BEA Tuxedo®

COBOL を使用した BEA Tuxedo アプリケーションのプログラミング

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 ... xiv
e-docs Web サイト ... xv
マニュアルの印刷方法 ... xv
関連情報 ... xv
サポート情報 ... xvi
表記上の規則 ... xvii

1. BEA Tuxedo プログラミングの概要

BEA Tuxedo 分散アプリケーションのプログラミング ... 1-1
コミュニケーション・パラダイム ... 1-3
BEA Tuxedo クライアント ... 1-5
BEA Tuxedo サーバ ... 1-6
　サーバの基本的な動作 ... 1-6
　要求元としてのサーバ ... 1-8
BEA Tuxedo API: ATMI ... 1-9

2. プログラミング環境

UBBCONFIG コンフィギュレーション・ファイルの更新 ... 2-1
環境変数の設定 ... 2-6
C 言語のデータ型に相当する COBOL 言語のデータ型の定義 ... 2-9
アプリケーションの起動と停止 ... 2-10

3. 型付きレコードの管理

型付きレコードの概要 ... 3-1
型付きレコードの定義 ... 3-7
VIEW 型レコード ... 3-9
　VIEW 型レコードの環境変数の設定 ... 3-10
　VIEW 記述ファイルの作成 ... 3-10
　VIEW コンパイラの実行 ... 3-14
FML 型レコード ... 3-17
　FML 型レコードの環境変数の設定 ... 3-17

フィールド・テーブル・ファイルの作成 3-18
型付きレコードの初期化 3-20
FML ヘッダ・ファイルの作成 3-23
XML 型レコード 3-24

4. クライアントのコーディング

アプリケーションへの参加 4-1
TPINFDEF-REC レコードの機能 4-4
クライアント命名 4-5
任意通知型通知の処理 4-6
システム・アクセス・モード 4-8
リソース・マネージャとの対応付け 4-9
クライアント認証 4-9
アプリケーションからの分離 4-10
クライアントのビルド 4-11
関連項目 4-13
クライアント・プロセスの例 4-13

5. サーバのコーディング

BEA Tuxedo システムの制御プログラム 5-1
システムで提供されるサーバおよびサービス 5-3
システムで提供されるサーバ : AUTHSVR( ) 5-4
システムで提供されるサービス : TPSVRINIT ルーチン 5-5
コマンド行オプションの取得 5-6
リソース・マネージャのオープン 5-7
システムで提供されるサービス : TPSVRDONE ルーチン 5-9
サーバのコーディングのためのガイドライン 5-10
サービスの定義 5-12
サービス・ルーチンの終了 5-20
応答の送信 5-20
記述子の無効化 5-26
要求の転送 5-27
サービスの宣言と宣言の取り消し 5-30
サービスの宣言 5-31
サービス宣言の取り消し 5-32
例 : サービスの動的な宣言と宣言の取り消し 5-32

サーバのビルド 5-34
関連項目 5-36

6. クライアントおよびサーバへの要求 / 応答のコーディング

要求 / 応答通信の概要...6-1
同期メッセージの送信...6-2
例 : 要求メッセージと応答メッセージに同じレコードを使用する.......6-4
例 : TPSIGRSTRT フラグを設定した同期メッセージの送信................6-6
例 : TPNOTRAN フラグを設定した同期メッセージの送信..................6-8
非同期メッセージの送信..6-10
非同期要求の送信 6-11
非同期応答の受信 6-14
メッセージの優先順位の設定および取得 6-14
メッセージの優先順位の設定 6-15
メッセージの優先順位の取得 6-17

7. 会話型クライアントおよびサーバのコーディング

会話型通信の概要 7-2
アプリケーションへの参加 7-4
接続の確立...7-4
メッセージの送受信...7-6
メッセージの送信 7-6
メッセージの受信 7-7
会話の終了 7-8
例 : 単純な会話の終了 7-10
例 : 階層構造の会話の終了..7-11
会話の切断...7-13
会話型のクライアントおよびサーバのビルド...7-14
会話型通信イベント...7-14

8. イベント・ベースのクライアントおよびサーバのコーディング

イベントの概要 8-2
任意通知型イベント 8-2
ブローカ・イベント 8-2
通知処理 8-3

イベント・ブローカ・サーバ 8-4
システム定義のイベント 8-5
イベント・ブローカ・プログラミング・インターフェイス 8-5
任意通知型メッセージ・ハンドラの定義 8-6
任意通知型メッセージの送信 8-7
名前によるメッセージのブロードキャスト 8-8
識別子によるメッセージのブロードキャスト 8-10
任意通知型メッセージの確認 8-10
任意通知型メッセージの取得 8-11
イベントのサブスクライブ 8-13
イベントに対するサブスクリプションの削除 8-17
イベントのポスト 8-17

9. グローバル・トランザクションのコーディング

グローバル・トランザクションとは 9-1
トランザクションの開始 9-3
トランザクションの終了 9-10
現在のトランザクションのコミット 9-11
トランザクションをコミットするための条件 9-11
2 フェーズ・コミット・プロトコル 9-12
現在のトランザクションのアボート 9-14
例 : 会話モードによるトランザクションのコミット 9-14
例 : パーティシパントのエラーの確認 9-16
グローバル・トランザクションの暗黙的な定義 9-18
XA 準拠のサーバ・グループに対するグローバル・トランザクションの定義 9-19
トランザクションが開始されたことの確認 9-20
関連項目 9-22

