BEA WebLogic Server™

BEA WebLogic Express™

WebLogic JTA プログラマーズ ガイド

BEA WebLogic Server バージョン 6.1
マニュアルの日付：2002 年 6 月 24 日

著作権

限定的権利条項

商標または登録商標

WebLogic JTA プログラマーズ ガイド

パート番号	マニュアルの日付	ソフトウェアのバージョン
なし	2002 年 6 月 24 日	BEA WebLogic Server 6.1

目次

このマニュアルの内容

1. トランザクションについて

このマニュアルの内容

このマニュアルでは、BEA WebLogic Server™ 環境で動作する EJB および RMI アプリケーションでのトランザクションの使い方について説明します。

このマニュアルの内容は以下のとおりです。

- 第 1 章「トランザクションについて」では、WebLogic Server 環境で動作する EJB および RMI アプリケーションにおけるトランザクションについて説明します。また、エンタープライズ アプリケーションの分散トランザクションと 2 フェーズ コミット プロトコルについても説明します。
- 第 2 章「トランザクションのコンフィグレーションと管理」では、WebLogic Server 環境におけるトランザクションの管理方法について説明します。
- 第 3 章「トランザクション サービス」では、WebLogic Server トランザクション サービスについて説明します。
- 第 4 章「Java Transaction API と BEA WebLogic の拡張機能」では、Java Transaction API (JTA) の概要について簡単に説明します。
- 第 5 章「EJB アプリケーションのトランザクション」では、EJB アプリケーションにおけるトランザクションの実装方法について説明します。
- 第 6 章「RMI アプリケーションのトランザクション」では、RMI アプリケーションにおけるトランザクションの実装方法について説明します。
- 第 7 章「WebLogic Server でのサードパーティ JDBC XA ドライバの使い方」では、サードパーティ製の XA ドライバをコンフィグレーションしてトランザクションで使用する方法について説明します。
- 第 8 章「WebLogic Server XA リソース プロバイダの要件」では、WebLogic Server で分散トランザクションに参加する XA リソースの要件について説明します。
- 第 9 章「トランザクションのトラブルシューティング」では、JTA を使用したアプリケーションのトラブルシューティング タスクの実行方法について説明します。

対象読者

このマニュアルは、WebLogic Server プラットフォームで動作するトランザクション対応 Java アプリケーションの構築に関心があるアプリケーション開発者を対象としています。WebLogic Server プラットフォーム、Java™ 2, Enterprise Edition (J2EE) プログラミング、およびトランザクション処理の概念に読者が精通していることを前提として書かれています。

e-docs Web サイト

BEA 製品のドキュメントは、BEA の Web サイトで入手できます。BEA ホームページの [製品のドキュメント] をクリックするか、WebLogic Server 製品ドキュメント ページ（http://edocs.beasys.co.jp/e-docs/wls61）を直接表示してください。

このマニュアルの印刷方法

Web ブラウザの [ファイル｜印刷] オプションを使用すると、Web ブラウザからこのマニュアルのメイン トピックを一度に 1 つずつ印刷できます。

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

Adobe Acrobat Reader は、Adobe の Web サイト（http://www.adobe.co.jp）から無料で入手できます。

サポート情報

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

電子メールのメッセージには、ご使用のソフトウェア名とバージョン名、およびマニュアルのタイトルと作成日付をお書き添えください。本バージョンの BEA WebLogic Server について不明な点がある場合、または BEA WebLogic Server のインストールおよび動作に問題がある場合は、BEA WebSUPPORT（www.bea.com）を通じて BEA カスタマ サポートまでお問い合わせください。カスタマ サポートへの連絡方法については、製品パッケージに同梱されているカスタマ サポート カードにも記載されています。

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

- お名前、電子メール アドレス、電話番号、ファクス番号
- 会社の名前と住所
- お使いの機種とコード番号
- 製品の名前とバージョン
- 問題の状況と表示されるエラー メッセージの内容

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
〔Ctrl〕+ 〔Tab〕	同時に押すキーを示す。
斜体	強調または本のタイトルを示す。

等幅テキスト	コード サンプル、コマンドとそのオプション、Java クラス、データ型、ディレクトリ、およびファイル名とその拡張子を示す。等幅テキストはキーボードから入力するテキストも示す。
	例 :
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

斜体の等幅テキスト	コード内の変数を示す。 例 : `String *CustomerName*;`
すべて大文字のテキスト	デバイス名、環境変数、および論理演算子を示す。 例 : LPT1 BEA_HOME OR
{ }	構文内の複数の選択肢を示す。

表記法	適用
[ ]	構文内の任意指定の項目を示す。 例 : `java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]`
|	構文の中で相互に排他的な選択肢を区切る。 例 : `java weblogic.deploy [list|deploy|undeploy|update] password {application} {source}`
...	コマンドラインで以下のいずれかを示す。 ■ 引数を複数回繰り返すことができる。 ■ 任意指定の引数が省略されている。 ■ パラメータや値などの情報を追加入力できる。
. . .	コード サンプルまたは構文で項目が省略されていることを示す。

1 トランザクションについて

この章では、WebLogic Server アプリケーション内の WebLogic トランザクションの概要を説明します。

- トランザクションの ACID プロパティ
- サポートされているプログラミング モデル
- サポートされている API モデル
- 分散トランザクションと 2 フェーズ コミット プロトコル
- ビジネス トランザクションのサポート
- どのような状況で使用するのか
- トランザクションを使用しない状況
- トランザクションの仕組み
- トランザクションのサンプル コード

トランザクションの ACID プロパティ

WebLogic Server システムの最も基本的な機能の 1 つはトランザクション管理です。トランザクションは、データベース トランザクションを正確に完了させるための手段です。また、トランザクションによって、以下のような高性能トランザクションのすべての ACID プロパティがデータベース トランザクションに備わります。

- 原子性 ― トランザクションによるデータベースへの変更はすべて永続的に行われ、そうでない場合にはすべての変更がロールバックされます。
- 一貫性 ― トランザクションが成功すると、データベースは直前の有効な状態から新規の有効な状態へ変換されます。
- 隔離性 ― トランザクションによるデータベースへの変更は、トランザクションが処理を完了するまでほかの処理からは見えません。

- 持続性 ― トランザクションによるデータベースへの変更は、システムまたは媒体の障害が発生しても失われません。

WebLogic Server はデータベースの更新が正確に行われるようにする優れたインフラストラクチャを備えているので、さまざまなリソース マネージャにまたがる場合であってもトランザクションの整合性が保護されます。1 つの処理でエラーが起きた場合は、関連するすべての処理がロールバックされます。

サポートされているプログラミング モデル

WebLogic Server では、Sun Microsystems の Java™ 2 Enterprise Edition（J2EE）プログラミング モデルのトランザクションがサポートされます。また、WebLogic Server では、エンタープライズ JavaBeans 仕様 2.0（Sun Microsystems 発行）に準拠しているエンタープライズ JavaBean を使用する Java アプリケーションのトランザクションもフルにサポートされています。さらに、WebLogic Server では、Java Transaction API（JTA）仕様 1.0.1（これも Sun Microsystems の発行）もサポートされています。

サポートされている API モデル

WebLogic Server では、Sun Microsystems の Java Transaction API（JTA）がサポートされています。JTA は、以下のアプリケーションで使用されます。

- WebLogic Server EJB コンテナ内のエンタープライズ JavaBean（EJB）アプリケーション
- WebLogic Server インフラストラクチャ内の Remote Method Invocation（RMI）アプリケーション

JTA については、以下の資料を参照してください。

- `javax.transaction` および `javax.transaction.xa` パッケージ API
- Sun Microsystems 発行の Java Tra nsaction API 仕様

分散トランザクションと 2 フェーズ コミット プロトコル

WebLogic Server では、エンタープライズ アプリケーションの分散トランザクションと 2 フェーズ コミット プロトコルがサポートされます。分散トランザクションとは、複数のリソース マネージャ（データベースなど）が調和的に更新されるトランザクションのことです。一方、ローカル トランザクションは、内部的に API 呼び出しを調整する 1 つのリソース マネージャに対して、トランザクションを開始およびコミットします。このため、トランザクション マネージャはありません。2 フェーズ コミット プロトコルは、複数のリソース マネージャにまたがって 1 つのトランザクションを調整する手段です。これにより、トランザクションによる更新を関連データベースのすべてにコミットするか、またはすべてのデータベースから完全にロールバックし、トランザクションによる状態の前の状態に戻すことで、データの完全性が保証されます。つまり、関連するすべてのデータベースが更新されるか、またはどのデータベースも更新されないか、のどちらかだということです。

分散トランザクションには、以下のものが関与します。

- トランザクション オリジネータ ― トランザクションを開始します。トランザクション オリジネータとしては、ユーザ アプリケーション、エンタープライズ JavaBean、または JMS クライアントがあります。
- トランザクション マネージャ ― アプリケーション プログラムに代わってトランザクションを管理します。トランザクション マネージャは、トランザクションの開始と完了を行うアプリケーション プログラムからのコマンドを、それらのトランザクションに関わるすべてのリソース マネージャと通信することで調整します。リソース マネージャがトランザクション中に失敗した場合、トランザクション マネージャはリソース マネージャが保留中のトランザクションをコミットするかロールバックするかを決定するのを支援します。
- 回復可能なリソース ― データの永続ストレージを提供します。ほとんどの場合はリソースとしてデータベースが使用されます。
- リソース マネージャ ― 情報やプロセスへのアクセス手段を提供する。リソース マネージャとしてはトランザクション対応 JDBC ドライバがよく使用されます。リソース マネージャはトランザクションの機能とアクションの永続性を提供します（これらは分散トランザクションの中でアクセスおよび管

理されるエンティティ）。リソース マネージャと特定のリソースの通信は、**トランザクション ブランチ**と呼ばれます。

2 フェーズ コミット プロトコルの最初のフェーズは準備フェーズと呼ばれます。要求された更新がトランザクション ログ ファイルに記録されたら、リソースはリソース マネージャを通じて変更の準備ができていることを示さなければなりません。リソースは、更新をコミットするのか、または前の状態にロールバックするのかを意思表示できます。第 2 フェーズで何が行われるかは、リソースの意思表示によって決まります。すべてのリソースがコミットを支持すると、トランザクションに関わっているすべてのリソースが更新されます。1 つまたは複数のリソースがロールバックを支持した場合は、トランザクションに関わっているすべてのリソースが前の状態にロールバックされます。

ビジネス トランザクションのサポート

WebLogic JTA では、ビジネス トランザクションが以下のようにサポートされています。

- クライアント アプリケーションがトランザクションを開始したときにユニークなトランザクション識別子が作成されます。
- トランザクションが表すビジネス プロセスを説明するトランザクション名（省略可能）がサポートされます。トランザクション名を使用すると統計やエラー メッセージがもっと有意義になります。
- トランザクションに関わっており、したがってトランザクションのコミット段階で調整しなければならないオブジェクトを追跡するために WebLogic Server インフラストラクチャが連係して機能します。
- トランザクションでアクセスされるときに、リソース マネージャ（たいていはデータベース）が通知を受けます。通知を受けたリソース マネージャは、トランザクションが終了するまでアクセス対象のレコードをロックします。
- トランザクションの完了時に 2 フェーズ コミットが調整される。この調整により、トランザクションに関わっているすべてのリソースで更新が同時にコミットされます。コミットの調整は、Open Group の XA プロトコルを使用して更新されるすべてのデータベースで行われます。この規格は、普及している多くのリレーショナル データベースでサポートされています。
- トランザクションを停止する必要があるときにロールバック処理が実行されます。

- 障害が起きたときに回復処理が実行されます。クラッシュの時点でアクティブだったトランザクションが確認され、そのトランザクションをロールバックするのかコミットするのかが判断されます。
- トランザクションのタイムアウトが管理されます。ビジネス処理にあまりにも多くの時間がかかる場合、またはビジネス処理が障害のために途中で終了している場合は、トランザクションのタイムアウトが自動的に発行され、データベース ロックなどのリソースが解放されます。

どのような状況で使用するのか

トランザクションは、以下のような状況で使用するのが適してます。各状況は、WebLogic Server システムでサポートされているトランザクション モデルを表現してます。分散トランザクションは、ユーザ入力の複数の画面にまたがってはいけないことに注意してください。もっと複雑でハイレベルなトランザクションは、連続する分散トランザクションで実装する必要があります。

たとえば、インターネットベースのオンライン ショッピング カート アプリケーションを考えてください。クライアント アプリケーションのユーザは、オンライン カタログを参照し、複数の商品を選択します。購入するすべての商品を選択した後、ユーザは精算へと進み、クレジット カードの情報を入力します。クレジット カードの確認が失敗した場合、ショッピング アプリケーションではショッピング カートにある未処理の商品をすべてキャンセルするか、対話の中で行われた購入トランザクションをすべてロールバックすることが必要となります。

- オブジェクトに対する 1 回のクライアント呼び出しで、オブジェクトによりデータベースのデータが複数回にわたって編集されます。いずれかの編集が失敗した場合、オブジェクトではすべての編集をロールバックすることが必要となります。この状況では、個々のデータベース編集は必ずしも EJB または RMI の呼び出しではありません。アプレットなどのクライアントでは、JNDI を使用して `Transaction` オブジェクトおよび `TransactionManager` オブジェクトの参照を取得し、トランザクションを開始することができます。

  たとえば、銀行のアプリケーションを考えてください。クライアントでは、出納オブジェクトに対して送金処理を要求します。送金処理においては、出納オブジェクトは銀行のデータベースで以下の呼び出しを行わなければなりません。

