じてこの要素を使用し、XML 型を Java 型にマップできます。Java 型は、<mapping> サブ要素を介して指定し、XML 型は、<mapping> の <xmlType> サブ要素を介して指定します。

<typeMap> 要素には、属性はありません。

<use-prefix>

親要素： <bind-prefix>

子要素： なし

必須 / オプション 親要素が使用されている場合は必須。使用数 : 1 つ以上。

特定の XML 名前空間を表す接頭辞を使用する場合は、<bind-prefix> 要素および <use-prefix> サブ要素を使用します。各 <use-prefix> サブ要素は、1 つの接頭辞を指定します。使用される要素値は、名前空間を表す接頭辞を指定する、名前空間および接頭辞の属性を指定します。次に例を示します。

```
<bind-prefix>
   <use-prefix prefix="xs">http://www.w3.org/2001/XMLSchema</use-prefix>
   <use-prefix prefix="xqs">http://xmlns.oracle.com/ias/xqs</use-prefix>
   <use-prefix prefix="myeis">http://customer.myeis.com/</use-prefix>
</bind-prefix>
```

これらの設定により、名前空間を指定する必要がある XQS 構成要素では、たとえば、namespace="http://customer.myeis.com/" のかわりに prefix="myeis" を使用できます。

> **注意：** <use-prefix> の指定は、XQuery 式ではなく、この指定が出現する XQS 構成ファイルのみで有効です。

表 8-17 <use-prefix> 属性

名前	説明
prefix	値 : 文字列 デフォルト : 該当なし（必須） この属性は、目的の接頭辞を指定します。

<username>

親要素： <document-source>

子要素： なし

必須 / オプション オプション。使用数 : 0 または 1 つ。

この要素は、将来のセキュリティ拡張のために予約されています。

<username> 要素には、属性はありません。

<wsdl-source>

親要素： <xqs-sources>

子要素： <cache-properties>、<error-message>、<function-name>、<input-parameters>、<operation>、<output-element>、<port>、<portType>、<service>、<typeMap>、<wsdlURL>

必須 / オプション オプション。使用数 : 0 以上。

構成する XQS 関数のうち、WSDL ベースのソースにアクセスするそれぞれの関数に、<wsdl-source> 要素および（必要に応じて）そのサブ要素を使用します。（XQS で使用できる WSDL ベースのソースの種類は、8-8 ページの「データソースのサポート対象カテゴリ」を参照してください。）

表 8-18 <wsdl-source> 属性

名前	説明
isCached	値 : ブール デフォルト : false データソースから結果をキャッシュするには、true に設定します。<cache-properties> 要素の詳細は、8-65 ページの「XQS キャッシングの構成」も参照してください。
largeData	値 : ブール デフォルト : false 大量のデータを処理するために、メモリー節約内部最適化オプションを使用するように XQS に指示する場合は、true に設定します。（必要な場合のみ使用してください。）8-67 ページの「ラージ・データ用の XQS ドキュメント・ソースの構成」も参照してください。
onError	値 : dynamicError | emptySequence | errorMessage デフォルト : dynamicError XQS で使用されるエラー処理モードを決定します。8-13 ページの「XQS エラー処理の概要」を参照してください。

<wsdlURL>

親要素： <wsdl-source>

子要素： なし

必須 / オプション 必須。使用数 : 1 つのみ。使用場所 : 出現する各親要素の内部。

実行する操作を定義する WSDL ドキュメントをポイントするために使用します。WSDL へのアクセスは、実行時行われます。次に例を示します。

```
<wsdlURL>http://api.mySearch.com/mySearch.wsdl</wsdlURL>
```

<wsdlURL> 要素には、属性はありません。

> **注意：** ファイアウォールを使用しており、指定した URL で外部インターネット・アクセスが要求される場合は、適切なプロキシ設定を使用して OC4J も構成する必要があります。

<xqs-config>

親要素： 該当なし（ルート）

子要素： <bind-prefix>、<xqs-sources>

必須 / オプション 必須。使用数 : 1 つのみ。

これは、xqs-config.xml のルート要素です。最低限必要な要素は、このルート要素、そのサブ要素 <xqs-sources>、および <xqs-sources> のサブ要素（<document-source>、<wsdl-source>、<xqsview-source>）のうち少なくとも 1 つです。

表 8-19 <xqs-config> 属性

名前	説明
version	値 : XML の名前トークン（NMTOKEN） デフォルト : 1.0（現在のバージョン） この属性は、xqs-config.xml スキーマ定義のバージョン番号を記録します。現行の XQS 実装では、デフォルト値を使用します。

<xqs-sources>

親要素： <xqs-config>

子要素： <document-source>、<wsdl-source>、<xqsview-source>

必須 / オプション 必須。使用数 : 1 つのみ。

これは、構成される全 XQS 関数のソース要素の親要素です。属性はありません。

<xqsview-source>

親要素： <xqs-sources>

子要素： <cache-properties>、<error-message>、<function-name>、<input-parameters>、<output-element>、<queryName>、<repository>、<dataSource>

必須 / オプション オプション。使用数 : 0 以上。

構成する XQS 関数のうち、XQS ビューを使用するそれぞれの関数に、`<xqsview-source>` 要素および（必要に応じて）そのサブ要素を使用します。

表 8-20 <xqsview-source> 属性

名前	説明
fetchSize	値 : 整数 デフォルト : 1 この属性は、リレーショナル・データベースに接続する SQL ベースの XQS ビューにのみ有効です。この属性はオプションです。 この `fetchSize` 属性は、`java.sql.PreparedStatement` の `setFetchSize` メソッドへのコールに変換されます。この属性は、JDBC へのヒントとして機能します。このヒントを使用して、1 回の処理でデータベースからフェッチされる結果セットの行数が定義されます。JDBC の `setFetchSize` パラメータは（つまり、XQS の `fetchSize` 属性も）単なるヒントであり、バインディングではないことに注意してください。
isCached	値 : ブール デフォルト : false データソースから結果をキャッシュするには、`true` に設定します。`<cache-properties>` 要素の詳細は、8-65 ページの「XQS キャッシングの構成」も参照してください。
largeData	値 : ブール デフォルト : false 大量のデータを処理するために、メモリー節約内部最適化オプションを使用するように XQS に指示する場合は、`true` に設定します。（必要な場合のみ使用してください。）8-67 ページの「ラージ・データ用の XQS ドキュメント・ソースの構成」も参照してください。
onError	値 : dynamicError | emptySequence | errorMessage デフォルト : dynamicError XQS で使用されるエラー処理モードを決定します。8-13 ページの「XQS エラー処理の概要」を参照してください。
WSDLvisibility	値 : ブール デフォルト : false アプリケーションの Web サービス・コンポーネントで、XQS を Web サービス操作として公開する場合は、これを `true` に設定します。この属性は、SQL ベースの XQS ビューには適用できません。8-56 ページの「Web サービス操作として公開された XQS ビューの使用方法」も参照してください。

表 8-20 <xqsview-source> 属性（続き）

名前	説明
SQLResultType	値 : relational | XMLType デフォルト : relational この属性は、SQL ベースのビューを使用している場合にのみ適用可能です。この属性では、データベース問合せで、データをリレーショナル（表）形式または XML のいずれで戻すかを設定します。Oracle XML データベースを使用している場合は、`SQLResultType` 属性を `XMLType` に設定することで、SQL ベースの XQS ビューを使用して（それ以上変換せずに）データベースから XML 型の値を選択できます。8-24 ページの「XQS ビューを使用する XQS 関数の構成」も参照してください。
rowTag	値 : 文字列 デフォルト : ROW この属性は、SQL ベースのビューを使用している場合にのみ適用可能です。この属性が指定されている場合、データベースの結果のリレーショナルから XML への変換で使用された要素のデフォルト名は、この値で上書きされます。この属性は、`SQLResultType` 属性が `relational` に設定されている場合にのみ適用可能です。 8-24 ページの「XQS ビューを使用する XQS 関数の構成」も参照してください。
rowIdAttr	値 : 文字列 デフォルト : NULL この属性は、SQL ベースのビューを使用している場合にのみ適用可能です。この属性が指定されている場合、結果に含まれる行要素の番号付けが制御されます。この属性は、`SQLResultType` 属性が `relational` に設定されている場合にのみ適用可能です。 `rowIdAttr` 属性の値は、結果の行要素の番号付け属性の名前です。行は 1 から開始して、`1`、`2`、`3` のように番号付けされます。`""` の設定はデフォルトの動作と同じで、行の番号付け属性を抑止します。 `rowTag` および `rowIdAttr` 属性は、それぞれ別々に使用されます。つまり、指定できる属性は 1 つのみで、もう一方にはデフォルトを使用します。 8-24 ページの「XQS ビューを使用する XQS 関数の構成」も参照してください。
XMLFormat	値 : XSU デフォルト : XSU この属性は、SQL ベースのビューを使用している場合にのみ適用可能です。この属性の値は `XSU` のみです。その他の値は今後のリリースで使用できるようになり、WebRowset 形式の結果がサポートされます。現行リリースの制限は、JDK 1.5 機能への依存性が原因です。