10. マルチスレッドおよびマルチコンテキスト・アプリケーションのプログラミング

マルチスレッドおよびマルチコンテキスト・アプリケーションのプログラミングに対するサポート 10-2
マルチスレッドおよびマルチコンテキスト・アプリケーションに関するプラットフォーム固有の検討事項 10-3

マルチスレッドおよびマルチコンテキスト・アプリケーションの計画と設計 10-4
マルチスレッドおよびマルチコンテキストとは 10-5
マルチスレッドとは 10-5
マルチコンテキストとは 10-7
マルチスレッド・アプリケーションまたはマルチコンテキスト・アプリケーションのライセンス 10-9
マルチスレッドおよびマルチコンテキスト・アプリケーションの利点と問題点 10-10
マルチスレッドおよびマルチコンテキスト・アプリケーションの利点 . 10-10
マルチスレッドおよびマルチコンテキスト・アプリケーションの問題点 10-12
クライアントでのマルチスレッドとマルチコンテキストの動作 10-13
起動フェーズ 10-14
クライアント・スレッドの複数コンテキストへの参加 10-14
クライアント・スレッドの既存のコンテキストへの切り替わり . . . 10-15
作業フェーズ 10-16
サービス要求 10-16
サービス要求に対する応答 10-16
トランザクション 10-16
任意通知型メッセージ 10-17
ユーザ・ログで保持されるスレッド固有の情報 10-19
完了フェーズ 10-19
サーバでのマルチスレッドとマルチコンテキストの動作 10-20
起動フェーズ 10-21
作業フェーズ 10-21
サーバ・ディスパッチ・スレッド 10-22
アプリケーション生成のスレッド 10-23
BBL によるシステム・プロセスの正常性チェック 10-24
システムで保持されるサーバ・スレッドの統計 10-24
ユーザ・ログで保持されるスレッド固有の情報 10-24
完了フェーズ 10-25
マルチスレッドおよびマルチコンテキスト・アプリケーションの設計上の検討事項 10-26

環境の要件 10-27
設計の要件 10-28
マルチスレッドやマルチコンテキストに適するアプリケーションのタスク 10-28
必要なアプリケーションと接続の数 10-29
同期に関する検討事項 10-30
アプリケーションの移植 10-30
最適なスレッド・モデル 10-30
ワークステーション・クライアントの相互運用性に関する制約 10-31
マルチスレッドおよびマルチコンテキスト・アプリケーションのインプリメント 10-32
マルチスレッドおよびマルチコンテキスト・アプリケーションのプログラミング開始前のガイドライン 10-32
マルチスレッド・アプリケーションに必要な条件 10-33
マルチスレッド・アプリケーションのプログラミングでの一般的な検討事項 10-33
並列性に関する検討事項 10-34
クライアントでマルチコンテキストを使用するためのコーディング 10-35
コンテキストの属性 10-36
初期化時のマルチコンテキストの設定 10-37
マルチコンテキスト・クライアントのセキュリティのインプリメント 10-38
クライアント終了前のスレッドの同期 10-38
コンテキストの切り替え 10-39
任意通知型メッセージの処理 10-42
マルチスレッドおよびマルチコンテキスト・アプリケーションにおけるトランザクションのコーディング規則 10-43
サーバでマルチコンテキストとマルチスレッドを使用するためのコーディング 10-44
コンテキストの属性 10-44
マルチコンテキスト・サーバのコーディング規則 10-45
サーバおよびサーバ・スレッドの初期化と終了 10-46
スレッドを生成するためのサーバのプログラミング 10-47
スレッドの生成 10-47
コンテキストへのスレッドの対応付け 10-47

マルチコンテキスト・サーバでアプリケーション・スレッドを生成するためのコード例 10-48
マルチスレッド・クライアントのコーディング 10-50
マルチスレッド・クライアントのコーディング規則 10-50
クライアントの複数のコンテキストへの初期化 10-52
クライアント・スレッドのコンテキスト状態の変化 10-53
マルチスレッド環境での応答の取得 10-55
マルチスレッド・マルチコンテキスト環境の環境変数 10-56
マルチスレッド・クライアントでのコンテキスト単位の関数とデータ構造体 10-58
マルチスレッド・クライアントでのプロセス単位の関数とデータ構造体 10-61
マルチスレッド・クライアントでのスレッド単位の関数とデータ構造体 10-62
マルチスレッド・クライアントのコード例 10-62
マルチスレッド・サーバのコーディング 10-65
マルチスレッドおよびマルチコンテキスト・アプリケーションのコードのコンパイル 10-65
マルチスレッドおよびマルチコンテキスト・アプリケーションのテスト ... 10-67
マルチスレッドおよびマルチコンテキスト・アプリケーションのテスト時の推奨事項 10-67
マルチスレッドおよびマルチコンテキスト・アプリケーションのトラブル・シューティング 10-68
tpinit( ) の TPMULTICONTEXTS フラグの間違った使用 10-68
TPMULTICONTEXTS が設定されていない場合の tpinit() の呼び出し 10-68
スレッドのスタック・サイズの不足 10-69
マルチスレッドおよびマルチコンテキスト・アプリケーションのエラー処理 10-69