- 1 つの口座で送金メソッドを呼び出します。
- 別の口座で入金メソッドを呼び出します。

データベースで入金が失敗した場合、アプリケーションでは直前の送金をロールバックすることが必要となります。

- クライアント アプリケーションで、サーバ アプリケーションで管理されるオブジェクトとの対話を必要とし、特定のオブジェクト インスタンスに対する複数の呼び出しを行う必要があります。その対話には、以下のような特徴の 1 つまたは複数が備わっています。
  - データは、連続する各呼び出しの間または後でメモリにキャッシュされるか、データベースに書き込まれます。
  - データは、対話の終わりにデータベースに書き込まれます。
  - クライアント アプリケーションでは、各呼び出し間のインメモリ コンテキストを保持するオブジェクトが必要となります。つまり、連続する各呼び出しでは、対話の全体を通してメモリに保持されているデータが使用されます。
  - 対話の最後に、クライアント アプリケーションでは対話の途中または最後に行われたデータベースへの書き込み処理をすべて取り消す機能が必要となります。

トランザクションを使用しない状況

トランザクションは常に適しているわけではありません。たとえば、一連のトランザクションに時間がかかる場合は、連続する分散トランザクションで実装します。トランザクションの不適切な使用例を示します。

- クライアント アプリケーションで複数のオブジェクトに対する呼び出しを行う必要があり、その一連の呼び出しには 1 つまたは複数のデータベースへの書き込み処理が伴っています。いずれかの呼び出しが失敗した場合、メモリまたはデータベースで書き込まれた状態はロールバックされなければならなりません。

  たとえば、旅行代理店のアプリケーションを考えてください。クライアント アプリケーションでは、たとえばフランスのストラスブールからオーストラリアのアリス スプリングスまでのような長距離の旅行を手配する必要があります。そのような長距離の旅行では、どうしても複数のフライトの予約を個

別に行わなければなりません。クライアント アプリケーションでは、旅程の各区分の予約が順番に行われます。たとえば、ストラスブールからパリ、パリからニューヨーク、ニューヨークからロサンゼルスの予約が順番に行われます。ただし、いずれかのフライトの予約ができない場合、クライアント アプリケーションではそれまでに行ったフライトの予約をすべてキャンセルする必要があります。

トランザクションの仕組み

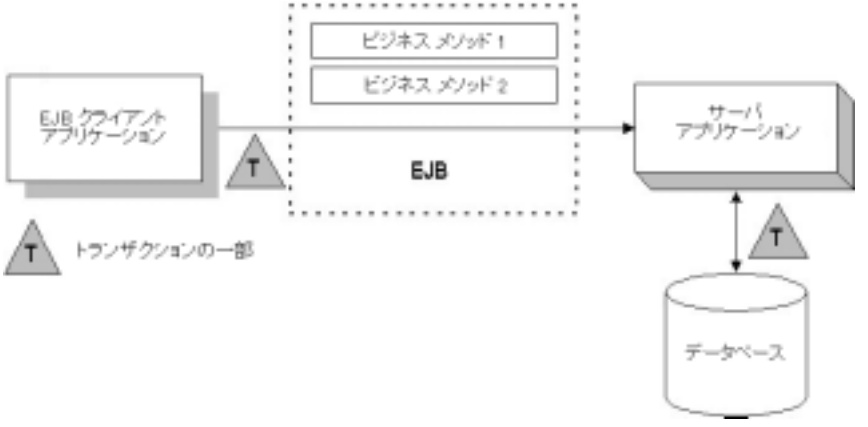
ここでは以下について説明します。

- WebLogic Server EJB アプリケーションのトランザクションの概要
- WebLogic Server RMI アプリケーションのトランザクションの概要

WebLogic Server EJB アプリケーションのトランザクションの概要

図 1-1 は、WebLogic Server EJB アプリケーションでトランザクションがどのように機能するのかを示しています。

図 1-1　WebLogic Server EJB アプリケーションでのトランザクションの仕組み



WebLogic Server では、WebLogic Server EJB アプリケーションで以下の 2 種類のトランザクションがサポートされています。

- **コンテナ管理のトランザクション**では、WebLogic Server EJB コンテナによってトランザクションの境界設定が管理されます。EJB デプロイメント記述子のトランザクション属性では、各メソッド呼び出しで WebLogic Server EJB コンテナがどのようにトランザクションを処理するのかが指定されます。デプロイメント記述子の詳細については、『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』を参照してください。
- **Bean 管理のトランザクション**では、EJB によってトランザクションの境界設定が管理されます。EJB は、`UserTransaction` オブジェクトに対する明示的なメソッド呼び出しを行って、トランザクションの開始、コミット、およびロールバックを行います。`UserTransaction` オブジェクトの詳細については、WebLogic の Javadoc を参照してください。

トランザクション イベントのシーケンスは、コンテナ管理のトランザクションと Bean 管理のトランザクションで異なります。

コンテナ管理のトランザクション

トランザクションがコンテナで管理される EJB アプリケーションの場合、基本的なトランザクションは次のように機能します。

1. EJB のデプロイメント記述子で、トランザクションの種類（`transaction-type` 要素）としてコンテナ管理の境界設定（`Container`）を指定します。
2. EJB のデプロイメント記述子で、EJB のデフォルトのトランザクション属性（`trans-attribute` 要素）を指定します。トランザクション属性としては、`NotSupported`、`Required`、`Supports`、`RequiresNew`、`Mandatory`、または `Never` を設定します。これらの設定の詳細については、Sun Microsystems が発行している EJB 仕様 2.0 のセクション 16.7.2 を参照してください。
3. 必要に応じて、EJB のデプロイメント記述子で、1 つまたは複数のメソッドの `trans-attribute` を指定します。
4. クライアント アプリケーションによって EJB のメソッドが呼び出されると、EJB コンテナはデプロイメント記述子でそのメソッドの `trans-attribute` 設定を調べます。メソッドの設定が指定されていない場合は、その EJB のデフォルトの `trans-attribute` 設定が使用されます。

5. EJB コンテナは、trans-attribute の設定に応じて適切に動作します。
   - たとえば、trans-attribute の設定が Required の場合、EJB コンテナは既存のトランザクション コンテキストでメソッドを呼び出します。クライアントがトランザクション コンテキストなしで呼び出しを行った場合、EJB コンテナはメソッドを実行する前に新しいトランザクションを開始します。
   - trans-attribute の設定が Mandatory の場合、EJB コンテナは既存のトランザクション コンテキストでメソッドを呼び出します。クライアントがトランザクション コンテキストなしで呼び出しを行った場合、EJB コンテナは javax.transaction.TransactionRequiredException 例外を送出します。
6. ビジネス メソッドの呼び出しの途中で、ロールバックが必要だと判断された場合、ビジネス メソッドでは EJBContext.setRollbackOnly メソッドが呼び出されます。このメソッドでは、メソッド呼び出しの終了時にトランザクションをロールバックしなければならないことが EJB コンテナに通知されます。

   注意： EJBContext.setRollbackOnly メソッドの呼び出しは、意味のあるトランザクション コンテキストを持つメソッドの場合だけ許可されます。
7. メソッドの実行が終了して、その結果がクライアントに送信される前に、EJB コンテナはコミットするかまたはロールバックすることによってトランザクションを完了します。トランザクションがロールバックされるのは、EJBContext.setRollbackOnly メソッドが呼び出された場合です。

Administration Console を使用して trans-timeout-seconds 要素を設定すれば、トランザクションのタイムアウトを指定できます。

Bean 管理のトランザクション

トランザクションの境界設定が Bean で管理される EJB アプリケーションの場合、基本的なトランザクションは次のように機能します。

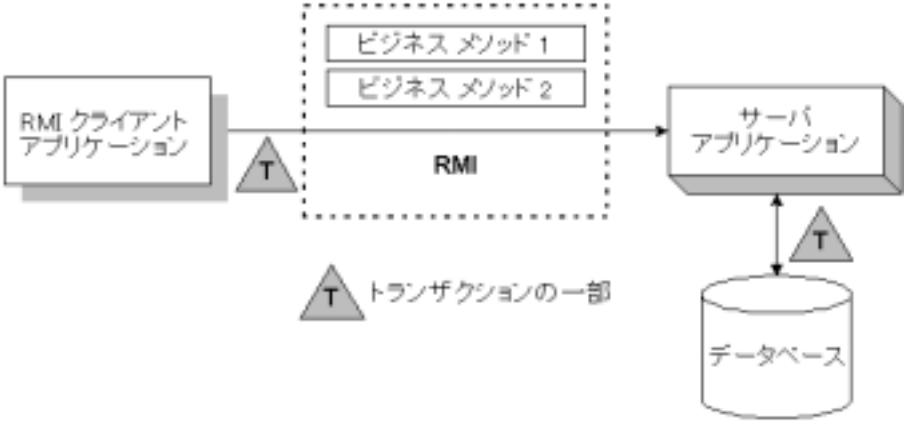
1. EJB のデプロイメント記述子で、トランザクションの種類（transaction-type 要素）として Bean 管理の境界設定（Bean）を指定します。

2. クライアント アプリケーションは、JNDI を使用して、WebLogic Server ドメインの `UserTransaction` オブジェクトへのオブジェクト参照を取得します。
3. クライアント アプリケーションは、`UserTransaction.begin` メソッドを使用してトランザクションを開始し、EJB コンテナを通じて EJB への要求を発行します。EJB でのすべての処理は、トランザクションのスコープ内で実行されます。
   - いずれかの処理の呼び出しで例外が生成された（明示的に、または通信エラーの結果として）場合は、その例外を捕捉し、`UserTransaction.rollback` メソッドを使用してトランザクションをロールバックすることができます。
   - 例外が生成されない場合、クライアント アプリケーションは `UserTransaction.commit` メソッドを使用して現在のトランザクションをコミットします。このメソッドは、トランザクションを終了して処理を開始します。トランザクションは、そのトランザクションに関わっているすべてのリソースがコミットに同意した場合だけコミットされます。
4. `UserTransaction.commit` メソッドが呼び出されると、EJB コンテナはトランザクションを完了するためにトランザクション マネージャを呼び出します。
5. トランザクション マネージャの役割は、リソース マネージャと協力してデータベースを更新することです。

WebLogic Server RMI アプリケーションのトランザクションの概要

図 1-2 は、WebLogic Server RMI アプリケーションでトランザクションがどのように機能するのかを示しています。

図 1-2 WebLogic Server **RMI アプリケーションでのトランザクションの仕組み**



RMI のクライアント アプリケーションとサーバ アプリケーションの場合、基本的なトランザクションは次のように機能します。

1. アプリケーションは、JNDI を使用して、WebLogic Server ドメインの `UserTransaction` オブジェクトへのオブジェクト参照を返します。

   オブジェクト参照を取得すると、アプリケーションとそのオブジェクトが対話状態に入ります。対話状態は、コミットかロールバックによってトランザクションが完了するまで続きます。インスタンス化された RMI オブジェクトは、解放されるまで（たいていはサーバの停止時）メモリ内でアクティブな状態を維持します。トランザクションの間は、WebLogic Server インフラストラクチャによって非アクティベーションやアクティベーションは実行されません。

2. クライアント アプリケーションは、`UserTransaction.begin` メソッドを使用してトランザクションを開始し、サーバ アプリケーションへの要求を発行します。サーバ アプリケーションでのすべての処理は、トランザクションのスコープ内で実行されます。

   - いずれかの処理の呼び出しで例外が生成された（明示的に、または通信エラーの結果として）場合は、その例外を捕捉し、`UserTransaction.rollback` メソッドを使用してトランザクションをロールバックすることができます。
   - 例外が生成されない場合、クライアント アプリケーションは `UserTransaction.commit` メソッドを使用して現在のトランザクションをコミットします。このメソッドは、トランザクションを終了して処理

を開始します。トランザクションは、そのトランザクションに関わっているすべてのリソースがコミットに同意した場合だけコミットされます。

3. `UserTransaction.commit` メソッドが呼び出されると、WebLogic Server はトランザクションを完了するためにトランザクション マネージャを呼び出します。
4. トランザクション マネージャの役割は、リソース マネージャと協力してデータベースを更新することです。

詳細については、第 6 章「RMI アプリケーションのトランザクション」を参照してください。

トランザクションのサンプル コード

ここでは以下について説明します。

- トランザクションのサンプル EJB コード
- トランザクションのサンプル RMI コード

トランザクションのサンプル EJB コード

この節では、EJB アプリケーションのクラスから抜粋したサンプル コードを利用して段階的な説明を行います。ここでは以下について説明します。

- パッケージをインポートする
- JNDI を使用してオブジェクト参照を返す
- トランザクションの開始
- トランザクションの完了

サンプル コードでは、Bean 管理によるトランザクションの境界設定で `UserTransaction` オブジェクトが使用されます。この Bean のデプロイメント記述子では、トランザクションの種類（`transaction-type` 要素）として Bean 管理によるトランザクションの境界設定（`Bean`）が指定されます。

注意： 以下のサンプル コードは、WebLogic Server に付属するサンプル アプリケーションから抜粋したものではありません。単に、EJB アプリケーションでの `UserTransaction` オブジェクトの利用を説明するために用意されたものです。

グローバル トランザクションでは、EJB が動作している WebLogic Server インスタンス上のローカルな TxDataSource からのデータベース接続を使用します。リモート WebLogic Server インスタンス上の TxDataSource からの接続は使用しないでください。

パッケージをインポートする

コード リスト 1-1 は、トランザクションで必要なパッケージのインポートを示しています。必要なパッケージには以下のものがあります。

- `javax.transaction.UserTransaction`。このオブジェクトに関連付けられているメソッドのリストについては、オンラインの Javadoc を参照してください。
- システム例外。例外のリストについては、オンラインの Javadoc を参照してください。

コード リスト 1-1　パッケージをインポートする

```
import javax.naming.*;
import javax.transaction.UserTransaction;
import javax.transaction.SystemException;
import javax.transaction.HeuristicMixedException
import javax.transaction.HeuristicRollbackException
import javax.transaction.NotSupportedException
import javax.transaction.RollbackException
import javax.transaction.IllegalStateException
import javax.transaction.SecurityException
import java.sql.*;
import java.util.*;
```