<XMLTranslate>

親要素： <document-source>

子要素： <schema-file>

必須 / オプション オプション。使用数 : 0 または 1。使用場所 : 出現する各親要素の内部。

ドキュメント・ソースが、XML 以外の、D3L ツールによってサポートされている書式である場合は、<XMLTranslate> 要素および <schema-file> サブ要素を使用して、そのドキュメントを XML に変換するための情報を XQS に渡します。この要素は、Excel のカンマ区切り（CSV）ファイルなど、別の書式のファイルを XML に変換する場合に便利です。D3L の詳細は、8-15 ページの「XML 以外のドキュメント・ソースを使用するための準備」を参照してください。

<XMLTranslate> の method 属性は、使用する変換ツールを指定します。現在、XQS でサポートされているのは、D3L 変換ツールです。

<schema-file> サブ要素は必須で、変換に使用するスキーマを指定します。

表 8-21 <XMLTranslate> 属性

名前	説明
method	値 : D3L デフォルト : 該当なし（必須） XQS で使用する変換ツールを指定します。XQS で現在サポートされているのは、D3L です。

<xmlType>

親要素： <mapping>

子要素： なし

必須 / オプション 必須。使用数 : 1 つのみ。使用場所 : 出現する各親要素の内部。

Java または EJB のバインディングを伴う WSDL では、<mapping> 親要素と併用され、Java 型と XML 型をマップします。属性設定を使用して XML 型の名前空間を指定します。名前空間を直接指定する場合は、namespace 属性を使用します。すでに、<bind-prefix> 要素を介して名前空間の接頭辞を定義している場合は、prefix 属性をショートカットとして使用します。namespace および prefix を同時に使用することはできません。必ずどちらか一方のみを使用してください。

<mapping> 親要素は、Java 型を指定します。

次に例を示します。

```
<typeMap>
   <mapping typeClass="org.w3c.dom.Node">
      <xmlType prefix="myeis">Customer</xmlType>
   </mapping>
   ...
</typeMap>
```

表 8-22 <xmlType> 属性

名前	説明
namespace	値 : URI デフォルト : なし（prefix が使用されていない場合は必須） 名前空間を直接指定します。

表 8-22 <xmlType> 属性（続き）

名前	説明
prefix	値 : 文字列 デフォルト : なし（namespace が使用されていない場合は必須） 事前定義済の接頭辞を介して名前空間を指定します。

<xquery-sequence>

親要素 : <part>

子要素 : <itemType>

必須 / オプション 親要素または XQS ビューが出現するたびに、その内部でこの要素または <schema-type> を使用する必要があります。使用数 : 1 つのみ。

XQS ビューの、パラメータ・タイプが XQuery/XPath 順序の入力パラメータごとに、<part> 親要素内でこの要素を使用して、タイプを指定する必要があります（これに対して、順序以外のタイプには、<schema-type> 要素が使用されます）。

WSDL ソースでは、入力順序がサポートされていないため、この要素は関係ありません。

<xquery-sequence> 要素は、複数のタイプの項目を含む異機種間の順序に使用できます。その場合は、この要素を空にする必要があります。もしくは、<itemType> サブ要素を含めて、異機種間順序の各メンバーの共通タイプを指定できます。

<xquery-sequence> 要素には、属性はありません。

OC4JPackager のリファレンス

この項では、XQS のパッケージ化ユーティリティである OC4JPackager のパラメータおよびプロパティに関するリファレンス・ドキュメントを提供します。概要は、8-12 ページの「OC4JPackager の概要」を参照してください。使用方法は、8-57 ページの「OC4JPackager を使用した XQS アプリケーションのパッケージ方法」を参照してください。

注意 : パラメータまたはプロパティの設定でパスを指定する際に使用できるように、OC4JPackager では、現在のディレクトリへの相対パス（ディレクトリ表記の「.」および「..」の使用を含む）または絶対パスがサポートされています。

OC4JPackager のパラメータ

この項では、アルファベット順に、OC4JPackager のパラメータの説明および使用例を示します。OC4JPackager は、次のように実行します（% はコマンド・プロンプトです）。

```
% java -jar OC4JPackager.jar -option1 value1 -option2 value2 ... -optionN valueN
```

appArchives

必須。ディレクトリ・パスを指定します。

アプリケーションの場所を指定するために使用します。アプリケーションの場所は、EAR ファイルが含まれているディレクトリ、または EAR 構造のトップレベル・ディレクトリです。

```
-appArchives /dir1/myappdir
```

help

オプション。これは、切替えフラグです。

OC4JPackager の使用状況およびパラメータ・リストを示すために使用します。その際、その他のパラメータは使用しません。

```
-help
```

jsp

オプション（XQS JSP タグ・ライブラリを使用する場合は必須）。これは、切替えフラグです。

XQS JSP タグ・ライブラリの JAR ファイル（xquerytaglib.jar）を出力 EAR ファイルに含めるために使用します。

```
-jsp
```

name

必須。EAR ファイルの名前を指定します。

出力 EAR ファイルの目的の名前（.ear 拡張子の有無は任意）を指定するために使用します。このファイルの場所は、-output パラメータによって決まります。

```
-name myxqsapp.ear
```

no_ws

オプション。これは、切替えフラグです。

Web サービスを生成しないようにする場合に使用します。

```
-no_ws
```

output

必須。ディレクトリ・パスを指定します。

OC4JPackager によって作成される EAR ファイルの配置場所を指定するために使用します。ファイル名は、-name パラメータに基づきます。

```
-output /dir1/mydeployments
```

repository

オプション。ディレクトリ・パスを指定します。

XQS ビューが格納される XQS リポジトリの場所を指定するために使用します。

```
-repository ORACLE_HOME/xds/samples/repository
```

sf

オプション（XQS ステートフル EJB クライアント API を使用する場合は必須）。これは、切替えフラグです。

XQS ステートフル EJB クライアントの JAR ファイル（xqsclient-stateful.jar）を出力 EAR ファイルに含めるために使用します。

```
-sf
```

また、これにより、次の <module> 要素が application.xml ファイルに追加されます。

```
<module>
   <ejb>xqsclient-stateful.jar</ejb>
</module>
```

sl

オプション（XQS ステートレス EJB クライアント API を使用する場合は必須）。これは、切替えフラグです。

XQS ステートレス EJB クライアントの JAR ファイル（xqsclient-stateless.jar）を出力 EAR ファイルに含めるために使用します。

```
-sl
```

また、これにより、次の <module> 要素が application.xml ファイルに追加されます。

```
<module>
   <ejb>xqsclient-stateless.jar</ejb>
</module>
```

xqsConfigFile

必須。ファイルのパスを指定します。

アプリケーションに対して、XQS 構成ファイル xqs-config.xml のパス（ファイル名を含む）を指定するために使用します。

```
-xqsConfigFile /dir1/subdir/myconfigdir/xqs-config.xml
```

OC4JPackager 用の Java プロパティ

この項では、アルファベット順に、OC4JPackager でサポートされている Java プロパティの説明および使用例を示します。Java -D オプションは、次のように使用します。

```
java -Dproperty1=value1 -Dproperty2=value2 -jar OC4JPackager.jar \
     -option1 value1 -option2 value2 ... -optionN valueN
```

oracle.home

オプション。ディレクトリ・パスを指定します。

OC4JPackager では、Oracle ホーム・ディレクトリを指定するための oracle.home がサポートされています。次に例を示します。

```
-Doracle.home=/myroot/myoraclehome
```

Oracle ホーム・ディレクトリを指定すると、-jsp、-sf および -sl の各フラグを介して要求されたクライアント JAR ファイルの検索、およびこれらのファイルのアプリケーションへのバンドルが可能になります。oracle.home を設定しない場合は、これらのフラグが無視されます。

java.home

必須。ディレクトリ・パスを指定します。

OC4JPackager が作業（JAR ファイルのバンドル解除やバンドルなど）のために java コマンドを実行する、Java ホーム・ディレクトリへのパスを指定します。

```
-Djava.home=/dir1/myjavahome
```

このプロパティを指定しない場合、OC4JPackager は終了します。

java.util.logging.properties.file

オプション。ファイルのパスを指定します。

Java ロギング・プロパティを指定するファイルへのパス（ファイル名を含む）を示すために使用します。次の例では、myfile.properties が現在のディレクトリにあります。

```
-Djava.util.logging.properties.file=myfile.properties
```

ロギング・プロパティ・ファイルを指定しない場合は、Java エラーの出力を確認できません。

OC4JPackager では、次の場所で示されるように、標準の J2SE ロギング・フレームワークを使用して、メッセージがログに記録されます。

```
http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/
```

OC4JPackager により、プロパティ・ファイル内では、OC4JPackager からのメッセージのロギング・レベルを指定できるプロパティ oracle.xds.tools.OC4JPackager.level がサポートされています。指定できるレベルは、SEVERE、WARNING、INFO または FINE です。高レベルの進捗メッセージには INFO、低レベルのデバッグ・メッセージには FINE を使用します。

メッセージをコンソールに送り、OC4JPackager からのメッセージに INFO レベルを使用する場合の、ロギング・プロパティ・ファイルのエントリの例を示します。

```
#  output to console
handlers= java.util.logging.ConsoleHandler
...
# set packager level to INFO
oracle.xds.tools.OC4JPackager.level=INFO
```

xds.packager.work.dir

オプション。ディレクトリ・パスを指定します。

これを使用して、EAR ファイルのバンドル解除および再作成の際に OC4JPackager が使用する、作業ディレクトリを指定します。

```
-Dxds.packager.work.dir=/some/dir
```