11. エラーの管理

システム・エラー 11-1
アボート・エラー 11-3
BEA Tuxedo のシステム・エラー 11-3
通信ハンドルのエラー 11-4
上限値に関するエラー 11-4

無効な記述子によるエラー 11-5
会話に関するエラー 11-5
複製オブジェクトに関するエラー 11-6
一般的な通信呼び出しのエラー 11-6
TPESVCFAIL および TPESVCERR エラー...11-7
TPEBLOCK および TPGOTSIG エラー 11-7
無効な引数によるエラー 11-8
エントリがないために発生するエラー 11-8
オペレーティング・システムのエラー 11-9
パーミッション・エラー 11-10
プロトコル・エラー..11-10
キューに関するエラー 11-11
リリース間の互換性に関するエラー 11-11
リソース・マネージャ・エラー..11-11
タイムアウト・エラー 11-12
トランザクション・エラー 11-13
型付きレコードのエラー 11-13
アプリケーション・エラー 11-15
エラー処理..11-15
トランザクションについて 11-16
通信規則..11-16
トランザクション・エラー 11-18
致命的ではないトランザクション・エラー 11-18
致命的なトランザクション・エラー...11-19
ヒューリスティックな判断に関するエラー 11-21
トランザクション・タイムアウト 11-22
TPCOMMIT 呼び出し 11-22
TPNOTRAN..11-22
TPRETURN および TPFORWAR 呼び出し...11-23
tpterm( ) 関数 11-24
リソース・マネージャ 11-24
トランザクションのサンプル・シナリオ...11-25
呼び出し元と同じトランザクションでのサービス呼び出し..............11-26
AUTOTRAN が設定された別のトランザクションでのサービス呼び出し 11-26

新しい明示的なトランザクションを開始するサービスの呼び出し ..11-28
BEA Tuxedo システムで提供されるサブルーチン 11-29
中央イベント・ログ...11-30
ログの名前...11-30
ログ・エントリの形式...11-31
イベント・ログへの書き込み 11-32

12. Workstation コンポーネントに対する COBOL 言語のバインディング

UNIX のバインディング 12-1
クライアント・プログラムの作成 12-2
クライアント・プログラムを作成する 12-2
環境変数を設定する 12-3
Microsoft Windows のバインディング 12-5
クライアント・プログラムの作成 12-5
クライアント・プログラムのビルド...12-5
ACCEPT/DISPLAY クライアントのビルド 12-7

このマニュアルについて

このマニュアルでは、COBOL 言語を使用して BEA Tuxedo ATMI アプリケーションをプログラミング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「BEA Tuxedo プログラミングの概要」—BEA Tuxedo プログラミングの概要。分散アプリケーション・プログラミング、クライアント、サーバ、および BEA Tuxedo 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) について説明します。
- 第 2 章「プログラミング環境」—BEA Tuxedo プログラミング環境。BEA Tuxedo システムの設定、環境変数の設定、およびアプリケーションの起動と停止について説明します。
- 第 3 章「型付きレコードの管理」—VIEW レコード、FML レコード、XML レコードなどの型付きレコードの管理および使用方法。
- 第 4 章「クライアントのコーディング」—COBOL 言語を使用して BEA Tuxedo クライアント・アプリケーションをコーディングおよび構築する手順。クライアント・プロセスのサンプルが同梱されています。
- 第 5 章「サーバのコーディング」—サービスの定義や宣言など、COBOL 言語を使用して BEA Tuxedo サーバ・アプリケーションをコーディングおよび構築する手順。
- 第 6 章「クライアントおよびサーバへの要求 / 応答のコーディング」—要求 / 応答型のクライアントおよびサーバのコーディング手順。同期 / 非同期メッセージング、メッセージの優先順位の設定について説明します。
- 第 7 章「会話型クライアントおよびサーバのコーディング」—会話型のクライアントおよびサーバのコーディング手順。アプリケーションへの

参加、接続の確立、メッセージの送受信、および会話の終了について説明します。

- 第 8 章「イベント・ベースのクライアントおよびサーバのコーディング」—イベント・ベースのクライアントおよびサーバのコーディング手順。任意通知型のメッセージおよびイベントの処理について説明します。
- 第 9 章「グローバル・トランザクションのコーディング」—グローバル・トランザクションのコーディング手順。トランザクションの開始と終了について説明します。
- 第 10 章「マルチスレッドおよびマルチコンテキスト・アプリケーションのプログラミング」—1 つのプロセスで同時に複数のタスクを実行するアプリケーションのコーディング手順。この章では、マルチスレッドおよびマルチコンテキストのアプリケーションをプログラミングする方法について説明します。マルチスレッド化されたアプリケーションでは、1 つのプロセスに複数の実行単位が含まれます。マルチコンテキスト化されたアプリケーションでは、1 つのプロセスがドメイン内で複数の接続を確立したり、複数のドメインに対して接続を確立できます。
- 第 11 章「エラーの管理」—システム・エラーとアプリケーション・エラーに対する処理の方法。
- 第 12 章「Workstation コンポーネントに対する COBOL 言語のバインディング」—UNIX および Microsoft Windows プラットフォームでの COBOL 言語のバインディング。