これらのクラスがインポートされた後に、`UserTransaction` オブジェクトのインスタンスが null に初期化されます。

JNDI を使用してオブジェクト参照を返す

コード リスト 1-2 は、JNDI を使用したオブジェクト参照のルックアップを示しています。

コード リスト 1-2 JNDI ルックアップの実行

```
Context ctx = null;
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
         "weblogic.jndi.WLInitialContextFactory");

// WebLogic Server の パラメータ
// 環境に合わせて適切なホスト名、ポート番号、
// ユーザ名、およびパスワードに置き換えること
env.put(Context.PROVIDER_URL, "t3://localhost:7001");
env.put(Context.SECURITY_PRINCIPAL, "Fred");
env.put(Context.SECURITY_CREDENTIALS, "secret");

ctx = new InitialContext(env);

UserTransaction tx = (UserTransaction)
  ctx.lookup("javax.transaction.UserTransaction");
```

トランザクションの開始

コード リスト 1-3 は、`UserTransaction` オブジェクトを取得し、`javax.transaction.UserTransaction.begin()` メソッドを呼び出すことによるトランザクションの開始を示しています。このメソッド呼び出しの後、トランザクションが完了するまでに行われるデータベースの処理は、このトランザクションのスコープ内に存在します。

コード リスト 1-3 トランザクションの開始

```
UserTransaction tx = (UserTransaction)
  ctx.lookup("javax.transaction.UserTransaction");
tx.begin();
```

トランザクションの完了

コード リスト 1-4 は、このトランザクションのスコープ内で試行されたいずれかのデータベース処理で例外が送出されたかどうかに依存するトランザクションの完了を示しています。

- いずれかのデータベース処理で例外が送出された場合は、アプリケーションによって `javax.transaction.UserTransaction.rollback()` メソッドが呼び出されます。
- 例外が送出されなかった場合は、アプリケーションによって `javax.transaction.UserTransaction.commit()` メソッドが呼び出され、すべてのデータベース処理が正常に終了した後にトランザクションのコミットが試行されます。このメソッドが呼び出されると、トランザクションが終了し、処理が開始されて、WebLogic Server EJB コンテナによってトランザクションを完了するためにトランザクション マネージャが呼び出されます。トランザクションは、そのトランザクションに関わっているすべてのリソースがコミットに同意した場合だけコミットされます。

コード リスト 1-4 トランザクションの完了

```
tx.commit();
// または
tx.rollback();
```

トランザクションのサンプル RMI コード

この節では、RMI アプリケーションのクラスから抜粋したサンプル コードを利用して段階的な説明を行います。ここでは以下について説明します。

- パッケージをインポートする
- JNDI を使用して UserTransaction オブジェクトへのオブジェクト参照を返す
- トランザクションの開始
- トランザクションの完了

サンプル コードでは、RMI トランザクションで UserTransaction オブジェクトが使用されます。RMI アプリケーションでのトランザクションの使用のガイドラインについては、第 6 章「RMI アプリケーションのトランザクション」を参照してください。

注意： 以下のサンプル コードは、WebLogic Server に付属するサンプル アプリケーションから抜粋したものではありません。単に、RMI アプリケーションでの UserTransaction オブジェクトの利用を説明するために用意されたものです。

パッケージをインポートする

コード リスト 1-5 は、必要なパッケージのインポートを示しています。必要なパッケージには、トランザクションの処理に使用する以下のパッケージが含まれます。

- javax.transaction.UserTransaction。このオブジェクトに関連付けられているメソッドのリストについては、オンラインの Javadoc を参照してください。
- システム例外。例外のリストについては、オンラインの Javadoc を参照してください。

コード リスト 1-5 パッケージをインポートする

```
import javax.naming.*;
import java.rmi.*;
import javax.transaction.UserTransaction;
import javax.transaction.SystemException;
import javax.transaction.HeuristicMixedException
import javax.transaction.HeuristicRollbackException
import javax.transaction.NotSupportedException
import javax.transaction.RollbackException
import javax.transaction.IllegalStateException
import javax.transaction.SecurityException
import java.sql.*;
import java.util.*;
```

これらのクラスがインポートされた後に、UserTransaction オブジェクトのインスタンスが null に初期化されます。

JNDI を使用して UserTransaction オブジェクトへのオブジェクト参照を返す

コード リスト 1-6 は、適切な WebLogic Server ドメインの UserTransaction オブジェクトに対するオブジェクト参照を返すための JNDI ツリーの検索を示しています。

注意： オブジェクト参照を取得すると、アプリケーションとそのオブジェクトが対話状態に入ります。対話状態は、コミットかロールバックによってトランザクションが完了するまで続きます。インスタンス化された RMI オブジェクトは、解放されるまで（たいていはサーバの停止時）メモリ内でアクティブな状態を維持します。トランザクションの間は、WebLogic Server インフラストラクチャによって非アクティベーションやアクティベーションは実行されません。

コード リスト 1-6 JNDI ルックアップの実行

```
Context ctx = null;
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
        "weblogic.jndi.WLInitialContextFactory");

// WebLogic Server の パラメータ
// 環境に合わせて適切なホスト名、ポート番号、
// ユーザ名、およびパスワードに置き換えること
env.put(Context.PROVIDER_URL, "t3://localhost:7001");
env.put(Context.SECURITY_PRINCIPAL, "Fred");
env.put(Context.SECURITY_CREDENTIALS, "secret");

ctx = new InitialContext(env);

UserTransaction tx = (UserTransaction)
  ctx.lookup("javax.transaction.UserTransaction");
```

トランザクションの開始

コード リスト 1-7 は、javax.transaction.UserTransaction.begin() メソッドを呼び出すことによるトランザクションの開始を示しています。このメソッド呼び出しの後、トランザクションが完了するまでに行われるデータベースの処理は、このトランザクションのスコープ内に存在します。

コード リスト 1-7 トランザクションの開始

```
UserTransaction tx = (UserTransaction)
  ctx.lookup("javax.transaction.UserTransaction");
tx.begin();
```

トランザクションの完了

コード リスト 1-8 は、このトランザクションのスコープ内で試行されたいずれかのデータベース処理で例外が送出されたかどうかに依存するトランザクションの完了を示しています。

- いずれかのデータベース処理で例外が送出された場合は、アプリケーションによって javax.transaction.UserTransaction.rollback() メソッドが呼び出されます。
- 例外が送出されなかった場合は、アプリケーションによって javax.transaction.UserTransaction.commit() メソッドが呼び出され、すべてのデータベース処理が正常に終了した後にトランザクションのコミットが試行されます。このメソッドが呼び出されると、トランザクションが終了し、処理が開始されて、WebLogic Server によってトランザクションを完了するためにトランザクション マネージャが呼び出されます。トランザクションは、そのトランザクションに関わっているすべてのリソースがコミットに同意した場合だけコミットされます。

コード リスト 1-8 トランザクションの完了

```
tx.commit();
// または
tx.rollback();
```

2 トランザクションのコンフィグレーションと管理

この章では、トランザクションに関連する一般的な管理作業について概説します。JTA コンフィグレーション作業の概要については、『管理者ガイド』の「トランザクションの管理」を参照してください。特定のコンフィグレーション属性および手順については、Administration Console オンライン ヘルプの JTA に関するトピックを参照してください。

- トランザクションのコンフィグレーション
- トランザクションのモニタ
- トランザクション リソース マネージャの追加

トランザクションのコンフィグレーション

Administration Console は、WebLogic JTA などの WebLogic Server の機能をコンフィグレーションするために使用するインタフェースを備えています。Administration Console を起動する手順については、「WebLogic Server とクラスタのコンフィグレーション」を参照してください。コンフィグレーション プロセスでは、属性の値を指定する必要があります。それらの属性によって、以下のようなトランザクション環境が定義されます。

- トランザクションのタイムアウトと制限
- トランザクション マネージャの動作

また、EJB、JDBC、JMS など、トランザクションに参加可能な J2EE コンポーネントの管理についてもよく理解しておく必要があります。

トランザクションのモニタ

ロギング、統計、およびモニタの機能を使用して、サーバ上のトランザクションをモニタできます。Administration Console では、それらの機能をコンフィグレーションしたり、結果出力を表示したりできます。

ロギング

トランザクション ログは、複数のファイルで構成されます。各ファイルの名前は、ファイル システムでの位置を示すプレフィックス（`TransactionLogFilePrefix` 属性で定義）、サーバ名、ユニークな数値のサフィックス、およびファイル拡張子で構成されます。`TransactionLogFilePrefix` 属性は、ドメイン内の各サーバで設定します。トランザクション ログで消費されるスペースの全体量は、ファイル システムの利用可能なディスク スペースによってのみ制限されます。サーバ ロギング属性の設定の詳細については、Administration Console オンライン ヘルプのサーバに関するトピックを参照してください。トラブルシューティングとデバッグでロギングを利用する方法については、第 9 章「トランザクションのトラブルシューティング」の「トランザクション ログ」を参照してください。

統計

WebLogic Server では、サーバ、リソース、およびトランザクション名に基づいてまとめられたトランザクションの統計が管理されます。統計を表示する方法については、Administration Console オンライン ヘルプの JTA に関するトピックを参照してください。トラブルシューティングとデバッグで統計を利用する方法については、第 9 章「トランザクションのトラブルシューティング」の「トランザクションの統計」を参照してください。

モニタ

Administration Console を使用して、進行中のトランザクションをモニタできます。トランザクションの情報は名前やリソースに基づいて表示でき、アクティブなすべてのトランザクションという基準で表示することもできます。トランザクションのモニタの詳細については、Administration Console オンライン ヘルプのサーバに関するトピックを参照してください。トラブルシューティングでデータのモニタを利用する方法については、第 9 章「トランザクションのトラブルシューティング」の「トランザクションのモニタ」を参照してください。

トランザクション リソース マネージャの追加

トランザクション リソース マネージャは、情報やプロセスへアクセスする手段を提供します。リソース マネージャとしてはトランザクション対応 JDBC ドライバがよく使用されます。JDBC ドライバを追加するときには、JTA と連係して適切に機能できるようにドライバ プロパティをコンフィグレーションする必要があります。JDBC のコンフィグレーションのガイドラインについては、『管理者ガイド』の「トランザクションの管理」を参照してください。

3 トランザクション サービス

この章では、WebLogic Server システム向けのトランザクション対応アプリケーションを記述するために必要な情報を提供します。

- トランザクション サービスについて
- 機能と制限
- EJB アプリケーションでのトランザクション サービス
- RMI アプリケーションでのトランザクション サービス

トランザクション サービスについて

WebLogic Server は、EJB アプリケーションと RMI アプリケーションのトランザクションをサポートするトランザクション サービスを備えています。WebLogic Server EJB コンテナでは、そのトランザクション サービスによって、Sun Microsystems 発行の EJB 仕様 2.0 で説明されているトランザクション サービスの実装が提供されます。

EJB と RMI のアプリケーションについて、WebLogic Server では Java アプリケーションの Java Transaction API（JTA）を実装する Sun Microsystems の `javax.transaction` パッケージおよび `javax.transaction.xa` パッケージも提供されます。JTA の詳細については、Sun Microsystems 発行 の Java Transaction API（JTA）仕様 1.0.1 を参照してください。トランザクションの境界を設定するためにアプリケーションで使用される `UserTransaction` オブジェクトの詳細については、WebLogic Server Javadoc を参照してください。

機能と制限

以下の節では、EJB と RMI のアプリケーションをサポートするトランザクション サービスの機能と制限について説明します。

軽量クライアントと委託コミット

軽量クライアントは、可用性が一定ではなく、管理のされていない、1 ユーザ対象のデスクトップ システムで動作します。デスクトップ システムは、使用されないときは所有者によって電源が落とされます。そのような管理のされていない、1 ユーザ対象のデスクトップ システムでは、トランザクションの調整などのネットワーク機能を行うべきではありません。特に、管理のされていないシステムはサーバ リソースが関わるトランザクションにおいて、障害の影響を受けず原子性、一貫性、隔離性、および持続性という ACID プロパティを維持することに責任を持つべきではありません。WebLogic Server のリモート クライアントは軽量クライアントです。

トランザクション サービスを利用すると、軽量クライアントでは委託コミットを実行できます。つまり、軽量クライアントでは、トランザクションの調整をサーバ マシン上のトランザクション マネージャにまかせてトランザクションを開始および終了することができるのです。クライアント アプリケーションは、ローカルのトランザクション サーバを必要としません。EJB クライアントまたは RMI クライアントで使用される `UserTransaction` のリモート実装によって、トランザクション調整の実際の作業がサーバ上のトランザクション マネージャに委託されます。

クライアントが開始するトランザクション

アプレットなどのクライアントでは、JNDI を使用して `UserTransaction` オブジェクトおよび `TransactionManager` オブジェクトの参照を取得できます。クライアントでは、いずれかのオブジェクト参照を使用してトランザクションを開始できます。現在のスレッドの `Transaction` オブジェクトを取得するには、クライアント プログラムは `((TransactionManager)tm).getTransaction()` メ

ソッドを呼び出さなければなりません。JNDI から返される Transaction オブジェクトでは、UserTransaction インタフェースと TransactionManager インタフェースの両方がサポートされます。

トランザクションの整合性

トランザクションの動作を管理すると、そのトランザクションに関わるすべてのトランザクション オブジェクトでトランザクション要求の処理が完了しない限り commit が成功しないのでトランザクションの整合性が保証されます。トランザクション サービスでは、Open Group によって定義されている要求と応答のプロセス間通信モデルで提供されるものと同等のトランザクション動作の管理が提供されます。