デフォルトの場所は、user.home ディレクトリです。

XQS のトラブルシューティング

この項では、OC4J ロギングを有効化する方法を説明し、XQS アプリケーションに関する様々なトラブルシューティングのヒントを提供します。

OC4J ロギングの有効化

Java プロパティ `java.util.logging.properties.file` を使用して、OC4J ロギングを有効化し、ロギング・プロパティ・ファイルを指定できます。このファイルでは、OC4JPackager のロギング・レベルを含む、Java ロギング設定を指定できます。ロギング・プロパティは、次の場所で指定されているように、標準の J2SE ロギングによって定義されます。

```
http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/
```

ロギングの詳細は、8-118 ページの「java.util.logging.properties.file」、『Oracle Containers for J2EE 構成および管理ガイド』の「OC4J でのロギング」の章、および『Oracle Containers for J2EE 開発者ガイド』の「ロギング実装のガイドライン」を参照してください。

XQS の主な兆候、原因および処置

この項では、XQS アプリケーションで発生する潜在的な問題およびその解決方法を説明します。

注意： XQS 実装には、ダイナミック・モニタリング・サービス（DMS）のメトリックは含まれていません。

大量のデータによる問題　サイズの大きい XML 文書をメモリーにロードできない場合は、`largeData` フラグを有効化すると処理が可能になることがあります。8-67 ページの「ラージ・データ用の XQS ドキュメント・ソースの構成」を参照してください。

XQS 関数が正常にロードされない　次のような内容のエラー・メッセージが表示されます（ここで、arity は関数が取る引数の数を表し、*ns* および *funcname* は関数の名前空間および名前に置き換えられます）。

```
XP0017: It is a static error if the expanded QName and number of arguments in a
function call do not match the name and arity of an in-scope function in the static
context. Detail: unknown function 'ns:funcname'
```

重要なのは、XQuery によって関数が認識されなかったということです。これは、関数が正しく構成されていなかったために、関数が XQS によって正常にロードされなかった可能性があることを示しています。OC4J エラー・ログ・ファイル（*ORACLE_HOME*`/j2ee/home/log/oc4j/log.xml` など）で、この関数で発生した問題の詳細な説明を検索してください。

外部関数に一貫性のないシグネチャが存在する　次のようなエラーが発生したとします。

```
external function "Foo" has inconsistent signatures: the one defined in function
library framework is different from the one declared in function declaration
prolog
```

XQS との関係において、function library framework（関数ライブラリ・フレームワーク）という用語は、XQS 構成ファイルの `<input-parameters>` 要素内で定義されたシグネチャを表すことに注意してください。`<input-parameters>` 要素の下にある `<part>` 定義は、必ず XQuery プロローグ内の関数のシグネチャと一致する必要があります。

XQuery シグネチャを構成と一致させることができない場合は、使用する XML 型がサポートされていない可能性があります。（8-38 ページの「サポートされている問合せパラメータのタイプ」を参照してください。）対応策として、XQuery パラメータまたは XQuery の戻り値について、プロローグでの型の宣言をすべて回避できます。たとえば、次のように完全に型が指定された関数の宣言があるとします。

```
declare function xqs:Foo ($s as element()) as xs:positiveInteger external;
```

型を指定せずにこの関数を宣言するには、次のようにします。

```
declare function xqs:Foo ($s) external;
```

ただし、XQS 構成を介して、入力パラメータの数を常に指定する必要があります。

XQS の例

XQS の例は、Oracle Technology Network の次の Web サイトで、XQS を検索してください。

```
http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html
```

9

アプリケーション・クライアント・コンテナの使用方法

この章では、アプリケーション・クライアント開発の基本的な要件、およびOracleアプリケーション・クライアント・コンテナの使用方法を説明します。この章には、次の項目が含まれます。

- コンテナの概要
- 要件
- アプリケーション・クライアントの開発
- アプリケーション・クライアントの起動

アプリケーション・クライアントを開発する場合、およびアプリケーション・クライアント・コンテナを使用する場合、ユーザーはJNDIおよびRMIを理解している必要があります。これらの項目は、このマニュアルのそれぞれに関連する章で説明されています。

アプリケーション・クライアント・コンテナのタスク

この章では、次のタスクを説明します。

- メイン・クラスの作成
- JNDIを使用したリソースへのアクセス
- JNDIプロパティの設定
- デプロイメント・ディスクリプタの使用方法
- クライアントのパッケージ化
- カスタム・セキュリティ・ハンドラの実装
- アプリケーション・クライアントの起動

追加ドキュメント

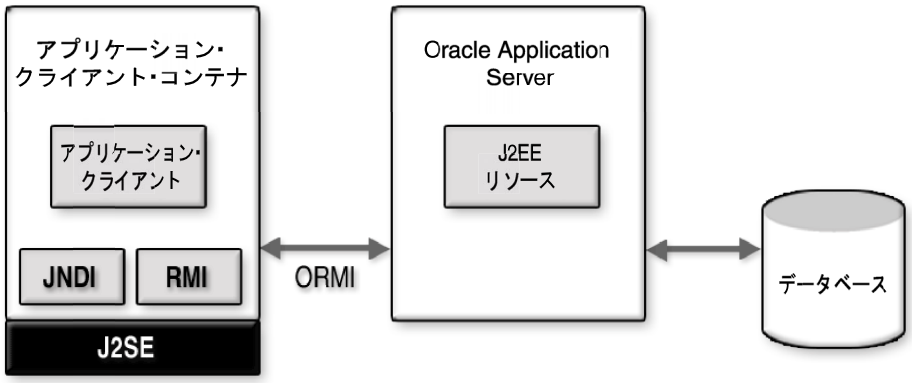
OC4Jのサイトにある How-To ドキュメントでは、OC4Jの機能に関する追加情報（EJBの例など）を提供しています。例には、アプリケーション・クライアント・コンテナの機能を示すアプリケーション・クライアントが含まれています。

```
http://www.oracle.com/technology/tech/java/oc4j/index.html
```

コンテナの概要

Web コンテナや EJB コンテナとは違い、アプリケーション・クライアント・コンテナは軽量の J2EE コンテナで、クライアント・コンピュータで稼働します。アプリケーション・クライアント・コンテナは、アプリケーション・クライアントと呼ばれるアプリケーション・コンポーネントのホスティングに使用されます。リモートの J2EE サーバーにあるリソースおよび機能へのアクセスを提供する際にアプリケーション・クライアント・コンテナに依存すること以外、アプリケーション・クライアントは、スタンドアロンの Java2 SE アプリケーションに似ています。実際的な面から見ると、アプリケーション・クライアント・コンテナを使用することにより、開発者は、データソースまたは EJB などの J2EE リソースに対するアクセス権も持つ、堅牢なスタンドアロンの Java アプリケーションを作成できます。図 9-1 に、アプリケーション・クライアント・コンテナ、アプリケーション・クライアントおよびリモートの J2EE リソースの概念図を示します。

図 9-1 アプリケーション・クライアント・コンテナの概念図



Oracle アプリケーション・クライアント・コンテナには、JNDI および RMI の実装が含まれます。実行時、コンテナは JNDI を使用して初期コンテキスト・ファクトリおよび RMI を作成し、クライアントに必要なリモート・リソースを取得します。

JNDI のサポート

アプリケーション・クライアント・コンテナでは、JNDI を使用してリモートの Oracle Application Server でホスティングされているリソースをルックアップできます。アプリケーション・クライアントのメイン・クラスにリソースが自動的に挿入されるようにするための注釈や、JNDI ルックアップを作成するための JNDI API も使用できます。JNDI を使用したことがない場合、JNDI の使用方法の詳細は第 2 章を参照してください。

RMI のサポート

リモート・リソースに対するすべてのリクエストは、RMI を介して実行されます。アプリケーション・クライアント・コンテナでは、ORMI、RMI/IIOP および ORMI トネリングを使用できます。また、OPMN を介した ORMI 接続もサポートされています。RMI を使用したことがない場合、RMI の使用方法の詳細は第 6 章を参照してください。

セキュリティのサポート

アプリケーション・クライアント・コンテナでは、カスタムのセキュリティ・コールバック・ハンドラを使用できます。カスタムのセキュリティ・コールバック・ハンドラを使用すると、カスタマイズした方法でユーザー資格証明を取得できます。

要件

アプリケーション・コンテナには、次のランタイム要件があります。

- JRE 1.4.X または 1.5.X

  リソース・インジェクション用の注釈の使用は、JRE 1.5 を使用している場合にのみサポートされます。JRE 1.4 を使用している場合、注釈の解釈はスキップされます。リソース・インジェクション（J2EE 1.5 の機能）は、リソース・インジェクション用のデプロイメント・ディスクリプタ・タグを使用している場合にのみ JRE 1.4 でサポートされます。

- OC4J クライアント・サイドのディストリビューション

  OC4J クライアント・サイドのディストリビューションには、アプリケーション・クライアント・コンテナの実行に必要なすべてのライブラリが含まれています。ディストリビューションは、OC4J のダウンロード・ページから入手できます。

