BEA Tuxedo®

BEA Tuxedo CORBA トランザクション

BEA Tuxedo 8.1
2003 年 1 月

Copyright

Copyright © 2003 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

目次

このマニュアルについて

対象読者 viii
e-docs Web サイト viii
マニュアルの印刷方法 viii
関連情報 ix
サポート情報 ix
表記上の規則 x

1. トランザクションについて

BEA Tuxedo CORBA アプリケーションのトランザクションの概要 1-2
- トランザクションの ACID 特性 1-2
- リソース・マネージャ 1-2
- サポートされているプログラミング・モデル 1-3
- サポートされている API モデル 1-3
- ビジネス・トランザクションのサポート 1-4
- 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 1-5

トランザクションの使用が必要な場合 1-5
BEA Tuxedo CORBA アプリケーションのトランザクションの使用方法 1-7
- BEA ブートストラップ処理メカニズムによるトランザクションの使用方法 1-8
- INS ブートストラップ処理メカニズムによるトランザクションの使用方法 1-9

Transactions サンプル・アプリケーションの記述 1-11
- Transactions サンプル・アプリケーションのワークフロー 1-11
- 開発手順 1-13

2. 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 2-2
機能と制限事項 2-2
- 委譲型コミットによるライトウェイト・クライアント 2-3
- INS を使用するサード・パーティ・クライアントのサポート 2-3

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 2-4
トランザクションの伝達 (CORBA のみ)...2-4
トランザクションの整合性...2-5
トランザクションの終了 2-5
フラット・トランザクション 2-6
CORBA リモート・クライアントと BEA Tuxedo ドメインの相互運用性 2-6
ドメイン内およびドメイン間の相互運用性 2-6
ネットワークの相互運用性 2-7
トランザクション・サービスとトランザクション処理の関係 2-7
プロセスの障害 2-8
一般的な制限事項 2-8
CORBA アプリケーションのトランザクション・サービス 2-10
Bootstrap オブジェクトを使用した TransactionCurrent オブジェクトへの初期リファレンスの取得...2-10
INS を使用した TransactionFactory オブジェクトへの初期リファレンスの取得 2-11
CORBA トランザクション・サービス API.......................................2-12
CORBA トランザクション・サービス API の拡張............................2-27
BEA Tuxedo CORBA アプリケーションのトランザクションの使用に関する注意事項..2-29
UserTransaction API 2-32
UserTransaction メソッド 2-32
UserTransaction メソッドがスローする例外.......................................2-34

3. CORBA サーバ・アプリケーションのトランザクション

BEA Tuxedo クライアントおよびサーバ・アプリケーションのトランザクションの統合...3-2
CORBA アプリケーションのトランザクション・サポート 3-2
オブジェクトを自動的にトランザクションに関与させる方法 3-4
オブジェクトのトランザクションへの参加の有効化..........................3-5
トランザクションのスコープ指定時のオブジェクト呼び出しの防止 3-7
実行中のトランザクションからのオブジェクトの除外 3-8
方針の割り当て 3-8
XA リソース・マネージャの使用方法...3-9
XA リソース・マネージャのオープン 3-10

XA リソース・マネージャのクローズ 3-10
トランザクション管理とオブジェクト状態管理 3-10
XA リソース・マネージャへのオブジェクト状態管理の委譲 3-11
トランザクションの作業が完了してから、データベースへの書き込みが始まるまでの待機 3-11
ユーザ定義の例外 3-14
ユーザ定義の例外 3-14
例外の定義...3-15
例外のスロー..3-15
Transactions University サンプル・アプリケーションのしくみ..................3-16
Transactions University サンプル・アプリケーションについて...........3-16
Transactions University サンプル・アプリケーションで使用するトランザクション・モデル 3-18
University サーバ・アプリケーションのオブジェクト状態に関する注意事項 3-18
Transactions サンプル・アプリケーションのコンフィギュレーションの要件 3-21

4. CORBA クライアント・アプリケーションのトランザクション

BEA Tuxedo CORBA トランザクションの概要.......................................4-3
トランザクションの開発プロセスの概要 4-3
ステップ 1: Bootstrap オブジェクトを使用して TransactionCurrent オブジェクトを取得する 4-4
C++ の例...4-5
Java の例...4-5
Visual Basic の例..4-5
ステップ 2: TransactionCurrent メソッドを使用する 4-6
C++ の例...4-8
Java の例...4-9
Visual Basic の例..4-9

5. トランザクションの管理

UBBCONFIG ファイルをトランザクションに対応させて変更する 5-2
手順の要約..5-2
ステップ 1: RESOURCES セクションでアプリケーション全体のトラン

ザクションを指定する 5-3
ステップ 2: トランザクション・ログ (TLOG) を作成する 5-4
ステップ 3: GROUPS セクションでリソース・マネージャ (RM) とトランザクション・マネージャ・サーバを定義する 5-7
ステップ 4: トランザクションを開始するためのインターフェイスを有効にする 5-9
トランザクションをサポートするように Domains コンフィギュレーション・ファイルを変更する (BEA Tuxedo CORBA サーバ) 5-13
DMTLOGDEV、DMTLOGNAME、DMTLOGSIZE、MAXRDTRAN、および MAXTRAN パラメータの特性 5-14
AUTOTRAN および TRANTIME パラメータの特性 (BEA Tuxedo CORBA および ATMI サーバ) 5-15
トランザクションを使用する分散アプリケーションの例 5-17
RESOURCES セクション...5-17
MACHINES セクション 5-18
GROUPS セクションおよび NETWORK セクション 5-20
SERVERS、SERVICES、および ROUTING セクション 5-21

索引

このマニュアルについて

このマニュアルでは、BEA Tuxedo 環境で実行される CORBA アプリケーションでトランザクションを使用する方法について説明します。

このマニュアルでは、以下の内容について説明します。

- 「第 1 章 トランザクションについて」では、BEA Tuxedo CORBA 環境で実行される CORBA アプリケーションにおけるトランザクションについて概説します。
- 「第 2 章 トランザクション・サービス」では、BEA Tuxedo のトランザクション・サービスについて説明します。
- 「第 3 章 CORBA サーバ・アプリケーションのトランザクション」では、CORBA C++ アプリケーションにトランザクションをインプリメントする方法について説明します。
- 「第 4 章 CORBA クライアント・アプリケーションのトランザクション」では、CORBA クライアント・アプリケーションにトランザクションをインプリメントする方法について説明します。
- 「第 5 章 トランザクションの管理」では、BEA Tuxedo CORBA 環境でトランザクションを管理する方法について説明します。

対象読者

このマニュアルは、BEA Tuxedo CORBA 環境で実行されるトランザクション対応 C++ アプリケーションをビルドするアプリケーション開発者を主な対象としています。BEA Tuxedo プラットフォーム、C++ 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 Tuxedo 製品のマニュアルは BEA 社の Web サイトで参照することができます。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、または http://edocs.beasys.co.jp/e-docs/index.html に直接アクセスしてください。

マニュアルの印刷方法

このマニュアルは、ご使用の Web ブラウザで一度に 1 ファイルずつ印刷できます。Web ブラウザの [ファイル] メニューにある [印刷] オプションを使用してください。

このマニュアルの PDF 版は、e-docs Web サイトの BEA Tuxedo マニュアル・ページから入手できます。また、マニュアルの CD-ROM にも収められています。この PDF を Adobe Acrobat Reader で開くと、マニュアル全体または一部をブック形式で印刷できます。PDF 形式を利用するには、BEA Tuxedo マニュアルのホーム・ページにある [PDF 版] ボタンをクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader をお持ちでない場合は、Adobe Web サイト (http://www.adobe.co.jp/) から無償でダウンロードできます。

関連情報

CORBA、BEA Tuxedo、分散オブジェクト・コンピューティング、トランザクション処理、および C++ プログラミングの詳細については、BEA Tuxedo オンライン・マニュアルの「Bibliography」を参照してください。

サポート情報

皆様の BEA Tuxedo マニュアルに対するフィードバックをお待ちしています。ご意見やご質問がありましたら、電子メールで docsupport-jp@bea.co.jp までお送りください。お寄せいただきましたご意見は、BEA Tuxedo マニュアルの作成および改訂を担当する BEA 社のスタッフが直接検討いたします。

電子メール メッセージには、BEA Tuxedo 8.0 リリースのマニュアルを使用していることを明記してください。

BEA Tuxedo に関するご質問、または BEA Tuxedo のインストールや使用に際して問題が発生した場合は、www.bea.com の BEA WebSUPPORT を通して BEA カスタマ・サポートにお問い合わせください。カスタマ・サポートへの問い合わせ方法は、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・サポートへお問い合わせの際には、以下の情報をご用意ください。

- お客様のお名前、電子メール・アドレス、電話番号、Fax 番号
- お客様の会社名と会社の住所
- ご使用のマシンの機種と認証コード

- ご使用の製品名とバージョン
- 問題の説明と関連するエラー・メッセージの内容

表記上の規則

このマニュアルでは、以下の表記規則が使用されています。

規則	項目
太字	用語集に定義されている用語を示します。
Ctrl + Tab	2 つ以上のキーを同時に押す操作を示します。
イタリック体	強調またはマニュアルのタイトルを示します。

`等幅テキスト`	コード・サンプル、コマンドとオプション、データ構造とメンバ、データ型、ディレクトリ、およびファイル名と拡張子を示します。また、キーボードから入力する文字も示します。
	例 :
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

等幅太字	コード内の重要な単語を示します。 例 : `void` **`commit`** `( )`
等幅イタリック体	コード内の変数を示します。 例 : `String` *`expr`*

規則	項目
大文字	デバイス名、環境変数、および論理演算子を示します。 例 : LPT1 SIGNON OR
{ }	構文の行で選択肢を示します。かっこは入力しません。

[ ]	構文の行で省略可能な項目を示します。かっこは入力しません。
	例 :
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

|	構文の行で、相互に排他的な選択肢を分離します。記号は入力しません。

...	コマンド行で次のいずれかを意味します。
	■ コマンド行で同じ引数を繰り返し指定できること
	■ 省略可能な引数が文で省略されていること
	■ 追加のパラメータ、値、その他の情報を入力できること
	省略符号は入力しません。
	例 :
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

. . .	コード例または構文の行で、項目が省略されていることを示します。省略符号は入力しません。

1 トランザクションについて

ここでは、以下の内容について説明します。

- BEA Tuxedo CORBA 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使用が必要な場合
- BEA Tuxedo CORBA アプリケーションのトランザクションの使用方法
- Transactions サンプル・アプリケーションの記述

BEA Tuxedo CORBA アプリケーションのトランザクションの概要

ここでは、以下の内容について説明します。

- トランザクションの ACID 特性
- リソース・マネージャ
- 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
- ビジネス・トランザクションのサポート
- 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

トランザクションの ACID 特性

BEA Tuxedo システムの最も基本的な機能の 1 つが、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・トランザクションが誤りなく完了し、高性能トランザクションのすべての ACID 特性 (原子性、一貫性、独立性、および持続性) が確保されるようにする手段です。BEA Tuxedo は、完全なインフラストラクチャを提供することでトランザクションの整合性を保護し、さまざまなリソース・マネージャ (RM) にわたってデータベースが正しく更新されるようにします。いずれか 1 つのオペレーションが失敗した場合、一連のオペレーションがすべてロールバックされます。

リソース・マネージャ

リソース・マネージャ (RM) は、データベース管理システムや BEA Tuxedo システムの Application Queuing Manager のようなデータ・リポジトリであり、データにアクセスするためのツールを備えています。BEA Tuxedo シス

テムは、1 つまたは複数の RM を使用してアプリケーションの状態を管理します。たとえば、銀行の預金残高のレコードは、RM に保存されています。預金引き出しサービスによってアプリケーションの状態が変わると、変更後の預金残高が、適切な RM に記録されます。

BEA Tuxedo システムは、XA インターフェイス対応の RM が関与するトランザクションを管理するのに役立ちます。BEA Tuxedo システムは、トランザクション・マネージャ (TM: Transaction Manager) として動作し、トランザクションに関連するすべての操作およびすべてのモジュールを調整します。

TM は、システム全体に渡るリソースが関与するグローバル・トランザクションを調整します。個々のリソースは、ローカルのリソース・マネージャ (RM) によって管理されます。トランザクション・マネージャ・サーバ (TMS: Transaction Manager Server) は、複数のリソースに関与するトランザクションの開始、コミット、およびアボートを行います。アプリケーション・コードは、RM に対して標準の埋め込み型 SQL インターフェイスを使用し、読み取りや更新を行います。TMS は、RM に対して XA インターフェイスを使用し、グローバル・トランザクション操作を実行します。

サポートされているプログラミング・モデル

BEA Tuxedo は、「The Common Object Request Broker: Architecture and Specification, Revision 2.4.2」(2001 年 1 月) に準拠し、C++ による Object Management Group の Common Object Request Broker (CORBA) をサポートします。

サポートされている API モデル

BEA Tuxedo は、CORBA のオブジェクト・トランザクション・サービス (OTS) をサポートしています。BEA Tuxedo は、OTS に対する C++ インターフェイスを提供し、OTS に基づいています。OTS には、環境オブジェクト `org.omg.CosTransactions.Current` を介してアクセスします。環境オブ

ジェクト `TransactionCurrent` を使用する際の詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』の「CORBA ブートストラップ処理のプログラミング・リファレンス」を参照してください。

注記　BEA Tuxedo では、CORBA インターオペラブル・ネーミング・サービス (INS) ブートストラップ処理メカニズムも使用できます。INS の詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』の「CORBA ブートストラップ処理のプログラミング・リファレンス」を参照してください。

ビジネス・トランザクションのサポート

OTS では、ビジネス・トランザクションについて以下のサポートが提供されます。

- クライアント・アプリケーションによってトランザクションが開始されたときにグローバル・トランザクション識別子が作成されます。
- BEA Tuxedo インフラストラクチャと共に機能します。トランザクションに関わっている (したがって、トランザクションがコミットできる状態になったときに調整する必要がある) オブジェクトを追跡します。
- リソース・マネージャ (データベースが多い) がトランザクションの代わりにアクセスされたときにそのことがリソース・マネージャに通知されます。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されたレコードをロックします。
- トランザクションが完了したときに 2 フェーズ・コミットが調整されます。その調整によって、トランザクションのすべてのパーティシパントで更新が同時にコミットされます。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コミットが調整されます。この規格は、ほぼすべてのリレーショナル・データベースでサポートされています。
- トランザクションが停止されるときにロールバック手続きが実行されます。
- 障害が発生したときに回復手続きが実行されます。クラッシュ発生時にマシンでどのトランザクションがアクティブだったのかが判別され、続

いて、トランザクションをロールバックすべきなのか、またはコミットすべきなのかが判断されます。

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

BEA Tuxedo CORBA は、エンタープライズ・アプリケーションに対して分散トランザクションと 2 フェーズ・コミット・プロトコルをサポートしています。分散トランザクションは、データベースなど複数のリソース・マネージャを調整しながら更新するトランザクションです。2 フェーズ・コミット (2PC) プロトコル は、1 つまたは複数のリソース・マネージャにわたる単一のトランザクションを調節する方法です。トランザクションで実行した、関連するすべてのデータベースへの更新処理をコミットするか、または完全にロールバックしてトランザクション開始時の状態に戻すことにより、データの整合性を保証できます。

トランザクションの使用が必要な場合

トランザクションは、以下で述べる場合に適しています。以下はそれぞれ、BEA Tuxedo CORBA でサポートされているトランザクション・モデルです。

- 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。呼び出しの 1 つが失敗した場合は、(メモリ、またはより一般的にはデータベースに) 書き込まれた状態をロールバックする必要があります。

  たとえば、旅行代理店アプリケーションがあるとします。クライアント・アプリケーションは、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・スプリングスまでなど、遠隔地への旅行を手配する必要があります。このような旅行では、複数の予約が必要になります。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。

- クライアント・アプリケーションは、サーバ・アプリケーションによって管理されているオブジェクトと会話し、指定したオブジェクト・インスタンスに対して複数の呼び出しを実行する必要があります。会話処理には、以下のような 1 つまたは複数の特徴があります。

  - データは、それぞれの呼び出しの最中またはその後にメモリにキャッシュされるか、またはデータベースに書き込まれます。
  - データは、会話の最後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、オブジェクトが各呼び出し間のメモリ内のコンテキストを保持するように要求します。つまり、連続する各呼び出しでは会話の全体を通してメモリ内に保持されているデータが使用されます。
  - 会話の最後に、クライアント・アプリケーションでは、会話の最中または最後に発生した可能性があるすべてのデータベース書き込みオペレーションをキャンセルできる必要があります。

  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・アプリケーションがあるとします。クライアント・アプリケーションのユーザは、オンライン・カタログをブラウズして、複数の購入を選択します。ユーザはすべての品目を選択して購入を指定すると、チェック・アウトに進み、クレジット・カード情報を入力して購入を確定します。クレジット・カードのチェックに失敗した場合、ショッピング・アプリケーションでは、ショッピング・カートで保留中のすべての購入選択をキャンセルするか、会話の最中に発生した購入トランザクションをロールバックする必要があります。

- オブジェクトに対する単一のクライアント呼び出しのスコープ内で、オブジェクトは、データベース内のデータに対する複数の編集を実行します。編集のいずれかが失敗した場合、オブジェクトはすべての編集をロールバックするメカニズムを必要とします。この状況では、個々のデータベース編集は必ずしも CORBA である必要はありません。

  たとえば、銀行取引アプリケーションがあるとします。クライアントは、窓口オブジェクトに対して振替オペレーションを呼び出します。振替オペレーションは、銀行データベースに対して以下の呼び出しを実行するために窓口オブジェクトを必要とします。

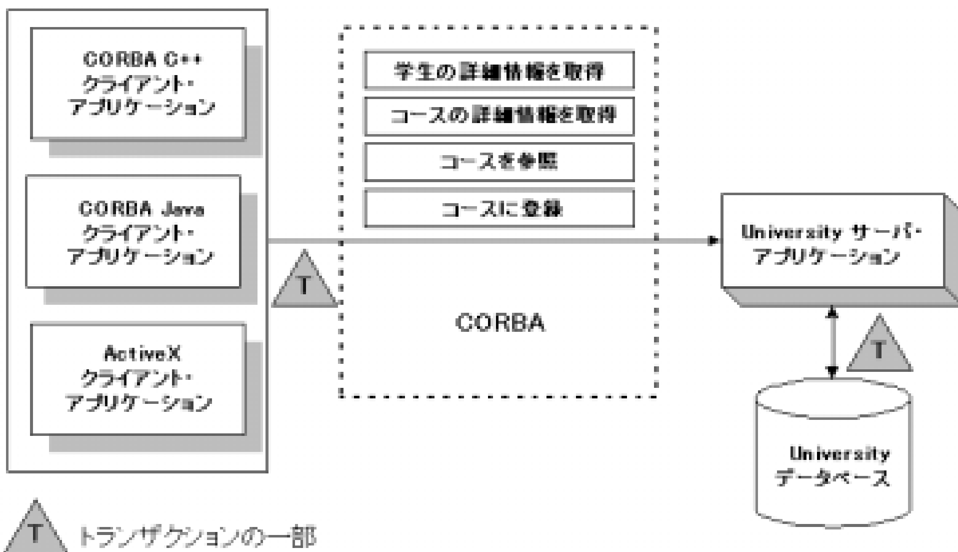
- ある口座に対する振替元メソッドの呼び出し
- 別の口座に対する振替先メソッドの呼び出し

銀行データベースの振替先呼び出しに失敗した場合、銀行取引アプリケーションではそれまでの振替元呼び出しをロールバックする機能が必要です。

BEA Tuxedo CORBA アプリケーションのトランザクションの使用方法

図 1-1 では、BEA Tuxedo CORBA アプリケーションのトランザクションを示しています。

図 1-1BEA Tuxedo CORBA アプリケーションのトランザクション



トランザクションの使用方法は、BEA ブートストラップ処理メカニズムを使用するか、インターオペラブル・ネーミング・サービス (INS) ブートストラップ処理メカニズムを使用するかによって異なります。