トランザクションの終了

WebLogic Server では、トランザクションはそのトランザクションを生成したクライアントによってのみ終了することができます。

注意： クライアントは、別のオブジェクトのサービスを要求したサーバ オブジェクトの場合もあります。

フラット トランザクション

WebLogic Server では、フラット トランザクション モデルが実装されています。ネスト トランザクションはサポートされていません。

トランザクション サービスのトランザクション処理との関係

トランザクション サービスは、さまざまなトランザクション処理のサーバ、インタフェース、プロトコル、および規格と以下のように関連しています。

- **Open Group XA インタフェースのサポート**。Open Group Resource Manager は 2 フェーズ コミット プロトコルを Open Group XA インタフェースを通じて管理できるようにすることで分散トランザクションに関与できるリソース マネージャです。WebLogic Server では Open Group Resource Manager との対話がサポートされています。
- **OSI TP プロトコルのサポート。**Open Systems Interconnect Transaction Processing（OSI TP）は国際標準化機構（ISO）によって定義されたトランザクション対応プロトコルです。WebLogic Server では OSI TP トランザクションとの対話はサポートされていません。
- **LU 6.2 プロトコルのサポート。**Systems Network Architecture（SNA）LU 6.2 は IBM によって定義されたトランザクション対応のプロトコル。WebLogic Server では LU 6.2 トランザクションとの対話はサポートされていません。
- **ODMG 規格のサポート。**ODMG-93 は Object Database Management Group（ODMG）によって定義された規格であり、Object Database Management System にアクセスするための移植可能なインタフェースを規定しています。WebLogic Server では、ODMG トランザクションとの対話はサポートされていません。

マルチスレッド トランザクション クライアントのサポート

WebLogic Server では、マルチスレッド トランザクション クライアントがサポートされています。クライアントでは、複数のスレッドで同時にトランザクション要求を行うことができます。

制約

トランザクション サービスには以下の制約があります。

- WebLogic Server では、クライアントまたはサーバ オブジェクトは別のトランザクションに参加しているオブジェクトのメソッドを呼び出すことができません。クライアントまたはサーバによってそのようなメソッド呼び出しが発行された場合は例外が返されます。

- WebLogic Server では、Java Transaction API（Java アプリケーション用）のサード パーティ実装を使用するクライアントはサポートされていません。

トランザクションのスコープ

トランザクションのスコープは、そのトランザクションが実行される環境を表します。WebLogic Server では、スタンドアロン サーバでのトランザクション、クラスタ化されていないサーバ間でのトランザクション、およびドメイン内のクラスタ化されたサーバ間でのトランザクションがサポートされています。複数のドメインにまたがるトランザクションはサポートされていません。

EJB アプリケーションでのトランザクション サービス

WebLogic Server EJB コンテナでは、WebLogic Server EJB アプリケーションでの以下の 2 種類のトランザクションをサポートするトランザクション サービスが提供されます。

- **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では、WebLogic Server EJB コンテナによってトランザクションの境界設定が管理されます。EJB デプロイメント記述子のトランザクション属性では、各メソッド呼び出しで WebLogic Server EJB コンテナがどのようにトランザクションを処理するのかが指定されます。
- **Bean 管理のトランザクション。**Bean 管理のトランザクションでは、EJB によってトランザクションの境界設定が管理されます。EJB は、`UserTransaction` オブジェクトに対する明示的なメソッド呼び出しを行って、トランザクションの開始、コミット、およびロールバックを行います。`UserTransaction` のメソッドの詳細については、オンラインの Javadoc を参照します。

EJB アプリケーションでのトランザクション管理の概要については、「トランザクションについて」の「WebLogic Server EJB アプリケーションのトランザクションの概要」および「トランザクションのサンプル EJB コード」を参照してください。

RMI アプリケーションでのトランザクション サービス

WebLogic Server は、WebLogic Server RMI アプリケーションのトランザクションをサポートするトランザクション サービスを備えています。RMI アプリケーションでは、クライアント アプリケーションまたはサーバ アプリケーションが `UserTransaction` オブジェクトに対する明示的なメソッド呼び出しを行って、トランザクションの開始、コミット、およびロールバックが行われます。

`UserTransaction` のメソッドの詳細については、オンラインの Javadoc を参照してください。RMI アプリケーションでのトランザクション管理の概要については、「トランザクションについて」の「WebLogic Server RMI アプリケーションのトランザクションの概要」および「トランザクションのサンプル RMI コード」を参照してください。

4 Java Transaction API と BEA WebLogic の拡張機能

この章では、Java Transaction API（JTA）と BEA Systems が提供する JTA の拡張機能について概説します。

- JTA API の概要
- JTA に対する BEA WebLogic の拡張機能

JTA API の概要

WebLogic Server では、Java アプリケーションの Java Transaction API（JTA）を実装する Sun Microsystems の `javax.transaction` パッケージと `javax.transaction.xa` パッケージがサポートされています。JTA の詳細については、Sun Microsystems 発行の Java Transaction API（JTA）仕様（バージョン 1.0.1）を参照してください。`javax.transaction` インタフェースと `javax.transaction.xa` インタフェースの詳細については、JTA の Javadoc を参照してください。

JTA の構成要素は以下のとおりです。

- アプリケーションからトランザクションの境界を設定したり、トランザクションを管理したりするためのインタフェースである `javax.transaction.UserTransaction`。このインタフェースは、Java クライアント プログラムの一部として、または EJB 内で Bean 管理のトランザクションの一部として使用します。
- トランザクション マネージャがアプリケーションのトランザクションの境界を設定したり、トランザクションを管理したりするためのインタフェースである `javax.transaction.TransactionManager`。このインタフェースはコンテナ管理のトランザクションの一部として EJB コンテナによって使用され

ます。このインタフェースでは、javax.transaction.Transaction インタフェースを使用して特定のトランザクションで処理が実行されます。

- トランザクション マネージャがアプリケーション サーバにステータスおよび同期の情報を提供するためのインタフェースである javax.transaction.Status と javax.transaction.Synchronization。これらのインタフェースは、トランザクション マネージャによってのみアクセスされ、アプリケーション プログラムの一部として使用することはできません。
- トランザクション マネージャが XA 準拠リソースのリソース マネージャと連係するためのインタフェースである javax.transaction.xa.XAResource、およびトランザクション マネージャがトランザクションの識別子を取り出すためのインタフェースである javax.transaction.xa.Xid。これらのインタフェースは、トランザクション マネージャによってのみアクセスされ、アプリケーション プログラムの一部として使用することはできません。

JTA に対する BEA WebLogic の拡張機能

Java Transactions API の拡張機能は、JTA の仕様で実装の細かな部分がカバーされていないところや補足的な機能が必要なところで提供されます。

BEA WebLogic では、JTA 仕様の解釈に基づいて以下の機能が提供されます。

- クライアントが開始するトランザクション ― JTA トランザクション マネージャ インタフェース（javax.transaction.TransactionManager）が JNDI を通じてクライアントや Bean プロバイダから利用可能になります。この機能により、クライアントや、Bean 管理のトランザクションを利用する EJB でトランザクションをサスペンドしたり再開したりすることができます。

  注意： サスペンドされたトランザクションは、それがサスペンドされた同じサーバ プロセスで再開する必要があります。

- トランザクションのスコープ ― トランザクションがクラスタ内とクラスタ間の両方で機能できます。

BEA WebLogic では、JTA の拡張機能として以下のクラスとインタフェースが提供されます。

- `weblogic.transaction.RollbackException`（`javax.transaction.RollbackException` を拡張）

  このクラスでは、より包括的な例外情報で使用するためにロールバックの元々の理由が保持されます。

- `weblogic.transaction.TransactionManager`（`javax.transaction.TransactionManager` を拡張）

  WebLogic JTA トランザクション マネージャ オブジェクトでは、XA リソースが起動時にトランザクション マネージャに対する登録を行ったり、登録を解除したりできるようにするこのインタフェースがサポートされています。このインタフェースを利用すると、トランザクションをサスペンド後に再開することもできます。

  このインタフェースには、以下のメソッドがあります。

  - `registerStaticResource`、`registerDynamicResource`、および `unregisterResource`
  - `getTransaction`
  - `forceResume` および `forceSuspend`

- `weblogic.transaction.Transaction`（`javax.transaction.Transaction` を拡張）

  WebLogic JTA トランザクション オブジェクトでは、ユーザがトランザクション プロパティを取得および設定できるようにするこのインタフェースがサポートされています。

  このインタフェースには、以下のメソッドがあります。

  - `setName` および `getName`
  - `addProperties`、`setProperty`、`getProperty`、および `getProperties`
  - `setRollbackReason` および `getRollbackReason`
  - `getHeuristicErrorMessage`
  - `getXID`
  - `getStatusAsString`
  - `getMillisSinceBegin`
  - `getTimeToLiveMillis`

- `weblogic.transaction.TxHelper`

  このクラスを使用すると、現在のトランザクション マネージャおよびトランザクションを取得できます。

  このインタフェースには、以下の静的メソッドがあります。

  - `getTransaction`、`getUserTransaction`、`getTransactionManager`
  - `status2String`

- `weblogic.transaction.XAResource`
  （`javax.transaction.xa.XAResource` を拡張）

  このクラスを使用すると、XA リソースをリストから削除できます。

  このインタフェースには、次のメソッドがあります。

  - `getDelistFlag`

`javax.transaction` インタフェースと `javax.transaction.xa` インタフェースに対する WebLogic 拡張機能の詳細については、`weblogic.transaction` パッケージの説明を参照してください。

5 EJB アプリケーションのトランザクション

この章では、EJB アプリケーションにおけるトランザクションの振る舞いと使用法を説明します。

- 始める前に
- ガイドライン
- トランザクション属性
- トランザクションへの関与
- トランザクション セマンティクス
- セッションの同期
- トランザクション時の同期
- トランザクション タイムアウトの設定
- EJB トランザクションでの例外処理

この章では、BEA WebLogic Server 上で動作するエンタープライズ JavaBean（EJB）アプリケーションでトランザクションを実装する方法を説明します。

始める前に

本題に入る前に、第 1 章「トランザクションについて」の特に以下の節を読んでおく必要があります。

- WebLogic Server EJB アプリケーションのトランザクションの概要
- トランザクションのサンプル EJB コード

ここでは、エンタープライズ JavaBean でのトランザクションの BEA WebLogic Server 実装について説明します。ここで説明する情報は、Sun Microsystems 発行の EJB 仕様 2.0 を補足するものです。

注意： この章を読み進める前に、EJB 仕様 2.0 のドキュメントの内容、特に第 16 章「Support for Transactions」で提示されている概念やデータをよく理解しておく必要があります。

WebLogic Server アプリケーションでエンタープライズ JavaBean を実装する方法については、『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』を参照してください。

ガイドライン

以下のガイドラインは、WebLogic Server 向けの EJB アプリケーションでトランザクションを実装するときに適用されます。

- EJB 仕様では、フラット トランザクションのみがサポートされています。トランザクションをネストすることはできません。
- EJB 仕様では、複数のリソース（データベースなど）にまたがる分散トランザクションと、EJB CMP 2.0 および EJB CMP 1.1 に対応した 2 フェーズ コミット プロトコルがサポートされています。
- WebLogic Server では、JTA 準拠の XA リソースがサポートされています。WebLogic Server で提供される XA リソース ドライバについては、『WebLogic jDriver for Oracle のインストールと使い方』の「トランザクションと WebLogic jDriver for Oracle」を参照してください。

- 標準的なプログラミング手法を使用してトランザクション処理を最適化します。たとえば、トランザクションの境界を適切に設定すると、トランザクションの処理が速くなります。
- EJB が動作している WebLogic Server インスタンス上のローカルな TxDataSource からのデータベース接続を使用します。リモート WebLogic Server インスタンス上の TxDataSource からの接続は使用しないでください。
- トランザクション対応の EJB アプリケーションでパフォーマンスを最大限に引き出すために、必ず EJB キャッシュを調整します。詳細については、『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』の「WebLogic Server EJB コンテナ」を参照してください。

WebLogic Server トランザクション サービスのガイドラインについては、「機能と制限」を参照してください。

トランザクション属性

ここでは以下について説明します。

- EJB のトランザクション属性について
- コンテナ管理のトランザクションのトランザクション属性
- Bean 管理のトランザクションのトランザクション属性

EJB のトランザクション属性について

トランザクション属性では、トランザクションが EJB アプリケーションでどのように管理されるのかが指定されます。各 EJB について、トランザクション属性により、トランザクションの境界が WebLogic Server EJB コンテナによって設定されるのか（コンテナ管理のトランザクション）、または EJB 自体によって設定されるのか（Bean 管理のトランザクション）が指定されます。EJB がコンテナ管理と Bean 管理のどちらであるのかは、デプロイメント記述子の `transaction-type` 要素の設定で決まります。`transaction-type` 要素の詳細については、EJB 仕様 2.0 の第 16 章「Support for Transactions」と第 21 章「Deployment Descriptor」を参照してください。

概して、Bean 管理のトランザクションよりもコンテナ管理のトランザクションの方がよく使用されます。なぜなら、その方がアプリケーションのコードがシンプルになるからです。たとえば、コンテナ管理のトランザクションでは、トランザクションを明示的に開始する必要がありません。

WebLogic Server では、EJB 仕様 2.0 のセクション 16.4 で定義されているメソッド レベルのトランザクション属性がフルサポートされています。

コンテナ管理のトランザクションのトランザクション属性

コンテナ管理のトランザクションの場合、トランザクション属性はデプロイメント記述子の `container-transaction` 要素で指定します。コンテナ管理のトランザクションには、すべてのエンティティ Bean と、`transaction-type` が `Container` に設定されているステートフルまたはステートレスのセッション Bean が含まれます。これらの要素の詳細については、『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』の WebLgoic Server 6.1 のプロパティを参照してください。

EJB とそのビジネス メソッドで指定できるトランザクション属性は以下のとおりです。

- `NotSupported`
- `Supports`
- `Required`
- `RequiresNew`
- `Mandatory`
- `Never`

`trans-attribute` 設定で WebLogic Server EJB コンテナがどのように機能するのかについては、EJB 仕様 2.0 のセクション 16.7.2 を参照してください。