  `http://www.oracle.com/technology/software/products/ias/htdocs/utilsoft.html`

  コンピュータの任意の場所にライブラリを解凍します。このマニュアルでは、この場所を `J2EE_HOME` と呼びます。開発およびテスト中は、`J2EE_HOME` を Oracle Application Server の `/j2ee/home` ディレクトリに設定した方が便利です。この場所には、必要なライブラリも格納されています。

アプリケーション・クライアントの開発

この項では、アプリケーション・クライアント・コンテナ用のアプリケーション・クライアントの開発に関する基本的な内容を説明します。一般的に、アプリケーション・クライアントは、J2SE アプリケーションの標準構造に準拠しています。ただし、アプリケーション・クライアントでは、コンテナによって提供される J2EE 機能も使用されます。

メイン・クラスの作成

アプリケーション・クライアントには、メイン・クラスとみなされるクラスがあります。メイン・クラスは、アプリケーション・クライアントの起動時のエントリ・ポイントです。このクラスには、`static main` メソッドが含まれている必要があります。`main` メソッドは、コンテナの初期化後に最初に起動されるメソッドです。単純なアプリケーション・クライアントでは、次のようなものを使用します。

```
public class Foo {
   public static void main(String[] args) {
...
   }
}
```

アプリケーション・クライアント・コンテナでは、起動時にアプリケーション・クライアント JAR の `Manifest.mf` ファイルを使用して、アプリケーション・クライアントのメイン・クラスを検索します。クラスは、`Manifest.mf` ファイルに `Main-Class` 属性を定義することでメイン・クラスとして指定されます。次に例を示します。

```
Main-Class: com.mycompany.MyMainClass
```

アプリケーション・クライアント JAR の作成方法の詳細は、「クライアントのパッケージ化」を参照してください。

JNDI を使用したリソースへのアクセス

アプリケーション・クライアント・コンテナには、リモート・リソースにアクセスするための JNDI サービスへのアクセス権のあるアプリケーション・クライアントが用意されています。そのため、アプリケーション・クライアントからのリソース（EJB やデータソースなど）の取得は、その他の J2EE コンポーネントで使用されるのと同じ JNDI プログラミング・モデルに従います。後続の各項では、JNDI の使用方法の概要を説明します。JNDI を使用したことがない場合、JNDI の使用方法の詳細は第 2 章を参照してください。

注意： JNDI を使用してアプリケーション・クライアントから EJB リソースをルックアップする詳細な例は、第 2 章の「J2EE アプリケーション・クライアントからのオブジェクトのルックアップ」を参照してください。また、JNDI ネーミング・コンテキストで使用可能なデータソースを使用してデータベースに接続する方法の詳細は、第 5 章の「データソースからの接続の取得」を参照してください。

注釈の使用方法

アプリケーション・クライアントのメイン・クラスでは、ネーミング・コンテキストからのリソース・インジェクションのリクエストや、ネーミング・コンテキストで必要なエントリの宣言を Java 言語の注釈を使用して実行できます。注釈は、コンテナの初期化手順の一部として処理されます。コンテナを開始してアプリケーション・クライアントが起動されると、注釈は処理されません。

注釈付きのメイン・クラスのメソッドおよびフィールドは、`static` である必要があります。この規則は、インジェクションをリクエストしているメソッドやフィールドを `static` にできない、典型的な J2EE リソースとは異なります。この違いの理由は、コンテナではアプリケーション・クライアントのインスタンスが作成されないことにあります。アプリケーション・クライアント・コンテナでは、アプリケーション・クライアント JAR のマニフェストに指定されているメイン・クラスをロードし、そのクラスの static `main` メソッドを起動します。この後、アプリケーション・クライアントは J2SE アプリケーションと同じように稼働します。

次の例では、`@EJB` という注釈を使用して、`HelloWorld` というリモートの EJB リソースへの参照を取得する方法を示します。

```
import javax.ejb.EJB;

public class HelloWorldClient {
   @EJB
      private static HelloWorld helloWorld;
      public  static void main(String[] args) {
         // Use the HellOWorld bean.
         ...
```

注意： アプリケーション・コンポーネントごとに、サポートされる注釈タグも異なります。それぞれのコンポーネントでの注釈の使用方法の詳細は、個々のアプリケーション・コンポーネントのマニュアル（『Oracle Containers for J2EE Enterprise JavaBeans 開発者ガイド』など）を参照してください。

JNDI API の使用方法

アプリケーション・クライアント・クラスでは、JNDI API を使用してネーミング・コンテキストのリソースをルックアップできます。注釈とは異なり、コンテナの初期化後も JNDI ルックアップを処理できます。次に、JNDI API を使用して、前の例と同じ HelloWorld EJB リソースを取得する例を示します。

```
import javax.naming.Context;
import javax.naming.InitialContext;

public class HelloWorldClient {
   public static void main(String[] args) {
   try {
      HelloWorld helloWorld = (HelloWorld) new InitialContext()
         .lookup("java:comp/env/ejb/HelloWorld");
      // Use the HelloWorld bean.
      ...
```

デプロイメント・ディスクリプタの使用方法

アプリケーション・クライアントでは、アプリケーション・クライアントのデプロイメント・ディスクリプタを使用して、ネーミング・コンテキストでのエントリの宣言およびエントリのインジェクションのリクエストを実行できます。次の例では、リソース・インジェクションをリクエストする方法、およびクライアントのデプロイメント・ディスクリプタに HelloWorld EJB を宣言する方法を示します。

```
<application-client>
   <ejb-ref>
      <ejb-ref-name>com.myCompany.HelloWorldClient/helloWorld</ejb-ref-name>
      <remote>com.myCompany.HelloWorld</remote>
      <injection-target>
         <injection-target-class>
            com.myCompany.HelloWorldClient
         </injection-target-class>
         <injection-target-name>helloWorld</injection-target-name>
      </injection-target>
   </ejb-ref>
</application-client>
```

例では、HelloWorld Bean が、HelloWorldClient クラスの helloWorld インスタンスに挿入されています。

アプリケーション・クライアントのデプロイメント・ディスクリプタの使用方法の詳細は、「デプロイメント・ディスクリプタの使用方法」を参照してください。

JNDI プロパティの設定

JNDI プロパティは jndi.properties ファイルに設定されます。このファイルは、実行時のコンテナに必要です。コンテナは、このファイルを使用して初期 JNDI コンテキストを作成し、リモート・リソースにアクセスするための RMI 接続設定を定義します。このファイルは、アプリケーション・クライアント JAR のルートに配置する必要があります。

次の例では、スタンドアロン OC4J サーバーでホスティングされている EJB にアクセスする際に必要なプロパティを示します。

```
java.naming.factory.initial=oracle.j2ee.naming.
   ApplicationClientInitialContextFactory
java.naming.provider.url=ormi://<host>:23791/my_ejb>
java.naming.security.principal=scott
java.naming.security.credentials=tiger
```

初期ファクトリは常に ApplicationClientInitialContextFactory です。また、リソースが OPMN を使用する Oracle Application Server に配置されている場合は、ネーミング・プロバイダ URL の一部として opmn:ormi プロトコルを使用します。次に例を示します。

```
java.naming.provider.url=opmn:ormi://<host>:6003:home/my_ejb
```

ORMI を使用してリモート・リソースに接続する方法の詳細は、6-8 ページの「RMI/ORMI を使用したリモート・オブジェクトのルックアップ」を参照してください。JNDI の使用方法の詳細は、第 2 章「JNDI の使用」を参照してください。

デプロイメント・ディスクリプタの使用方法

application-client.xml ファイルは、アプリケーション・クライアント用の標準の J2EE デプロイメント・ディスクリプタで、アプリケーション・クライアント JAR の META-INF/ ディレクトリに配置する必要があります。このファイルは、環境エントリやアプリケーション・クライアントに必要なリソースへの参照の定義に使用されます。ただし、リソース・インジェクションを実行するために、クライアント・コードで注釈のみが使用されている場合には、このファイルは不要です。

> **注意：** リソース参照は、注釈を使用して定義するだけでなく、application-client.xml ファイルにも定義できます。その場合、デプロイメント・ディスクリプタは注釈よりも優先されます。

application-client.xml ファイルでは、標準の J2EE 参照要素（env-entry、ejb-ref、resource-ref など）が使用されます。デプロイメント・ディスクリプタの詳細は、J2EE 仕様のアプリケーション・クライアント・コンテナに関する章を参照してください。この章には、application-client.xml ファイルの詳細な例および使用可能な要素のリストがあります。

クライアントのパッケージ化

アプリケーション・クライアントは、JAR ファイルとしてパッケージ化されます。アーカイブは、Java の jar ユーティリティを使用するか、類似の圧縮形式（zip など）を使用して拡張子を .jar に変更することで作成できます。

アーカイブ・ファイルのコンテンツは、次のような標準の構造で編成されています。

```
/
/jndi.properties
/META-INF
/META-INF/Manifest.mf
/META-INF/application-client.xml
```

カスタム・セキュリティ・ハンドラの実装

アプリケーション・クライアント・コンテナは、初期 JNDI コンテキストの作成時やリモート・メソッドの起動時にはリモート・サーバーに認証する必要があります。コンテナでは、JNDI プロパティ・ファイルに定義されているユーザー名、パスワードおよびプロバイダ URL を使用するデフォルトのコールバック・ハンドラが使用されます。これらのプロパティの設定方法の詳細は、「JNDI プロパティの設定」を参照してください。