注記　BEA Tuxedo CORBA クライアント・ソフトウェアを使用する場合は、BEA ブートストラップ処理メカニズムを使用する必要があります。サード・パーティのクライアントを使用する場合は、INS ブートストラップ処理メカニズムを使用する必要があります。

BEA ブートストラップ処理メカニズムによるトランザクションの使用方法

BEA 独自のブートストラップ処理メカニズムを使用する場合、以下の方法で基本的なトランザクションを使用します。

1. クライアント・アプリケーションが、Bootstrap オブジェクトを使用して BEA Tuxedo ドメインの TransactionCurrent オブジェクトのオブジェクト・リファレンスを取得します。
2. クライアント・アプリケーションが、`Tobj::TransactionCurrent::begin()` オペレーションを使用してトランザクションを開始し、TP フレームワークを通じて CORBA インターフェイスに要求を発行します。CORBA インターフェイスのすべてのオペレーションは、トランザクションのスコープ内で実行されます。
   - それらのいずれかのオペレーションの呼び出しで (明示的にまたは通信障害の結果として) 例外が生成された場合は、その例外をキャッチできます。
   - 発生する必要があるすべての変更が正常に発生し、データベース (またはオブジェクト) の状態に整合性がある場合、トランザクションはコミットされます。そうでない場合、トランザクションはロールバックされます。
   - クライアント・アプリケーションでは `Tobj::TransactionCurrent::commit()` オペレーションを使用して現在のトランザクションをコミットします。このオペレーションは、トランザクションを終了して、オペレーションの処理を開始します。ト

ランザクションは、トランザクションのすべてのパーティシパントがコミットに同意した場合にのみコミットされます。

3. `Tobj::TransactionCurrent::commit()` オペレーションを実行すると、TP フレームワークはトランザクション・マネージャを呼び出してトランザクションを完了します。

4. トランザクション・マネージャは、それぞれのリソース・マネージャを調整してデータベースを更新する役割を果たします。

INS ブートストラップ処理メカニズムによるトランザクションの使用方法

CORBA サービス・インターオペラブル・ネーミング・サービス (INS) のブートストラップ処理メカニズムを使用する場合、以下の方法で基本的なトランザクションを使用します。

1. クライアント・アプリケーションは、`ORB::resolve_initial_references()` オペレーションを使用して、BEA Tuxedo ドメインの FactoryFinder オブジェクトを取得します。

2. クライアント・アプリケーションは、FactoryFinder を使用して TransactionFactory を取得します。

   注記　TransactionFactory は、BEA の委譲型インターフェイスではなく、標準の CORBA のトランザクション・サービス・インターフェイスに準拠したオブジェクトを返します。これは、サード・パーティのクライアントが、それぞれの ORB の `resolve_initial_references()` 関数を使用して BEA Tuxedo CORBA サーバから TransactionFactory を取得し、標準 OMG IDL で生成されたスタブを使用して、返されたインスタンス上で機能できることを意味します。

3. クライアント・アプリケーションは、TransactionFactory の `create()` オペレーションを使用してトランザクションを開始し、TP フレームワークを介して CORBA インターフェイスへの要求を発行します。

4. `create()` オペレーションによって返された Contorol オブジェクトから、クライアント・アプリケーションは、`get_terminator()` オペレーションを使用してトランザクションの Terminator インターフェイスを取得します。

5. 次に、クライアント・アプリケーションは、Terminator インターフェイスの `commit()` オペレーションまたは `rollback()` オペレーションを使用して、トランザクションを終了またはアボートします。`commit()` オペレーションを実行すると、TP フレームワークはトランザクション・マネージャを呼び出してトランザクションを完了します。

6. トランザクション・マネージャは、それぞれのリソース・マネージャを調整してデータベースを更新する役割を果たします。

注記　CORBA インターフェイスのすべてのオペレーションは、トランザクションのスコープ内で実行されます。

- それらのいずれかのオペレーションの呼び出しで (明示的にまたは通信障害の結果として) 例外が生成された場合は、その例外をキャッチできます。
- 発生する必要があるすべての変更が正常に発生し、データベース (またはオブジェクト) の状態に整合性がある場合、トランザクションはコミットされます。そうでない場合、トランザクションはロールバックされます。
- クライアント・アプリケーションは、`Terminator::commit()` オペレーションを使用して現在のトランザクションをコミットします。このオペレーションは、トランザクションを終了して、オペレーションの処理を開始します。トランザクションは、トランザクションのすべてのパーティシパントがコミットに同意した場合にのみコミットされます。

注記　INS の詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』の「CORBA ブートストラップ処理のプログラミング・リファレンス」を参照してください。

Transactions サンプル・アプリケーションの記述

ここでは、以下の内容について説明します。

- Transactions サンプル・アプリケーションのワークフロー
- 開発手順

Transactions サンプル・アプリケーションのワークフロー

Transactions サンプル CORBA アプリケーションでは、コースの登録のオペレーションがトランザクションのスコープ内で実行されます。Transactions サンプル・アプリケーションで使用されるトランザクション・モデルは、会話モデル、および単一のクライアントの呼び出しによってデータベース上で複数のオペレーションが実行されるモデルの組み合わせです。

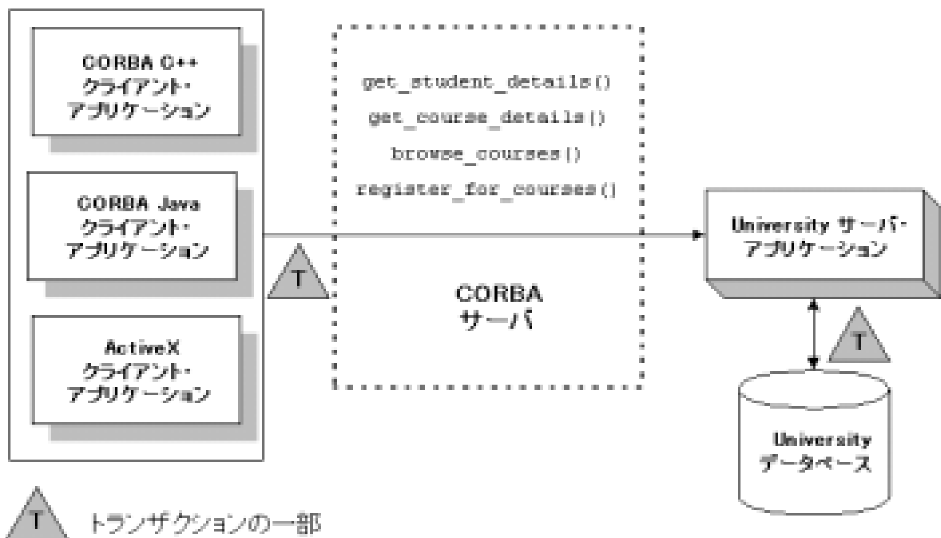
Transactions サンプル・アプリケーションは、以下のように機能します。

1. 学生は登録するコースのリストを送信します。
2. サーバ・アプリケーションは、リスト内の各コースに対して以下をチェックします。
   - コースがデータベースにあるかどうか
   - 学生がコースに登録済みであるかどうか
   - 学生が履修できる単位の最大数を超えているかどうか
3. 次のいずれかの処理が行われます。
   - コースがすべての基準を満たす場合は、サーバ・アプリケーションによってそのコースに学生が登録されます。

- コースがデータベースにない場合、またはそのコースに学生が既に登録されている場合は、サーバ・アプリケーションによって、その学生が登録できないコースのリストに追加されます。すべての登録要求の処理が完了したら、サーバ・アプリケーションは登録に失敗したコースのリストを返します。クライアント・アプリケーションでは、トランザクションをコミットするか (登録要求が成功したコースで学生が登録される)、またはトランザクションをロールバックするか (どのコースにも学生は登録されない) を選択できます。
- 学生が履修できる単位の最大数を超えている場合は、ユーザ例外 `TooManyCredits` がサーバ・アプリケーションからクライアント・アプリケーションに返されます。続いて、クライアント・アプリケーションによって、要求が拒否されたという内容の簡単なメッセージが表示されます。その際、クライアント・アプリケーションはトランザクションをロールバックします。

図 1-2 では、Transactions サンプル・アプリケーションのしくみを示しています。

図 1-2Transactions サンプル・アプリケーション



Transactions サンプル・アプリケーションは、次の 2 とおりの方法でトランザクションをロールバックします。

- 致命的ではない場合。コースがデータベースに存在しないか、学生が既に登録されているためにコースの登録が失敗する場合は、それらのコースの数がサーバ・アプリケーションからクライアント・アプリケーションに返されます。トランザクションをロールバックするかどうかは、クライアント・アプリケーションのユーザに任せられます (Transaction クライアント・アプリケーション・コードでは、自動的にトランザクションをロールバックします)。

- 致命的な場合。登録できる単位の最大数を超えているためにコースの登録が失敗する場合は、サーバ・アプリケーションによって CORBA 例外が生成され、その例外がクライアントに返されます。この場合も、トランザクションをロールバックするかどうかは、クライアント・アプリケーションに依存します。

  したがって、Transactions サンプル・アプリケーションでも、ユーザ定義の CORBA 例外をインプリメントする方法が示されています。たとえば、学生が登録可能なコースの最大数を超えているコースに登録しようとすると、サーバ・アプリケーションは、`TooManyCredits` 例外を返します。クライアント・アプリケーションは、この例外を受け取ると、自動的にトランザクションをロールバックします。

注記　BEA Tuxedo CORBA アプリケーションにトランザクションがインプリメントされるしくみについては、BEA Tuxedo オンライン・マニュアルの Transactions サンプルを参照してください。

開発手順

ここでは、トランザクション処理コードが含まれる BEA Tuxedo アプリケーションの以下の開発手順をを説明します。

- ステップ 1: OMG IDL の記述
- ステップ 2: インターフェイスのトランザクション方針の定義
- ステップ 3: サーバ・アプリケーションの記述

- ステップ 4: クライアント・アプリケーションの記述
- ステップ 5: コンフィギュレーション・ファイルの作成

この開発手順では、Transactions サンプル・アプリケーションを例に挙げて説明します。Transactions サンプル・アプリケーションのソース・ファイルは、BEA Tuxedo ソフトウェアの `\samples\corba\university` ディレクトリにあります。Transactions サンプル・アプリケーションのビルドと実行の詳細については、BEA Tuxedo オンライン・マニュアルの Transactions サンプルを参照してください。

ステップ 1: OMG IDL の記述

ほかの CORBA インターフェイスの場合と同じように、トランザクションに関与するインターフェイスを Object Management Group (OMG) インターフェイス定義言語 (IDL) で指定する必要があります。また、そのインターフェイスを使用して発生する可能性のあるすべてのユーザ例外を指定することも必要です。

Transactions サンプル・アプリケーションでは、`Registrar` インターフェイスと `register_for_courses()` オペレーションを OMG IDL で定義します。`register_for_courses()` オペレーションには、`NotRegisteredList` というパラメータがあります。このパラメータは、登録の失敗したコースのリストをクライアント・アプリケーションに返します。`NotRegisteredList` の値が空の場合、クライアント・アプリケーションはトランザクションをコミットします。また、`TooManyCredits` ユーザ例外も定義する必要があります。

リスト 1-1 は、Transactions サンプル・アプリケーションの OMG IDL を示しています。

コード リスト 1-1 Transactions **サンプル・アプリケーションの** OMG IDL

```
#pragma prefix "beasys.com"
module UniversityT

{
       typedef unsigned long CourseNumber;
       typedef sequence<CourseNumber> CourseNumberList;

       struct CourseSynopsis
```

```
        {
                CourseNumber   course_number;
                string         title;
        };
        typedef sequence<CourseSynopsis> CourseSynopsisList;

        interface CourseSynopsisEnumerator
        {
        // エントリがない場合は長さ 0 のリストを返す
        CourseSynopsisList get_next_n(
                in  unsigned long number_to_get, // 0 の場合はすべてが
返される
                out unsigned long number_remaining
        );

        void destroy();
        };
        typedef unsigned short Days;
        const Days MONDAY    =  1;
        const Days TUESDAY   =  2;
        const Days WEDNESDAY =  4;
        const Days THURSDAY  =  8;
        const Days FRIDAY    = 16;
}
// スケジューリングされたすべての曜日の、正時に始まる
// 同じ時間帯に制限されたクラス

struct ClassSchedule
{
        Days           class_days; // 日のビットマスク
        unsigned short start_hour; // 軍用時間による時間
                unsigned short duration;   // 分
};

struct CourseDetails
{
        CourseNumber   course_number;
        double         cost;
        unsigned short number_of_credits;
        ClassSchedule  class_schedule;
        unsigned short number_of_seats;
        string         title;
        string         professor;
        string         description;
};
        typedef sequence<CourseDetails> CourseDetailsList;
        typedef unsigned long StudentId;
```

```
struct StudentDetails
{
       StudentId         student_id;
       string            name;
       CourseDetailsList registered_courses;
};

enum NotRegisteredReason
{
       AlreadyRegistered,
       NoSuchCourse
};

struct NotRegistered
{
       CourseNumber        course_number;
       NotRegisteredReason not_registered_reason;
};
       typedef sequence<NotRegistered> NotRegisteredList;

exception TooManyCredits
{
       unsigned short maximum_credits;
};

// Registrar インターフェイスは、学生がデータベース
// にアクセスするためのメインのインターフェイス
interface Registrar
{
       CourseSynopsisList
       get_courses_synopsis(
              in string                    search_criteria,
              in unsigned long             number_to_get,
              out unsigned long            number_remaining,
              out CourseSynopsisEnumerator rest
);

       CourseDetailsList get_courses_details(in CourseNumberList
        courses);
       StudentDetails get_student_details(in StudentId student);
       NotRegisteredList register_for_courses(
              in StudentId        student,
              in CourseNumberList courses
       ) raises (
              TooManyCredits
       );

};

// RegistrarFactory インターフェイスが Registrar インターフェイスを検索
```

```
interface RegistrarFactory
{
       Registrar find_registrar(
       );
};
```

ステップ 2: インターフェイスのトランザクション方針の定義

トランザクション方針は、インターフェイスごとに使用されます。設計時に、BEA Tuxedo アプリケーション内のどのインターフェイスでトランザクションを処理するかが指定されます。表 1-1 では、CORBA トランザクション方針を説明します。

表 1-1 CORBA トランザクション方針

トランザクション方針	説明
always	インターフェイスは常にトランザクションの一部である必要があります。インターフェイスがトランザクションの一部ではない場合、トランザクションは TP フレームワークによって自動的に開始されます。
ignore	インターフェイスはトランザクションに関与しません。ただし、トランザクションのスコープ内でこのインターフェイスに要求を行うことはできます。このインターフェイスの UBBCONFIG ファイルで指定される AUTOTRAN パラメータは無視されます。
never	インターフェイスはトランザクションに関与しません。このインターフェイス用に作成されるオブジェクトは、トランザクションに関与できません。BEA Tuxedo システムは、この方針が設定されているインターフェイスがトランザクションに関与した場合に例外 (INVALID_TRANSACTION) を生成します。
optional	インターフェイスはトランザクションに関与できます。要求がトランザクションに関係する場合、オブジェクトはトランザクションに関与できます。このトランザクション方針はデフォルトです。

開発段階では、トランザクション方針を割り当てることで、どのインターフェイスがトランザクションで実行されるのかを定義します。インプリメンテーション・コンフィギュレーション • ファイル (ICF) で、トランザクション方針を指定します。`genicf` コマンドを使用すると、テンプレートの ICF ファイルが作成されます。ICF の詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』の「インプリメンテーション・コンフィギュレーション・ファイル (ICF)」を参照してください。

Transactions サンプル・アプリケーションでは、`Registrar` インターフェイスのトランザクション方針が `always` に設定されます。

ステップ 3: サーバ・アプリケーションの記述

サーバ・アプリケーションでトランザクションを使用する場合は、インターフェイスのオペレーションをインプリメントするメソッドを記述する必要があります。Transactions サンプル・アプリケーションでは、`register_for_courses()` オペレーションのメソッド・インプリメンテーションを記述します。

BEA Tuxedo アプリケーションでデータベースを使用する場合は、XA リソース・マネージャを開閉するコードが CORBA サーバ・アプリケーションで必要となります。それらのオペレーションは、Server オブジェクトの `Server::initialize()` オペレーションと `Server::release()` オペレーションに含められます。リスト 1-2 は、XA リソース・マネージャをオープンおよびクローズする Transactions サンプル・アプリケーションの Server オブジェクトのコードの一部分です。

注記　トランザクションをインプリメントする CORBA アプリケーションの完全な例については、BEA Tuxedo オンライン・マニュアルの Transactions サンプルを参照してください。

コード リスト 1-2 Transactions **サンプル・アプリケーションの** C++ Server **オブジェクト**

```
CORBA::Boolean Server::initialize(int argc, char* argv[])
{
        TRACE_METHOD("Server::initialize");
        try {
```

```
                open_database();
                begin_transactional();
                register_fact();
                return CORBA_TRUE;
}
        catch (CORBA::Exception& e) {
                LOG("CORBA exception : " <<e);
        }
        catch (SamplesDBException& e) {
                LOG("Can't connect to database");
        }
        catch (...) {
                LOG("Unexpected database error : " <<e);
        }
        catch (...) {
                LOG("Unexpected exception");
        }
        cleanup();
        return CORBA_FALSE;
}

void Server::release()
{
        TRACE_METHOD("Server::release");
        cleanup();
}

static void cleanup()
{
        unregister_factory();
        end_transactional();
        close_database();
}
// トランザクション・リソース・マネージャを管理するユーティリティ

CORBA::Boolean s_became_transactional = CORBA_FALSE;
static void begin_transactional()
{
        TP::open_xa_rm();
        s_became_transactional = CORBA_TRUE;
}
static void end_transactional()
{
        if(!s_became_transactional){
        return// クリーンアップは不要
}
try {
        TP::close_xa_rm ();
}
```

```
        catch (CORBA::Exception& e) {
                LOG("CORBA Exception : " << e);
        }
        catch (...) {
                LOG("unexpected exception");
        }

        s_became_transactional = CORBA_FALSE;
}
```

ステップ 4: クライアント・アプリケーションの記述

クライアント・アプリケーションでは、次のタスクを実行するコードが必要です。

1. Bootstrap オブジェクトから TransactionCurrent オブジェクトのリファレンスを取得します。
2. TransactionCurrent オブジェクトの `Tobj::TransactionCurrent::begin()` オペレーションを呼び出してトランザクションを開始します。
3. オブジェクトのオペレーションを呼び出します。Transactions サンプル・アプリケーションでは、クライアント・アプリケーションが (コースのリストを渡して) Registrar オブジェクトの `register_for_courses()` オペレーションを呼び出します。

リスト 1-3 は、トランザクションの開発手順を説明する Transactions サンプル・アプリケーションの CORBA C++ クライアント・アプリケーションの一部分です。

注記　リスト 1-3 で示されているサンプル・コードは、BEA ブートストラップ処理メカニズムの使用方法を示したものです。INS ブートストラップ処理メカニズムを使用する際の詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』の「CORBA ブートストラップ処理のプログラミング・リファレンス」を参照してください。

ActiveX クライアント・アプリケーションでのトランザクションの使用例については、「第 4 章 CORBA クライアント・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。