トランザクション属性の trans-timeout-seconds は、BEA WebLogic JTA 拡張機能に基づいています。デプロイメント記述子でタイムアウト値が定義されていない場合は、WebLogic Server EJB コンテナによって自動的にトランザクションのタイムアウトが設定されます。コンテナでは `trans-timeout-seconds` コンフィグレーション パラメータの値が使用されます。デフォルトの timeout 値は 300 秒です。

トランザクションのコンフィグレーション パラメータの詳細については、このマニュアルの第 2 章「トランザクションのコンフィグレーションと管理」および『管理者ガイド』の「トランザクションの管理」を参照してください。

コンテナ管理のトランザクションを使用する EJB の場合、EJB は `javax.transaction.UserTransaction` インタフェースにアクセスすることができず、また、開始と終了のトランザクション コンテキストが一致していなければなりません。さらに、コンテナ管理のトランザクションを使用する EJB では、`javax.ejb.EJBContext` インタフェースの `setRollbackOnly` メソッドと `getRollbackOnly` メソッドのサポートが制限されています（EJB 仕様 2.0 のセクション 16.4.4.2 と 16.4.4.3 で規定されている規則によって呼び出しが制限される）。

Bean 管理のトランザクションのトランザクション属性

Bean 管理のトランザクションの場合は、Bean が `javax.transaction.UserTransaction` インタフェースのメソッドを使用してトランザクションの境界設定を指定します。Bean 管理のトランザクションには、`transaction-type` が `Bean` に設定されているステートフルまたはステートレスのセッション Bean が含まれます。エンティティ Bean では、Bean 管理のトランザクションは使用できません。

ステートレス セッション Bean の場合は、開始と終了のトランザクション コンテキストが一致していなければなりません。ステートフル セッション Bean の場合は、開始と終了のトランザクション コンテキストが必ずしも一致している必要はありません。一致していない場合は、WebLogic Server EJB コンテナによって Bean と未終了のトランザクションの関連が維持されます。

Bean 管理のトランザクションを使用するセッション Bean では、`javax.ejb.EJBContext` インタフェースの `setRollbackOnly` メソッドと `getRollbackOnly` メソッドが使用できません。

トランザクションへの関与

EJB 仕様 2.0 で「トランザクションに関与」という表現が使用されるとき、それは Bean が以下のいずれかの条件を満たしているという意味であると解釈できます。

- Bean がトランザクション コンテキストで呼び出される場合（コンテナ管理のトランザクション）。
- Bean が、クライアントによって呼び出された Bean メソッドの中で UserTransaction API を使用してトランザクションを開始し（Bean 管理のトランザクション）、かつ、クライアントによって呼び出された対応する Bean メソッドの終了時にそのトランザクションをサスペンドまたは終了しない場合。

トランザクション セマンティクス

この節の内容は以下のとおりです。

- コンテナ管理のトランザクションのトランザクション セマンティクス
- Bean 管理のトランザクションのトランザクション セマンティクス

EJB 仕様 2.0 では、EJB の種類（エンティティ Bean、ステートレス セッション Bean、またはステートフル セッション Bean）とトランザクションの種類（コンテナ管理または Bean 管理）に基づいてトランザクション処理の動作を決定するセマンティクスが規定されています。それらのセマンティクスでは、メソッドが呼び出されるときのトランザクション コンテキストが規定されており、また、EJB が `javax.transaction.UserTransaction` インタフェースのメソッドにアクセスできるかどうかが定義されています。EJB アプリケーションは、それらのセマンティクスに注意して設計する必要があります。

コンテナ管理のトランザクションのトランザクション セマンティクス

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 セマンティクスは Bean の種類によって異なります。

ステートフル セッション Bean のトランザクション セマンティクス

表 5-1 は、コンテナ管理のトランザクションのステートフル セッション Bean のトランザクション セマンティクスを示しています。

表 5-1 コンテナ管理のトランザクションのステートフル セッション Bean のトランザクション セマンティクス

メソッド	メソッドが呼び出されたときのトランザクション コンテキスト	UserTransaction のメソッドへのアクセス
コンストラクタ	指定なし	不可
`setSessionContext()`	指定なし	不可
`ejbCreate()`	指定なし	不可
`ejbRemove()`	指定なし	不可
`ejbActivate()`	指定なし	不可
`ejbPassivate()`	指定なし	不可
ビジネス メソッド	トランザクション属性によって有または無	不可
`afterBegin()`	可	不可
`beforeCompletion()`	可	不可
`afterCompletion()`	不可	不可

ステートレス セッション Bean のトランザクション セマンティクス

表 5-2 は、コンテナ管理のトランザクションのステートレス セッション Bean のトランザクション セマンティクスを示しています。

表 5-2 コンテナ管理のトランザクションのステートレス セッション Bean のトランザクション セマンティクス

メソッド	メソッドが呼び出されたときのトランザクション コンテキスト	UserTransaction のメソッドへのアクセス
コンストラクタ	指定なし	不可
`setSessionContext()`	指定なし	不可
`ejbCreate()`	指定なし	不可
`ejbRemove()`	指定なし	不可
ビジネス メソッド	トランザクション属性によって有または無	不可

エンティティ Bean のトランザクション セマンティクス

表 5-3 は、コンテナ管理のトランザクションのエンティティ Bean のトランザクション セマンティクスを示しています。

表 5-3 コンテナ管理のトランザクションのエンティティ Bean のトランザクション セマンティクス

メソッド	メソッドが呼び出されたときのトランザクション コンテキスト	UserTransaction のメソッドへのアクセス
コンストラクタ	指定なし	不可
`setEntityContext()`	指定なし	不可
`unsetEntityContext()`	指定なし	不可

表 5-3 コンテナ管理のトランザクションのエンティティ Bean のトランザクション セマンティクス （続き）

メソッド	メソッドが呼び出されたときのトランザクション コンテキスト	UserTransaction のメソッドへのアクセス
`ejbCreate()`	対応する create のトランザクション属性で決まる	不可
`ejbPostCreate()`	対応する create のトランザクション属性で決まる	不可
`ejbRemove()`	対応する remove のトランザクション属性で決まる	不可
`ejbFind()`	対応する find のトランザクション属性で決まる	不可
`ejbActivate()`	指定なし	不可
`ejbPassivate()`	指定なし	不可
`ejbLoad()`	`ejbLoad()` を呼び出したビジネス メソッドのトランザクション属性で決まる	不可
`ejbStore()`	`ejbStore()` を呼び出したビジネス メソッドのトランザクション属性で決まる	不可
ビジネス メソッド	トランザクション属性によって有または無	不可

Bean 管理のトランザクションのトランザクション セマンティクス

Bean 管理のトランザクションの場合、トランザクション セマンティクスはステートフルとステートレスのセッション Bean で異なります。エンティティ Bean では、Bean 管理のトランザクションは利用できません。

ステートフル セッション Bean のトランザクション セマンティクス

表 5-4 は、Bean 管理のトランザクションのステートフル セッション Bean のトランザクション セマンティクスを示しています。

表 5-4 Bean 管理のトランザクションのステートフル セッション Bean のトランザクション セマンティクス

メソッド	メソッドが呼び出されたときのトランザクション コンテキスト	UserTransaction のメソッドへのアクセス
コンストラクタ	指定なし	不可
`setSessionContext()`	指定なし	不可
`ejbCreate()`	指定なし	可
`ejbRemove()`	指定なし	可
`ejbActivate()`	指定なし	可
`ejbPassivate()`	指定な x し	可
ビジネス メソッド	通常は無。Bean での直前のメソッド実行がトランザクション コンテキストの中で終了している場合を除く	可
`afterBegin()`	非適用	非適用
`beforeCompletion()`	非適用	非適用

表 5-4 Bean 管理のトランザクションのステートフル セッション Bean のトランザクション セマンティクス （続き）

メソッド	メソッドが呼び出されたときのトランザクション コンテキスト	UserTransaction のメソッドへのアクセス
`afterCompletion()`	非適用	非適用

ステートレス セッション Bean のトランザクション セマンティクス

表 5-5 は、Bean 管理のトランザクションのステートレス セッション Bean のトランザクション セマンティクスを示しています。

表 5-5 Bean 管理のトランザクションのステートレス セッション Bean のトランザクション セマンティクス

メソッド	メソッドが呼び出されたときのトランザクション コンテキスト	UserTransaction のメソッドへのアクセス
コンストラクタ	指定なし	不可
`setSessionContext()`	指定なし	不可
`ejbCreate()`	指定なし	可
`ejbRemove()`	指定なし	可
ビジネス メソッド	不可	可

セッションの同期

コンテナ管理のトランザクションを使用するステートフル セッション Bean では、javax.ejb.SessionSynchronization インタフェースを実装してトランザクション同期通知を提供できます。その場合、ステートフル セッション Bean のすべてのメソッドでは、REQUIRES_NEW、MANDATORY、または REQUIRED のいずれかのトランザクション属性をサポートしなければなりません。javax.ejb.SessionSynchronization インタフェースの詳細については、EJB 仕様 2.0 のセクション 6.5.3 を参照してください。

トランザクション時の同期

Bean で SessionSynchronization が実装されている場合、通常、WebLogic Server EJB コンテナではトランザクションのコミット時に Bean に対して以下のコールバックが行われます。

- afterBegin()
- beforeCompletion()
- afterCompletion()

EJB コンテナは、beforeCompletion メソッドで他の Bean を呼び出すか、または XA リソースを追加することができます。呼び出しの回数は、beforeCompletionIterationLimit 属性によって制限されます。この属性は、トランザクションがロールバックされるまでに処理されるコールバックのサイクル数を指定します。同期サイクルは、登録されているオブジェクトが beforeCompletion コールバックを受信し、リソースが追加されるか、または以前に同期されていたオブジェクトが再登録されたときに発生します。反復制限を利用することで、同期サイクルが無限に繰り返されることを防止できます。

トランザクション タイムアウトの設定

EJB アプリケーションのトランザクションには、タイムアウトを指定できます。トランザクションの存続期間がタイムアウトの設定値を超えた場合、トランザクション サービスによってそのトランザクションは自動的にロールバックされます。

注意： トランザクションで `begin()` を実行する前に、タイムアウトを設定する必要があります。タイムアウトを設定しても現在のトランザクションには影響しません。WebLogic Server の以前のバージョンとはこの点が異なります。以前のバージョンでは、タイムアウトが現在のトランザクションに影響していました。

タイムアウトの指定方法は、トランザクションの種類によって異なります。

- **コンテナ管理のトランザクション。** `weblogic-ejb-jar.xml` デプロイメント記述子で `trans-timeout-seconds` 属性をコンフィグレーションします。詳細については、『管理者ガイド』を参照してください。
- weblogic-ejb-jar.xml デプロイメント記述子で trans-timeout-seconds 属性をコンフィグレーションする必要があります。
- **Bean 管理のトランザクション。** アプリケーションによって `UserTransaction.setTransactionTimeout` メソッドが呼び出されます。

EJB トランザクションでの例外処理

WebLogic Server EJB アプリケーションでは、トランザクションの途中で送出された特定の例外を検出して処理する必要があります。例外処理の詳細については、Sun Microsystems 発行の EJB 仕様 2.0 の第 17 章「Exception Handling」を参照してください。

EJB トランザクションでビジネス メソッドによって例外がどのように送出されるのかについては、セクション 17.3 にある表 12（コンテナ管理のトランザクション）および表 13（Bean 管理のトランザクション）を参照してください。

クライアントの観点から見た例外については、セクション 17.4 を参照してください。特に、セクション 17.4.1（アプリケーション例外）、セクション 17.4.2（`java.rmi.RemoteException`）、セクション 17.4.2.1（`javax.transaction.TransactionRolledBackException`）、およびセクション 17.4.2.2（`javax.transaction.TransactionRequiredException`）の参照をお勧めします。

6 RMI アプリケーションのトランザクション

以下の節では、BEA WebLogic Server 環境で動作する RMI アプリケーションで、トランザクションを使用するためのガイドラインとその他のリファレンスを提供します。

- 始める前に
- ガイドライン

始める前に

本題に入る前に、「トランザクションについて」の特に以下の節を参照してください。

- WebLogic Server RMI アプリケーションのトランザクションの概要
- トランザクションのサンプル RMI コード

RMI アプリケーションの詳細については、『WebLogic RMI プログラマーズ ガイド』および 『WebLogic RMI over IIOP プログラマーズ ガイド』を参照してください。

ガイドライン

以下のガイドラインは、WebLogic Server 向けの RMI アプリケーションでトランザクションを実装するときに適用されます。

- WebLogic Server では、フラット トランザクションのみがサポートされています。トランザクションをネストすることはできません。
- 標準的なプログラミング手法を使用してトランザクション処理を最適化します。たとえば、トランザクションの境界を適切に設定すると、トランザクションの処理が速くなります。
- RMI アプリケーションでは、トランザクションでコールバック オブジェクトを使用することが推奨されていません（WebLogic Server で管理できないため）。コールバック オブジェクトの詳細については、『WebLogic RMI プログラマーズ ガイド』および『WebLogic RMI over IIOP プログラマーズ ガイド』を参照してください。
- RMI アプリケーションでは、RMI クライアントがトランザクションを開始できますが、トランザクションの処理はすべて、WebLogic Server によってホストされるサーバ オブジェクトまたはリモート オブジェクトで行われなければなりません。クライアント JVM でホストされるリモート オブジェクトは、トランザクションの処理に関わることはできません。

  対応策としては、クライアント JVM 上のリモート オブジェクトを呼び出す前にトランザクションをサスペンドし、リモート操作から返った後でトランザクションを再開します。

WebLogic Server トランザクション サービスのガイドラインについては、「機能と制限」を参照してください。

7 WebLogic Server でのサードパーティ JDBC XA ドライバの使い方

この章では、WebLogic Server トランザクション内で JDBC XA ドライバをどのように使うかを説明します。

- サードパーティ XA ドライバの概要
- サードパーティ ドライバのコンフィグレーションとパフォーマンス要件

サードパーティ XA ドライバの概要

この節では、分散トランザクションにおける WebLogic Server でのサードパーティ JDBC 2 層ドライバの使い方について概説します。これらのドライバは、WebLogic Server と DBMS の間の接続を提供します。分散トランザクションで使用するドライバは、後に「/XA」が続くドライバ名（Oracle Thin/XA ドライバなど）によって指定されます。