対象読者

このマニュアルは、BEA Tuxedo 環境で COBOL 言語を使用してアプリケーションをプログラミングするアプリケーション開発者を対象にしています。

このマニュアルは、BEA Tuxedo プラットフォームおよび COBOL 言語のプログラミングについて理解していることを前提としています。

e-docs Web サイト

BEA 製品のマニュアルは BEA 社の Web サイト上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 Documents ページの [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 をお持ちではない場合は、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

以下の BEA Tuxedo マニュアルには、BEA Tuxedo 環境で BEA Tuxedo /Q コンポーネントを使用する方法、およびメッセージ・キュー・アプリケーションを実装する方法についての関連情報が掲載されています。

- 『BEA Tuxedo コマンド・リファレンス』の `cobcc(1)`

- 『BEA Tuxedo COBOL リファレンス』
- 『BEA Tuxedo のファイル形式とデータ記述方法』の `tuxenv(5)`

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.com 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 BEA Tuxedo プログラミングの概要

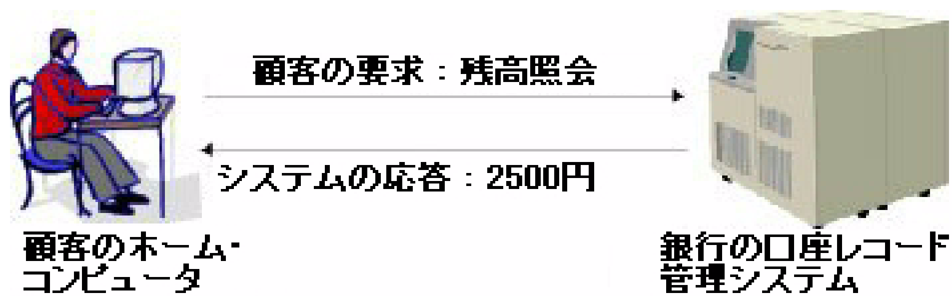
ここでは、次の内容について説明します。

- BEA Tuxedo 分散アプリケーションのプログラミング
- コミュニケーション・パラダイム
- BEA Tuxedo クライアント
- BEA Tuxedo サーバ
- BEA Tuxedo API: ATMI

BEA Tuxedo 分散アプリケーションのプログラミング

分散アプリケーションは、複数のハードウェア・システム上にあるソフトウェア・モジュールから構成され、これらのモジュールが互いに通信してアプリケーションのタスクが実行されます。たとえば、次の図に示すリモート・オンライン銀行業務システムの分散アプリケーションは、顧客のホーム・コンピュータで実行されるソフトウェア・モジュールと、すべての口座レコードを管理する銀行のコンピュータ・システムから構成されています。

図 1-1 分散アプリケーションの例 - オンライン銀行業務システム



たとえば、ログオンしてメニューからオプションを選択するだけで、残高を照会できます。この処理では、ローカル・ソフトウェア・モジュールが特別なアプリケーション・プログラミング・インターフェイス (API) ルーチンを使用して、リモート・ソフトウェア・モジュールと通信しています。

BEA Tuxedo 分散アプリケーション・プログラミング環境では、分散ソフトウェア・モジュール間で安全かつ信頼性の高い通信を行うために必要な API ルーチンが提供されています。BEA Tuxedo API は、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) と呼ばれます。

ATMI を使用すると、以下の操作を行うことができます。

- クライアントとサーバ間でのメッセージのやり取り。ネットワークを介して異機種マシン間でやり取りすることも可能です。
- クライアント命名およびセキュリティ機能の設定と使用。
- 複数の場所に格納されているデータを処理するトランザクションの定義と管理。
- データベース管理システム (DBMS) などのリソース・マネージャのオープンとクローズ。
- サービス要求のフロー管理、およびサービス要求を処理するサーバの可用性の管理。

コミュニケーション・パラダイム

次の表は、アプリケーション開発者が利用できる BEA Tuxedo のコミュニケーション・パラダイムを示しています。

表 1-1 コミュニケーション・パラダイム

パラダイム	説明
要求 / 応答型通信	要求 / 応答型通信では、あるソフトウェア・モジュールが 2 番目のソフトウェア・モジュールに要求を送り、その応答を受け取ります。要求元が応答を受け取るまで処理が行われずに待機する同期通信、または要求元が応答を待機する間も処理が継続される非同期通信の 2 種類があります。 このモードは、クライアント / サーバ相互作用とも呼ばれます。最初のソフトウェア・モジュールがクライアント、2 番目のソフトウェア・モジュールがサーバになります。 このパラダイムの詳細については、6-1 ページの「クライアントおよびサーバへの要求 / 応答のコーディング」を参照してください。
会話型通信	会話型通信は要求 / 応答型通信に似ていますが、「会話」が終了する前に複数の要求や応答が発生します。会話型通信では、会話が切断されるまでクライアントとサーバで状態の情報が保持されます。クライアントとサーバ間でメッセージをやり取りする方法は、使用するアプリケーション・プロトコルによって決定されます。 通常、会話型通信はサーバからクライアントへの長い応答のバッファとして使用されます。 このパラダイムの詳細については、7-1 ページの「会話型クライアントおよびサーバのコーディング」を参照してください。