コード リスト 1-3 CORBA C++ クライアント・アプリケーションのトランザクション・コード

```
CORBA::Object_var var_transaction_current_oref =
     Bootstrap.resolve_initial_references(“TransactionCurrent”);
CosTransactions::Current_var transaction_current_oref=
     CosTransactions::Current::_narrow(var_transaction_current_oref.in());
// トランザクションを開始
var_transaction_current_oref->begin();
try {
// トランザクション内でオペレーションを実行
     pointer_Registar_ref->register_for_courses(student_id, course_number_list);
     ...
     // オペレーションがエラーなしで実行された場合は、トランザクションをコミット
     CORBA::Boolean report_heuristics = CORBA_TRUE;
     var_transaction_current_ref->commit(report_heuristics);
    }
catch (...) {
      // オペレーションで実行エラーが発生した場合は、トランザクションをロールバック
      // 次に元の例外を再びスローする
      // ロールバックが失敗した場合、例外を無視し、
      // 元の例外を再びスローする
try {
     var_transaction_current_ref->rollback();
    }
catch (...) {
            TP::userlog("rollback failed");
            }
throw;
}
```

ステップ 5: コンフィギュレーション・ファイルの作成

トランザクション対応 BEA Tuxedo アプリケーションのコンフィギュレーション・ファイルには、次の情報を追加する必要があります。

- GROUPS セクション
  - OPENINFO パラメータで、データベースのリソース・マネージャをオープンするのに必要な情報を指定します。この情報は、データベースの製品マニュアルから取得します。デフォルト・バージョンの

com.beasys.Tobj.Server.initialize メソッドは、自動的にリソース・マネージャをオープンします。

- CLOSEINFO パラメータで、データベースのリソース・マネージャをクローズするのに必要な情報を指定します。デフォルトでは、CLOSEINFO パラメータは空です。
- TMSNAME パラメータと TMSCOUNT パラメータを指定して、XA リソース・マネージャを、指定したサーバ・グループに関連付けます。

- SERVERS セクションで、インターフェイスがあるサーバ・アプリケーションと、データベースを管理するサーバ・アプリケーションの両方を含むサーバ・グループを定義します。このサーバ・グループは、トランザクションとして指定する必要があります。
- TLOGDEVICE パラメータで、トランザクション・ログ (TLOG) にパス名を指定します。トランザクション・ログの詳細については、「第 5 章 トランザクションの管理」を参照してください。

リスト 1-4 は、Transactions サンプル・アプリケーションのコンフィギュレーション・ファイルでこの情報が定義されている部分です。

コード リスト 1-4 Transactions サンプル・アプリケーションのコンフィギュレーション・ファイル

```
*RESOURCES
       IPCKEY     55432
       DOMAINID   university
       MASTER     SITE1
       MODEL      SHM
       LDBAL      N
       SECURITY   APP_PW

*MACHINES
       BLOTTO
       LMID = SITE1
       APPDIR = C:\TRANSACTION_SAMPLE
       TUXCONFIG=C:\TRANSACTION_SAMPLE\tuxconfig
       TLOGDEVICE=C:\APP_DIR\TLOG
       TLOGNAME=TLOG
       TUXDIR="C:\tuxdir"
       MAXWSCLIENTS=10
```

```
*GROUPS
        SYS_GRP
          LMID      = SITE1
          GRPNO     = 1
        ORA_GRP
          LMID      = SITE1
          GRPNO     = 2

        OPENINFO  = "ORACLE_XA:Oracle_XA+SqlNet=ORCL+Acc=P
        /scott/tiger+SesTm=100+LogDir=.+MaxCur=5"
        CLOSEINFO = ""
        TMSNAME   = "TMS_ORA"
        TMSCOUNT  = 2

*SERVERS
        DEFAULT:
        RESTART = Y
        MAXGEN  = 5

        TMSYSEVT
          SRVGRP  = SYS_GRP
          SRVID   = 1

        TMFFNAME
          SRVGRP  = SYS_GRP
          SRVID   = 2
          CLOPT   = "-A -- -N -M"

        TMFFNAME
          SRVGRP  = SYS_GRP
          SRVID   = 3
          CLOPT   = "-A -- -N"

        TMFFNAME
          SRVGRP  = SYS_GRP
          SRVID   = 4
          CLOPT   = "-A -- -F"

        TMIFRSVR
          SRVGRP  = SYS_GRP
          SRVID   = 5

        UNIVT_SERVER
          SRVGRP  = ORA_GRP
          SRVID   = 1
          RESTART = N

        ISL
          SRVGRP  = SYS_GRP
```

```
         SRVID   = 6
         CLOPT   = -A -- -n //MACHINENAME:2500

*SERVICES
```

トランザクション・ログの詳細とコンフィギュレーション・ファイルのパラメータの定義については、「第 5 章 トランザクションの管理」を参照してください。

2 トランザクション・サービス

ここでは、以下の内容について説明します。

- トランザクション・サービスについて
- 機能と制限事項
- CORBA アプリケーションのトランザクション・サービス
- UserTransaction API

ここでは、BEA Tuxedo システムで トランザクション CORBA アプリケーションを作成するのに必要な情報を提供します。始める前に、第 1 章 「トランザクションについて」を読む必要があります。

トランザクション・サービスについて

BEA Tuxedo は、CORBA アプリケーションでのトランザクションをサポートするトランザクション・サービスを提供します。トランザクション・サービスは、OMG CORBA サービス Transaction Service Specification で記述されている CORBA サービス・トランザクション・サービスのインプリメンテーションを提供します。この仕様では、トランザクション機能を提供するオブジェクト・サービスのインターフェイスを定義しています。

機能と制限事項

ここでは、以下の内容について説明します。

- 委譲型コミットによるライトウェイト・クライアント
- INS を使用するサード・パーティ・クライアントのサポート
- 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
- トランザクションの整合性
- トランザクションの終了
- フラット・トランザクション
- CORBA リモート・クライアントと BEA Tuxedo ドメインの相互運用性
- ドメイン内およびドメイン間の相互運用性
- ネットワークの相互運用性
- トランザクション・サービスとトランザクション処理の関係
- プロセスの障害
- 一般的な制限事項

ここでは、CORBA 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事項を説明します。

委譲型コミットによるライトウェイト・クライアント

ライトウェイト・クライアントは、不定期的に使用される単一ユーザの未管理デスクトップ・システム上で動作します。所有者は、使用しない場合にそれぞれのデスクトップ・システムをオフにすることができます。単一ユーザの未管理デスクトップ・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の実行は要求できません。特に、未管理システムでは、サーバ・リソースを必要とするトランザクションの失敗に対して原子性、一貫性、独立性、持続性などの ACID 特性を保証する役割を果たすことはできません。BEA Tuxedo CORBA リモート・クライアントは、ライトウェイト・クライアントです。

トランザクション・サービスでは、ライトウェイト・クライアントは委譲型コミットを実行できます。つまり、サーバ・マシンで動作しているトランザクション・マネージャにトランザクションの調整が委譲されているときに、トランザクションを開始および終了できます。クライアント・アプリケーションは、ローカル・トランザクション・サーバを必要としません。CORBA クライアントが使用するリモート TransactionCurrent インプリメンテーションは、実際のトランザクションの調整をサーバ上のトランザクション・マネージャに委譲します。

INS を使用するサード・パーティ・クライアントのサポート

BEA Tuxedo リリース 8.0 以降では、CORBA インターオペラブル・ネーミング・サービス (INS) がサポートされます。したがって、CORBA のオブジェクト・トランザクション・サービス (OTS) をインプリメントしたクライアントは、BEA Tuxedo CORBA サーバと通信し、トランザクションを開始および終了できます。INS を使用すると、標準 OTS IDL ファイルをコンパイルし、使用可能なスタブ・ファイルを作成できるサード・パーティ・クラ

イアント ORB は、BEA Tuxedo CORBA トランザクション・マネージャと対話できます。ただし、サード・パーティ ORB をリソース・マネージャとして使用できるようにするトランザクション調整インターフェイスがサポートされていないため、この対話処理は制限されます。BEA 提供のリソース・マネージャや XA 準拠のリソース・マネージャのみが、トランザクションの調整に参加できます。さらに、BEA 提供および XA 準拠のリソース・マネージャは､トランザクションの調整に対して CORBA の OTS ではなく XA プロトコルを使用する場合にのみ、トランザクションの調整に参加できます。

つまり、サード・パーティ・クライアント ORB を使用して、トランザクションを開始できます。また、クライアントは、トランザクションのロールバックまたはコミットを要求できますが、クライアント ORB は、CORBA の OTS を使用して 2 フェーズ・コミット・プロトコルの調整に参加することはできません。

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

リリース 8.0 では、BEA Tuxedo CORBA は、トランザクション処理を行わないクライアントとトランザクション処理を行うクライアントに対してマルチスレッド・クライアントをサポートします。

トランザクションの伝達 (CORBA のみ)

CORBA アプリケーションの場合、OMG CORBA のトランザクション・サービス仕様で、クライアントがトランザクション・コンテキストの伝達を暗黙的にするか明示的にするかを選択できることが規定されています。BEA Tuxedo では、暗黙的な伝達を提供します。明示的な伝達はできる限り使用しないでください。

明示的なトランザクションの伝達を使用して渡されるトランザクション・コンテキストに関連付けられたオブジェクトは、暗黙的なトランザクション伝達 API と混在しないようにしください。ただし、明示的な伝達は、トランザ

クション・メソッドが処理可能な場合に対してはどのような制約も課しません。トランザクションをコミットする前にすべてのトランザクション・メソッドが完了するかどうかは保証できません。

トランザクションの整合性

チェックされたトランザクションの振る舞いでは、トランザクションに必要な、トランザクションに関与するすべてのオブジェクトがトランザクションに関する要求を完了するまで `commit` が成功しないようにすることでトランザクションの整合性を提供します。暗黙的なトランザクションの伝達を使用する場合、トランザクション・サービスは、チェックされたトランザクションの振る舞いを提供します。このトランザクションの振る舞いは、Open Group によって定義された要求 / 応答のプロセス間通信モデルで提供されたものと同じです。たとえば、CORBA アプリケーションの場合、トランザクション・サービスは、OMG CORBA のトランザクション・サービス仕様で記述されているとおり、`reply` チェック、`commit` チェック、`resume` チェックを実行します。

チェックされていないトランザクションの振る舞いは、トランザクションの整合性を提供するアプリケーションに完全に依存します。明示的な伝達を使用した場合、トランザクション・サービスではチェックされたトランザクションの振る舞いを提供しないため、トランザクションの整合性は保証されません。

トランザクションの終了

BEA Tuxedo CORBA では、トランザクションを作成したクライアントからのみトランザクションを終了できます。

注記　クライアントは、別のオブジェクトのサービスを要求するサーバ・オブジェクトとしても使用できます。

フラット・トランザクション

BEA Tuxedo CORBA ではフラット・トランザクション・モデルをインプリメントします。入れ子になったトランザクションはサポートされていません。

CORBA リモート・クライアントと BEA Tuxedo ドメインの相互運用性

BEA Tuxedo CORBA は、同じトランザクション内で、異なる BEA Tuxedo ドメインにあるサーバ・オブジェクトのメソッドを呼び出すリモート・クライアントをサポートします。

リモート CORBA クライアントでは、同じ BEA Tuxedo ドメインへの複数の接続がある場合、同じトランザクション内の別の接続上にあるサーバ・オブジェクトに対して呼び出しを実行できます。

ドメイン内およびドメイン間の相互運用性

BEA Tuxedo CORBA は、BEA Tuxedo ドメインでサーバ・オブジェクトのメソッドを呼び出すネイティブ・クライアントをサポートします。さらに、BEA Tuxedo は、同じ BEA Tuxedo ドメイン内の同じ、または異なるプロセスで、ほかのオブジェクトのメソッドを呼び出すサーバ・オブジェクトをサポートしています。

BEA Tuxedo アプリケーションでは、ファクトリ・ベース・ルーティングが複数のドメインにわたって正しくコンフィギュレーションされている限り、トランザクションは複数のドメインで使用できます。複数のドメインにわたるトランザクションをサポートするには、現在の (ローカル) ドメインで使用されるものの、別の (リモート) ドメインに存在するファクトリ・オブジェクトを識別するように `factory_finder.ini` ファイルをコンフィギュレーションする必要があります。詳細については、『BEA Tuxedo Domains コンポーネント』を参照してください。

ネットワークの相互運用性

クライアント・アプリケーションは、1 つのドメイン内にアクティブな Bootstrap オブジェクトと TransactionCurrent オブジェクトを 1 つだけ持つことができます。BEA Tuxedo CORBA では、リモート BEA Tuxedo ドメインとの間のトランザクションのエクスポートまたはインポートをサポートしていません。

ただし、トランザクションは、順次的に複数のドメインを含むことができます。たとえば、ドメイン A でアクティブなトランザクションを持つサーバは、それと同じトランザクション・コンテキスト内のドメイン B でサーバと通信できます。

トランザクション・サービスとトランザクション処理の関係

トランザクション・サービスは、以下のようにさまざまなトランザクション処理サーバ、インターフェイス、プロトコル、および標準に関連します。

- BEA Tuxedo ATMI サーバに対するサポート。BEA Tuxedo CORBA トランザクション・サービスを使用するサーバは、同じドメインのほかのアプリケーション・トランザクション・モニタ・インターフェイス (ATMI) サーバ・プロセスに対して呼び出しを実行できます。さらに、ATMI サービスは、トランザクションに関与するコンテキストでも、関与しないコンテキストでも、BEA Tuxedo ドメイン・ゲートウェイを介して、同じドメインおよび複数のドメインの両方で、CORBA オブジェクトを呼び出すことができます。ただし、BEA Tuxedo CORBA は、BEA Tuxedo ドメイン内の ATMI サービスを呼び出すリモート・クライアントまたはネイティブ・クライアントをサポートしません。
- Open Group の XA インターフェイスに対するサポート。Open Group のリソース・マネージャは、2 フェーズ・コミット・プロトコルを Open Group の XA インターフェイスで制御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。BEA Tuxedo

は、Open Group リソース・マネージャとの対話処理をサポートしています。

- OSI TP プロトコルに対するサポート。開放型システム間相互接続のトランザクション処理 (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・プロトコルです。BEA Tuxedo CORBA は、OSI TP トランザクションとの対話処理をサポートしません。

- LU 6.2 プロトコルに対するサポート。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・プロトコルです。BEA Tuxedo CORBA は、LU 6.2 トランザクションとの対話処理をサポートしません。

- ODMG 標準に対するサポート。ODMG-93 は Object Database Management Group (ODMG) によって定義された標準です。オブジェクト・データベース管理システムにアクセスするための移植可能なインターフェイスについて記述されています。BEA Tuxedo CORBA は、ODMG トランザクションとの対話処理をサポートしません。

プロセスの障害

トランザクション・サービスは、トランザクションのパーティシパントを監視し、障害や非アクティブの状態の有無をチェックします。BEA Tuxedo システムは、障害が発生した場合にアプリケーションの実行を維持するための管理ツールを提供します。BEA Tuxedo CORBA は、BEA Tuxedo トランザクション管理システムで構築されるので、アプリケーションの実行を維持するための BEA Tuxedo 機能を継承します。

一般的な制限事項

トランザクション・サービスには、以下の制限があります。

- BEA Tuxedo CORBA では、クライアント・オブジェクトまたはサーバ・オブジェクトは、別のトランザクションに関与している (または、参加している) オブジェクトのメソッドを呼び出すことはできません。クラ

イアントまたはサーバがメソッドを呼び出そうとすると、例外が返されます。

- CORBA アプリケーションでは、BEA Tuxedo CORBA トランザクション・サービス・ライブラリのトランザクションを使用しているサーバ・アプリケーション・オブジェクトは、TP フレームワーク機能を必要とします。TP フレームワークの詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』の「TP フレームワーク」を参照してください。
- CORBA アプリケーションの場合、`Current` オブジェクトの `rollback` メソッドからの戻り値は非同期的です。

  その結果、ロールバックされたトランザクションに関与していた (または、参加していた) オブジェクトは、BEA Tuxedo によってクリアされた状態を少し後で取得します。したがって、BEA Tuxedo がこれらのオブジェクトの状態をクリアするまで、他のクライアントは、これらのオブジェクトを別のトランザクションに関与させることはできません。この状態はごく短い時間しか存在しないため、通常は製品アプリケーションで問題になることはありません。この状態に対する簡単な対策は、短時間 (通常は 1 秒) の遅延の後で適切なオペレーションを試してみることです。
- BEA Tuxedo CORBA アプリケーションでは、クライアントは、トランザクションのコンテキスト内で、`NEVER`、`OPTIONAL`、または `ALWAYS` トランザクション方針を持つサーバ・オブジェクトに対する一方向のメソッド呼び出しを実行できません。

  一方向のメソッド呼び出しのため、エラーまたは例外はクライアントに返されません。ただし、サーバ・オブジェクトのメソッドは実行されず、適切なエラー・メッセージがログに書き込まれます。クライアントは、トランザクションのコンテキスト内で、`IGNORE` トランザクション方針を持つサーバ・オブジェクトに対する一方向のメソッド呼び出しを実行できます。この場合、サーバ・オブジェクトのメソッドは実行されますが、トランザクションのコンテキスト内ではありません。トランザクション方針の詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』の「インプリメンテーション・コンフィギュレーション・ファイル (ICF)」を参照してください。

CORBA アプリケーションのトランザクション・サービス

ここでは、以下の内容について説明します。

- Bootstrap オブジェクトを使用した TransactionCurrent オブジェクトへの初期リファレンスの取得
- INS を使用した TransactionFactory オブジェクトへの初期リファレンスの取得
- CORBA トランザクション・サービス API
- CORBA トランザクション・サービス API の拡張
- BEA Tuxedo CORBA アプリケーションのトランザクションの使用に関する注意事項

ここでは、インプリメンテーション固有のものとして CORBA のオブジェクト・トランザクション・サービスの一部を特に取り上げ、BEA Tuxedo がどのように OTS をインプリメントするかを説明します。また、トランザクションの開始、終了、中断、または再開、およびトランザクションに関する情報の取得に使用する OTS アプリケーション・プログラミング・インターフェイス (API) について説明します。

Bootstrap オブジェクトを使用した TransactionCurrent オブジェクトへの初期リファレンスの取得

TransactionCurrent オブジェクトを使用してトランザクション・サービス API およびトランザクション・サービス API の拡張 (後述) にアクセスするには、アプリケーションは、以下のオペレーションを完了する必要があります。

1. Bootstrap オブジェクトを作成します。Bootstrap オブジェクト作成の詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』の「インプリメンテーション・コンフィギュレーション・ファイル (ICF)」を参照してください。

2. Bootstrap オブジェクトの `resolve_initial_reference("TransactionCurrent")` メソッドを呼び出します。標準 CORBA オブジェクト・ポインタが返されます。この Bootstrap オブジェクト・メソッドの詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』を参照してください。

3. アプリケーションは、トランザクション・サービス API のみを必要とする場合、上記のステップ 2 で返されたオブジェクト・ポインタに対して `org.omg.CosTransactions.Current.narrow()` (Java の場合) または `CosTransactionsCurrent::_narrow()` (C++ の場合) を発行する必要があります。

   アプリケーションは、拡張を含むトランザクション・サービス API を必要とする場合、上記のステップ 2 で返されたオブジェクト・ポインタに対して `com.beasys.Tobj.TransactionCurrent.narrow()` (Java の場合) または `Tobj::TransactionCurrent::_narrow()` (C++ の場合) を発行する必要があります。

INS を使用した TransactionFactory オブジェクトへの初期リファレンスの取得

BEA Tuxedo では、サード・パーティ・クライアントによる CORBA インターオペラブル・ネーミング・サービス (INS) も使用して初期トランザクション・オブジェクト・リファレンスを取得することもできます。INS は、`ORB::resolve_initial_references()` オペレーションを使用します。

リスト 2-1 では、クライアント・アプリケーションが INS を使用して TransactionFactory オブジェクトを取得するしくみを示します。完全なコード例については、University Sample のクライアント・アプリケーションを参照してください。