サードパーティ XA ドライバの表

次の表に、サード パーティ JDBC/XA ドライバを WebLogic Server 6.1 で使用する場合の確認済みの機能について示します。

表 7-1 2 層 JDBC/XA ドライバ

ドライバ / データベースのバージョン	説明
Type 2 XA ドライバ（ネイティブ .dll）	
IBM DB2 ■ バージョン 7.2 ■ プラットフォーム : NT	7-14 ページの「DB2 7.2/XA ドライバの使い方」を参照。
Type 4 XA ドライバ（オール Java）	
Oracle Thin ドライバ XA ■ ドライバ バージョン 8.1.7 ■ データベース バージョン 8.1.7	7-5 ページの「Oracle Thin 8.1.7/XA ドライバの使い方」を参照。
Sybase jConnect/XA ■ バージョン 5.2.1 ■ Adaptive Server Enterprise 12.0	7-9 ページの「Sybase jConnect 5.2.1/XA ドライバの使い方」を参照。
Cloudscape ■ バージョン 3.5.1	7-12 ページの「Cloudscape 3.5.1/XA ドライバのソフトウェア要件」を参照。

サードパーティ ドライバのコンフィグレーションとパフォーマンス要件

次に、WebLogic Server で特定のサードパーティ X/A ドライバを使用するための要件とガイドラインを示します。

注意： ここの一覧で示されていないサードパーティ製ドライバを使用するときは、接続プールの追加プロパティの設定が必要になる場合があります。詳細については、『管理者ガイド』の「その他の XA 接続プール プロパティ」を参照してください。

Oracle Thin 8.1.7/XA ドライバの使い方

以下の節では、WebLogic Server 6.1 で Type 4 Oracle Thin 8.1.7/XA ドライバを使用するための情報を示します。

Oracle Thin 8.1.7/XA ドライバのソフトウェア要件

Oracle Thin 8.1.7/XA ドライバのソフトウェア要件は次のとおりです。

- JDK 1.2.x
- XA 機能を使用するための Oracle 8.1.7 サーバ（非 XA での使用には制限は適用されない）

Oracle Thin 8.1.7/XA ドライバの確認済みの問題

次に、確認済みの問題と BEA の回避策を示します。

表 7-2 Oracle Thin ドライバの確認済みの問題と回避策

説明	Oracle のバグ	説明 /WebLogic Server 6.1 の回避策
ORA-01002 -「フェッチ順序が無効です」例外。XAResource.end(TMSUSPEND) 結果および XAResource.start(TMRESUME) 結果の後で結果セットを反復すると ORA-01002 になる。	—	回避策として、文のフェッチサイズを結果セットのサイズ以上に設定する。つまり、この回避策を使用しない限り、Oracle Thin 8.1.7 ドライバをクライアント サイドでは使用できない。または、メソッド呼び出しの間、Bean では結果セットを開いておくことができない。
XAResource.end(TMSUSPEND) に XAResource.end(TMSUCCESS) を続けると、XAER_RMERR になる。	1527725	WebLogic Server では、このバグに対する内部的な回避策を提供している。
マルチスレッド XA で使用する場合、ドライバはハングするか、XAER_RMERR になる。	1569235	WebLogic Server では、このバグに対する内部的な回避策を提供している。
グローバル トランザクションのない更新がサポートされない。更新時にグローバル トランザクションがない場合、更新を実行するためにローカル トランザクションが暗黙的に開始され、グローバル トランザクションと同じ XA 接続が再利用されるために XAER_RMERR になる。 さらに、アプリケーションがローカル トランザクションのコミットを試みる（自動コミットを true に設定する、または Connection.commit() を明示的に呼び出す）と、Oracle XA ドライバが「SQLException: Use explicit XA call」を返す。	—	アプリケーションで XA ドライバを使用して更新を行う場合は、有効なグローバル トランザクション コンテキストが存在していなければならない。つまり、Bean メソッドには、トランザクション属性として Required、RequiresNew、または Mandatory を持たせる必要がある。

表 7-2 Oracle Thin ドライバの確認済みの問題と回避策（続き）

説明	Oracle のバグ	説明 /WebLogic Server 6.1 の回避策
XAResource.recover が、入力フラグに関係なく、不明な Xid のセットを繰り返し返す。XA 仕様では、トランザクション マネージャはまず TMSTARTRSCAN で XAResource.recover を呼び出し、つづいて Xid が返されなくなるまで繰り返し TMNOFLAGS で XAResource.recover を呼び出す。このバグが原因で無限反復が発生し、Oracle カーソルが使い果たされたことを示す「ORA-01000: 最大オープン カーソル数を超えました」というエラーが表示される。	—	WebLogic Server では、この問題に対する内部的な回避策を提供している。

Oracle Thin 8.1.7/XA ドライバの環境の設定

次のように環境を設定します。

XA 用データベース サーバを有効にします。

- システム ユーザとして sqlplus にログオンします（`sqlplus sys/CHANGE_ON_INSTALL@<DATABASE ALIAS NAME>` など）。
- 次の sql を実行します。grant select on `DBA_PENDING_TRANSACTIONS` to `public`

上の手順をデータベース サーバ上で実行しない場合、通常の XA データベース クエリと更新は正常に動作できます。ただし、クラッシュ後の再起動に続き、WebLogic Server トランザクション マネージャが回復を行うときに、Oracle リソース の回復は `XAER_RMERR` になり失敗します。

Oracle Thin 8.1.7/XA ドライバのコンフィグレーション プロパティ

次の表に、接続プールのコンフィグレーションのためのサンプル コードを示します。

表 7-3 Oracle Thin 8.1.7/XA ドライバ : 接続プールのコンフィグレーション

プロパティ名	プロパティ値
Name	`jtaXAPool`
Targets	`myserver,server1`
URL	`jdbc:oracle:thin:@baybridge:1521:bay817`
DriverClassname	`oracle.jdbc.xa.client.OracleXADataSource`
Initial Capacity	`1`
MaxCapacity	`20`
CapacityIncrement	`2`
Properties	`user=scott;password=tiger`

次の表に、TxDataSource のコンフィグレーションのためのサンプル コードを示します。

表 7-4 Oracle Thin 8.1.7/XA ドライバ : TxDataSource のコンフィグレーション プロパティ

プロパティ名	プロパティ値
Name	`jtaXADS`
Targets	`myserver,server1`
JNDIName	`jtaXADS`
PoolName	`jtaXAPool`

Sybase jConnect 5.2.1/XA ドライバの使い方

以下の節では、Sybase jConnect 5.2.1/XA ドライバを使用する場合の重要なコンフィグレーション情報とパフォーマンス問題を示します。

Sybase jConnect 5.2.1/XA ドライバの確認済みの問題

次に、確認済みの問題と BEA の回避策を示します。

表 7-5 Sybase jConnect 5.2.1 の確認済みの問題と回避策

説明	Sybase のバグ	説明 /WebLogic Server 6.1 の回避策
setAutoCommit(true) を呼び出すと次の例外が送出される。 `java.sql.SQLException: JZ0S3: The inherited method setAutoCommit(true) cannot be used in this subclass.`	10726192	回避策はなし。ベンダによる修正が必要。
分散トランザクションでドライバを使用する場合、`XAResource.end(TMSUSPEND)` に続けて `XAResource.end(TMSUCCESS)` を呼び出すと、`XAER_RMERR` になる。	10727617	WebLogic Server では、このバグに対する内部的な回避策を提供している。 接続プールのプロパティを `XAEndOnlyOnce="true"` に設定する。 ベンダによる修正が要求されている。

Sybase jConnect/XA ドライバの環境の設定

次の手順に従って環境を設定します。

- `set CLASSPATH=.;%SYBASE_INSTALL_DIR%\jCONNECT-5_2\classes\jconn2.jar`

  ここで、`SYBASE_INSTALL_DIR` は Sybase ドライバをインストールしたディレクトリです。

- 分散トランザクション管理のライセンスをインストールします。

- sp_configure "enable DTM",1 を実行して、トランザクションを有効にします。
- sp_configure "enable xat coordination",1 を実行します。
- <USER_NAME> にロール dtm_role を付与します。
- SYBASE_INSTALL\OCS-12_0\sample\xa-dtm サブディレクトリにあるサンプルの xa_config ファイルを、3 レベル上位の SYBASE_INSTALL へコピーします。SYBASE_INSTALL は、Sybase サーバをインストールしたディレクトリです。次に例を示します。

  ```
  $ SYBASE_INSTALL\xa_config
  ```

- xa_config ファイルを編集します。最初の [xa] セクションで、サンプル サーバ名を適切なサーバ名に変更します。

トランザクションを実行するときのデッドロックを防ぐために、デフォルトで低レベル ロックを有効にします。

- sp_configure "lock scheme",0,datarows を実行します。

Sybase jConnect 5.2.1/XA ドライバの接続プール

次の表に、接続プールのコンフィグレーションのためのサンプル コードを示します。

表 7-6 Sybase jConnect 5.2.1/XA ドライバ : サンプルの接続プールのコンフィグレーション

プロパティ名	プロパティ値
Name	jtaXAPool
Targets	myserver,server1
DriverClassname	com.sybase.jdbc2.jdbc.SybXADataSource

表 7-6 Sybase jConnect 5.2.1/XA ドライバ : サンプルの接続プールのコンフィグレーション

プロパティ名	プロパティ値

Properties	`User=dbuser;`
	`DatabaseName=dbname;`
	`ServerName=server_name_or_IP_address;`
	`PortNumber=serverPortNumber;`
	`NetworkProtocol=Tds;`
	`resourceManagerName=Lrm_name_in_xa_config;`
	`resourceManagerType=2`

Password	*dbpassword*
Initial Capacity	1
MaxCapacity	10
CapacityIncrement	1

Lrm_name は、論理リソース マネージャの名前です。

注意 : config.xml ファイルの接続プール タグに、KeepXAConnTillTxComplete="true" も追加する必要があります。詳細については、『管理者ガイド』の「その他の XA 接続プール プロパティ」を参照してください。

次の表に、TxDataSource のコンフィグレーションのためのサンプル コードを示します。

表 7-7 Sybase jConnect 5.2.1/XA ドライバ : TxDataSource のコンフィグレーション

プロパティ名	プロパティ値
Name	jtaXADS
Targets	server1
JNDIName	jtaXADS
PoolName	jtaXAPool

Java クライアントのコンフィグレーション プロパティ

Java クライアントを実行する場合の以下のコンフィグレーション プロパティを設定します。

表 7-8 Sybase jConnect 5.2.1/XA ドライバ : Java クライアントの接続プロパティ

プロパティ名	プロパティ値
ds.setPassword	<password>
ds.setUser	<username>
ds.setNetworkProtocol	Tds
ds.setDatabaseName	<database-name>
ds.setResourceManagerName	<Lrm name in xa_config file>
ds.setResourceManagerType	2
ds.setServerName	<machine host name>
ds.setPortNumber	4100

Cloudscape 3.5.1/XA ドライバの使い方

以下の節では、WebLogic Server 6.1 で Type 2 Cloudscape 3.5.1/XA ドライバを使用するための情報を示します。

Cloudscape 3.5.1/XA ドライバのソフトウェア要件

Cloudscape 3.5.1/XA ドライバは JDK 1.3 RC1 をサポートしています。詳細については、http://www.cloudscape.com/support/techinfo.jsp を参照してください。

Cloudscape 3.5.1/XA ドライバの確認済みの問題

次の表に、確認済みの問題を示します。

表 7-9 Cloudscape 3.5.1/XA ドライバの確認済みの問題

説明	Cloudscape の拡張機能の要求	説明 /WebLogic Server 6.1 の回避策
確認済みの問題はなし。		.