パラダイム	説明
アプリケーション・キュー・ベースの通信	アプリケーション・キュー・ベースの通信では、遅延通信、つまり時間に依存しない通信が行われます。この通信では、クライアントとサーバがアプリケーション・キューを使用して通信します。BEA Tuxedo/Q 機能を使用すると、メッセージを永続的な記憶装置 (ディスク) や一時的な記憶装置 (メモリ) のキューに入れることができ、後で処理したり取り出すことができます。 アプリケーション・キュー・ベースの通信は、たとえば、メンテナンスのためにシステムをオフラインにしたときに要求をキューに登録する場合や、クライアントとサーバの処理速度が異なる場合に通信をバッファに格納する場合に使用します。 /Q 機能の詳細については、『BEA Tuxedo /Q コンポーネント』を参照してください。
イベント・ベースの通信	イベント・ベースの通信では、特別な状況 (イベント) が発生した場合に、クライアントやサーバがそれをクライアントに通知します。 イベントの通知には、次の 2 つの方法があります。 ■ 任意通知型イベント。クライアントやサーバからクライアントに直接通知される予測不能な状況です。 ■ ブローカ・イベント。予測不能な状況、または発生は予測できても発生時間を予測できない状況です。イベントは、メッセージの受信と転送を行う「無名ブローカ」プログラムによって、サーバからクライアントに間接的に通知されます。 イベント・ベースの通信は、BEA Tuxedo のイベント・ブローカ機能に基づいています。 このパラダイムの詳細については、8-1 ページの「イベント・ベースのクライアントおよびサーバのコーディング」を参照してください。

BEA Tuxedo クライアント

BEA Tuxedo クライアントとは、ユーザの要求を収集し、その要求に対するサービスを提供するサーバにその要求を転送するソフトウェア・モジュールです。ほとんどのソフトウェア・モジュールは、BEA Tuxedo クライアントとして動作できます。その場合、ATMI クライアント初期化ルーチンを呼び出して、BEA Tuxedo アプリケーションに「参加」します。以降、クライアントはサーバと情報をやり取りできるようになります。

クライアントは、ATMI 終了ルーチンを呼び出してアプリケーションから「分離」し、BEA Tuxedo システムにクライアントをトラッキングする必要がなくなったことを通知します。これにより、ほかの操作で BEA Tuxedo アプリケーションのリソースを使用できるようになります。

次は、基本的なクライアント・プロセスの処理を示す擬似コードです。

コード リスト 1-1　クライアントの擬似コード

```
START PROGRAM
BEA TUXEDO アプリケーションのクライアントとして登録
初期クライアント ID をデータ構造体に格納
最後まで実行
ユーザー入力を取得
ユーザ入力を DATA-REC に格納
サービス要求を送信
応答を受信
応答をユーザに送信
実行を終了
アプリケーションの終了
END PROGRAM
```

この擬似コードに示したほとんどの処理は、ATMI 呼び出しでインプリメントされます。ただし、ユーザ入力を DATA-REC に格納する処理と、ユーザに応答を返す処理は、COBOL ルーチンでインプリメントされます。

クライアントは、アプリケーションから分離する前に、サービス要求をいくつでも送受信できます。クライアントは、これらの要求を一連の要求 / 応答型呼び出しとして送信することができます。また、ある呼び出しから別の呼び出しに状態の情報を渡す必要がある場合は、会話型サーバに接続して要求を送ることもできます。どちらの方法でもクライアント・プログラムのロジックは同じですが、使用する ATMI 呼び出しは異なります。

ATMI クライアントを実行するには、`buildclient -C` コマンドを実行してクライアントをコンパイルし、BEA Tuxedo ATMI および必要なライブラリとリンクします。`buildclient(1)` コマンドの詳細については、4-1 ページの「クライアントのコーディング」を参照してください。

BEA Tuxedo サーバ

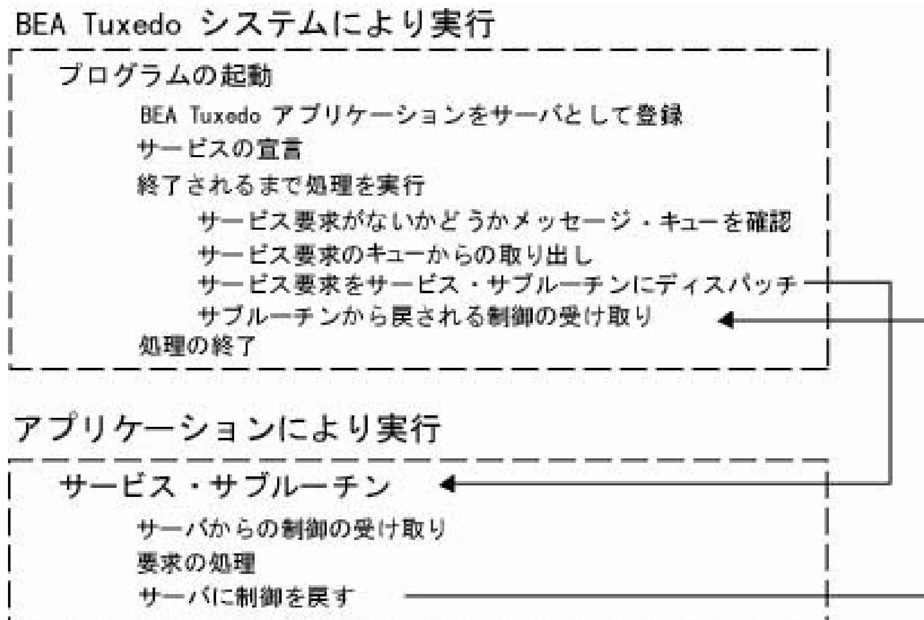
BEA Tuxedo サーバとは、クライアントに対して 1 つ以上のサービスを提供するプロセスです。サービスとは、クライアントが処理を要求する特定のビジネス・タスクです。サーバは、クライアントから要求を受け取り、それらを適切なサービス・サブルーチンにディスパッチします。