コード リスト 2-1 INS **を使用したクライアント・アプリケーションのコード例**

```
// ORB からファクトリ・ファインダを取得
CORBA::Object_var v_fact_finder_oref =
   orb->resolve_initial_references("FactoryFinder");

// ファクトリ・ファインダをナロー変換
Tobj::FactoryFinder_var v_fact_finder_ref =
   Tobj::FactoryFinder::_narrow(v_fact_finder_oref.in());

// FactoryFinder から TransactionFactory を取得
CORBA::Object_var v_txn_fac_oref =
   v_fact_finder_ref->find_one_factory_by_id(
            "IDL:omg.org/CosTransactions/TransactionFactory:1.0");

// TransactionFactory オブジェクト・リファレンスをナロー変換
CosTransactions::TransactionFactory_var v_txn_fac_ref =
   CosTransactions::TransactionFactory::_narrow(
                                          v_txn_fac_oref.in());
```

`ORB::resolve_initial_references()` オペレーションの使い方については、『BEA Tuxedo CORBA プログラミング・リファレンス』の「CORBA ブートストラップ処理のプログラミング・リファレンス」を参照してください。

CORBA トランザクション・サービス API

ここでは、以下の内容について説明します。

- データ型
- 例外
- Current インターフェイス
- Control インターフェイス
- TransactionalObject インターフェイス

ここでは、BEA Tuxedo がトランザクション・サービスをサポートするためにインプリメントする `CosTransactions` モジュールの CORBA ベース・コンポーネントについて説明します。これらのコンポーネントの詳細については、「OMG CORBA Services Transaction Service Specification, Version 1.1」(2000 年 5 月) を参照してください。

データ型

リスト 2-2 には、サポートされているデータ型が示されています。

コード リスト 2-2 トランザクション・サービスでサポートされているデータ型

```
enum Status {

        StatusActive,
        StatusMarkedRollback,
        StatusPrepared,
        StatusCommitted,
        StatusRolledBack,
        StatusUnknown,
        StatusNoTransaction,
        StatusPreparing,
        StatusCommitting,
        StatusRollingBack
};
// この情報は、「OMG Transaction Service Specification,
// Version 1.1」(2000 年 5 月 ) から。OMG の使用許可を得て
// 使用
```

例外

リスト 2-3 には、サポートされている IDL コードの例外が示されています。

コード リスト 2-3 トランザクション・サービスでサポートされている例外

```
// ヒューリスティックな例外
exception HeuristicMixed {};
exception HeuristicHazard {};

// その他のトランザクション固有の例外
exception SubtransactionsUnavailable {};
exception NoTransaction {};
exception InvalidControl {};
exception Unavailable {};
```

表 2-1 では、例外を説明します。

注記　これらの情報は、OMG から使用許可を得て、「OMG CORBA Services Transaction Service Specification, Version 1.1」(2000 年 5 月) のものを利用しました。

表 2-1 トランザクション・サービスでサポートされている例外

例外	説明
HeuristicMixed	ヒューリスティックな決定がなされたこと、関連する更新の一部がコミットされ、ほかはロールバックされたことを通知するために、要求に対してこの例外が発生します。
HeuristicHazard	ヒューリスティックな決定がなされたこと、関連するすべての更新の処理が不明であること、および処理が認識された更新について、すべてがコミットされたかロールバックされたことを通知するために、要求に対してこの例外が発生します。したがって、HeuristicMixed 例外は、HeuristicHazard 例外よりも優先されます。
SubtransactionsUnavailable	クライアントに既に関連するトランザクションがある場合、Current インターフェイスの begin メソッドに対して、この例外が発生します。

表 2-1 トランザクション・サービスでサポートされている例外（続き）

例外	説明
NoTransaction	クライアント・アプリケーションに関連付けられているトランザクションがない場合、Current インターフェイスの rollback および rollback_only メソッドに対して、この例外が発生します。
InvalidControl	パラメータが現在の実行環境で有効ではない場合、Current インターフェイスの resume メソッドに対して、この例外が発生します。
Unavailable	Control インターフェイスが要求されたオブジェクトを提供できない場合、Control インターフェイスの get_terminator メソッドと get_coordinator メソッドに対して、この例外が発生します。

Current インターフェイス

Current インターフェイスは、トランザクション・サービスのクライアントがスレッドとトランザクションの関連付けを明示的に管理できるようにするメソッドを定義します。また、Current インターフェイスは、ほとんどのアプリケーションでトランザクション・サービスを簡単に使用できるようにするメソッドも定義します。これらのメソッドは、トランザクションの開始、終了、中断、再開、および現在のトランザクションに関する情報の取得に使用できます。

CosTransactions モジュールは、Current インターフェイスを定義します (リスト 2-4 を参照)。

コード リスト 2-4 Current インターフェイスの IDL

```
// Current のトランザクション
interface Current : CORBA::Current {
        void begin()
                raises(SubtransactionsUnavailable);
        void commit(in boolean report_heuristics)
                raises(
                        NoTransaction,
                        HeuristicMixed,
```

```
                    HeuristicHazard
            );
        void rollback()
                raises(NoTransaction);
        void rollback_only()
                raises(NoTransaction);
        Status get_status();
        string get_transaction_name();
        void set_timeout(in unsigned long seconds);
        Control get_control();
        Control suspend();
        void resume(in Control which)
                raises(InvalidControl);

};

// この情報は、「OMG Transaction Service Specification,
// Version 1.1」(2000 年 5 月 ) から。OMG の使用許可を得て
// 使用
```

表 2-2 では、Current トランザクション・メソッドについて説明します。

注記　これらの情報は、OMG から使用許可を得て、「OMG CORBA Services Transaction Service Specification, Version 1.1」(2000 年 5 月) のものを利用しました。

表 2-2 Current オブジェクトのトランザクション・メソッド

メソッド	説明
begin	新しいトランザクションを作成します。スレッドが新しいトランザクションに関連付けられるように、クライアント・アプリケーションのトランザクション・コンテキストが変更されます。クライアント・アプリケーションが既にトランザクションに関連付けられている場合は、SubtransactionsUnavailable 例外が発生します。トランザクション開始時にエラーが発生したためにクライアント・アプリケーションをトランザクション・モードに配置できない場合は、標準システム例外 INVALID_TRANSACTION が発生します。呼び出し時のコンテキストが無効な場合は、標準システム例外 BAD_INV_ORDER が発生します。

表 2-2 Current オブジェクトのトランザクション・メソッド (続き)

メソッド	説明
commit	クライアント・アプリケーションに関連付けられているトランザクションがない場合は、NoTransaction 例外が発生します。 呼び出し時のコンテキストが無効な場合は、標準システム例外 BAD_INV_ORDER が発生します。 システムでトランザクションのロールバックを決定すると、標準例外 TRANSACTION_ROLLEDBACK が発生し、スレッドのトランザクション・コンテキストが NULL に設定されます。 ヒューリスティックな決定がなされたこと、関連する更新の一部がコミットされ、ほかはロールバックされたことを通知するために、HeuristicMixed 例外が発生します。ヒューリスティックな決定がなされたこと、関連するすべての更新の処理が不明であること、および処理が認識された更新について、すべてがコミットされたかロールバックされたことを通知するために、HeuristicHazard 例外が発生します。HeuristicMixed 例外は、HeuristicHazard 例外よりも優先されます。ヒューリスティックな例外が発生するか、オペレーションが正常に完了した場合、スレッドのトランザクション例外コンテキストは NULL に設定されます。 オペレーションが正常に完了した場合、スレッドのトランザクション・コンテキストは NULL に設定されます。
rollback	クライアント・アプリケーションに関連付けられているトランザクションがない場合は、NoTransaction 例外が発生します。 呼び出し時のコンテキストが無効な場合は、標準システム例外 BAD_INV_ORDER が発生します。 オペレーションが正常に完了した場合、スレッドのトランザクション・コンテキストは NULL に設定されます。

表 2-2 Current オブジェクトのトランザクション・メソッド (続き)

メソッド	説明
rollback_only	クライアント・アプリケーションに関連付けられているトランザクションがない場合は、NoTransaction 例外が発生します。それ以外の場合、トランザクションの可能な結果のみがロールバックされるように、クライアント・アプリケーションに関連付けられているトランザクションが変更されます。
get_status	クライアント・アプリケーションに関連付けられているトランザクションがない場合は、StatusNoTransaction 値が返されます。それ以外の場合、このメソッドは、クライアント・アプリケーションに関連付けられているトランザクションのステータスを返します。
get_transaction_name	クライアント・アプリケーションに関連付けられているトランザクションがない場合は、空の文字列が返されます。それ以外の場合、このメソッドは、トランザクションを説明する出力可能な文字列 (特に、Open Group で指定されている XID) を返します。返された文字列は、デバッグ時のサポートを目的としています。

表 2-2 Current オブジェクトのトランザクション・メソッド (続き)

メソッド	説明
set_timeout	このメソッドは、タイムアウト値に影響を与えるターゲット・オブジェクトに関連付けられている状態変数を変更します。タイムアウト値は、その後の begin メソッドの呼び出しによって作成されるトランザクションに関連付けられています。 トランザクション・タイムアウトの初期値は 300 秒です。0 より大きい引数値を付けて set_timeout() を呼び出すと、新しいタイムアウト値が指定されます。引数 0 を付けて set_timeout() を呼び出すと、タイムアウト値がデフォルトの 300 秒に戻ります。 set_timeout() を呼び出すと、その後の begin の呼び出しで作成されたトランザクションは、指定した作成後の秒数が経過するまでに完了しなかった場合にロールバックの対象になります。 注記　トランザクション・タイムアウトの初期値は 300 秒です。begin メソッドを使用せずに、AUTOTRAN を介してトランザクションを開始した場合、タイムアウト値は、BEA Tuxedo コンフィギュレーション・ファイルの TRANTIME パラメータで指定した値です。詳細については、第 5 章「トランザクションの管理」を参照してください。
get_control	クライアントがトランザクションに関連付けられていない場合は、NULL オブジェクト・リファレンスが返されます。それ以外の場合、クライアント・アプリケーションに関連付けられているトランザクション・コンテキストを表す Control オブジェクトが返されます。このオブジェクトは、このコンテキストを再確立するために resume メソッドに提供できます。

表 2-2 Current オブジェクトのトランザクション・メソッド (続き)

メソッド	説明
suspend	クライアント・アプリケーションがトランザクションに関連付けられていない場合は、NULL オブジェクト・リファレンスが返されます。 関連付けられているトランザクションが、可能なトランザクションの結果のみをロールバックする状態にある場合、標準システム例外 TRANSACTION_ROLLEDBACK が発生し、クライアント・アプリケーションに関連付けられるトランザクションはなくなります。 呼び出し時のコンテキストが無効な場合は、標準システム例外 BAD_INV_ORDER が発生します。トランザクションに関する呼び出し元の状態は変化しません。 それ以外の場合、クライアント・アプリケーションに関連付けられているトランザクション・コンテキストを表すオブジェクトが返されます。同じクライアントは、このコンテキストを再確立するために、このオブジェクトを resume メソッドに提供できます。さらに、クライアント・アプリケーションに関連付けられるトランザクションはなくなります。 注記　「The Common Object Request Broker: Architecture and Specification, Revision 2.4」にあるとおり、標準システム例外 TRANSACTION_ROLLEDBACK は、要求に関連付けられているトランザクションが既にロールバックされているか、ロールバックとしてマークされていることを示します。したがって、トランザクションの代わりに実行される処理は無効になるので、要求されたメソッドが実行できなかったか、実行されなかったことを示します。

表 2-2 Current オブジェクトのトランザクション・メソッド (続き)

メソッド	説明
resume	クライアントに既に関連付けられているトランザクションが、可能なトランザクションの結果のみをロールバックする状態にある場合、標準システム例外 TRANSACTION_ROLLEDBACK が発生し、クライアント・アプリケーションに関連付けられるトランザクションはなくなります。 呼び出し時のコンテキストが無効な場合は、標準システム例外 BAD_INV_ORDER が発生します。 呼び出し元が、1 つまたは複数のリソース・マネージャとグローバル・トランザクション外の作業に関与しているため、システムがグローバル・トランザクションを再開できない場合、標準システム例外 INVALID_TRANSACTION が発生します。 パラメータが NULL オブジェクト・リファレンスの場合、クライアント・アプリケーションに関連付けられるトランザクションはなくなります。現在の実行環境でパラメータが有効な場合、クライアント・アプリケーションは、(以前のトランザクションに代わって) そのトランザクションに関連付けられます。それ以外の場合、InvalidControl 例外が発生します。 注記　標準システム例外 TRANSACTION_ROLLEDBACK の定義については、suspend を参照してください。

Control インターフェイス

Control インターフェイスを使用すると、プログラムがトランザクション・コンテキストを明示的に管理および伝達できるようになります。Control インターフェイスをサポートするオブジェクトは、特定のトランザクションに暗黙的に関連付けられます。

リスト 2-5 では、CosTransactions モジュールで定義されている Control インターフェイスを示します。

コード リスト 2-5 Control **インターフェイス**

```
interface Control {
       Terminator get_terminator()
              raises(Unavailable);
       Coordinator get_coordinator()
              raises(Unavailable);
};

// この情報は、「OMG Transaction Service Specification,
// Version 1.1」(2000 年 5 月 ) から。OMG の使用許可を得て
// 使用
```

Control インターフェイスは、suspend メソッドおよび resume メソッドでのみ使用します。

Terminator インターフェイス

Terminator インターフェイスは、トランザクションをコミットまたはロールバックするオペレーションをサポートします。通常、これらのオペレーションは、トランザクションの開始元によって使用されます。トランザクション・サービスのインプリメンテーションは、Terminator の使用範囲を制限できます。最低で、単一スレッド内で使用できます。

リスト 2-6 では、Terminator インターフェイスを示します。

コード リスト 2-6 Terminator **インターフェイス**

```
interface Terminator {
     void commit(in boolean report_heuristics)
         raises(
             HeuristicMixed,
             HeuristicHazard
);
    void rollback();
};

// この情報は、「OMG Transaction Service Specification,
// Version 1.1」(2000 年 5 月 ) から。OMG の使用許可を得て
// 使用
```

表 2-3 では、Terminator インターフェイスのメソッドを示します。

表 2-3 Termination インターフェイスのメソッド

メソッド	説明
`commit`	トランザクションが rollback only としてマークされておらず、トランザクションのすべてのパーティシパントがコミットに同意した場合、トランザクションはコミットされ、オペレーションは正常に終了します。それ以外の場合、トランザクションはロールバックされ (下記の `rollback` メソッドを参照)、標準例外 `TRANSACTION_ROLLEDBACK` が発生します。 `report_heuristics` パラメータが true の場合、トランザクション・サービスは、`HeuristicMixed` 例外と `HeuristicHazard` 例外を使用して、矛盾している (またはその可能性がある) 結果を通知します。トランザクション・サービスのインプリメンテーションでは、オプションの CORBA ノーティフィケーション・サービスを使用すると、ヒューリスティックな決定をレポートできます。 コミットまたはロールバックされていないトランザクションのサブトランザクションがある場合、または完了していないトランザクションに関連付けられている (あるいはその可能性がある) アクティビティがある場合、`commit` オペレーションはトランザクションをロールバックできます。このようなエラー・チェックの性質と程度は、インプリメンテーションによって異なります。最上位トランザクションがコミットされると、回復可能なオブジェクトに対する、このトランザクションの範囲内の変更は確定され、他のトランザクションまたはクライアントでも認識可能なものになります。サブトランザクションがコミットされた場合は、リソースで適用された独立性の程度に応じて、ほかの関連するトランザクションでも変更が認識可能になります。

表 2-3 Termination インターフェイスのメソッド

メソッド	説明
`rollback`	トランザクションをロールバックします。 トランザクションがロールバックされると、回復可能なオブジェクトに対する、このトランザクションの範囲内の変更 (下位トランザクションによる変更も含む) がロールバックされます。 トランザクションでロックされたすべてのリソースは、リソースで適用された独立性の程度に応じて、ほかの関連するトランザクションで使用可能になります。

TransactionalObject インターフェイス

BEA Tuxedo リリース 8.0 以降では、`CosTransactions::TransactionalObject` は、トランザクションに関与することを示すオブジェクトによって使用されなくなりました。インターフェイスが TransactionalObject から継承され、ICF が異なるトランザクション方針を示している場合、警告が発行されます。TransactionalObject は、ほかの目的には使用されません。オブジェクトをトランザクションに関与させるために設定する必要があるトランザクション方針については、『BEA Tuxedo CORBA プログラミング・リファレンス』の「インプリメンテーション・コンフィギュレーション・ファイル (ICF)」を参照してください。

`CosTransactions` モジュールは、`TransactionalObject` インターフェイスを定義します (リスト 2-7 を参照)。このインターフェイスで定義されるメソッドはありません。単にマーカとして使用されます。

コード リスト 2-7 TransactionalObject インターフェイス

```
interface TransactionalObject {
};

// この情報は、「OMG Transaction Service Specification,
// Version 1.1」（2000 年 5 月 ）から。OMG の使用許可を得て
// 使用
```

TransactionFactory インターフェイス

TransactionFactory インターフェイスは、トランザクションの開始元がトランザクションを開始できるようにするためのものです。このインターフェイスは、最上位トランザクションの新しい表現を作成する、作成と再作成の 2 つのオペレーションを定義します。TransactionFactory は、ORB インターフェイスの resolve_initial_reference() オペレーションではなく、ライフ・サイクル・サービスの FactoryFinder インターフェイスを使用して検索します。

リスト 2-8 では、TransactionFactory インターフェイスを示します。

注記 TransactionFactory インターフェイスの Control recreate メソッドはサポートされていません。

コード リスト 2-8 TransactionFactory **インターフェイス**

```
interface TransactionFactory {
     Control create(in unsigned long time_out);
     Control recreate(in PropagationContext ctx);
};

// この情報は、「OMG Transaction Service Specification,
// Version 1.1」(2000 年 5 月 ) から。OMG の使用許可を得て
// 使用
```

表 2-4 では、TransactionFactory インターフェイスのメソッドを示します。

表 2-4 TransactionFactory インターフェイスのメソッド

メソッド	説明
create	新しい最上位トランザクションを作成し、`Control` オブジェクトを返します。`Control` オブジェクトは、新しいトランザクションのパーティシパントを管理するのに使用できます。トランザクション・サービスのインプリメンテーションは、`Control` オブジェクトの機能を制限し、他の実行環境に送信し、そこで使用できるようにすることができます。最低で、クライアント・アプリケーションで使用できます。 パラメータが 0 以外の値 `n` の場合、新しいトランザクションは、`n` 秒が経過するまでに完了しなかったときにロールバックの対象になります。パラメータが 0 の場合、アプリケーション指定のタイムアウトは設定されません。
recreate	サポートされていません。

その他の CORBA のオブジェクト・トランザクション・サービス・インターフェイス

その他の CORBA のオブジェクト・トランザクション・サービス・インターフェイスはサポートされていません。前述の `Current` インターフェイスは、Bootstrap オブジェクトから取得した場合にのみサポートされます。前述の `Control` インターフェイスは、Current オブジェクトの `get_control` メソッドおよび `suspend` メソッドで取得した場合にのみサポートされます。

CORBA トランザクション・サービス API の拡張

ここでは、前述の CORBA のオブジェクト・トランザクション・サービス API の特定の拡張について説明します。ここで説明する API を使用すると、アプリケーションは、Open Group リソース・マネージャをオープンまたはクローズできるようになります。

以下の API は、2 フェーズ・コミット・プロトコルを Open Group の XA インターフェイスで制御できるようにすることで、リソース・マネージャが簡単に分散トランザクションに関与できるようにします。