アプリケーション・クライアントで JAAS コールバック・ハンドラを実装すると、必要な資格証明を取得するためにログイン画面を表示するなどのカスタム機能を提供できます。そのような場合、コンテナにより起動時にコールバック・ハンドラがインスタンス化され、JNDI プロパティ・ファイルに定義されている設定は無視されます。コールバック・ハンドラが失敗すると、コンテナは JNDI プロパティ・ファイルのデフォルトの設定に戻ります。

カスタム・セキュリティ・ハンドラを実装するには、次のようにします。

1. `javax.security.auth.callback.CallbackHandler` インタフェースを実装する、引数コンストラクタのないハンドラ・クラスを作成します。
2. アプリケーション・クライアント JAR ファイルにコールバック・ハンドラを追加するか、実行時に `CLASSPATH` でハンドラが検出されることを確認します。
3. ハンドラ・クラスへの参照を `application-client.xml` ファイルに追加します。次に例を示します。

```
<application-client>
   <callback-handler>
      com.myCompany.MyCallbackHandler
   </callback-handler>
</application-client>
```

アプリケーション・クライアントの起動

アプリケーション・クライアント・コンテナは、起動時にクライアントを起動します。起動中、コンテナはまず JNDI コンテキストを作成し、定義されている任意のリモート・サーバーに接続して、クライアントがリクエストした J2EE リソースを取得します。クライアントは、コンテナによる初期化が完了した後にのみ起動されます。

> **注意：** クライアントを起動する前には、リクエストされたリソースがリモート・サーバーにデプロイされていることと、サーバーが起動されていることを確認してください。コンテナがリモート・サーバーのリソースを検出できない場合、コンテナは初期化に失敗し、クライアントも起動されません。

アプリケーション・クライアント・コンテナの起動クラス（`oracle.oc4j.appclient.AppClientContainer`）は、`oc4jclient.jar` のメイン・クラスとして設定されます。JAR を使用してアプリケーション・クライアントを起動するには、コマンドラインから次のコマンドを入力します。

```
java -jar J2EE_HOME¥oc4jclient.jar <aplication_client_jar> <arguments>
```

アプリケーション・クライアント JAR ファイルへのフルパスを指定する必要があります。また、コマンドラインで指定されたクライアント引数は、起動時にクライアントの `main()` メソッドに渡され、コンテナでは処理されません。

`oc4jclient.jar` ファイルのデフォルトの動作では、`Manifest.mf` ファイルに定義されている `CLASSPATH` を使用してコンテナが起動されます。ただし、起動時にその他のライブラリまたはディレクトリが必要な場合には、アプリケーション・クライアント・コンテナの起動クラスおよび Java の `-cp` オプションを使用します。次に例を示します。

```
java -cp J2EE_HOME¥oc4jclient.jar;foo.jar oracle.oc4j.appclient.AppClientContainer
<aplication_client_jar> <arguments>
```

注意： Java ライブラリと同様に、アプリケーション・クライアント・ライブラリでは、マニフェストに `Class-Path` エントリを含めることができます。これにより、アプリケーション・クライアントのクラスパスに外部 JAR を追加し、詳細なクラスパス構成を提供できます。ただし、JAR を使用できるのはアプリケーション・クライアントのみで、アプリケーション・クライアント・コンテナでは使用できません。

A

サード・パーティ・ライセンス

この付録には、このマニュアルの内容に関連するサード・パーティ製品のサード・パーティ・ライセンスが含まれています。

ANTLR

このプログラムには、ANTLR 提供のサード・パーティ・コードが含まれています。ANTLR のライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（ANTLR ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはないことに注意してください。

ANTLR ライセンス

```
Software License

We reserve no legal rights to the ANTLR--it is fully in the public domain. An
individual or company may do whatever they wish with source code distributed with ANTLR
or the code generated by ANTLR, including the incorporation of ANTLR, or its output,
into commerical software.
We encourage users to develop software with ANTLR. However, we do ask that credit is
given to us for developing ANTLR. By "credit", we mean that if you use ANTLR or
incorporate any source code into one of your programs (commercial product, research
project, or otherwise) that you acknowledge this fact somewhere in the documentation,
research report, etc... If you like ANTLR and have developed a nice tool with the
output, please mention that you developed it using ANTLR. In addition, we ask that the
headers remain intact in our source code. As long as these guidelines are kept, we
expect to continue enhancing this system and expect to make other tools available as
they are completed.
```

Apache

このプログラムには、Apache Software Foundation（Apache）提供のサード・パーティ・コードが含まれています。Apache のライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（Apache ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはないことに注意してください。

Apache ソフトウェア・ライセンス

Apache Web Server 1.3.29 のライセンス

```
/* ====================================================================
 * The Apache Software License, Version 1.1
 *
 * Copyright (c) 2000-2002 The Apache Software Foundation.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 *    if any, must include the following acknowledgment:
 *       "This product includes software developed by the
 *        Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself,
```

```
 *    if and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "Apache" and "Apache Software Foundation" must
 *    not be used to endorse or promote products derived from this
 *    software without prior written permission. For written
 *    permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *    nor may "Apache" appear in their name, without prior written
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ====================================================================
 *
 * This software consists of voluntary contributions made by many
 * individuals on behalf of the Apache Software Foundation.  For more
 * information on the Apache Software Foundation, please see
 * <http://www.apache.org/>.
 *
 * Portions of this software are based upon public domain software
 * originally written at the National Center for Supercomputing
Applications,
 * University of Illinois, Urbana-Champaign.
```

Apache Web Server 2.0 のライセンス

```
Copyright (c) 1999-2004, The Apache Software Foundation
Licensed under the Apache License, Version 2.0 (the "License"); you may not use this
file except in compliance with the License.  You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software distributed under
the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
KIND, either express or implied.  See the License for the specific language governing
permissions and limitations under the License.
Copyright (c) 1999-2004, The Apache Software Foundation
                                 Apache License
                            Version 2.0, January 2004
                         http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
```

"control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s)

with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

Apache SOAP

このプログラムには、Apache Software Foundation（Apache）提供のサード・パーティ・コードが含まれています。Apache のライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（Apache ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはないことに注意してください。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、Apache ソフトウェアは現状のままでオラクル社から提供されるものであり、いかなる種類の保証またはサポートもオラクル社または Apache から提供されません。

Apache SOAP ライセンス

Apache SOAP ライセンス 2.3.1

Copyright (c) 1999 The Apache Software Foundation. All rights reserved.
TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION
1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the

direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You

institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

DBI モジュール

このプログラムには、Tim Bunce 氏提供のサード・パーティ・コードが含まれています。Tim Bunce 氏のライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（Tim Bunce 氏のソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはないことに注意してください。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、Tim Bunce 氏のソフトウェアは現状のままでオラクル社から提供されるものであり、いかなる種類の保証またはサポートもオラクル社または Tim Bunce 氏から提供されません。

The DBI module is Copyright (c) 1994-2002 Tim Bunce. Ireland. All rights reserved.

Perl の README ファイルに記載されているとおり、GNU 一般公衆利用許諾契約書または Artistic ライセンスの条項に基づいて配布できます。

Perl Artistic ライセンス

The "Artistic License"

Preamble

The intent of this document is to state the conditions under which a Package may be copied, such that the Copyright Holder maintains some semblance of artistic control over the development of the package, while giving the users of the package the right to use and distribute the Package in a more-or-less customary fashion, plus the right to make reasonable modifications.

Definitions

"Package" refers to the collection of files distributed by the Copyright Holder, and derivatives of that collection of files created through textual modification.

"Standard Version" refers to such a Package if it has not been modified, or has been modified in accordance with the wishes of the Copyright Holder as specified below.

"Copyright Holder" is whoever is named in the copyright or copyrights for the package.

"You" is you, if you're thinking about copying or distributing this Package.

"Reasonable copying fee" is whatever you can justify on the basis of media cost, duplication charges, time of people involved, and so on. (You will not be required to justify it to the Copyright Holder, but only to the computing community at large as a market that must bear the fee.)

"Freely Available" means that no fee is charged for the item itself, though there may be fees involved in handling the item. It also means that recipients of the item may redistribute it under the same conditions they received it.

1. You may make and give away verbatim copies of the source form of the Standard Version of this Package without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.
2. You may apply bug fixes, portability fixes and other modifications derived from the Public Domain or from the Copyright Holder. A Package modified in such a way shall still be considered the Standard Version.
3. You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file stating how and when you changed that file, and provided that you do at least ONE of the following:
   a. place your modifications in the Public Domain or otherwise make them Freely Available, such as by posting said modifications to Usenet or an equivalent medium, or placing the modifications on a major archive site such as uunet.uu.net, or by allowing the Copyright Holder to include your modifications in the Standard Version of the Package.
   b. use the modified Package only within your corporation or organization.
   c. rename any non-standard executables so the names do not conflict with standard executables, which must also be provided, and provide a separate manual page for each non-standard executable that clearly documents how it differs from the Standard Version.
   d. make other distribution arrangements with the Copyright Holder.
4. You may distribute the programs of this Package in object code or executable form, provided that you do at least ONE of the following:
   a. distribute a Standard Version of the executables and library files, together with instructions (in the manual page or equivalent) on where to get the Standard Version.
   b. accompany the distribution with the machine-readable source of the Package with your modifications.

   c. give non-standard executables non-standard names, and clearly document the differences in manual pages (or equivalent), together with instructions on where to get the Standard Version.

   d. make other distribution arrangements with the Copyright Holder.