サーバの基本的な動作

サーバのビルドでは、サービス・サブルーチンと、BEA Tuxedo システムで提供される制御プログラムとがアプリケーションによって組み合わされます。この制御プログラムはシステムによって提供され、定義済みのルーチンから構成されています。この制御プログラムは、サーバの初期化と終了を行ったり、ユーザ入力をデータ構造体に格納して、要求を受信してサービス・ルーチンへのディスパッチを行ったりします。このすべての処理は、アプリケーションに透過的です。

次の図は、サーバとサービス・ルーチン間の相互作用を示す疑似コードです。

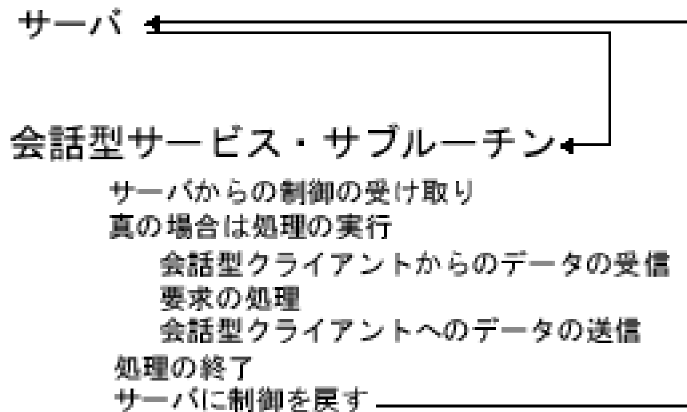
図 1-2 要求 / 応答型サーバとサービス・サブルーチンの擬似コード



初期化後、サーバは要求メッセージがメッセージ・キューに登録されるまで待機し、その要求をキューから取り出してサービス・サブルーチンに送り、要求を処理します。応答が必要な場合は、その応答は要求処理の一部と見なされます。

会話型パラダイムは、次の図の疑似コードで示すように、要求 / 応答型パラダイムとは多少異なります。

図 1-3 会話型サービス・サブルーチンの擬似コード



BEA Tuxedo システム提供の制御プログラムには、プロセスをサーバとして登録し、サービスを宣言し、キューから要求メッセージを取り出すために必要なコードが記述されています。ATMI 呼び出しは、要求を処理するサービス・サブルーチンで使用されます。サービス・サブルーチンのコンパイルとテストを行う準備ができたら、これらをサーバとリンクして、実行可能サーバを生成します。その場合、`buildserver -C` コマンドを実行します。

要求元としてのサーバ

クライアントが複数のサービスや同じサービスを複数回要求する場合、サービスのサブセットを別のサーバに転送して実行することができます。その場合、サーバはクライアント、つまりサービスの要求元として動作します。つまり、クライアントとサーバの両方が要求元になることができます。ただし、クライアントは要求元にしかなれません。このようなモデルは、BEA Tuxedo の ATMI 呼び出しを使用すると簡単にコーディングできます。

注記　要求 / 応答型サーバも、要求を別のサーバに転送できます。その場合、サーバはクライアント (要求元) としては動作しません。応答を必要としているのは、要求を転送したサーバではなく、元のクライアントだからです。

BEA Tuxedo API: ATMI

アプリケーションのロジックを記述する COBOL 言語のほかに、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) を使用する必要があります。この ATMI は、アプリケーションと BEA Tuxedo システム間のインターフェイスになります。

ATMI は、リソースのオープンとクローズ、トランザクションの開始と終了、およびクライアントとサーバ間の通信のサポートなどの処理に使用されるコンパクトな呼び出しのセットです。次の表は、ATMI 呼び出しをまとめたものです。各呼び出しについては、『BEA Tuxedo COBOL リファレンス』を参照してください。