以下の定義とインターフェイスは、Tobj モジュールで定義されます。

Exception

以下の例外がサポートされています。

```
exception RMfailed {};
```

要求により、リソース・マネージャのオープンまたはクローズに失敗したことを通知するために例外が発生します。

TransactionCurrent インターフェイス

このインターフェイスは、CosTransactions モジュールの Current インターフェイスのすべてのメソッドをサポートしています。これについては、『BEA Tuxedo CORBA プログラミング・リファレンス』の「CORBA ブートストラップ処理のプログラミング・リファレンス」で説明しています。さらに、このインターフェイスは、リソース・マネージャをオープンまたはクローズする API をサポートします。

リスト 2-9 では、Tobj モジュールで定義されている TransactionCurrent インターフェイスを示します。

コード リスト 2-9 TransactionCurrent **インターフェイス**

```
Interface TransactionCurrent: CosTransactions::Current {
      void open_xa_rm()
             raises(RMfailed);
      void close_xa_rm()
             raises(Rmfailed);
}
```

表 2-5 では、リソース・マネージャ固有の API を説明しています。これらの API の詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』を参照してください。

表 2-5 Current **インターフェイスのリソース・マネージャ** API

メソッド	説明
`open_xa_rm`	このメソッドは、このプロセスがリンクされている Open Group のリソース・マネージャをオープンします。リソース・マネージャをオープンしているときに障害が発生すると、`RMfailed` 例外が発生します。 リモート・クライアントまたはネイティブ・クライアントでこのメソッドを呼び出そうとすると、標準システム例外 `NO_IMPLEMENT` が発生します。
`close_xa_rm`	このメソッドは、このプロセスがリンクされている Open Group のリソース・マネージャをクローズします。リソース・マネージャをクローズしているときに障害が発生すると、`RMfailed` 例外が発生します。関数呼び出し時のコンテキストが無効な場合 (呼び出し側がトランザクション・モードの場合など) は、標準システム例外 `BAD_INV_ORDER` が発生します。 リモート・クライアントまたはネイティブ・クライアントでこのメソッドを呼び出そうとすると、標準システム例外 `NO_IMPLEMENT` が発生します。

BEA Tuxedo CORBA アプリケーションのトランザクションの使用に関する注意事項

BEA Tuxedo CORBA クライアント / サーバ・アプリケーションにトランザクションを統合する場合は、以下のガイドライン考慮してください。

- 入れ子になったトランザクションは、BEA Tuxedo システムではサポートされていません。既存のトランザクションが既に活性化している場合に、新しいトランザクションを開始することはできません。既存のトランザクションを中断すれば、新しいトランザクションを開始できます。しかし、中断したトランザクションを後で再開できるのは、トランザクションを中断したオブジェクトだけです。

- トランザクションを終了できるエンティティは、トランザクションを開始したオブジェクトのみです。厳密には、このオブジェクトはクライアント / アプリケーション、TP フレームワーク、またはオブジェクト (サーバ・アプリケーションによって管理されているもの) のいずれかです。トランザクションの範囲内で呼び出されたオブジェクトは、トランザクションを中断および再開できます (また、トランザクションが中断されている間、オブジェクトは、ほかのトランザクションを開始および終了できます)。しかし、トランザクションを開始しない限り、オブジェクト内でトランザクションを終了できません。

- BEA Tuxedo は、並列トランザクションをサポートしていません。オブジェクトは、一度に 1 つのトランザクションにのみ関与できます。オブジェクトは、トランザクション全体の期間で、トランザクションに関与します。また、現在のトランザクションが完了した後でのみ別のトランザクションに関与できます。

- BEA Tuxedo は、現在トランザクションに関与するオブジェクトに対して、キューに要求を登録しません。トランザクションに関与しないクライアント・アプリケーションが、現在トランザクション内にあるオブジェクトに対してオペレーションを呼び出そうとすると、以下のエラー・メッセージが発行されます。

  Java

  ```
  org.omg.CORBA.OBJ_ADAPTER
  ```

  C++

  ```
  CORBA::OBJ_ADAPTER
  ```

  トランザクション内にあるクライアントが、現在別のトランザクション内にあるオブジェクトに対してオペレーションを呼び出そうとすると、以下のエラー・メッセージが発行されます。

  Java

  ```
  org.omg.CORBA.INVALID_TRANSACTION
  ```

  C++

  ```
  CORBA::INVALID_TRANSACTION
  ```

- トランザクション・バウンド・オブジェクトの場合、`Tobj_ServantBase::deactivate_object()` オペレーションですべての状態を処理することを検討してください。トランザクションの結果は、

deactivate_object() の呼び出し時に認識されるので、これによって、オブジェクトは状態を正しくかつ簡単に処理できるようになります。

- 複数のオペレーションを持っているものの、オブジェクトの永続的な状態に影響を与えるものはわずかしかないメソッド・バウンド・オブジェクトでは、以下を検討してください。
  - トランザクション方針として optional を割り当てる。
  - TransactionCurrent オブジェクトに対して呼び出しを実行することで、書き込みオペレーションをトランザクション内にスコープ指定する。

  トランザクション外でオブジェクトが呼び出された場合、データの読み取りに対するトランザクションのスコープにオーバーヘッドは生じません。この方法で、トランザクション内でオブジェクトが呼び出されるかどうかに関係なく、オブジェクトのすべての書き込みオペレーションは、トランザクションに関与する形で処理されます。

- トランザクションのロールバックは非同期的です。したがって、トランザクション・コンテキストがアクティブな場合でも、オブジェクトを呼び出すことは可能です。このようなオブジェクトを呼び出そうとすると、例外が発生します。

- トランザクション方針として always を割り当てたオブジェクトが、クライアント・アプリケーションではなく BEA Tuxedo システムで開始されたトランザクションに関与している場合は、以下に注意してください。
  - サーバ・アプリケーションがトランザクションを rollback only としてマークし、サーバが CORBA 例外をスローした場合、クライアント・アプリケーションは CORBA 例外を受け取ります。
  - サーバ・アプリケーションがトランザクションを rollback only としてマークし、サーバが CORBA 例外をスローしない場合、クライアント・アプリケーションは OBJ_ADAPTER 例外を受け取ります。この場合、BEA Tuxedo システムは、トランザクションをロールバックします。ただし、クライアント・アプリケーションは、トランザクションが BEA Tuxedo ドメインでスコープ指定されていないことをまったく認識していません。

- 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、サーバ・アプリケーションがトランザクションをロールバックとしてマークしている場合、以下のいずれかが行われます。

- サーバが CORBA 例外をスローした場合、クライアント・アプリケーションは、CORBA 例外を受け取ります。
- サーバが CORBA 例外をスローしなかった場合、クライアント・アプリケーションは、TRANSACTION_ROLLEDBACK 例外を受け取ります。

UserTransaction API

ここでは、以下の内容について説明します。

- UserTransaction メソッド
- UserTransaction メソッドがスローする例外

UserTransaction メソッド

表 2-6 では、UserTransaction オブジェクトのメソッドについて説明します。

表 2-6 UserTransaction オブジェクトのメソッド

メソッド名	説明
begin	現在のスレッドでトランザクションを開始します。
commit	現在のスレッドに関連付けられているトランザクションをコミットします。

表 2-6 UserTransaction オブジェクトのメソッド (続き)

メソッド名	説明
getStatus	トランザクション・ステータスを返します。現在のスレッドに関連付けられているトランザクションがない場合は、STATUS_NO_TRANSACTION を返します。 トランザクション・ステータスは、以下のいずれかの値です。 ■ STATUS_ACTIVE ■ STATUS_COMMITTED ■ STATUS_COMMITTING ■ STATUS_MARKED_ROLLBACK ■ STATUS_NO_TRANSACTION ■ STATUS_PREPARED ■ STATUS_PREPARING ■ STATUS_ROLLEDBACK ■ STATUS_ROLLING_BACK ■ STATUS_UNKNOWN
rollback	現在のスレッドに関連付けられているトランザクションをロールバックします。
setRollbackOnly	トランザクションの可能な結果のみがロールバックされるように、現在のスレッドに関連付けられているトランザクションをマークします。
setTransactionTimeout	begin メソッドで現在のスレッドが開始したトランザクションのタイムアウト値を指定します。アプリケーションが begin メソッドを呼び出していない場合、トランザクション・サービスは、トランザクション・タイムアウトのデフォルト値を使用します。

UserTransaction メソッドがスローする例外

表 2-7 では、UserTransaction オブジェクトのメソッドがスローする例外について説明します。

表 2-7 UserTransaction メソッドがスローする例外

例外	説明
`HeuristicMixedException`	ヒューリスティックな決定がなされたこと、関連する更新の一部がコミットされ、ほかはロールバックされたことを通知するためにスローされます。
`HeuristicRollbackException`	ヒューリスティックな決定がなされたこと、関連する更新の一部がロールバックされたことを通知するためにスローされます。
`NotSupportedException`	入れ子になったトランザクションなど、要求されたオペレーションがサポートされていないときにスローされます。
`RollbackException`	トランザクションが rollback only としてマークされているか、トランザクションがコミットではなくロールバックされたときにスローされます。
`IllegalStateException`	現在のスレッドがトランザクションに関連付けられていない場合にスローされます。
`SecurityException`	スレッドによるトランザクションのコミットが認められないことを通知するためにスローされます。
`SystemException`	トランザクション・マネージャで予期しないエラーが発生し、以降のトランザクション・サービスを続行できなくなったことを示すために、トランザクション・マネージャによって送出されます。

3 CORBA サーバ・アプリケーションのトランザクション

ここでは、以下の内容について説明します。

- BEA Tuxedo クライアントおよびサーバ・アプリケーションのトランザクションの統合
- トランザクション管理とオブジェクト状態管理
- ユーザ定義の例外

ここでは、トランザクションを BEA Tuxedo サーバ・アプリケーションに統合する方法について説明します。始める前に、第 1 章 「トランザクションについて」を読む必要があります。

BEA Tuxedo クライアントおよびサーバ・アプリケーションのトランザクションの統合

ここでは、以下の内容について説明します。

- CORBA アプリケーションのトランザクション・サポート
- オブジェクトを自動的にトランザクションに関与させる方法
- オブジェクトのトランザクションへの参加の有効化
- トランザクションのスコープ指定時のオブジェクト呼び出しの防止
- 実行中のトランザクションからのオブジェクトの除外
- 方針の割り当て
- XA リソース・マネージャの使用方法
- XA リソース・マネージャのオープン
- XA リソース・マネージャのクローズ

CORBA アプリケーションのトランザクション・サポート

BEA Tuxedo は、次の方法でトランザクションをサポートします。

- クライアント・アプリケーションまたはサーバ・アプリケーションは、TransactionCurrent オブジェクトに対して呼び出しを実行することで、トランザクションを明示的に開始および終了できます。TransactionCurrent オブジェクトの詳細については、第 4 章「CORBA クライアント・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。

- オブジェクトのインターフェイスにトランザクション方針を割り当てることができます。そうすることで、オブジェクトが呼び出されたときに、トランザクションが既に開始されていなければ、BEA Tuxedo システムがオブジェクトのために自動的にトランザクションを開始し、メソッド呼び出しが完了したときに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ができるようになります。すべてのトランザクションのコミット権限およびロールバック権限を XA リソース・マネージャに委譲する場合は、その XA リソース・マネージャおよびデータベースも考慮に入れて、オブジェクトのトランザクション方針を使用します。

- トランザクションに関与しているオブジェクトは、トランザクションを強制的にロールバックできます。つまり、トランザクションの範囲内でオブジェクトが呼び出されると、オブジェクトは、TransactionCurrent オブジェクトの `rollback_only()` を呼び出して、トランザクションを rollback only としてマークします。これによって、現在のトランザクションがコミットされるのを防ぐことができます。エンティティ (通常はデータベース) が破損データまたは不正確なデータで更新される危険がある場合、オブジェクトは、トランザクションを rollback としてマークしなければならない場合があります。

- トランザクションに関与するオブジェクトは、最初に呼び出されたときから、トランザクションをコミットまたはロールバックする準備ができたときまでメモリに保持できます。間もなくコミットされるトランザクションの場合、これらのオブジェクトは、リソース・マネージャがこのトランザクションのコミットを準備する直前に BEA Tuxedo システムによってポーリングされます。ここでは、ポーリングとはオブジェクトの `Tobj_ServantBase::deactive_object()` オペレーションを呼び出して reason 値を渡すことを意味します。

  オブジェクトは、ポーリング時に TransactionCurrent オブジェクトの `rollback_only()` を呼び出すと、現在のトランザクションを拒否できます。さらに、現在のトランザクションがロールバックされる場合、オブジェクトは、データベースへの書き込みをスキップできます。現在のトランザクションを拒否するオブジェクトがない場合、トランザクションはコミットされます。

以下の節では、オブジェクトの活性化方針およびトランザクション方針を使用し、オブジェクト内でどのように目的のトランザクションの振る舞いを決定するかを説明します。これらの方針は、インターフェイスに適用されます。したがって、そのインターフェイスをインプリメントしているすべてのオペレーションとオブジェクトに適用されます。

注記　サーバ・アプリケーションが、トランザクションに参加するオブジェクトを管理している場合、そのアプリケーションの Server オブジェクトは、`TP::open_xa_rm()` オペレーションおよび `TP::close_xa_rm()` オペレーションを呼び出す必要があります。データベース接続の詳細については、第 3 章の 10 ページ「XA リソース・マネージャのオープン」を参照してください。

オブジェクトを自動的にトランザクションに関与させる方法

BEA Tuxedo システムは、`always` トランザクション方針を提供します。これは、オブジェクトが呼び出されたときにトランザクションがまだスコープ指定されていない場合、BEA Tuxedo システムがトランザクションを自動的に開始するように、そのオブジェクトのインターフェイスを定義します。そのオブジェクトの呼び出しが完了すると、BEA Tuxedo システムは、自動的に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。サーバ・アプリケーションもオブジェクト・インプリメンテーションも、この状態で TransactionCurrent オブジェクトを呼び出す必要はありません。つまり、BEA Tuxedo システムは、サーバ・アプリケーションの代わりに自動的に TransactionCurrent オブジェクトを呼び出します。

`always` トランザクション方針をオブジェクトのインターフェイスに割り当てるのは、以下の場合です。

- オブジェクトがデータベースに書き込みをし、このオブジェクトが呼び出されたときに、すべてのデータベースのコミットまたはロールバックの権限を XA リソース・マネージャに委譲する必要がある場合。
- クライアント・アプリケーションが、複数のオブジェクトに対して呼び出しを実行する大規模なトランザクションでオブジェクトを含めること

ができるようにし、呼び出しがすべて成功するか、呼び出しが失敗した場合にロールバックする必要がある場合。

オブジェクトを自動的にトランザクションに関与させる必要がある場合、インプリメンテーション・コンフィギュレーション・ファイルで、そのオブジェクトのインターフェイスに以下の方針を割り当てます。

活性化方針	**トランザクション方針**
■ `process` ■ `method` ■ `transaction`	`always`

注記　データベース・カーソルは、複数のトランザクションにまたがることができません。ただし、C++ では、BEA Tuxedo University サンプル・アプリケーションの CourseSynopsisEnumerator オブジェクトは、データベース・カーソルを使用して、University データベースからコースの概要に一致するものを検索します。データベース・カーソルは、複数のトランザクションにまたがることはできないので、CourseSynopsisEnumerator オブジェクトの `activate_object()` オペレーションは、一致したすべてのコースの講義をメモリに読み込みます。カーソルは、イテレータ・クラスによって管理されるので、CourseSynopsisEnumerator オブジェクトでは認識できません。

オブジェクトのトランザクションへの参加の有効化

オブジェクトをトランザクションのスコープ内で呼び出すことができるようにする必要がある場合、`optional` トランザクション方針をそのオブジェクトのインターフェイスに割り当てることができます。`optional` トランザクション方針は、データベース書き込みオペレーションを実行しないものの、トランザクション時に呼び出すことができるようにする必要があるオブジェクトに適しています。

以下の方針を、そのオブジェクトのインターフェイスに対してインプリメンテーション・コンフィギュレーション・ファイルで指定し、オブジェクトを必要に応じてトランザクションに関与させることができます。

活性化方針	トランザクション方針
■ process ■ method ■ transaction	optional

トランザクション方針が optional のときに、アプリケーションの UBBCONFIG ファイルで AUTOTRAN パラメータが有効になっている場合、インプリメンテーションはトランザクションに関与します。トランザクションに関与するオブジェクトを含むサーバは、XA 準拠のリソース・マネージャに関連付けられているグループ内で設定する必要があります。

オブジェクトがデータベース書き込みオペレーションを実行しており、オブジェクトがトランザクションに関与できるようにする必要がある場合は、always トランザクション方針を割り当てる方が適切です。ただし、目的に応じて optional 方針を使用し、TransactionCurrent オブジェクトに対する呼び出しで書き込みオペレーションをカプセル化できます。つまり、オブジェクトがまだトランザクション内にスコープ指定されていない場合、データを書き込むオペレーション内で、トランザクションを開始およびコミットまたはロールバックするために TransactionCurrent オブジェクトを呼び出し、write 文の周囲にトランザクションをスコープします。これによって、データベース書き込みオペレーションがトランザクションに関与する形で処理されます。また、性能も効率的に発揮できるようになります。オブジェクトがトランザクションのスコープ内で呼び出されなかった場合、すべてのデータベース読み取りオペレーションは、トランザクションに関与しないため、より効率的になります。

注記　トランザクション方針を選択してオブジェクトに割り当てる場合、使用している XA リソース・マネージャの要件を把握します。たとえば、XA リソース・マネージャ (Oracle 7 トランザクション・マネージャ・サーバなど) では、トランザクションに参加するオブジェクトが、データベース書き込みオペレーションだけでなく、読み取りオペレーションもトランザクション内でスコープ指定する必要があります (ただし、自身のトランザクションをスコープ指定することはできます)。他のリソース・マネージャ (Oracle8i など) では、読み取りオペレーションおよび書き込みオペレーションのトランザクション・コンテキストを必要としません。アプリケーションが、トランザクション・コンテキストなしに書き込みオペレーションを実行しよう

とすると、Oracle8i は、アプリケーションがローカル・トランザクションを明示的にコミットする必要がある場合に、暗黙的にローカル・トランザクションを開始します。

トランザクションのスコープ指定時のオブジェクト呼び出しの防止

多くの場合、オブジェクトをトランザクションから除外することは危険です。このようなオブジェクトがトランザクション時に呼び出されると、オブジェクトは例外を返し、トランザクションがロールバックされることがあります。BEA Tuxedo CORBA は、`never` トランザクション方針を提供します。このトランザクション方針をオブジェクトのインターフェイスに割り当てると、現在のトランザクションが中断されている場合でも、そのオブジェクトがトランザクションの過程で呼び出されなくなります。

このトランザクション方針は、XA リソース・マネージャによって管理されないディスクにデータを書き込むオブジェクトなど、ロールバックできない永続的な状態をディスクに書き込むオブジェクトに適しています。クライアント・アプリケーションで、呼び出しの一部がトランザクションのスコープ指定を引き起こしているかどうかを認識できない場合、クライアント / サーバ・アプリケーションでこの機能を使用することは重要です。したがって、トランザクションがスコープ指定されている場合、この方針を持つオブジェクトが呼び出されると、トランザクションをロールバックできるようになります。

トランザクションがスコープ指定されているときにオブジェクトの呼び出しを防ぐには、インプリメンテーション・コンフィギュレーション・ファイルで、そのオブジェクトのインターフェイスに以下の方針を割り当てます。

活性化方針	トランザクション方針
■ `process` ■ `method`	`never`

実行中のトランザクションからのオブジェクトの除外

トランザクションの過程でオブジェクトの呼び出しを許可し、ただしそのオブジェクトをトランザクションの一部にはしないことがふさわしい場合もあります。このようなオブジェクトがトランザクションの最中に呼び出された場合、トランザクションは自動的に中断します。オブジェクトに対する呼び出しが完了すると、トランザクションは自動的に再開します。BEA Tuxedo CORBA は、この目的のために `ignore` トランザクション方針を提供します。