5. You may charge a reasonable copying fee for any distribution of this Package. You may charge any fee you choose for support of this Package. You may not charge a fee for this Package itself. However, you may distribute this Package in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution provided that you do not advertise this Package as a product of your own. You may embed this Package's interpreter within an executable of yours (by linking); this shall be construed as a mere form of aggregation, provided that the complete Standard Version of the interpreter is so embedded.

6. The scripts and library files supplied as input to or produced as output from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whoever generated them, and may be sold commercially, and may be aggregated with this Package. If such scripts or library files are aggregated with this Package through the so-called "undump" or "unexec" methods of producing a binary executable image, then distribution of such an image shall neither be construed as a distribution of this Package nor shall it fall under the restrictions of Paragraphs 3 and 4, provided that you do not represent such an executable image as a Standard Version of this Package.

7. C subroutines (or comparably compiled subroutines in other languages) supplied by you and linked into this Package in order to emulate subroutines and variables of the language defined by this Package shall not be considered part of this Package, but are the equivalent of input as in Paragraph 6, provided these subroutines do not change the language in any way that would cause it to fail the regression tests for the language.

8. Aggregation of this Package with a commercial distribution is always permitted provided that the use of this Package is embedded; that is, when no overt attempt is made to make this Package's interfaces visible to the end user of the commercial distribution. Such use shall not be construed as a distribution of this Package.

9. The name of the Copyright Holder may not be used to endorse or promote products derived from this software without specific prior written permission.

10. THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The End

expat

このプログラムには、CPAN 提供のサード・パーティ・コードが含まれています。CPAN のライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（CPAN ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはないことに注意してください。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、CPAN ソフトウェアは現状のままでオラクル社から提供されるものであり、いかなる種類の保証またはサポートもオラクル社または CPAN から提供されません。

FastCGI

このプログラムには、Open Market 社提供のサード・パーティ・コードが含まれています。Open Market のライセンス条件に基づき、オラクル社は次の条件で Open Market ソフトウェアをライセンスすることが求められています。この製品に付随する Oracle プログラム・ライセンスの条件は Open Market ソフトウェアに適用されず、このソフトウェアを使用する権利は次に記載されている内容のみであることに注意してください。オラクル社は、Open Market ソフトウェアのパフォーマンスに対して責任を負わず、技術サポートも提供しません。オラクル社は、このソフトウェアの使用に起因するいかなる損害についても責任を負いかねます。

FastCGI Developer's Kit ライセンス

This FastCGI application library source and object code (the "Software") and its documentation (the "Documentation") are copyrighted by Open Market, Inc ("Open Market"). The following terms apply to all files associated with the Software and Documentation unless explicitly disclaimed in individual files.

Open Market permits you to use, copy, modify, distribute, and license this Software and the Documentation solely for the purpose of implementing the FastCGI specification defined by Open Market or derivative specifications publicly endorsed by Open Market and promulgated by an open standards organization and for no other purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions.

No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this Software and Documentation may be copyrighted by their authors and need not follow the licensing terms described here, but the modified Software and Documentation must be used for the sole purpose of implementing the FastCGI specification defined by Open Market or derivative specifications publicly endorsed by Open Market and promulgated by an open standards organization and for no other purpose. If modifications to this Software and Documentation have new licensing terms, the new terms must protect Open Market's proprietary rights in the Software and Documentation to the same extent as these licensing terms and must be clearly indicated on the first page of each file where they apply.

Open Market shall retain all right, title and interest in and to the Software and Documentation, including without limitation all patent, copyright, trade secret and other proprietary rights.

OPEN MARKET MAKES NO EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE SOFTWARE OR THE DOCUMENTATION, INCLUDING WITHOUT LIMITATION ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL OPEN MARKET BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY DAMAGES ARISING FROM OR RELATING TO THIS SOFTWARE OR THE DOCUMENTATION, INCLUDING, WITHOUT LIMITATION, ANY INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES OR SIMILAR DAMAGES, INCLUDING LOST PROFITS OR LOST DATA, EVEN IF OPEN MARKET HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS". OPEN MARKET HAS NO LIABILITY IN CONTRACT, TORT, NEGLIGENCE OR OTHERWISE ARISING OUT OF THIS SOFTWARE OR THE DOCUMENTATION.

モジュール mod_fastcgi ライセンス

This FastCGI application library source and object code (the "Software") and its documentation (the "Documentation") are copyrighted by Open Market, Inc ("Open Market"). The following terms apply to all files associated with the Software and Documentation unless explicitly disclaimed in individual files.

Open Market permits you to use, copy, modify, distribute, and license this Software and the Documentation solely for the purpose of implementing the FastCGI specification defined by Open Market or derivative specifications publicly endorsed by Open Market and promulgated by an open standards organization and for no other purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions.

No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this Software and Documentation may be copyrighted by their authors and need not follow the licensing terms described here, but the modified Software and Documentation must be used for the sole purpose of implementing the FastCGI specification defined by Open Market or derivative specifications publicly endorsed by Open Market and promulgated by an open standards organization and for no other purpose. If modifications to this Software and Documentation have new licensing terms, the new terms must protect Open Market's proprietary rights in the Software and Documentation to the same extent as these licensing terms and must be clearly indicated on the first page of each file where they apply.

Open Market shall retain all right, title and interest in and to the Software and Documentation, including without limitation all patent, copyright, trade secret and other proprietary rights.

OPEN MARKET MAKES NO EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE SOFTWARE OR THE DOCUMENTATION, INCLUDING WITHOUT LIMITATION ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL OPEN MARKET BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY DAMAGES ARISING FROM OR RELATING TO THIS SOFTWARE OR THE DOCUMENTATION, INCLUDING, WITHOUT LIMITATION, ANY INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES OR SIMILAR DAMAGES, INCLUDING LOST PROFITS OR LOST DATA, EVEN IF OPEN MARKET HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS". OPEN MARKET HAS NO LIABILITY IN CONTRACT, TORT, NEGLIGENCE OR OTHERWISE ARISING OUT OF THIS SOFTWARE OR THE DOCUMENTATION.

Info-ZIP Unzip パッケージ

このプログラムには、Info-ZIP 提供のサード・パーティ・コードが含まれています。Info-ZIP のライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（Info-ZIP ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはないことに注意してください。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、Info-ZIP のソフトウェアは現状のままでオラクル社から提供されるものであり、いかなる種類の保証またはサポートもオラクル社または Info-ZIP から提供されません。

Info-ZIP Unzip パッケージ・ライセンス

```
Copyright (c) 1990-1999 Info-ZIP. All rights reserved. For the purposes of this
copyright and license, "Info-ZIP" is defined as the following set of individuals:

Mark Adler, John Bush, Karl Davis, Harald Denker, Jean-Michel Dubois, Jean-loup Gailly,
Hunter Goatley, Ian Gorman, Chris Herborth, Dirk Haase, Greg Hartwig, Robert Heath,
Jonathan Hudson, Paul Kienitz, David Kirschbaum, Johnny Lee, Onno van der Linden, Igor
Mandrichenko, Steve P. Miller, Sergio Monesi, Keith Owens, George Petrov, Greg Roelofs,
Kai Uwe Rommel, Steve Salisbury, Dave Smith, Christian Spieler, Antoince Verheijen,
Paul von Behren, Rich Wales, Mike White

This software is provided "AS IS," without warranty of any kind, express or implied.
In no event shall InfoZIP or its contributors be held liable for any direct, indirect,
incidental, special or consequential damages arising out of the use of or inability to
use this software."
```

Jabberbeans

このプログラムには、Jabber 社提供のサード・パーティ・コードが含まれています。Jabber 社のオープン・ソース・ライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（JabberBeans ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはないことに注意してください。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、JabberBeans ソフトウェアは現状のままでオラクル社から提供されるものであり、いかなる種類の保証またはサポートもオラクル社または Jabber 社から提供されません。

JabberBeans のソース・コードは、Jabber 社の Web サイト（`www.jabber.com`）から入手可能です。

JSR 110

このプログラムには、IBM 社提供のサード・パーティ・コードが含まれています。IBM 社のライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（IBM ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはないことに注意してください。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、IBM ソフトウェアは現状のままでオラクル社から提供されるものであり、いかなる種類の保証またはサポートもオラクル社または IBM 社から提供されません。

```
Copyright IBM Corporation 2003 - All rights reserved

Java APIs for the WSDL specification are available at:
http://www-124.ibm.com/developerworks/projects/wsdl4j/
```

Jaxen

このプログラムには、Apache Software Foundation（Apache）および Jaxen Project（Jaxen）提供のサード・パーティ・コードが含まれています。Apache および Jaxen のライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（Apache ソフトウェアおよび Jaxen ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはないことに注意してください。