表 1-2ATMI **呼び出し**

タスク ■訳文不要■ . .	COBOL **関数 . .**	**目的 . .**	**参照先 . .**
クライアントのメンバーシップ	`TPINITIALIZE`	クライアントをアプリケーションに参加させます。	4-1 ページの「クライアントのコーディング」
	`TPTERM`	クライアントをアプリケーションから分離します。	
複数のアプリケーション・コンテキストの管理	TPGETCTXT(3cbl)	現在のスレッドのコンテキストの識別子を取得します。	10-1 ページの「マルチスレッドおよびマルチコンテキスト・アプリケーションのプログラミング」
	TPSETCTXT(3cbl)	マルチコンテキスト・プロセスに現在のスレッドのコンテキストを設定します。	
サービスの登録と応答	`TPSVCSTART`	サービス情報を取得します。	5-1 ページの「サーバのコーディング」
	`TPSVRINIT`	サーバを初期化します。	
	`TPSVRDONE`	サーバを終了します。	
	`TPRETURN`	サービス・ルーチンを終了します。	
	`TPFORWAR`	要求を転送します。	

表 1-2ATMI 呼び出し

タスク ■訳文不要■ . .	COBOL **関数 . .**	**目的 . .**	**参照先 . .**
動的な宣言	TPADVERTISE	サービス名を宣言します。	5-1 ページの「サーバのコーディング」
	TPUNADVERTISE	サービス名の宣言を取り消します。	
メッセージの優先順位	TPGPRIO	最後の要求の優先順位を取得します。	5-1 ページの「サーバのコーディング」
	TPSPRIO	次の要求の優先順位を設定します。	
要求 / 応答型通信	TPCALL	サービスへの同期要求 / 応答を開始します。	■ 5-1 ページの「サーバのコーディング」 ■ 6-1 ページの「クライアントおよびサーバへの要求 / 応答のコーディング」
	TPACALL	非同期要求 (ファンアウト) を開始します。	
	TPGETRPLY	非同期応答を受け取ります。	
	TPCANCEL	非同期要求を取り消します。	
会話型通信	TPCONNECT	サービスとの会話を開始します。	7-1 ページの「会話型クライアントおよびサーバのコーディング」
	TPDISCON	会話を異常終了します。	
	TPSEND	会話中にメッセージを送信します。	
	TPRECV	会話中にメッセージを受信します。	
高信頼性キュー	TPENQUEUE(3cbl)	メッセージをメッセージ・キューに登録します。	『BEA Tuxedo /Q コンポーネント』
	TPDEQUEUE(3cbl)	メッセージをメッセージ・キューから取り出します。	

表 1-2ATMI 呼び出し

タスク ■訳文不要■ . .	COBOL 関数 . .	目的 . .	参照先 . .
イベント・ベースの通信	TPNOTIFY	クライアントに任意通知型メッセージを送信します。	8-1 ページの「イベント・ベースのクライアントおよびサーバのコーディング」
	TPBROADCAST	複数のクライアントにメッセージを送信します。	
	TPSETUNSOL	任意通知型メッセージのコールバックを設定します。	
	TPCHKUNSOL	任意通知型メッセージの到着を確認します。	
	TPGETUNSOL	任意通知型メッセージを取得します。	
	TPPOST	イベント・メッセージをポストします。	
	TPSUBSCRIBE	イベント・メッセージをサブスクライブします。	
	TPUNSUBSCRIBE	イベント・メッセージのサブスクリプションを削除します。	
トランザクション管理	TPBEGIN	トランザクションを開始します。	9-1 ページの「グローバル・トランザクションのコーディング」
	TPCOMMIT	現在のトランザクションをコミットします。	
	TPABORT	現在のトランザクションをロールバックします。	
	TPGETLEV	トランザクション・モードであるかどうかを確認します。	

表 1-2ATMI 呼び出し

タスク ■訳文不要■ . .	COBOL **関数 . .**	**目的 . .**	**参照先 . .**
リソース管理	TPOPEN(3cbl)	リソース・マネージャをオープンします。	■ 10-1 ページの「マルチスレッドおよびマルチコンテキスト・アプリケーションのプログラミング」 ■ 『BEA Tuxedo CORBA アプリケーション入門』
	TPCLOSE(3cbl)	リソース・マネージャをクローズします。	
セキュリティ	TPKEYOPEN(3cbl)	デジタル署名の生成、メッセージの暗号化 / 暗号解読のためのキー・ハンドルをオープンします。	『BEA Tuxedo CORBA アプリケーションのセキュリティ機能』
	TPKEYGETINFO(3cbl)	キー・ハンドルに対応付けられた情報を取得します。	
	TPKEYSETINFO(3cbl)	キー・ハンドルに対応付けられた属性 (省略可能) を設定します。	
	TPKEYCLOSE(3cbl)	TPKEYOPEN を使用して既にオープンされているキー・ハンドルをクローズします。	

2 プログラミング環境

ここでは、次の内容について説明します。

- UBBCONFIG コンフィギュレーション・ファイルの更新
- 環境変数の設定
- C 言語のデータ型に相当する COBOL 言語のデータ型の定義
- アプリケーションの起動と停止

UBBCONFIG コンフィギュレーション・ファイルの更新

アプリケーション管理者は、最初に UBBCONFIG コンフィギュレーション・ファイルにアプリケーションのコンフィギュレーションを定義します。プログラミング環境をカスタマイズするには、コンフィギュレーション・ファイルを作成するか更新します。