`ignore` トランザクション方針は、通常はデータをディスクに書き込まないファクトリなどのオブジェクトに適している場合があります。ファクトリをトランザクションから除外することで、そのファクトリは、トランザクションの最中でもほかのクライアントの呼び出しに使用できるようになります。さらに、この方針を使用すると、トランザクションに関与しているオブジェクトを呼び出す際のオーバーヘッドが軽減されるので、サーバ・アプリケーションの処理効率が向上します。

トランザクションがオブジェクトに伝達されないようにするには、インプリメンテーション・コンフィギュレーション・ファイルで、そのオブジェクトのインターフェイスに以下の方針を割り当てます。

活性化方針	トランザクション方針
■ `process` ■ `method`	`ignore`

方針の割り当て

インプリメンテーション・コンフィギュレーション・ファイルの作成方法とオブジェクトに対する方針の指定方法については、『BEA Tuxedo CORBA プログラミング・リファレンス』の「BEA Tuxedo CORBA サーバ・アプリケーションの作成手順」の「ステップ 4: オブジェクトのメモリ内での振る舞いの定義」を参照してください。

XA リソース・マネージャの使用方法

トランザクション・マネージャ・サーバ (TMS) は、オブジェクトの状態データを自動的に処理します。たとえば、`drive:\TUX8\samples\corba\university\transactions` ディレクトリの University サンプル C++ アプリケーションは、リレーショナル・データベース管理サービス (RDBMS) の例として Oracle TMS を使用します。

XA リソース・マネージャを使用すると、サーバ・アプリケーションで管理される別のオブジェクトが、データベースとの間のデータの読み書きをどのように実行するかについて、以下のような特定の要件が適用されます。

- Oracle7 などの一部の XA リソース・マネージャでは、すべてのデータベース・オペレーションをトランザクション内にスコープ指定する必要があります。つまり、DBaccess オブジェクトは、データベースから読み取りを実行するので、このオブジェクトに対するすべてのメソッド呼び出しをトランザクション内にスコープ指定する必要があります。トランザクションは、クライアントまたは BEA Tuxedo システムで開始できます。

  他のリソース・マネージャ (Oracle8i など) では、読み取りオペレーションおよび書き込みオペレーションのトランザクション・コンテキストを必要としません。アプリケーションが、トランザクション・コンテキストなしに書き込みオペレーションを実行しようとすると、Oracle8i は、アプリケーションがローカル・トランザクションを明示的にコミットする必要がある場合に、暗黙的にローカル・トランザクションを開始します。

- 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、XA リソース・マネージャは、そのコミットまたはロールバックが示す永続的な状態を自動的に処理します。つまり、トランザクションをコミットすると、XA リソース・マネージャは、すべてのデータベース更新を恒久的なものにします。同様に、トランザクションをロールバックすると、XA リソース・マネージャは、データベースをトランザクション開始前の状態に復元します。

  XA リソース・マネージャの特徴は、ロールバック時のオブジェクト状態データの処理に関する設計の問題をより簡単なものにすることです。トランザクションに関与するオブジェクトは、コミットおよびロール

バック権限を XA リソース・マネージャに委譲します。これによって、サーバ・アプリケーションのインプリメントは、大幅に単純化されます。

XA リソース・マネージャのオープン

オブジェクトのインターフェイスに always または optional トランザクション方針が適用されている場合、Server オブジェクトの Server::initialize() オペレーションの TP::open_xa_rm() オペレーションを呼び出す必要があります。リソース・マネージャは、UBBCONFIG ファイルの GROUPS セクションにある OPENINFO パラメータで提供された情報を基にオープンされます。デフォルト・バージョンの Server::initialize() オペレーションは、自動的にリソース・マネージャをオープンします。

データをディスクに書き込まず、トランザクションに参加しているオブジェクト (通常、トランザクション方針は optional) がある場合、TP::open_xa_rm() オペレーションへの呼び出しを含める必要があります。その呼び出しでは、ヌル・リソース・マネージャを指定します。

XA リソース・マネージャのクローズ

Server オブジェクトの Server::initialize() オペレーションが、XA リソース・マネージャをオープンする場合は、Server::release() オペレーションに以下の呼び出しを含めます。

```
TP::close_xa_rm();
```

トランザクション管理とオブジェクト状態管理

ここでは、以下の内容について説明します。

- XA リソース・マネージャへのオブジェクト状態管理の委譲
- トランザクションの作業が完了してから、データベースへの書き込みが始まるまでの待機

BEA Tuxedo CORBA クライアント / サーバ・アプリケーションでトランザクションが必要な場合、トランザクションをいくつかの方法でオブジェクト状態管理に統合できます。通常、BEA Tuxedo CORBA は、アプリケーションのロジック、またはオブジェクトが永続状態をディスクに書き込む方法を変更せずに、オペレーション呼び出しの間、自動的にトランザクションをスコープ指定できます。

XA リソース・マネージャへのオブジェクト状態管理の委譲

通常、Oracle などの XA リソース・マネージャを使用すると、ロールバック時のオブジェクト状態データの処理に関する設計の問題をより簡単にできます (Oracle リソース・マネージャは、BEA Tuxedo CORBA University サンプル C++ アプリケーションで使用されます)。トランザクションに関与するオブジェクトは、コミットおよびロールバック権限を XA リソース・マネージャに委譲します。これによって、サーバ・アプリケーションのインプリメントは、大幅に単純化されます。つまり、トランザクションに関与するプロセス・バウンドまたはメソッド・バウンド・オブジェクトは、トランザクション時にデータベースに書き込みを実行し、トランザクションのロールバック時にリソース・マネージャに従ってデータベースに書き込まれたデータをロールバックできます。

トランザクションの作業が完了してから、データベースへの書き込みが始まるまでの待機

`transaction` 活性化方針は、トランザクションの作業が完了するまで書き込みたくない、または書き込めないメモリ内の状態をディスクに保持するオブジェクトに適しています。`transaction` 活性化方針をオブジェクトに割り当てると、オブジェクトは、以下のようになります。

- トランザクションのスコープ内で最初に呼び出されたときに、メモリに書き込まれます。
- トランザクションがコミットされるかロールバックされるまで、メモリ内に残ります。

トランザクションの作業が完了したら、BEA Tuxedo CORBA は、DR_TRANS_COMMITTING または DR_TRANS_ABORTED のいずれかの可能性がある reason コードを渡す、各トランザクション・バウンド・オブジェクトの Tobj_ServantBase::deactivate_object() オペレーションを呼び出します。変数が DR_TRANS_COMMITTING の場合、オブジェクトは、データベース書き込みオペレーションを呼び出すことができます。変数が DR_TRANS_ABORTED の場合、オブジェクトは、データベース書き込みオペレーションをスキップします。

transaction 活性化方針の割り当てが必要な場合

transaction 活性化方針のオブジェクトへの割り当ては、以下のような場合に適しています。

- トランザクションが完了したときに、オブジェクトが永続状態をディスクに書き込めるようにする場合。

  ロールバックの対象となる可能性のあるデータベース書き込みオペレーションの数を減らすことができるので、これによって、パフォーマンスがより効率的になります。

- オブジェクトが、間もなくコミットされるトランザクションを拒否できるようにする場合。

  BEA Tuxedo CORBA が、reason コード DR_TRANS_COMMITTING を渡す場合、オブジェクトは必要に応じて TransactionCurrent オブジェクトの rollback_only() を呼び出すことができます。Tobj_ServantBase::deactivate_object() オペレーション内で rollback_only() を呼び出した場合、deactivate_object() は再び呼び出されません。

- オブジェクトがバッチ更新を実行できるようにする場合。

- 1 つのトランザクションに複数回関与する可能性が高いオブジェクトがあり、そのトランザクション時にオブジェクトを連続して活性化または非活性化する際のオーバーヘッドを避けるようにする場合。

transaction 活性化方針で使用するトランザクション方針

トランザクションがコミットされてからデータベースに書き込まれるまでオブジェクトが待機できるようにするには、インプリメンテーション・コンフィギュレーション • ファイルで、そのオブジェクトのインターフェイスに以下の方針を割り当てます。

活性化方針	トランザクション方針
`transaction`	`always` または `optional`

注記　トランザクション・バウンド・オブジェクトは、`Tobj_ServantBase::deactivate_object()` オペレーション内でトランザクションを開始したり、ほかのオブジェクトを呼び出したりすることはできません。`deactivate_object()` 内でトランザクション・バウンド・オブジェクトが唯一可能な呼び出しは、データベースへの書き込みオペレーションです。

また、トランザクションに関与するオブジェクトがある場合、そのオブジェクトを管理する Server オブジェクトは、データをディスクに書き込まない場合でも、XA リソース・マネージャをオープンまたはクローズするための呼び出しを含める必要があります (データをディスクに書き込まない、トランザクションに関与するオブジェクトがある場合、ヌル・リソース・マネージャを指定します)。XA リソース・マネージャのオープンおよびクローズの詳細については、第 3 章の 10 ページ「XA リソース・マネージャのオープン」と第 3 章の 10 ページ「XA リソース・マネージャのクローズ」を参照してください。

ユーザ定義の例外

ここでは、以下の内容について説明します。

- ユーザ定義の例外
- 例外の定義
- 例外のスロー

ユーザ定義の例外

BEA Tuxedo CORBA クライアント / サーバ・アプリケーションにユーザ定義の例外を含めるには、以下の手順を実行する必要があります。

1. OMG IDL ファイルで、例外を定義し、それを使用できるオペレーションを指定します。
2. インプリメンテーション・ファイルに例外をスローするコードを含めます。
3. クライアント・アプリケーションのソース・ファイルに、例外をキャッチして処理するコードを含めます。

たとえば、Transactions サンプル C++ アプリケーションは、ユーザ定義の例外 `TooManyCredits` のインスタンスを含んでいます。クライアント・アプリケーションが学生をコースに登録しようとしたときに、学生が登録可能なコースの最大数を超えている場合、サーバ・アプリケーションはこの例外を返します。クライアント・アプリケーションは、この例外を受け取ると、学生をコースに登録するトランザクションをロールバックします。ここでは、例として `TooManyCredits` 例外を使用し、BEA Tuxedo CORBA クライアント / サーバ・アプリケーションでユーザ定義の例外をどのように定義およびインプリメントできるかを説明します。

例外の定義

クライアント / サーバ・アプリケーションの OMG IDL ファイルでは、以下の作業を行います。

1. 例外の定義、および例外によって送信されるデータの定義。たとえば、TooManyCredits 例外は、学生が登録できる単位の最大数を表す short 型の整数値を渡すために定義します。したがって、TooManyCredits 例外の定義には、以下の OMG IDL 文が含まれます。

   ```
   exception TooManyCredits
   {
       unsigned short maximum_credits;
   };
   ```

2. 例外をスローするオペレーションの定義で、例外を含めます。次の例は、Registrar インターフェイスの register_for_courses() オペレーションに対する OMG IDL 文を示したものです。

   ```
   NotRegisteredList register_for_courses(
       in StudentId        student,
       in CourseNumberList courses
   ) raises (
       TooManyCredits
   );
   ```

例外のスロー

例外を使用するオペレーションのインプリメンテーションで、次の C++ 例のように例外をスローするコードを記述します。

```
if ( ... ) {
    UniversityZ::TooManyCredits e;
    e.maximum_credits = 18;
    throw e;
```

Transactions University サンプル・アプリケーションのしくみ

ここでは、以下の内容について説明します。

- Transactions University サンプル・アプリケーションについて
- Transactions University サンプル・アプリケーションで使用するトランザクション・モデル
- University サーバ・アプリケーションのオブジェクト状態に関する注意事項
- Transactions サンプル・アプリケーションのコンフィギュレーションの要件

Transactions University サンプル・アプリケーションについて

学生を登録するプロセスをインプリメントするために、Transactions サンプル・アプリケーションは、以下の作業を実行します。

- クライアント・アプリケーションは、Bootstrap オブジェクトから TransactionCurrent オブジェクトへのリファレンスを取得します。
- 学生が登録希望のコースのリストを提出したときに、クライアント・アプリケーションは以下を実行します。
  a. TransactionCurrent オブジェクトの `Current::begin()` オペレーションを呼び出してトランザクションを開始します。
  b. Register オブジェクトの `register_for_courses()` オペレーションを呼び出し、コースのリストを渡します。

- Register オブジェクトの `register_for_courses()` オペレーションは、リスト内の各コースに対して、ループ処理で以下の作業を繰り返し実行し、登録要求を処理します。

  a. その学生が既に登録している単位の数をチェックします。

  b. その学生が登録しているコースのリストにコースを追加します。

  Registrar オブジェクトは、トランザクションをコミットできないようにする可能性がある以下の問題をチェックします。

  - 学生がそのコースを既に登録しているかどうか。
  - リスト内にコースが存在しているかどうか。
  - 学生が履修できる単位の最大数を超えているかどうか。

- アプリケーションの OMG IDL での定義に従って、`register_for_courses()` オペレーションは、クライアント・アプリケーションにパラメータ `NotRegisteredList` を返します。このパラメータには、登録に失敗したコースのリストが含まれています。

  `NotRegisteredList` の値が空の場合、クライアント・アプリケーションはトランザクションをコミットします。

  `NotRegisteredList` 値にコースが含まれていない場合、クライアント・アプリケーションは、登録に成功したコースに対する登録プロセスを完了するかどうかを指定するよう学生に要求します。登録を完了するように選択した場合、クライアント・アプリケーションはトランザクションをコミットします。登録を取り消すように選択した場合、クライアント・アプリケーションはトランザクションをロールバックします。

- 学生が、履修できる単位の最大数を超えているためにコースの登録が失敗した場合、Register オブジェクトは `TooManyCredits` 例外をクライアント・アプリケーションに返し、クライアント・アプリケーションはトランザクション全体をロールバックします。

Transactions University サンプル・アプリケーションで使用するトランザクション・モデル

Transactions サンプル・アプリケーションの基本設計原理は、コース登録を一度に 1 つではなくグループ単位で処理することです。この設計原理によって、Register オブジェクトに対するリモート呼び出しの数を最小化できます。

この設計をインプリメントするにあたって、Transactions サンプル・アプリケーションでは、第 3 章の 2 ページ「BEA Tuxedo クライアントおよびサーバ・アプリケーションのトランザクションの統合」で説明したトランザクションの使用モデルを 1 つ示しています。このモデルの特徴は、次のとおりです。

- クライアントは、TransactionCurrent オブジェクトの `begin()` オペレーションを呼び出してから、Register オブジェクトの `register_for_courses()` オペレーションを呼び出すことでトランザクションを開始します。

  Register オブジェクトは、学生を可能なコースに登録し、登録プロセスが失敗したコースのリストを返します。クライアント・アプリケーションは、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選択できます。トランザクションは、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間の会話をカプセル化します。

- `register_for_courses()` オペレーションは、University データベースの複数チェックを実行します。いずれか 1 つのチェックが失敗した場合、トランザクションをロールバックできます。

University サーバ・アプリケーションのオブジェクト状態に関する注意事項

Transactions University サンプル・アプリケーションはトランザクションに関与するため、University サーバ・アプリケーションは通常、オブジェクト状態に関していくつかの点 (特にロールバック時の) を考慮する必要がありま

す。ロールバックが発生した場合、サーバ・アプリケーションは、関連するすべてのオブジェクトが、正しい状態に復元された永続状態を持つことを保証する必要があります。

Registrar オブジェクトは、データベース・トランザクションで使用されるため、このオブジェクトに対する最適な設計は、そのオブジェクトがトランザクションに関与するようにすること (`always` トランザクション方針をこのオブジェクトのインターフェイスに割り当てること) です。オブジェクト呼び出し時にトランザクションがまだスコープ指定されていない場合、BEA Tuxedo システムは、トランザクションを自動的に開始します。

Register オブジェクトが自動的にトランザクションに関与するようにすることで、このオブジェクトによって実行されるすべてのデータベース書き込みオペレーションは、クライアント・アプリケーションが開始したかどうかに関係なく、常にトランザクションのスコープ内で完了します。サーバ・アプリケーションは XA リソース・マネージャを使用し、またオブジェクトは、データベースに書き込みを実行するときにトランザクションにあることが保証されます。したがって、XA リソース・マネージャがオブジェクトの代わりにロールバックまたはコミットの権限を持つことになるので、オブジェクトにはロールバックまたはコミット権限がありません。

ただし、RegistrarFactory オブジェクトは、トランザクション時に使用するデータを管理しないので、トランザクションから除外できます。オブジェクトをトランザクションから除外することで、トランザクションに課せられる処理のオーバーヘッドを最小化します。

Registrar オブジェクトで定義されたオブジェクト方針

Registrar オブジェクトがトランザクションに関与するようにするために、ICF ファイルは、`Registrar` インターフェイスに対して `always` トランザクション方針を指定します。したがって、Transactions サンプル・アプリケーションでは、ICF ファイルで、`Registrar` インターフェイスに対して以下のオブジェクト方針を指定します。

活性化方針	トランザクション方針
`process`	`always`

RegistrarFactory オブジェクトで定義されたオブジェクト方針

RegistrarFactory オブジェクトをトランザクションから除外するために、ICF ファイルでは、`Registrar` インターフェイスに対して `ignore` トランザクション方針を指定します。したがって、Transactions サンプル・アプリケーションでは、ICF ファイルで、`RegistrarFactory` インターフェイスに対して以下のオブジェクト方針を指定します。

活性化方針	トランザクション方針
`process`	`ignore`

Transactions サンプル・アプリケーションでの XA リソース・マネージャの使用

Transactions サンプル・アプリケーションは、オブジェクト状態データを自動的に処理する Oracle トランザクション・マネージャ・サーバ (TMS) を使用します。XA リソース・マネージャを使用すると、サーバ・アプリケーションで管理される別のオブジェクトが、データベースとの間のデータの読み書きをどのように実行するかについて、以下のような特定の要件が適用されます。

- Oracle7 などの一部の XA リソース・マネージャでは、すべてのデータベース・オペレーションをトランザクション内にスコープ指定する必要があります。つまり、CourseSynopsisEnumerator オブジェクトは、データベースから読み取りを実行するので、このオブジェクトをトランザクション内にスコープ指定する必要があります。

- 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、XA リソース・マネージャは、そのコミットまたはロールバックが示す永続的な状態を自動的に処理します。つまり、トランザクションをコミットすると、XA リソース・マネージャは、すべてのデータベース更新を恒久的なものにします。同様に、トランザクションをロールバックすると、XA リソース・マネージャは、データベースをトランザクション開始前の状態に復元します。

  XA リソース・マネージャの特徴は、ロールバック時のオブジェクト状態データの処理に関する設計の問題をより簡単なものにすることです。

トランザクションに関与するオブジェクトは、コミットおよびロールバック権限を XA リソース・マネージャに委譲します。これによって、サーバ・アプリケーションのインプリメントは、大幅に単純化されます。

Transactions サンプル・アプリケーションのコンフィギュレーションの要件

University サンプル・アプリケーションでは Oracle トランザクション・マネージャ・サーバ (TMS) を使用します。Oracle データベースを使用するには、Oracle 提供の特定のファイルをサーバ・アプリケーションのビルド・プロセスに含める必要があります。Transactions サンプル・アプリケーションのビルド、コンフィギュレーション、および実行の詳細については、BEA Tuxedo オンライン・マニュアルの「Transactions サンプル・アプリケーション」を参照してください。UBBCONFIG ファイルのコンフィギュレーション可能な設定の詳細については、第 5 章の 2 ページ「UBBCONFIG ファイルをトランザクションに対応させて変更する」を参照してください。