Cloudscape 3.5.1/XA ドライバの環境の設定

次のように環境変数を設定します（NT 構文が前提）。

- `set CLOUDSCAPE_INSTALL=<directory where Cloudscape is installed>`
- `set CLASSPATH=.;%cloudscape_install%\lib\cloudscape.jar; %cloudscape_install%\lib\cloudsync.jar; %cloudscape_install%\lib\client.jar;%cloudscape_install%\lib\ tools.jar;c:\weblogic\dev\src\3rdparty\weblogicaux.jar`

注意： `weblogicaux.jar` は javax クラスの場合のみです。

Cloudscape 3.5.1/XA ドライバのコンフィグレーション プロパティ

次の表に、接続プールのコンフィグレーションのためのサンプル コードを示します。

表 7-10 Cloudscape 3.5.1/XA ドライバ : 接続プールのコンフィグレーション

プロパティ名	プロパティ値
Name	`jtaXAPool`
Targets	`myserver,server1`
DriverClassname	`COM.cloudscape.core.XaDataSource`
Initial Capacity	`1`

表 7-10 Cloudscape 3.5.1/XA ドライバ : 接続プールのコンフィグレーション

プロパティ名	プロパティ値
Max Capacity	`10`
Capacity Increment	`2`
Properties	`databaseName=CloudscapeDB;` `createDatabase=create`
Supports Local Transaction	True

次の表に、TxDataSource のコンフィグレーションのためのサンプル コードを示します。

表 7-11 Cloudscape 3.5.1/XA ドライバ : TxDataSource のコンフィグレーション

プロパティ名	プロパティ値
Name	`jtaXADS`
Targets	`myserver,server1`
JNDIName	`jtaXADS`
PoolName	`jtaXAPool`

DB2 7.2/XA ドライバの使い方

以下の節では、WebLogic Server 6.1 で Type2 DB2 7.2/XA ドライバを使用するために環境を設定する方法について説明します。

DB2 7.2/XA ドライバの環境の設定

次のように環境を設定します。

- `<db2>\java12` ディレクトリにあるバッチファイル `usejdbc2.bat` を実行して、正しいバージョンの `db2java.zip` ファイルを展開し、適切な場所に移

動します。これで、ドライバの JDBC 2.0 機能が有効になります。このバッチ ファイルを実行する前に、実行中の DB2 プロセスがないことを確認してください。

- CLASSPATH に <db2>/java/db2java.zip を含めます。
- PATH に <db2>/bin を含めます。

ここで、<db2> は DB2 サーバがインストールされているディレクトリを表します。

XAResource として DB2 を使用する場合の制限

1. 複数の接続プールを持つコンフィグレーションの場合、各接続プールは個別のデータベース インスタンスを持つ必要があります。
2. トランザクションは、中断されたトランザクションに既に関連付けられているリソースでは開始できません。この場合、javax.transaction.InvalidTransactionException（非アクティブなトランザクションを再開しようとした）が送出されます。中断と再開の間に、中間トランザクションが、中断されたトランザクションで使用されたものと同じリソースを使用すると、javax.transaction.invalidtransation 例外が送出されます。異なるリソースが中間トランザクションの内部で使用される場合は、正常に動作します。

DB2 7.2/XA ドライバのコンフィグレーション プロパティ

次の表に、接続プールのコンフィグレーションのためのサンプル コードを示します。

表 7-12 DB2 7.2/XA ドライバ : 接続プールのコンフィグレーション

プロパティ名	プロパティ値
Name	jtaXAPool
Targets	server1
DriverClassname	COM.ibm.db2.jdbc.DB2XADataSource
Initial Capacity	1

表 7-12 DB2 7.2/XA ドライバ : 接続プールのコンフィグレーション

プロパティ名	プロパティ値
MaxCapacity	`10`
CapacityIncrement	`2`

Properties	`user=db2admin;`
	`password=db2admin;`
	`DatabaseName=NEWDEMO`

注意 : `config.xml` ファイルの接続プール タグに、`KeepXAConnTillTxComplete="true"` も追加する必要があります。詳細については、『管理者ガイド』の「その他の XA 接続プール プロパティ」を参照してください。

次の表に、TxDataSource のコンフィグレーションのためのサンプル コードを示します。

表 7-13 DB2 7.2/XA ドライバ : TxDataSource のコンフィグレーション

プロパティ名	プロパティ値
Name	`jtaXADS`
Targets	`server1`
JNDIName	`jtaXADS`
PoolName	`jtaXAPool`

他のサードパーティ XA ドライバ

他のサードパーティ製 XA 準拠 JDBC ドライバを使用するには、ドライバのクラス ライブラリに対するパスを `CLASSPATH` に追加する必要があります。

8 WebLogic Server XA リソース プロバイダの要件

BEA WebLogic Server では Java Transaction API（JTA）がサポートされており、XA 準拠のリソースを使用して分散トランザクションを調整するトランザクション マネージャを備えています。この章では、WebLogic Server で分散トランザクションに参加する上での XA リソースの要件について説明します。この情報は、サードパーティ アプリケーションのインテグレータ向けに書かれたものです。

JTA の詳細については、Sun Microsystems 発行の Java Transaction API（JTA）仕様バージョン 1.0.1 を参照してください。

- XA リソース プロバイダ要件の概要
- トランザクション マネージャへの登録
- XAResource の取得と解放
- オプションの weblogic.transaction.XAResource インタフェース

XA リソース プロバイダ要件の概要

XA リソース プロバイダは、WebLogic Server での分散トランザクションへの参加を可能にするため、スレッド アフィニティ制限のない JTA `XAResource` インタフェースをサポートしている必要があります。

非 JDBC リソースについては、以下の作業もリソース プロバイダが行います。

- 起動時に、WebLogic Server のトランザクション マネージャに XA リソースを登録します。
- 必要に応じて、XA リソースの使用前に WebLogic Server のトランザクション マネージャからリソースを取得し、使用後にそれを解放します。

- 必要に応じて、weblogic.transaction.XAResource インタフェースを実装します。

トランザクション マネージャへの登録

JTA 仕様では、XA リソースをサーバのトランザクション マネージャにブートストラップする方法は定義されていません。そのため、WebLogic Server では登録用の API が定義されています。

XA リソース プロバイダでは、起動時にローカル トランザクション マネージャに XAResource 実装を登録するため、次の手順が実行されます。

1. JNDI または TxHelper インタフェースを使用して、JTA トランザクション マネージャを取得します。次のコードは、TxHelper インタフェースを使用して現在のトランザクション マネージャを取得する方法を示しています。

```
import javax.transaction.xa.XAResource;
import javax.transaction.TransactionManager;
import weblogic.transaction.TxHelper;

TransactionManager tm = TxHelper.getTransactionManager();
```

2. JTA トランザクション マネージャに XA リソースを登録します。

```
XAResource res = ...// リソース プロバイダによる XAResource の実装

tm.registerStaticResource(name, res); // 静的なリソースの取得
tm.registerDynamicResource(name, res2); // 動的なリソースの取得
```

リソース プロバイダは、静的または動的な登録メソッドを使用して名前と実装を提供します。静的および動的な取得についての詳細は、「XAResource の取得と解放」を参照してください。

リソース名によってトランザクション ブランチが決まります。リソースが複数のインスタンスをサポートする場合、それぞれのリソース インスタンスは異なる名前を使用する必要があります。このリソース名は管理にも使用されます。

XA リソースは、トランザクション作業で取得するよりも前に、トランザクション マネージャに登録されている必要がある点に注意してください。

3. JTA トランザクション マネージャから、XA リソースを登録解除します。

```
XAResource res = _ // リソース プロバイダによる XAResource の実装
tm.unregisterResource(name);
```

現在のサーバ上でこの名前に関連付けられているリソース プロバイダが登録解除されます。このリソースに未処理のトランザクションがある場合、リソースの登録を解除すると、トランザクションはロールバックされ、トランザクション ブランチは破棄されます。

XAResource の取得と解放

JTA アーキテクチャでは、トランザクション マネージャでのリソースの暗黙的な取得と解放を含め、トランザクション リソース マネージャの役割をアプリケーション サーバが果たします。

WebLogic Server では、XA リソースの静的な取得と動的な取得がサポートされています。

静的な取得と解放

XA リソースを静的に取得して登録する場合、WebLogic Server がアプリケーション サーバの役割を果たし、XA リソース プロバイダのために暗黙的な取得と解放を実行します。

WebLogic Server は、トランザクションの開始時と再開時にリソースを取得します。トランザクションが暗黙的に再開されるのは、それ以前に関連付けられたトランザクションを持つ Bean 管理のトランザクション Bean を呼び出すメソッドが開始されたとき、および別のサーバの呼び出しが返されたときです。

WebLogic Server は同様に、トランザクションの中断時、コミット時、およびロールバック時にリソースを解放します。トランザクションが暗黙的に中断されるのは、別のサーバを呼び出すとき、およびメソッド呼び出しがクライアントに返されたときです。使用される解放フラグは、リソース プロバイダでサポートされている場合には `getDelistFlag()` メソッドで、`getDelistFlag()` メソッドがサポートされていない場合には `TMSUSPEND` から取得します。`getDelistFlag()` メソッドの詳細については、「オプションの weblogic.transaction.XAResource インタフェース」を参照してください。

動的な取得と解放

XA リソース プロバイダ自体を動的に取得して登録することによって、取得および解放を実行することもできます。この場合、取得と解放の実行において、XA リソース プロバイダが部分的に JTA アプリケーション サーバの役割を果たします。動的な取得の利点は、リソース プロバイダが必要に応じてゆるやかな取得を実行することで、不必要なリソースの取得を避けることができる点です。

XA リソースを動的に取得する場合は、XA リソース プロバイダで以下が実行されます。

```
import javax.transaction.Transaction;
// JNDI または TxHelper を介して現在のトランザクションを取得する
Transaction tx = TxHelper.getTransaction();
tx.enlistResource(res);
```

リソースの取得と解放には、かなりの負荷がかかるおそれがあります。WebLogic Server のトランザクション マネージャでは、最適化のため同じスレッド内の同じリソースが重複して取得されないようになっています。

XA リソースを動的に解放する場合は、XA リソース プロバイダが上記のようにトランザクションを取得し、解放フラグを提供すると同時に delistResource メソッドを呼び出します。

```
tx.delistResource(res, flag);
```

リソースを動的に取得して登録する場合、取得の手順は省略できません。解放の手順は省略可能です。

WebLogic Server のトランザクション マネージャは遅延解放を実行します。遅延解放の場合は、トランザクションの中断時と終了時に XA リソースに対して `end()` メソッドが呼び出されます。トランザクション マネージャは、次の順番で解放フラグを取得します。

- リソース プロバイダがすでに解放された XA リソースを保持している場合は、以前に提供された解放フラグを使用します。
- リソース プロバイダで `getDelistFlag()` メソッドがサポートされている場合は、そのメソッドを呼び出して解放フラグを取得します。詳細については、「オプションの weblogic.transaction.XAResource インタフェース」を参照してください。
- リソース プロバイダが上記のいずれをもサポートしていない場合、これまで通り `TMSUSPEND` を使用します。

オプションの weblogic.transaction.XAResource インタフェース

静的な解放の場合、または動的な解放を省略した場合、WebLogic Server JTA トランザクション マネージャは、これまで通りメソッド呼び出しの間に TMSUSPEND を使用して XA リソースを解放し、オープンしている可能性のあるカーソルを保持します。しかし、リソースによっては、TMSUCCESS で解放するよりも多くの内部リソースを必要とするおそれがあります。XA リソース プロバイダは、オプションの weblogic.transaction.XAResource インタフェースをサポートしており、WebLogic Server で使用するデフォルトの解放フラグをオーバーライドできます。

weblogic.transaction.XAResource インタフェースでは、次のメソッドがサポートされています。

```
package weblogic.transaction;

public interface XAResource extends
javax.transaction.xa.XAResource {
  int getDelistFlag();
}
```

このメソッドがリソース プロバイダでサポートされている場合、WebLogic Server のトランザクション マネージャは getDelistFlag() メソッドを呼び出して、トランザクションの中断時とメソッドの終了時にリソースの解放に使用する解放フラグを取得します。

9 トランザクションのトラブルシューティング

この章では、トランザクション失敗の原因を突き止めたり、問題修正の手段を決定したりするためのトラブルシューティング ツールおよびタスクについて説明します。

- トランザクションのトラブルシューティングの概要
- トラブルシューティング ツール
- デバッグのヒント
- ヒューリスティックな終了の処理
- トランザクション システムの回復処理

トランザクションのトラブルシューティングの概要

WebLogic Server では、実行中のトランザクションをモニタし、ヒューリスティックな終了のケースで適切な情報を確実に取り込むことができます。また、データベース クエリ、トランザクション要求、および Bean メソッドのパフォーマンスをモニタすることもできます。

トラブルシューティング ツール

WebLogic Server では、トランザクションのトラブルシューティングで以下の支援を利用できます。

- 例外
- トランザクション識別子
- トランザクションの名前とプロパティ
- トランザクション ステータス
- トランザクションの統計
- トランザクションのモニタ
- トランザクションのロギング

例外

WebLogic JTA では、すべての標準の JTA 例外がサポートされています。標準の JTA 例外の詳細については、`javax.transaction` および `javax.transaction.xa` パッケージ API の Javadoc を参照してください。

標準の JTA 例外に加えて、WebLogic Server では `weblogic.transaction.RollbackException` クラスが提供されます。このクラスは `javax.transaction.RollbackException` を拡張し、ロールバックの元々の理由を保持します。トランザクションをロールバックする前、つまり `rollbackonly` に設定する前に、アプリケーションではそのロールバックの理由を提供できます。トランザクション サービスの中で発生したすべてのロールバックではその理由が設定されます。たとえば、タイムアウト、XA エラー、`beforeCompletion` での非検査例外、あるいはトランザクション マネージャへのアクセス不能などです。一度設定された理由は上書きできません。

トランザクション識別子

トランザクション サービスでは、各トランザクションにトランザクション識別子（Xid）が割り当てられます。この ID は、ログ ファイル内で特定トランザクションの情報を他と区別するために使用できます。トランザクション識別子は、weblogic.transaction.Transaction インタフェースの getXID メソッドを使用して取り出すことができます。トランザクション識別子を取得するメソッドの詳細については、weblogic.transaction.Transaction の Javadoc を参照してください。

トランザクションの名前とプロパティ

WebLogic JTA では、トランザクションの命名とユーザ定義プロパティをサポートする javax.transaction.Transaction の拡張機能が提供されます。それらの拡張機能は、weblogic.transaction.Transaction インタフェースに含まれています。

トランザクション名は、トランザクションの種類（送金やチケット購入など）を示します。したがって、サーバ上のユニークなトランザクションを識別するトランザクション ID とは混同しないでください。トランザクション名を利用すると、例外やログ ファイルでトランザクションの種類を簡単に識別できます。

ユーザ定義のプロパティはキーと値の組み合わせです。この場合のキーはプロパティを識別する文字列であり、値はプロパティに割り当てられた現在の値です。トランザクション プロパティの値は、Serializable インタフェースを実装するオブジェクトでなければなりません。アプリケーションでは、weblogic.transaction.Transaction インタフェースで定義されている set メソッドと get メソッドを使用してプロパティを管理します。一度設定されたプロパティは、有効期間が終わるまでトランザクションと共に存在し、トランザクションがシステムを移動するときにはマシン間で受け渡しされます。プロパティはトランザクション ログに保存され、クラッシュの回復処理で復元されます。トランザクション プロパティが複数回にわたって設定された場合は、最新の値が維持されます。

トランザクション名とトランザクション プロパティを設定および取得するメソッドの詳細については、weblogic.transaction.Transaction の Javadoc を参照してください。