Jaxen ライセンス

JGroups

このプログラムには、GNU 提供のサード・パーティ・コードが含まれています。GNU のライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（GNU ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはないことに注意してください。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、GNU ソフトウェアは現状のままでオラクル社から提供されるものであり、いかなる種類の保証またはサポートもオラクル社または GNU から提供されません。

GNU ライセンス

GNU Lesser General Public License
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble
The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced

by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION
0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without

limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium

does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must

include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will

be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS
How to Apply These Terms to Your New Libraries
If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the library's name and an idea of what it does.> Copyright (C) <year> <name of author>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

JTidy

このプログラムには、World Wide Web Consortium（W3C）および共同作成者（Dave Raggett <dsr@w3.org>、Andy Quick <ac.quick@sympatico.ca>（Java への変換）、Gary L Peskin <garyp@firstech.com>（Java 開発）、Sami Lempinen <sami@lempinen.net>（リリースの管理））提供の Java HTML Tidy（JTidy）として知られるサード・パーティ・コードが含まれています。ただし、Oracle プログラム（W3C JTidy ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定されることに注意してください。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、W3C JTidy ソフトウェアは現状のままでオラクル社から提供されるものであり、いかなる種類の保証またはサポートもオラクル社、W3C または共同作成者から提供されません。

mod_dav

このプログラムには、Greg Stein 氏提供のサード・パーティ・コードが含まれています。Greg Stein 氏のライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（Greg Stein 氏のソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはないことに注意してください。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、Greg Stein 氏のソフトウェアは現状のままでオラクル社から提供されるものであり、いかなる種類の保証またはサポートもオラクル社または Greg Stein 氏から提供されません。

```
Copyright © 1998-2001 Greg Stein. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are
permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice,
     this list of conditions and the following disclaimer.

  2. Redistributions in binary form must reproduce the above copyright
     notice, this list of conditions and the following disclaimer in the
     documentation and/or other materials provided with the distribution.

  3. All advertising materials mentioning features or use of this software
     must display the following acknowledgment:

          This product includes software developed by Greg Stein
          <gstein@lyra.org> for use in the mod_dav module for Apache
          (http://www.webdav.org/mod_dav/).

  4. Products derived from this software may not be called "mod_dav" nor may
     "mod_dav" appear in their names without prior written permission of
     Greg Stein. For written permission, please contact gstein@lyra.org.

  5. Redistributions of any form whatsoever must retain the following
     acknowledgment:

          This product includes software developed by Greg Stein
          <gstein@lyra.org> for use in the mod_dav module for Apache
          (http://www.webdav.org/mod_dav/).

THIS SOFTWARE IS PROVIDED BY GREG STEIN "AS IS" AND ANY EXPRESSED OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
EVENT SHALL GREG STEIN OR THE SOFTWARE'S CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
```

```
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

  ----------------------------------------------------------------------
Greg Stein
Last modified: Thu Feb 3 17:34:42 PST 2000
```

mod_mm および mod_ssl

このプログラムには、Ralf S. Engelschall 氏（以降「Engelschall 氏」）提供のサード・パーティ・コードが含まれています。Engelschall 氏のライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（Engelschall 氏のソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはないことに注意してください。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、mod_mm ソフトウェアは現状のままでオラクル社から提供されるものであり、いかなる種類の保証またはサポートもオラクル社または Engelschall 氏から提供されません。

mod_mm

```
Copyright (c) 1999 - 2000 Ralf S. Engelschall.  All rights reserved.
This product includes software developed by Ralf S. Engelschall <rse@engelschall.com>
for use in the mod_ssl project (http://www.modssl.org/).
```

mod_ssl

```
Copyright (c) 1998-2001 Ralf S. Engelschall.  All rights reserved.
This product includes software developed by Ralf S. Engelschall <rse@engelschall.com>
for use in the mod_ssl project (http://www.modssl.org/).
```

OpenSSL

このプログラムには、OpenSSL Project 提供のサード・パーティ・コードが含まれています。OpenSSL Project のライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（OpenSSL ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはないことに注意してください。

OpenSSL ライセンス

```
/* ====================================================================
 * Copyright (c) 1998-2005 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
```

```
 *    prior written permission. For written permission, please contact
 *    openssl-core@openssl.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT "AS IS" AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 *
 * This product includes cryptographic software written by Eric Young
 * (eay@cryptsoft.com).  This product includes software written by Tim
 * Hudson (tjh@cryptsoft.com).
 *
 */

 Original SSLeay License
 -----------------------

/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
 * All rights reserved.
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to.  The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
```

```
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    "This product includes cryptographic software written by
 *     Eric Young (eay@cryptsoft.com)"
 *    The word 'cryptographic' can be left out if the rouines from the library
 *    being used are not cryptographic related :-).
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledgement:
 *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
 *
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed.  i.e. this code cannot simply be
 * copied and put under another distribution licence
 * [including the GNU Public Licence.]
 */
```

Perl

このプログラムには、Comprehensive Perl Archive Network（CPAN）提供のサード・パーティ・コードが含まれています。CPAN のライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（CPAN ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはないことに注意してください。

Perl Kit Readme

This program is free software; you can redistribute it and/or modify it under the terms of either:

1. the GNU General Public License as published by the Free Software Foundation; either version 1, or (at your option) any later version, or
2. the "Artistic License" which comes with this Kit.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See either the GNU General Public License or the Artistic License for more details.

You should have received a copy of the Artistic License with this Kit, in the file named "Artistic". If not, I'll be glad to provide one.

You should also have received a copy of the GNU General Public License along with this program in the file named "Copying". If not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA or visit their Web page on the internet at http://www.gnu.org/copyleft/gpl.html.

For those of you that choose to use the GNU General Public License, my interpretation of the GNU General Public License is that no Perl script falls under the terms of the GPL unless you explicitly put said script under the terms of the GPL yourself. Furthermore, any object code linked with perl does not automatically fall under the terms of the GPL, provided such object code only adds definitions of subroutines and variables, and does not otherwise impair the resulting interpreter from executing any standard Perl script. I consider linking in C subroutines in this manner to be the moral equivalent of defining subroutines in the Perl language itself. You may sell such an object file as proprietary provided that you provide or offer to provide the Perl source, as specified by the GNU General Public License. (This is merely an alternate way of specifying input to the program.) You may also sell a binary produced by the dumping of a running Perl script that belongs to you, provided that you provide or offer to provide the Perl source as specified by the GPL. (The fact that a Perl interpreter and your code are in the same binary file is, in this case, a form of mere aggregation.) This is my interpretation of the GPL. If you still have concerns or difficulties understanding my intent, feel free to contact me. Of course, the Artistic License spells all this out for your protection, so you may prefer to use that.

mod_perl 1.29 ライセンス

```
/* ====================================================================
 * The Apache Software License, Version 1.1
 *
 * Copyright (c) 1996-2000 The Apache Software Foundation.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 *    if any, must include the following acknowledgment:
 *       "This product includes software developed by the
 *        Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself,
 *    if and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "Apache" and "Apache Software Foundation" must
 *    not be used to endorse or promote products derived from this
 *    software without prior written permission. For written
 *    permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *    nor may "Apache" appear in their name, without prior written
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
```

```
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ====================================================================
 */
```

mod_perl 1.99_16 ライセンス

```
Copyright (c) 1999-2004, The Apache Software Foundation
Licensed under the Apache License, Version 2.0 (the "License"); you may not use this
file except in compliance with the License.  You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software distributed under
the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
KIND, either express or implied.  See the License for the specific language governing
permissions and limitations under the License.
Copyright (c) 1999-2004, The Apache Software Foundation
                                 Apache License
                            Version 2.0, January 2004
                         http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      "control" means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      "You" (or "Your") shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      "Source" form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      "Object" form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
      and conversions to other media types.

      "Work" shall mean the work of authorship, whether in Source or
      Object form, made available under the License, as indicated by a
      copyright notice that is included in or attached to the work
      (an example is provided in the Appendix below).

      "Derivative Works" shall mean any work, whether in Source or Object
      form, that is based on (or derived from) the Work and for which the
      editorial revisions, annotations, elaborations, or other modifications
      represent, as a whole, an original work of authorship. For the purposes
      of this License, Derivative Works shall not include works that remain
      separable from, or merely link (or bind by name) to the interfaces of,
      the Work and Derivative Works thereof.
```

"Contribution" shall mean any work of authorship, including
the original version of the Work and any modifications or additions
to that Work or Derivative Works thereof, that is intentionally
submitted to Licensor for inclusion in the Work by the copyright owner
or by an individual or Legal Entity authorized to submit on behalf of
the copyright owner. For the purposes of this definition, "submitted"
means any form of electronic, verbal, or written communication sent
to the Licensor or its representatives, including but not limited to
communication on electronic mailing lists, source code control systems,
and issue tracking systems that are managed by, or on behalf of, the
Licensor for the purpose of discussing and improving the Work, but
excluding communication that is conspicuously marked or otherwise
designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity
on behalf of whom a Contribution has been received by Licensor and
subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
this License, each Contributor hereby grants to You a perpetual,
worldwide, non-exclusive, no-charge, royalty-free, irrevocable
copyright license to reproduce, prepare Derivative Works of,
publicly display, publicly perform, sublicense, and distribute the
Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
this License, each Contributor hereby grants to You a perpetual,
worldwide, non-exclusive, no-charge, royalty-free, irrevocable
(except as stated in this section) patent license to make, have made,
use, offer to sell, sell, import, and otherwise transfer the Work,
where such license applies only to those patent claims licensable
by such Contributor that are necessarily infringed by their
Contribution(s) alone or by combination of their Contribution(s)
with the Work to which such Contribution(s) was submitted. If You
institute patent litigation against any entity (including a
cross-claim or counterclaim in a lawsuit) alleging that the Work
or a Contribution incorporated within the Work constitutes direct
or contributory patent infringement, then any patent licenses
granted to You under this License for that Work shall terminate
as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
Work or Derivative Works thereof in any medium, with or without
modifications, and in Source or Object form, provided that You
meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one

of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability

```
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.
```

Perl Artistic ライセンス

The "Artistic License"

Preamble

The intent of this document is to state the conditions under which a Package may be copied, such that the Copyright Holder maintains some semblance of artistic control over the development of the package, while giving the users of the package the right to use and distribute the Package in a more-or-less customary fashion, plus the right to make reasonable modifications.