4 CORBA クライアント・アプリケーションのトランザクション

ここでは、以下の内容について説明します。

- BEA Tuxedo CORBA トランザクションの概要
- トランザクションの開発プロセスの概要
- ステップ 1: Bootstrap オブジェクトを使用して TransactionCurrent オブジェクトを取得する
- ステップ 2: TransactionCurrent メソッドを使用する

ここでは、BEA Tuxedo CORBA ソフトウェアの CORBA C++、Java、および ActiveX クライアント・アプリケーションでトランザクションを使用する方法を説明します。始める前に、第 1 章 「トランザクションについて」を読む必要があります。

注記　BEA Tuxedo では、CORBA インターオペラブル・ネーミング・サービス (INS) ブートストラップ処理メカニズムも使用できます。INS の詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』の「CORBA ブートストラップ処理のプログラミング・リファ

レンス」を参照してください。INS を使用する際の制限については、第 2 章の 3 ページ「INS を使用するサード・パーティ・クライアントのサポート」を参照してください。

作業クライアント・アプリケーションにトランザクションがインプリメントされるしくみの例については、BEA Tuxedo オンライン・マニュアルの「Transactions サンプル・アプリケーション」を参照してください。TransactionCurrent オブジェクトの概要については、『BEA Tuxedo CORBA クライアント・アプリケーションの開発方法』の「クライアント・アプリケーションの開発概念」を参照してください。

BEA Tuxedo CORBA トランザクションの概要

クライアント・アプリケーションは、トランザクション処理を使用してデータの有効性、一貫性、永続性を保証します。BEA Tuxedo ソフトウェアのトランザクションを使用すると、クライアント・アプリケーションはトランザクションを開始および終了し、トランザクションのステータスを取得できます。BEA Tuxedo ソフトウェアでは、CORBA のオブジェクト・トランザクション・サービスで定義されたトランザクションを使いやすいように拡張して使用します。

トランザクションは、インターフェイスで定義されます。アプリケーション設計者は、BEA Tuxedo クライアント / サーバ・アプリケーション内のどのインターフェイスでトランザクションを処理するかを指定します。インプリメンテーション・コンフィギュレーション・ファイル (ICF) で、各インターフェイスのトランザクション方針がサーバ・アプリケーション用に定義されます。一般に、使用可能なインターフェイスの ICF ファイルは、アプリケーション設計者からクライアント・プログラマに提供されます。

トランザクションの開発プロセスの概要

トランザクションをクライアント・アプリケーションに追加するには、以下の手順を実行します。

- ステップ 1: Bootstrap オブジェクトを使用して TransactionCurrent オブジェクトを取得する
- ステップ 2: TransactionCurrent メソッドを使用する

以下の節では、Transactions University サンプル・アプリケーションにあるクライアント・アプリケーションの一部を使用して、この手順を説明します。Transactions University サンプル・アプリケーションの詳細については、BEA Tuxedo オンライン・マニュアルの「Transactions サンプル・アプリケーション」を参照してください。

Transactions University サンプル・アプリケーションは、BEA Tuxedo ソフトウェア・キットの次のディレクトリにあります。

- Microsoft Windows システムの場合
  `drive:\tuxdir\samples\corba\university\transactions`
- UNIX システムの場合
  `drive:/tuxdir/samples/corba/university/transactions`

ステップ 1: Bootstrap オブジェクトを使用して TransactionCurrent オブジェクトを取得する

BEA Tuxedo CORBA クライアント・ソフトウェアを使用している場合、Bootstrap オブジェクトを使用し、指定した BEA Tuxedo ドメインで TransactionCurrent オブジェクトへのオブジェクト・リファレンスを取得します。TransactionCurrent オブジェクトの詳細については、『BEA Tuxedo CORBA クライアント・アプリケーションの開発方法』の「CORBA クライアント・アプリケーションの開発概念」を参照してください。

注記　サード・パーティ・クライアント ORB を使用している場合、CORBA インターオペラブル・ネーミング・サービス (INS) の `CORBA::ORB::resolve_initial_references` オペレーションを使用して、BEA Tuxedo ドメインの FactoryFinder オブジェクトへのリファレンスを取得します。INS を使用してトランザクション・クライアントの初期オブジェクト・リファレンスを取得する方法の詳細につい

ては、『BEA Tuxedo CORBA プログラミング・リファレンス』の「CORBA ブートストラップ処理のプログラミング・リファレンス」を参照してください。

次の C++、Java、Visual Basic の例は、Bootstrap オブジェクトを使用して TransactionCurrent オブジェクトを返す方法を示します。

C++ の例

```
CORBA::Object_var var_transaction_current_oref =
     Bootstrap.resolve_initial_references(“TransactionCurrent”);
CosTransactions::Current_var transaction_current_oref=
     CosTransactions::Current::_narrow(
              var_transaction_current_oref.in());
```

Java の例

```
org.omg.CORBA.Object transCurObj =
     gBootstrapObjRef.resolve_initial_references(
                         “TransactionCurrent”);
org.omg.CosTransactions.Current gTransCur=
     org.omg.CosTransactions.CurrentHelper.narrow(transCurObj);
```

Visual Basic の例

```
Set objTransactionCurrent =
             objBootstrap.CreateObject(“Tobj.TransactionCurrent”)
```

ステップ 2: TransactionCurrent メソッドを使用する

TransactionCurrent オブジェクトには、クライアント・アプリケーションでトランザクションを管理できるようにするメソッドがあります。これらのメソッドを使用すると、トランザクションを開始および終了して、現在のトランザクションに関する情報を取得できます。

注記　または、CORBA Java クライアントは、UserTransaction オブジェクトを代わりに使用することもできます。

表 4-1 では、TransactionCurrent オブジェクトのメソッドについて説明します。

表 4-1 TransactionCurrent オブジェクトのメソッド

メソッド	説明
`begin`	新しいトランザクションを作成します。以降のオペレーションは、このトランザクションのスコープ内で発生します。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、デフォルトのトランザクション・タイムアウトは 300 秒です。`set_timeout` メソッドで、このデフォルト値を変更できます。
`commit`	トランザクションを正常に終了します。このクライアント・アプリケーションですべてのオペレーションが正常に終了したことを示します。
`rollback`	トランザクションを強制的にロールバックします。
`rollback_only`	可能なアクションのみがロールバックされるようにトランザクションをマークします。通常、このメソッドは、サーバ・アプリケーションでのみ使用されます。

表 4-1 TransactionCurrent **オブジェクトのメソッド** (**続き**)

メソッド	説明
`suspend`	現在のトランザクションの参加を一時停止します。このメソッドは、トランザクションを示すオブジェクトを返し、クライアント・アプリケーションが後でトランザクションを再開できるようにします。
`resume`	指定したトランザクションの参加を再開します。
`get_status`	クライアント・アプリケーションでトランザクションのステータスを返します。
`get_transaction_name`	トランザクションを示す出力可能な文字列を返します。
`set_timeout`	トランザクションに関連付けられたタイムアウトを修正します。デフォルトのトランザクション・タイムアウト値は 300 秒です。`begin` メソッドで明示的に開始するのではなく、トランザクションが自動的に開始した場合、タイムアウト値は、`UBBCONFIG` ファイルの `TRANTIME` パラメータで指定した値です。`TRANTIME` パラメータの設定方法については、第 5 章 「トランザクションの管理」を参照してください。
`get_control`	トランザクションを表すコントロール・オブジェクトを返します。

基本的なトランザクションは、以下のように動作します。

1. クライアント・アプリケーションは、`Tobj::TransactionCurrent::begin` メソッドを使用してトランザクションを開始します。このメソッドは値を返しません。

2. CORBA インターフェイスのオペレーションは、トランザクションのスコープ内で実行されます。これらのオペレーションの一部を呼び出して例外が発生した場合 (明示的に、または通信の失敗の結果として)、その例外がキャッチされ、トランザクションはロールバックされます。

3. Tobj::TransactionCurrent::commit メソッドを使用すると、現在のトランザクションがコミットされます。このメソッドは、トランザクションを終了して、オペレーションの処理を開始します。トランザクションは、トランザクションのすべてのパーティシパントがコミットに同意した場合にのみコミットされます。

   トランザクションとクライアント・アプリケーションの関連付けは、アプリケーションが Tobj::TransactionCurrent::commit メソッドまたは Tobj::TransactionCurrent::rollback メソッドを呼び出したときに解除されます。次の C++、Java、Visual Basic 例は、クラスに登録している生徒のオペレーションをカプセル化するためにトランザクションを使用する場合を示しています。

C++ の例

```
// トランザクションを開始
transaction_current_oref->begin();
try {
//Perform the operation inside the transaction
     pointer_Registar_ref->register_for_courses(student_id, course_number_list);
     ...
     // オペレーションがエラーなしで実行された場合は、トランザクションをコミット
     CORBA::Boolean report_heuristics = CORBA_TRUE;
     transaction_current_ref->commit(report_heuristics);
}
catch (CORBA::Exception &) {
      // オペレーションで実行エラーが発生した場合は、トランザクションをロールバック
      // 次に元の例外を再びスローする
      // ロールバックが失敗した場合、例外を無視し、
      // 元の例外を再びスローする
try {
     transaction_current_ref->rollback();
}
catch (CORBA::Exception &) {
            TP::userlog("rollback failed");

throw;
}
```

Java の例

```
try{
   gTransCur.begin();
// トランザクション内でオペレーションを実行
   not_registered =
      gRegistrarObjRef.register_for_courses(student_id,selected_course_numbers);


   if (not_registered != null)

     // オペレーションがエラーなしで実行された場合は、トランザクションをコミット
     boolean report_heuristics = true;
     gTransCur.commit(report_heuristics);

   } else gTransCur.rollback();


} catch(org.omg.CosTransactions.NoTransaction nte) {
    System.err.println("NoTransaction: " + nte);
    System.exit(1);
} catch(org.omg.CosTransactions.SubtransactionsUnavailable e) {
    System.err.println("Subtransactions Unavailable: " + e);
    System.exit(1);
} catch(org.omg.CosTransactions.HeuristicHazard e) {
    System.err.println("HeuristicHazard: " + e);
    System.exit(1);
} catch(org.omg.CosTransactions.HeuristicMixed e) {
    System.err.println("HeuristicMixed: " + e);
    System.exit(1);
}
```

Visual Basic の例

```
' トランザクションを開始
'
objTransactionCurrent.begin
'
' コースの登録を試行
'
NotRegisteredList = objRegistrar.register_for_courses(mStudentID,
     CourseList, exception)
'
```

```
If exception.EX_majorCode = NO_EXCEPTION then
     ' 要求が成功したら、トランザクションをコミット
     '
     Dim report_heuristics As Boolean
     report_heuristics = True
     objTransactionCurrent.commit report_heuristics
Else
     ' 要求が失敗したら、トランザクションをロールバック
     '
     objTransactionCurrent.rollback
          MsgBox "Transaction Rolled Back"
End If
```

5 トランザクションの管理

ここでは、以下の内容について説明します。

- UBBCONFIG ファイルをトランザクションに対応させて変更する
- トランザクションをサポートするように Domains コンフィギュレーション・ファイルを変更する (BEA Tuxedo CORBA サーバ)
- トランザクションを使用する分散アプリケーションの例

始める前に、第 1 章 「トランザクションについて」を読む必要があります。

注記　管理情報は、BEA Tuxedo ORB に対する初期オブジェクト・リファレンスを取得するために、Bootstrap オブジェクトまたは CORBA インターオペラブル・ネーミング・サービス (INS) のいずれを使用している場合でも適用されます。

UBBCONFIG ファイルをトランザクションに対応させて変更する

ここでは、以下の内容について説明します。

- 手順の要約
- ステップ 1: RESOURCES セクションでアプリケーション全体のトランザクションを指定する
- ステップ 2: トランザクション・ログ (TLOG) を作成する
- ステップ 3: GROUPS セクションでリソース・マネージャ (RM) とトランザクション・マネージャ・サーバを定義する
- ステップ 4: トランザクションを開始するためのインターフェイスを有効にする

手順の要約

アプリケーションの UBBCONFIG ファイルをトランザクションに対応させるには、RESOURCES、MACHINES、GROUPS、および INTERFACES または SERVICES セクションを変更する必要があります。

- RESOURCES セクションでは、アプリケーション全体で設定できるトランザクションの数、およびコミット制御フラグの値を指定します。
- MACHINES セクションでは、各マシンの TLOG 情報を作成します。
- GROUPS セクションでは、各リソース・マネージャおよびトランザクション・マネージャ・サーバに関する情報を指定します。
- INTERFACES セクション (BEA Tuxedo CORBA アプリケーションのみ) または SERVICES セクション (BEA Tuxedo ATMI アプリケーションのみ) では、自動トランザクション・オプションを有効にします。

UBBCONFIG ファイルのこれらのセクションの変更については、『BEA Tuxedo アプリケーションの設定』の「コンフィギュレーション・ファイルの作成」を参照してください。

ステップ 1: RESOURCES セクションでアプリケーション全体のトランザクションを指定する

表 5-1 では、コンフィギュレーション・ファイルの RESOURCES セクションで指定可能なトランザクション関連のパラメータを説明しています。

表 5-1 RESOURCES セクションのトランザクション関連のパラメータ

パラメータ	説明
MAXGTT	一度に 1 台のマシン上で設定できるグローバル・トランザクション識別子 (GTRID) の数を制限します。指定可能な最大値は 2048、最小値は 0、デフォルトは 100 です。MACHINES セクションで、マシンごとにこの値をオーバーライドできます。 グローバル・トランザクションがアクティブな間だけ、テーブル内にそのエントリがあるため、このパラメータには同時トランザクションの数に制限を設ける効果があります。

表 5-1 RESOURCES **セクションのトランザクション関連のパラメータ** (**続**

パラメータ	**説明**
CMTRET	TP_COMMIT_CONTROL 特性の初期設定を指定します。デフォルト値は COMPLETE です。以下のいずれかの設定が可能です。 ■ LOGGED — TP_COMMIT_CONTROL 特性は TP_CMT_LOGGED に設定されます。すべてのパーティシパントが正常にプリコミットを行った場合に tpcommit() が返されることを示します。 ■ COMPLETE — TP_COMMIT_CONTROL 特性は TP_CMT_COMPLETE に設定されます。すべてのパーティシパントが正常にコミットするまで tpcommit() が返されないことを示します。 注記　RM ベンダに問い合わせて、適切な設定を決める必要があります。XA 標準の レイト・コミット・インプリメンテーションを使用するリソース・マネージャがある場合、COMPLETE に設定します。すべてのリソース・マネージャが アーリー・コミット・インプリメンテーションを使用する場合、性能上の理由により LOGGED に設定します。この設定は tpscmt() を使用して上書きできます。

ステップ 2: トランザクション・ログ (TLOG) を作成する

ここでは、トランザクションが終了するまで、トランザクションに関する情報が保持されているトランザクション・ログ (TLOG) の作成について説明します。

UDL を作成する

汎用デバイス・リスト (UDL) は、BEA Tuxedo ファイル・システムの地図のようなものです。UDL は、アプリケーションの起動後、共用メモリにロードされます。UDL 内に TLOG デバイス用のエントリを作成するには、グローバル・トランザクションを使用して各マシンに UDL を作成します。

TLOGDEVICE が 2 台のマシン間でミラーリングされる場合は、一方のマシンに UDL を作成するだけで十分です。BBL は、起動時に TLOG を初期化してオープンします。

UDL を作成するには、アプリケーションを起動する前に次の形式のコマンドを実行します。

```
tmadmin -c crdl -z config -b blocks
```

この例の説明は次のとおりです。:

-z *config*	UDL の作成場所となるデバイスの絶対パス名を指定します。
-b *blocks*	デバイスに割り当てられるブロック数を指定します。
config	これは、UBBCONFIG ファイルの MACHINES セクションにある TLOGDEVICE パラメータの値と一致している必要があります。

注記　通常、blocks で指定する値は、TLOGSIZE より小さい値にはしません。たとえば、TLOGSIZE に 200 ブロックが指定されている場合、-b 500 を指定すると、性能は低下しません。

TLOG の保存方法については、『BEA Tuxedo システムのインストール』を参照してください。

MACHINES セクションでトランザクション関連のパラメータを定義する

UBBCONFIG ファイルの MACHINES セクションでパラメータをいくつか設定して、グローバル・トランザクション・ログ (TLOG) を定義できます。TLOGDEVICE のデバイス・リストのエントリは、TLOG が必要な各マシンで手動で作成する必要があります。これは、TUXCONFIG がロードされる前でも後でも作成できますが、必ずシステムを起動する前に作成してください。

注記　トランザクションを使用しない場合、TLOG パラメータは不要です。

表 5-2 では、コンフィギュレーション・ファイルの MACHINES セクションで指定可能なトランザクション関連のパラメータを説明しています。

表 5-2 MACHINES セクションのトランザクション関連のパラメータ

パラメータ	説明
TLOGNAME	このマシンの DTP トランザクション・ログの名前を指定します。
TLOGDEVICE	このマシンの DTP トランザクション・ログ (TLOG) を格納する BEA Tuxedo または BEA Tuxedo のファイル・システムを指定します。このパラメータを指定しない場合、そのマシンには TLOG がないものと見なされます。文字列の最大値は 64 文字です。
TLOGSIZE	TLOG ファイルのサイズを物理ページ単位で指定します。この値には、1 ～ 2048 の範囲の値を設定します。デフォルト値は 100 です。この値を指定した時点で、未処理トランザクションをマシン上で十分保持できるサイズにします。1 ページにつき 1 つのトランザクションがログに記録されます。ほとんどのアプリケーションでは、デフォルト値で十分です。
TLOGOFFSET	TLOGDEVICE の始めから、このマシンのトランザクション・ログを含む VTOC の開始までのページのオフセット (ページ数) を指定します。この値は 0 以上で、デバイス上のページ数より小さい値でなければなりません。デフォルト値は 0 です。 TLOGOFFSET が必要になることはほとんどありません。ただし、2 つの VTOC が同じデバイスを共有したり、VTOC が別のアプリケーションに共有されるデバイス (ファイル・システムなど) 上に格納される場合は、TLOGOFFSET を使用してデバイスのアドレスに関連する開始アドレスを指定できます。

Domains トランザクション・ログを作成する (BEA Tuxedo ATMI サーバのみ)

ここで説明することは、ATMI サーバにのみ適用されます。

Domains ゲートウェイ・グループを開始する前に、次のコマンドで Domains トランザクション・ログを作成できます。

```
dmadmin(1) crdmlog (crdlog) -d local_domain_name
```

Domains トランザクション・ログは、現在のマシン (dmadmin を実行中のマシン) の指定されたローカル・ドメインに作成してください。このコマンドでは、DMCONFIG ファイルで指定したパラメータが使用されます。指定のローカル・ドメインが現在のマシン上でアクティブであるか、ログがすでに存在する場合は、このコマンドは失敗します。トランザクション・ログがない場合は、Domains ゲートウェイ・グループの起動時に作成されます。

ステップ 3: GROUPS セクションでリソース・マネージャ (RM) とトランザクション・マネージャ・サーバを定義する

GROUPS セクションへの追加は、以下の 2 つのカテゴリに分類されます。

- グローバル・トランザクションを制御するほとんどの作業を実行するトランザクション・マネージャ・サーバの定義。
  - TMSNAME パラメータは、実行可能なサーバ名を指定します。
  - TMSCOUNT パラメータは、起動するトランザクション・マネージャ・サーバの数 (最小 2、最大 10、デフォルト 3) を指定します。

  ヌル・トランザクション・マネージャ・サーバは、リソース・マネージャと通信しません。このサーバは、回復可能な実際の環境でアプリケーションをテストする前に、トランザクションに関与するプリミティブをアプリケーションで実際に使用するために使用します。TMS と呼ばれるこのサーバは、リソース・マネージャと会話せずに単純にトランザクションを開始、コミット、および終了します。