トランザクション ステータス

Java Transaction API では、javax.transaction.Status クラスを使用してトランザクションのステータス コードが提供されます。

weblogic.transaction.Transaction の getStatusAsString メソッドを使用すると、トランザクションのステータスを文字列として返すことができます。文字列には、javax.transaction.Status で指定されているメジャー ステートと、補足的なマイナー ステート（logging や pre-preparing など）が含まれます。

トランザクションの統計

トランザクションの統計は、サーバ上のトランザクション マネージャで処理されるすべてのトランザクションについて提供されます。それらの統計には、トランザクションの総数、特定の結果（コミット、ロールバック、ヒューリスティックな終了など）を持つトランザクションの数、ロールバックされたトランザクションの理由別の数、およびトランザクションがアクティブな状態を維持した合計時間が含まれます。トランザクションの統計の詳細については、Administration Console のオンライン ヘルプを参照してください。

トランザクションのモニタ

進行中のトランザクションは、WebLogic Console を使用してモニタできます。「トランザクションの統計」で説明されている統計に加えて、以下の情報を表示できます。

- 名前別のトランザクション。ロールバックやアクティブ時間の情報が含まれます。
- リソース別のトランザクション。全トランザクション、コミットされたトランザクション、およびロールバックされたトランザクションの統計が含まれます。
- すべてのアクティブなトランザクション。ステータス、サーバ、リソース、プロパティ、およびトランザクション識別子の情報が含まれます。

トランザクション ログ

各サーバには、システムでのトランザクションの伝播に関する情報を記録するトランザクション ログがあります。トランザクション ログは、永続ストレージに書き込まれ、システムのクラッシュやネットワークの障害からサーバが復旧するのを支援します。トランザクション ログには、直接にはアクセスできません。トランザクション ログのファイルはバイナリ フォーマットです。

トランザクション ログは、複数のファイルで構成されます。各ファイルはガベージ コレクションの影響を受けます。したがって、トランザクション ログ ファイルに必要な記録がない場合は、システムによってファイルが削除され、そのディスク スペースがファイル システムに返されます。また、トランザクション ログ ファイルのサイズが大きくなりすぎた場合は、システムによって新しいログ ファイルが作成されます。

トランザクション ログ ファイルには、パス名プレフィックス、サーバ名、4 桁の数値サフィックス、およびファイル拡張子で構成されたユニークな名前が付けられます。WebLogic Console を使用して `TransactionLogFilePrefix` サーバ属性の値を指定すれば、パス名プレフィックスを設定できます。トランザクション ログ ファイルを RAID デバイスなどの可用性の高いファイル システムで作成する場合は、必ず `TransactionLogFilePrefix` 属性を設定します。

`websvr` というサーバ名の UNIX システムでは、以下のような名前でログ ファイルが作成されます。

```
/usr7/applog1/websvr0000.tlog
/usr7/applog1/websvr0001.tlog
/usr7/applog1/websvr0002.tlog
```

同様に、Windows NT システムでは以下のような名前でログ ファイルが作成されます。

```
C:\weblogic\logA\websvr0000.tlog
C:\weblogic\logA\websvr0001.tlog
C:\weblogic\logA\websvr0002.tlog
```

システムで大量のトランザクション ログ ファイルが作成されている場合は、完了していない長時間のトランザクションが 1 つまたは複数存在している可能性があります。その原因としては、リソース マネージャの障害や、トランザクションのタイムアウト値が著しく大きいことが考えられます。

トランザクション ログを格納するファイル システムでスペースが不足すると、commit() によって SystemException が送出され、トランザクション マネージャによってメッセージがシステム エラー ログに配置されます。利用可能なスペースが増えるまで、トランザクションはコミットされません。

サーバを別のマシンに移動するときには、すべてのトランザクション ログ ファイルも一緒にまとめて移動してください。

ヒューリスティック ログ ファイル

外部のトランザクション マネージャから WebLogic Server にトランザクションをインポートすると、WebLogic Server のトランザクション マネージャは、外部のトランザクション マネージャによって調整される XA リソースとして機能します。トランザクション破棄タイムアウトが経過した後や、WebLogic Server によってインポートされたトランザクションに参加する XA リソースがヒューリスティックな例外を送出する場合のような、まれに発生する致命的状況では、WebLogic Server のトランザクション マネージャはヒューリスティックな決定を行います。つまり、WebLogic Server のトランザクション マネージャは、外部のトランザクション マネージャからの入力を利用しないで、トランザクションをコミットまたはロールバックすることを決定します。ヒューリスティックな決定を行った WebLogic Server のトランザクション マネージャは、外部のトランザクション マネージャからトランザクションの無視を指示する通知を受け取るまで、ヒューリスティックな決定についての情報をヒューリスティック ログ ファイルに格納します。

ヒューリスティック ログ ファイルは、トランザクション ログ ファイルと共に格納され、.tlog 拡張子の前に .heur の付いたトランザクション ログ ファイルと同様の名前が与えられます。ヒューリスティック ログ ファイルのフォーマットは次のとおりです。

```
<TLOG_file_prefix>/<server_name><4-digit number>.heur.tlog
```

websvr というサーバ名の UNIX システムでは、次のようなヒューリスティック ログ ファイルが作成されます。

```
/usr7/applog1/websvr0000.heur.tlog
/usr7/applog1/websvr0001.heur.tlog
/usr7/applog1/websvr0002.heur.tlog
```

同様に、Windows システムでは、次のようなヒューリスティック ログ ファイルが作成されます。

```
C:/weblogic/logA/websvr0000.heur.tlog
C:/weblogic/logA/websvr0001.heur.tlog
C:/weblogic/logA/websvr0002.heur.tlog
```

デバッグのヒント

トランザクションの命名プロパティを使用すると、問題のあるトランザクションを分離および識別できます。

トランザクションをデバッグするときには、WebLogic JTA と関与しているリソースの間で障害を分離する必要があります。

ヒューリスティックな終了の処理

ヒューリスティックな終了（ヒューリスティックな決定）は、更新をコミットするかロールバックする分散トランザクションの終了段階でリソースが一方だけの決定を行ったときに発生します。これにより、分散されたデータは不確定な状態のままになります。ヒューリスティックな終了の原因としては、ネットワークの障害またはトランザクションのタイムアウトが考えられます。ヒューリスティックな終了が発生すると、以下のヒューリスティックな結果例外のいずれかが送出されます。

- `HeuristicRollback` — トランザクションに参加する 1 つのリソースが、準備してコミットの決定を待つことに同意しているにもかかわらず、処理のロールバックを自発的に決定した場合。トランザクション マネージャがトランザクションのコミットを決定すると、そのリソースによるヒューリスティックなロールバックの決定は不正になり、トランザクションの他のブランチはコミットされるため、矛盾した結果を招きます。
- `HeuristicCommit` — トランザクションに参加する 1 つのリソースが、準備してコミットの決定を待つことに同意しているにもかかわらず、処理のコミットを自発的に決定した場合。トランザクション マネージャがトランザクションのロールバックを決定すると、そのリソースによるヒューリスティックなコミットの決定は不正になり、トランザクションの他のブランチはロールバックされるため、矛盾した結果を招きます。

- `HeuristicMixed` ― トランザクションが、参加リソースの一部がコミットし、一部がロールバックするという混合した結果になったことを、トランザクション マネージャで認識している場合。主に、1 つまたは複数の参加リソースが、ヒューリスティックなロールバックまたはヒューリスティックなコミットを決定したことが原因で発生します。
- `HeuristicHazard` ― トランザクションが、参加リソースの一部がコミットし、一部がロールバックするという混合した結果になったことを、トランザクション マネージャで認識している場合。ただし、システムまたはリソースに障害があるため、HeuristicMixed の結果が確かに発生したかどうかを確認できない場合。主に、1 つまたは複数の参加リソースが、ヒューリスティックなロールバックまたはヒューリスティックなコミットを決定したことが原因で発生します。

ヒューリスティックな終了が発生すると、サーバ ログにメッセージが書き込まれます。ヒューリスティックな終了を解決する方法については、データベース ベンダが提供するドキュメントを参照してください。

一部のリソース マネージャでは、ヒューリスティックな終了のコンテキスト情報が保存されます。この情報は、リソース マネージャのデータの矛盾を解決するときに便利です。WebLogic Console の JTA パネルで `ForgetHeuristics` 属性が true に設定されている場合、この情報はヒューリスティックな終了の後で削除されます。コンテキスト情報を保存するリソース マネージャを使用するときには、`ForgetHeuristics` 属性を false に設定することをお勧めします。

トランザクション システムの回復処理

WebLogic Server のトランザクション マネージャは、ユーザによる最低限の介入でシステムのクラッシュから回復するように設計されています。準備されたトランザクションは、たとえ複数のクラッシュの後であっても、トランザクション マネージャからコミットまたはロールバックが行われることなくリソース マネージャで未解決のまま残ることはありません。

指定した時間が過ぎた後にトランザクションを破棄することもできます。`AbandonTimeoutSeconds` 属性を使用すると、トランザクション コーディネータがトランザクションの完了を試みる最大限の時間を秒単位で設定できます。デ

フォルト値は 86,400 秒、つまり 24 時間です。この時間を過ぎると、利用できないリソースや、トランザクションの結果を承認できないリソースが関わるトランザクションでそれ以上の解決は試行されません。

システムがクラッシュした後、トランザクション マネージャには以下の責任があります。

- 複数のリソースにわたって一貫性を維持します。

  トランザクションがクラッシュの前にコミットされ、`XAResource.recover()` によってトランザクション ID が返される場合、トランザクション マネージャでは一貫して `XAResource.commit()` が呼び出されます。トランザクションがクラッシュの前にコミットされず、`XAResource.recover()` によってトランザクション ID が返される場合、トランザクション マネージャでは一貫して `XAResource.rollback()` が呼び出されます。つまり、トランザクション マネージャのクラッシュだけで、一部のブランチがコミットされ一部がロールバックされる混合ヒューリスティックの終了が生じることはありません。

- 準備されたトランザクションを解決します。

  リソース マネージャとのトランザクションが準備されている場合、トランザクション マネージャではそのリソース マネージャの回復処理時に `XAResource.recover()` を呼び出し、最終的には、`commit()`、`rollback()`、または `forget()` を呼び出すことによって、`recover()` で返されたすべてのトランザクション ID を解決しなければなりません。

- トランザクションの解決に固執します。

  リソース マネージャがクラッシュした場合でも、トランザクション マネージャでは準備されている各トランザクションについて `commit()` または `rollback()` を呼び出さなければなりません。`commit()` または `rollback()` の呼び出しは成功するまで続けられます。トランザクション解決の試行は、`AbandonTimeoutSeconds` コンフィグレーション属性を設定することで制限できます。

- ヒューリスティックな終了を報告します。

  リソース マネージャによってヒューリスティックなコミットまたはヒューリスティックなロールバックが報告された場合、それはトランザクション マネージャによってサーバ ログに記録され、`Forget Heuristics` コンフィグレーション属性が有効な場合は `forget()` が呼び出されます。`Forget Heuristics` コンフィグレーション属性が有効ではない場合のヒューリス

ティックな終了の解決については、データベース ベンダが提供するドキュメントを参照してください。

A　用語集

ローカル トランザクション（local transaction）

単一のリソース マネージャにのみローカルなトランザクション。1 つのデータベースにのみ関連するトランザクションなど。

分散トランザクション（distributed transaction）

複数のリソース マネージャにわたる 2 フェース コミット プロトコルを介して、外部のトランザクション マネージャにより境界の設定と調整が行われるトランザクション。グローバル トランザクションとも呼ばれます。

グローバル トランザクション（global transaction）

「分散トランザクション」を参照してください。

トランザクション ブランチ（transaction branch）

グローバル トランザクションをサポートする、各リソース マネージャの内部処理ユニットは、厳密には 1 つのトランザクション ブランチの一部です。トランザクション マネージャがリソース マネージャに提供する各グローバル トランザクション識別子（GTRID または XID）によって、分散トランザクションと個々のブランチの両方が識別されます。

ヒューリスティックな決定（heuristic decision）

ヒューリスティックな決定（ヒューリスティックな終了）は、分散トランザクションの終了段階においてリソースが更新をコミットするかロールバックするかを一方的に決定することで発生します。これにより、分散されたデータは不確定な状態のままになります。ヒューリスティックな決定の原因としては、ネットワークの障害またはトランザクションのタイムアウトが考えられます。

HeuristicRollback

トランザクションに参加する 1 つのリソースが、準備してコミットの決定を待つことに同意しているにもかかわらず、処理のロールバックを自発的に決定した場合。トランザクション マネージャがトランザクションのコミットを決定すると、そのリソースによるヒューリスティックなロールバックの決定は不正になり、トランザクションの他のブランチはコミットされるため、矛盾した結果を招きます。

HeuristicCommit

トランザクションに参加する 1 つのリソースが、準備してコミットの決定を待つことに同意しているにもかかわらず、処理のコミットを自発的に決定した場合。トランザクション マネージャがトランザクションのロールバックを決定すると、そのリソースによるヒューリスティックなコミットの決定は不正になり、トランザクションの他のブランチはロールバックされるため、矛盾した結果を招きます。

HeuristicMixed

トランザクションが、参加リソースの一部がコミットし、一部がロールバックするという混合した結果になったことを、トランザクション マネージャで認識している場合。主に、1 つまたは複数の参加リソースが、ヒューリスティックなロールバックまたはヒューリスティックなコミットを決定したことが原因で発生します。

HeuristicHazard

トランザクションが、参加リソースの一部がコミットし、一部がロールバックするという混合した結果になったことを、トランザクション マネージャで認識している場合。ただし、システムまたはリソースに障害があるため、HeuristicMixed の結果が確かに発生したかどうかを確認できない場合。主に、1 つまたは複数の参加リソースが、ヒューリスティックなロールバックまたはヒューリスティックなコミットを決定したことが原因で発生します。

索引

S

T

U

X

い

え

か

く

け

こ

さ

し

す

せ

と

ね

は

ひ

ふ

ま

む

も

り

れ

ろ