Definitions

"Package" refers to the collection of files distributed by the Copyright Holder, and derivatives of that collection of files created through textual modification.

"Standard Version" refers to such a Package if it has not been modified, or has been modified in accordance with the wishes of the Copyright Holder as specified below.

"Copyright Holder" is whoever is named in the copyright or copyrights for the package.

"You" is you, if you're thinking about copying or distributing this Package.

"Reasonable copying fee" is whatever you can justify on the basis of media cost, duplication charges, time of people involved, and so on. (You will not be required to justify it to the Copyright Holder, but only to the computing community at large as a market that must bear the fee.)

"Freely Available" means that no fee is charged for the item itself, though there may be fees involved in handling the item. It also means that recipients of the item may redistribute it under the same conditions they received it.

1. You may make and give away verbatim copies of the source form of the Standard Version of this Package without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.
2. You may apply bug fixes, portability fixes and other modifications derived from the Public Domain or from the Copyright Holder. A Package modified in such a way shall still be considered the Standard Version.
3. You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file stating how and when you changed that file, and provided that you do at least ONE of the following:
   a. place your modifications in the Public Domain or otherwise make them Freely Available, such as by posting said modifications to Usenet or an equivalent medium, or placing the modifications on a major archive site such as uunet.uu.net, or by allowing the Copyright Holder to include your modifications in the Standard Version of the Package.
   b. use the modified Package only within your corporation or organization.
   c. rename any non-standard executables so the names do not conflict with standard executables, which must also be provided, and provide a separate manual page for each non-standard executable that clearly documents how it differs from the Standard Version.
   d. make other distribution arrangements with the Copyright Holder.

4. You may distribute the programs of this Package in object code or executable form, provided that you do at least ONE of the following:

   a. distribute a Standard Version of the executables and library files, together with instructions (in the manual page or equivalent) on where to get the Standard Version.

   b. accompany the distribution with the machine-readable source of the Package with your modifications.

   c. give non-standard executables non-standard names, and clearly document the differences in manual pages (or equivalent), together with instructions on where to get the Standard Version.

   d. make other distribution arrangements with the Copyright Holder.

5. You may charge a reasonable copying fee for any distribution of this Package. You may charge any fee you choose for support of this Package. You may not charge a fee for this Package itself. However, you may distribute this Package in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution provided that you do not advertise this Package as a product of your own. You may embed this Package's interpreter within an executable of yours (by linking); this shall be construed as a mere form of aggregation, provided that the complete Standard Version of the interpreter is so embedded.

6. The scripts and library files supplied as input to or produced as output from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whoever generated them, and may be sold commercially, and may be aggregated with this Package. If such scripts or library files are aggregated with this Package through the so-called "undump" or "unexec" methods of producing a binary executable image, then distribution of such an image shall neither be construed as a distribution of this Package nor shall it fall under the restrictions of Paragraphs 3 and 4, provided that you do not represent such an executable image as a Standard Version of this Package.

7. C subroutines (or comparably compiled subroutines in other languages) supplied by you and linked into this Package in order to emulate subroutines and variables of the language defined by this Package shall not be considered part of this Package, but are the equivalent of input as in Paragraph 6, provided these subroutines do not change the language in any way that would cause it to fail the regression tests for the language.

8. Aggregation of this Package with a commercial distribution is always permitted provided that the use of this Package is embedded; that is, when no overt attempt is made to make this Package's interfaces visible to the end user of the commercial distribution. Such use shall not be construed as a distribution of this Package.

9. The name of the Copyright Holder may not be used to endorse or promote products derived from this software without specific prior written permission.

10. THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The End

PHP

このプログラムには、Henry Spencer 氏および PHP Group 提供のサード・パーティ・コードが含まれています。Henry Spencer 氏の著作権情報および PHP のライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（PHP ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはないことに注意してください。

PHP ライセンス

```
--------------------------------------------------------------------
                  The PHP License, version 3.01
Copyright (c) 1999 - 2005 The PHP Group. All rights reserved.
--------------------------------------------------------------------

Redistribution and use in source and binary forms, with or without
modification, is permitted provided that the following conditions
are met:

  1. Redistributions of source code must retain the above copyright
     notice, this list of conditions and the following disclaimer.

  2. Redistributions in binary form must reproduce the above copyright
     notice, this list of conditions and the following disclaimer in
     the documentation and/or other materials provided with the
     distribution.

  3. The name "PHP" must not be used to endorse or promote products
     derived from this software without prior written permission. For
     written permission, please contact group@php.net.

  4. Products derived from this software may not be called "PHP", nor
     may "PHP" appear in their name, without prior written permission
     from group@php.net.  You may indicate that your software works in
     conjunction with PHP by saying "Foo for PHP" instead of calling
     it "PHP Foo" or "phpfoo"

  5. The PHP Group may publish revised and/or new versions of the
     license from time to time. Each version will be given a
     distinguishing version number.
     Once covered code has been published under a particular version
     of the license, you may always continue to use it under the terms
     of that version. You may also choose to use such covered code
     under the terms of any subsequent version of the license
     published by the PHP Group. No one other than the PHP Group has
     the right to modify the terms applicable to covered code created
     under this License.

  6. Redistributions of any form whatsoever must retain the following
     acknowledgment:
     "This product includes PHP software, freely available from
     <http://www.php.net/software/>".

THIS SOFTWARE IS PROVIDED BY THE PHP DEVELOPMENT TEAM "AS IS" AND
ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE PHP
DEVELOPMENT TEAM OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
```

```
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.

-----------------------------------------------------------------------
```

SAXPath

このプログラムには、SAXPath 提供のサード・パーティ・コードが含まれています。SAXPath のライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（SAXPath ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはないことに注意してください。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、SAXPath ソフトウェアは現状のままでオラクル社から提供されるものであり、いかなる種類の保証またはサポートもオラクル社または SAXPath から提供されません。

SAXPath ライセンス

```
Copyright (C) 2000-2002 werken digital. All rights reserved. Redistribution and use in
source and binary forms, with or without modification, are permitted provided that the
following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of
conditions, and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of
conditions, and the disclaimer that follows these conditions in the documentation
and/or other materials provided with the distribution.

The name "SAXPath" must not be used to endorse or promote products derived from this
software without prior written permission. For written permission, please contact
license@saxpath.org.

Products derived from this software may not be called "SAXPath", nor may "SAXPath"
appear in their name, without prior written permission from the SAXPath Project
Management (pm@saxpath.org).

In addition, we request (but do not require) that you include in the end-user
documentation provided with the redistribution and/or in the software itself an
acknowledgment equivalent to the following: "This product includes software developed
by the SAXPath Project (http://www.saxpath.org/)." Alternatively, the acknowledgment
may be graphical using the logos available at http://www.saxpath.org/.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE SAXPath AUTHORS OR THE PROJECT
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This software consists of voluntary
contributions made by many individuals on behalf of the SAXPath Project and was
originally created by bob mcwhirter and James Strachan . For more information on the
SAXPath Project, please see http://www.saxpath.org/.
```

Sun 社

このプログラムには、Sun 社提供のサード・パーティ・コードが含まれています。Sun 社のライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（Sun ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはないことに注意してください。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、Sun ソフトウェアは現状のままでオラクル社から提供されるものであり、いかなる種類の保証またはサポートもオラクル社または Sun 社から提供されません。

Java ロゴ

W3C DOM

このプログラムには、World Wide Web Consortium（W3C）提供のサード・パーティ・コードが含まれています。W3C のライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（W3C ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはないことに注意してください。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、W3C ソフトウェアは現状のままでオラクル社から提供されるものであり、いかなる種類の保証またはサポートもオラクル社または W3C から提供されません。

W3C ライセンス

W3C® SOFTWARE NOTICE AND LICENSE
http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
This work (and included software, documentation such as READMEs, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions.

Permission to copy, modify, and distribute this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications:

The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.
Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, the W3C Software Short Notice should be included (hypertext is preferred, text is permitted) within the body of any redistributed or derivative code.
Notice of any changes or modifications to the files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)
THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO,

索引

数字

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

あ

い

え

お

か

き

く

こ

さ

し

す