- 各リソース・マネージャの情報のオープンおよびクローズを定義するには、以下のパラメータを使用します。
  - OPENINFO は、リソース・マネージャのオープンに使用する情報の文字列です。
  - CLOSEINFO は、リソース・マネージャのクローズに使用します。

GROUPS セクションの例

次の GROUPS セクションの例は、bankapp 銀行取引アプリケーションから抜粋したものです。

```
BANKB1 GRPNO=1 TMSNAME=TMS_SQL TMSCOUNT=2
OPENINFO="TUXEDO/SQL:<APPDIR>/bankdl1:bankdb:readwrite"
```

表 5-3 では、この GROUPS セクションの例で指定したトランザクション値について説明します。

表 5-3 サンプル UBBCONFIG ファイルの GROUPS セクションのトランザクション値

トランザクション値	説明
BANKB1 GRPNO=1 TMSNAME=TMS_SQL\ TMSCOUNT=2	トランザクション・マネージャ・サーバの名前 (TMS_SQL) と、グループ BANKB1 で起動するサーバの数 (2) を指定します。
TUXEDO/SQL	リソース・マネージャの公開名です。
<APPDIR>/bankdl1	デバイス名を指定します。
bankdb	データベース名です。
readwrite	アクセス・モードです。

TMSNAME、TMSCOUNT、OPENINFO、および CLOSEINFO パラメータの特性

表 5-4 では、TMSNAME、TMSCOUNT、OPENINFO、および CLOSEINFO パラメータの特性を説明します。

表 5-4 TMSNAME、TMSCOUNT、OPENINFO、**および** CLOSEINFO **パラメータの特性**

パラメータ	説明
TMSNAME	実行可能なトランザクション・マネージャ・サーバの名前です。 トランザクションのコンフィギュレーションでは必須のパラメータです。 TMS は、ヌル・トランザクション・マネージャ・サーバです。
TMSCOUNT	トランザクション・マネージャ・サーバの数。2 ～ 10 の範囲の値を設定します。 デフォルト値は 3 です。
OPENINFO CLOSEINFO	リソース・マネージャをオープンまたはクローズするための情報を表します。 内容は、リソース・マネージャによって異なります。 リソース・マネージャの名前で始まります。 この値を省略すると、リソース・マネージャがオープンするための情報を必要としないことを示します。

ステップ 4: トランザクションを開始するためのインターフェイスを有効にする

トランザクションを開始するためのインターフェイスを有効にするには、UBBCONFIG ファイル内のセクションを変更します。変更対象のセクションは、BEA Tuxedo CORBA サーバをコンフィギュレーションするか、BEA Tuxedo ATMI サーバをコンフィギュレーションするかによって異なります。

- INTERFACES セクションの変更 (BEA Tuxedo CORBA サーバ)
- SERVICES セクションの変更 (BEA Tuxedo ATMI サーバ)

INTERFACES セクションの変更 (BEA Tuxedo CORBA サーバ)

UBBCONFIG ファイルの INTERFACES セクションは、BEA Tuxedo CORBA インターフェイスをサポートします。

- オペレーション呼び出しを受信したときにトランザクションを自動的に開始する場合、各 CORBA インターフェイスに対して、AUTOTRAN を Y に設定します。AUTOTRAN=Y は、インターフェイスが既にトランザクション・モードにある場合は無効です。デフォルト値は N です。AUTOTRAN に値を指定した場合の効果は、インプリメンテーション・コンフィギュレーション・ファイル (ICF) でインターフェイスの開発者が指定したトランザクション方針によって異なります。このトランザクション方針は実行時に、関連する T_IFQUEUE MIB オブジェクトのトランザクション方針の属性になります。この値がアプリケーションの動作に影響するのは、開発者が optional トランザクション方針を指定した場合だけです。

  注記　この機能を正しく動作させるには、設計者と管理者の間の共同作業が不可欠です。インターフェイスの ICF で開発者が定義したトランザクション方針を知らずに、管理者がこの値を Y に設定すると、パラメータを実際に実行した場合の影響が分からなくなる可能性があります。

- AUTOTRAN が Y に設定されている場合、TRANTIME パラメータを設定する必要があります。これは、作成するトランザクションに対するタイムアウト値 (秒) を指定します。0 ～ 2,147,483,647 (2^{31} - 1、つまり約 70 年) の値を指定してください。0 は、トランザクションにタイムアウトが設定されていないことを示します。デフォルト値は 30 秒です。

表 5-5 では、AUTOTRAN、TRANTIME、および FACTORYROUTING パラメータの特性を説明します。

表 5-5 AUTOTRAN、TRANTIME、および FACTORYROUTING パラメータの特性

パラメータ	説明
AUTOTRAN	■ インターフェイスをトランザクションのイニシエータにします。 ■ この機能を正しく動作させるには、システム設計者とシステム管理者の間の共同作業が不可欠です。ICF で開発者が定義したトランザクション方針を知らずに、管理者がこの値を Y に設定すると、パラメータを実際に実行した場合の影響が分からなくなる可能性があります。 ■ この値がアプリケーションの動作に影響するのは、開発者が optional トランザクション方針を指定した場合だけです。 ■ トランザクションがすでに存在している場合、新しいトランザクションは開始しません。 ■ デフォルト値は N です。
TRANTIME	■ AUTOTRAN トランザクションのタイムアウトを表します。 ■ 有効な値は 0 ～ 2^{31} - 1 の範囲です。 ■ 0 はタイムアウト値がないことを示します。 ■ デフォルト値は 30 秒です。
FACTORYROUTING	■ この CORBA インターフェイスのファクトリ・ベース・ルーティングに使用するルーティング条件の名前を指定します。 ■ ファクトリ・ベース・ルーティングを要求しているインターフェイスに対して FACTORYROUTING パラメータを指定する必要があります。

SERVICES セクションの変更 (BEA Tuxedo ATMI サーバ)

以下の 3 つのパラメータは、SERVICES セクションに指定するトランザクション関連のパラメータです。

- クライアントではなくサービスでトランザクションを開始する場合、AUTOTRAN フラグを Y に設定する必要があります。この設定は、サービスが大規模なトランザクションの一部でない場合や、トランザクションの決定に関わるクライアント側の負担を軽減したい場合に役立ちます。既存のトランザクションがある場合にサービスが呼び出されると、この呼び出しは既存のトランザクションの一部になります。デフォルト値は N です。

  注記　一般的に、サービスは大規模なトランザクションに参加する可能性があるため、トランザクションのイニシエータとしてはクライアントが最も適しています。

- AUTOTRAN が Y に設定されている場合、TRANTIME パラメータを設定する必要があります。これは、作成するトランザクションに対するタイムアウト値 (秒) です。0 ～ 2,147,483,647 (2^{31} - 1、つまり約 70 年) の値を指定してください。0 は、トランザクションにタイムアウトが設定されていないことを示します。デフォルト値は 30 秒です。

- データ依存型ルーティングを要求するトランザクションには、ROUTING パラメータを指定する必要があります。

表 5-6 では、AUTOTRAN、TRANTIME、および ROUTING パラメータの特性を説明します。

表 5-6 AUTOTRAN、TRANTIME、および ROUTING パラメータの特性

パラメータ	説明
AUTOTRAN	トランザクションのイニシエータとしてサービスを指定します。 トランザクションに関するクライアントの負荷を軽減します。 トランザクションがすでに存在している場合、新しいトランザクションは開始しません。 デフォルト値は N です。
TRANTIME	AUTOTRAN トランザクションのタイムアウトを表します。 有効な値は 0 ～ 2^{31} - 1 の範囲です。 0 は、タイムアウトが発生しないことを示します。 デフォルト値は 30 秒です。

表 5-6 AUTOTRAN、TRANTIME、および ROUTING パラメータの特性 (続

パラメータ	説明
ROUTING	このサービスを要求するトランザクションにデータ依存型ルーティングが指定されている ROUTING セクション内のエントリを指します。

トランザクションをサポートするように Domains コンフィギュレーション・ファイルを変更する (BEA Tuxedo CORBA サーバ)

ここでは、以下の内容について説明します。

- DMTLOGDEV、DMTLOGNAME、DMTLOGSIZE、MAXRDTRAN、および MAXTRAN パラメータの特性
- AUTOTRAN および TRANTIME パラメータの特性 (BEA Tuxedo CORBA および ATMI サーバ)

ドメインを介してトランザクションを有効にするには、Domains コンフィギュレーション・ファイル (DMCONFIG) の DM_LOCAL_DOMAINS および DM_REMOTE_SERVICES セクション内のパラメータを設定する必要があります。DM_LOCAL_DOMAINS セクションのエントリでは、ローカル・ドメインの特性を定義します。DM_REMOTE_SERVICES セクションのエントリでは、インポートされたサービスや、リモート・ドメインで使用可能なサービスに関する情報を定義します。

DMTLOGDEV、DMTLOGNAME、DMTLOGSIZE、MAXRDTRAN、および MAXTRAN パラメータの特性

Domains コンフィギュレーション・ファイルの DM_LOCAL_DOMAINS セクションでは、ローカル・ドメインおよびそれに関連するゲートウェイ・グループを指定します。このセクションは、ゲートウェイ・グループ (ローカル・ドメイン) ごとにエントリを持つ必要があります。各エントリは、そのグループで実行されている Domains ゲートウェイ・プロセスで必要なパラメータを指定します。

表 5-7 では、このセクションの 5 つのトランザクション関連パラメータ (DMTLOGDEV、DMTLOGNAME、DMTLOGSIZE、MAXRDTRAN、および MAXTRAN) を説明します。

表 5-7 DMTLOGDEV、DMTLOGNAME、DMTLOGSIZE、MAXRDTRAN、**および** MAXTRAN **パラメータの特性**

パラメータ	説明
DMTLOGDEV	このマシンの Domains トランザクション・ログ (DMTLOG) を含む BEA Tuxedo ファイル・システムを指定します。DMTLOG は、BEA Tuxedo システムの VTOC テーブルとしてデバイスに格納されています。このパラメータを指定しない場合、Domains ゲートウェイ・グループは要求をトランザクション・モードで処理できません。同じマシン上で実行するローカル・ドメインは、同じ DMTLOGDEV ファイル・システムを共用できますが、ローカル・ドメインごとに、DMTLOGNAME キーワードで指定した個別のログ (DMTLOGDEV の表) を作成する必要があります。
DMTLOGNAME	このドメインの Domains トランザクション・ログの名前を指定します。この名前は、複数のローカル・ドメインで同じ DMTLOGDEV を使用する場合、一意のものでなければなりません。値を指定しない場合は、デフォルトで DMTLOG 文字列が設定されます。名前は、30 文字以内にする必要があります。

表 5-7 DMTLOGDEV、DMTLOGNAME、DMTLOGSIZE、MAXRDTRAN、および MAXTRAN パラメータの特性 (続き)

パラメータ	説明
DMTLOGSIZE	このマシンの Domains トランザクション・ログのサイズをページ数単位で指定します。0 より大きく、BEA Tuxedo ファイル・システム上の空き領域より小さい値を指定します。値を指定しない場合は、デフォルトで 100 ページが設定されます。 注記　トランザクション内のドメインの数により、DMTLOGSIZE パラメータで指定するページ数が決まります。トランザクションは、必ずログ・ページと同じわけではありません。
MAXRDTRAN	トランザクションに含めることのできるドメインの最大数を指定します。0 より大きく、32,768 未満の値を指定します。値を指定しない場合、デフォルト値は 16 です。
MAXTRAN	このローカル・ドメイン上で同時に実行できるグローバル・トランザクションの最大数を指定します。0 以上で、TUXCONFIG ファイルに定義されている MAXGTT パラメータ以下の値を指定します。値を指定しない場合は、デフォルトの MAXGTT が指定されます。

AUTOTRAN および TRANTIME パラメータの特性 (BEA Tuxedo CORBA および ATMI サーバ)

Domains コンフィギュレーション・ファイルの DM_REMOTE_SERVICES セクションでは、インポートされ、リモート・ドメインで使用可能になったサービスに関する情報を示します。各リモート・サービスは、特定のリモート・ドメインに関連付けられています。

表 5-8 では、このセクションに指定するトランザクション関連のパラメータ (AUTORUN および TRANTIME) を説明します。

表 5-8 AUTOTRAN および TRANTIME パラメータの特性

パラメータ	説明
AUTOTRAN	リモート・サービスのトランザクションを自動的に開始 / 終了するために、ゲートウェイが使用します。この機能は、リモート・サービスに対して、信頼性のあるネットワーク通信を行う場合に必要です。この機能を指定するには、対応するリモート・サービス定義の中のパラメータ AUTOTRAN を Y に設定します。
TRANTIME	関連するサービスに対するトランザクションを自動的に開始するまでのデフォルトのタイムアウト値を秒単位で指定します。この値は 0 以上 2147483648 未満でなければなりません。デフォルト値は 30 秒です。0 は、マシンの最大タイムアウト値を示します。

トランザクションを使用する分散アプリケーションの例

ここでは、以下の内容について説明します。

- RESOURCES セクション
- MACHINES セクション
- GROUPS セクションおよび NETWORK セクション
- SERVERS、SERVICES、および ROUTING セクション

ここでは、トランザクションを有効にし、アプリケーションを 3 つのサイトに分散するサンプル CORBA アプリケーションのサンプル・コンフィギュレーション・ファイルを説明します。このアプリケーションには、以下の機能があります。

- ACCOUNT_ID に対してデータ依存型ルーティングが実行されます。
- データが 3 つのデータベースに分散されます。
- ATMI インターフェイスを使用してシステムと通信する BRIDGE プロセスが実行されます。
- 1 つのサイトからシステムを管理します。

ファイルは、7 つのセクション (RESOURCES、MACHINES、GROUPS、NETWORK、SERVERS、SERVICES、および ROUTING) で構成されます。

RESOURCES セクション

リスト 5-1 に示されている RESOURCES セクションは、以下のパラメータを指定します。

- MAXSERVERS、MAXSERVICES、および MAXGTT はデフォルトより少ない値です。これによって、掲示板のサイズが小さくなります。
- MASTER は SITE3 で、バックアップ・マスタは SITE1 です。
- MODEL が MP に設定されており、OPTIONS が LAN、MIGRATE に設定されているため、ネットワーク接続されたコンフィギュレーションを使用して移行を行うことができます。
- BBLQUERY が 180 に設定されており、SCANUNIT が 10 に設定されているため、DBBL は、リモートの BBL を 1800 秒おき (30 分おき) にチェックします。

コード リスト 5-1 RESOURCES **セクションの例**

```
*RESOURCES
#
IPCKEY         99999
UID            1
GID            0
PERM           0660
MAXACCESSERS 25
MAXSERVERS     25
MAXSERVICES    40
MAXGTT         20
MASTER         SITE3, SITE1
SCANUNIT       10
SANITYSCAN     12
BBLQUERY       180
BLOCKTIME      30
DBBLWAIT       6
OPTIONS        LAN, MIGRATE
MODEL          MP
LDBAL           Y
```

MACHINES セクション

リスト 5-2 に示されている MACHINES セクションは、以下のパラメータを指定します。

- TLOGDEVICE および TLOGNAME は、トランザクションが行われることを示します。
- TYPE パラメータはすべて異なりますが、これはマシン間で送られるすべてのメッセージに符号化 / 復号化が行われることを示します。

コード リスト 5-2　MACHINES **セクションの例**

```
*MACHINES
Gisela        LMID=SITE1
               TUXDIR="/usr/tuxedo"
               APPDIR="/usr/home"
               ENVFILE="/usr/home/ENVFILE"
               TLOGDEVICE="/usr/home/TLOG"
               TLOGNAME=TLOG
               TUXCONFIG="/usr/home/tuxconfig"
               TYPE="3B600"

romeo          LMID=SITE2
               TUXDIR="/usr/tuxedo"
               APPDIR="/usr/home"
               ENVFILE="/usr/home/ENVFILE"
               TLOGDEVICE="/usr/home/TLOG"
               TLOGNAME=TLOG
               TUXCONFIG="/usr/home/tuxconfig"
               TYPE="SEQUENT"

juliet        LMID=SITE3
              TUXDIR="/usr/tuxedo"
              APPDIR='/usr/home"
              ENVFILE="/usr/home/ENVFILE"
              TLOGDEVICE="/usr/home/TLOG"
              TLOGNAME=TLOG
              TUXCONFIG="/usr/home/tuxconfig"
              TYPE="AMDAHL"
```

GROUPS セクションおよび NETWORK セクション

リスト 5-3 に示されている GROUPS セクションと NETWORK セクションは、以下のパラメータを指定します。

- TMSCOUNT は 2 に設定されますが、これはグループあたり 2 つの TMS_SQL トランザクション・マネージャ・サーバだけが起動されることを示します。
- OPENINFO 文字列は、アプリケーションがデータベース・アクセスを行うことを示します。

コード リスト 5-3 GROUPS **セクションおよび** NETWORK **セクションの例**

```
*GROUPS
DEFAULT:         TMSNAME=TMS_SQL       TMSCOUNT=2
BANKB1           LMID=SITE1            GRPNO=1
 OPENINFO="TUXEDO/SQL:/usr/home/bankdl1:bankdb:readwrite"
BANKB2           LMID=SITE2            GRPNO=2
 OPENINFO="TUXEDO/SQL:/usr/home/bankdl2:bankdb:readwrite"
BANKB3            LMID=SITE3            GRPNO=3
 OPENINFO="TUXEDO/SQL:/usr/home/bankdl3:bankdb:readwrite"

*NETWORK
SITE1             NADDR="0X0002ab117B2D4359"
                  BRIDGE="/dev/tcp"
                  NLSADDR="0X0002ab127B2D4359"

SITE2             NADDR="0X0002ab117B2D4360"
                  BRIDGE="/dev/tcp"
                  NLSADDR="0X0002ab127B2D4360"

SITE3             NADDR="0X0002ab117B2D4361"
                  BRIDGE="/dev/tcp"
                  NLSADDR="0X0002ab127B2D4361"
```

SERVERS、SERVICES、および ROUTING セクション

リスト 5-4 に示されている SERVERS セクション、SERVICES セクション、および ROUTING セクションは、以下のパラメータを指定します。

- TLR サーバには、*tpsrvrinit()* 関数に渡される -T number があります。
- サービスに対するすべての要求は ACCOUNT_ID フィールドでルーティングされます。
- AUTOTRAN モードではサービスは行われません。

コード リスト 5-4 SERVERS、SERVICES、**および** ROUTING **セクションの例**

```
*SERVERS
DEFAULT: RESTART=Y MAXGEN=5 REPLYQ=N CLOPT=”-A”
TLR       SRVGRP=BANKB1    SRVID=1    CLOPT=”-A -- -T 100"
TLR       SRVGRP=BANKB2    SRVID=3    CLOPT=”-A -- -T 400"
TLR       SRVGRP=BANKB3    SRVID=4    CLOPT=”-A -- -T 700"
XFER      SRVGRP=BANKB1    SRVID=5    REPLYQ=Y
XFER      SRVGRP=BANKB2    SRVID=6    REPLYQ=Y
XFER      SRVGRP=BANKB3    SRVID=7    REPLYQ=Y

*SERVICES
DEFAULT:    AUTOTRAN=N
WITHDRAW        ROUTING=ACCOUNT_ID
DEPOSIT         ROUTING=ACCOUNT_ID
TRANSFER        ROUTING=ACCOUNT_ID
INQUIRY         ROUTING=ACCOUNT_ID

*ROUTING
ACCOUNT_ID      FIELD=ACCOUNT_ID     BUFTYPE=”FML”
                     RANGES=”MON - 9999:*,
                     10000 - 39999:BANKB1
                     40000 - 69999:BANKB2
                     70000 - 100000:BANKB3
                        “”
```

索引

い

印刷、製品のマニュアル viii

か

カスタマ・サポート情報 ix
関連情報 ix

さ

サポート
　技術情報 ix

ま

マニュアル、入手先 viii