コンフィギュレーション・ファイルの作成または更新を行う場合は、次のガイドラインを参考にしてください。

- 既存のファイルをコピーして編集します。たとえば、サンプル・アプリケーション bankapp に付属するファイル ubbshm から作業を開始します。

- 複雑性を最小限にします。テストを行う場合は、共用メモリを使用する単一プロセッサ・システムとしてアプリケーションを設定します。データとしては、通常のオペレーティング・システムのファイルを使用します。
- コンフィギュレーション・ファイルの IPCKEY パラメータが、インストールされているシステムで使用されているほかのパラメータと競合しないようにします。詳細については、BEA Tuxedo アプリケーション管理者に確認し、『BEA Tuxedo アプリケーションの設定』を参照してください。
- UID および GID パラメータを設定し、定義したコンフィギュレーションが自分自身のものであることを示します。
- マニュアルを参照します。コンフィギュレーション・ファイルについては、『BEA Tuxedo のファイル形式とデータ記述方法』の UBBCONFIG(5) で説明しています。

次の表は、プログラミング環境に影響する UBBCONFIG コンフィギュレーション・ファイルのパラメータを示しています。パラメータは機能別に分類されています。

表 2-1 プログラミング関連の UBBCONFIG パラメータ (機能別)

機能	パラメータ	セクション	説明
グローバル・リソースの制限	MAXSERVERS	RESOURCES	コンフィギュレーション内のサーバの最大数。この値を設定する場合は、すべてのサーバの MAX 値を考慮する必要があります。
	MAXSERVICES	RESOURCES	コンフィギュレーション内のサービスの最大総数。
データ依存型ルーティング	BUFTYPE	ROUTING	このルーティング・エントリが有効なデータ・レコードのタイプとサブタイプのリスト。

表 2-1 **プログラミング関連の** UBBCONFIG **パラメータ (機能別) (続き)**

機能	パラメータ	セクション	説明
リンク・レベルの暗号	MINENCRYPTBITS	NETWORK	プロセスで使用できる暗号化の最低レベル。
	MAXENCRYPTBITS	NETWORK	プロセスで使用できる暗号化の最高レベル。
ロード・バランシング	LDBAL	RESOURCES	ロード・バランシングを有効にするかどうかを示すフラグ。ロード・バランシングが有効になっている場合、BEA Tuxedo システムは要求の負荷をネットワークで分散します。
	NETLOAD	MACHINES	呼び出し元クライアントからの要求をリモートに送る場合に、サービスのロード・ファクタに追加される数値。リモート・サーバ上でローカル・サーバを選択する際の基準になります。ロード・バランシングが有効になっていること (LDBAL に Y が設定されていること) が必要です。
	LOAD	SERVICES	サービス・インスタンスに対応付けられた相対的なロード・ファクタ。デフォルト値は 50 です。
セキュリティ	AUTHSVC	RESOURCES	システムに参加している各クライアントに対して、システムによって呼び出されるアプリケーション認証サービスの名前。
	SECURITY	RESOURCES	実行するアプリケーション・セキュリティの種類。

表 2-1 プログラミング関連の UBBCONFIG **パラメータ (機能別) (続き)**

機能	パラメータ	セクション	説明
会話型通信	MAXCONV	RESOURCES	特定のマシン上で同時に関与できる会話の最大数。0 〜 32,767 の値を指定します。SERVERS セクションに会話型サーバが定義されている場合、デフォルト値は 64 になります。それ以外の場合、デフォルト値は 1 になります。このパラメータに指定された値は、MACHINES セクションのマシンごとに上書きできます。
	CONV	SERVERS	会話型通信がサポートされているかどうかを示す値。このパラメータが N に設定されているか、値が指定されてない場合、サービスに対する TPCONNECT の呼び出しは失敗します。
	MIN/MAX	SERVERS	tmboot(1) によって起動するサーバのオカレンスの最小数と最大数。指定がない場合には、MIN と MAX はそれぞれ、1 と MIN がデフォルト値となります。要求 / 応答型サーバでも同じパラメータを利用できます。ただし、会話型サーバは、必要に応じて自動的に追加されます。そのため、MIN=1、MAX=10 と設定されている場合、tmboot によって最初に 1 つのサーバが起動します。そのサーバが提供するサービスに対して TPCONNECT が呼び出されると、システムによって 2 番目のサーバが起動します。同じように、新しいサーバが上限の 10 まで起動します。

表 2-1 プログラミング関連の UBBCONFIG **パラメータ (機能別) (続き)**

機能	パラメータ	セクション	説明
トランザクション管理	AUTOTRAN	SERVICES	サービス・ルーチンでトランザクション・モードを開始するかどうかを示す値。このパラメータに Y が設定されている場合、別のプロセスから要求メッセージを受信すると、サービス・サブルーチンでトランザクションが自動的に開始します。
マルチスレッド・サーバ	MAXDISPATCHTHREADS	SERVERS	各サーバ・プロセスによってスレッドが追加された場合、同時にディスパッチされるスレッドの最大数。
	MINDISPATCHTHREADS	SERVERS	最初のサーバの起動時に開始されるサーバ・ディスパッチ・スレッドの数。

コンフィギュレーション・ファイルは、オペレーティング・システムのテキスト・ファイルです。このファイルを実際にシステムで使用する場合は、tmloadcf(1) を実行して、バイナリ・ファイルに変換する必要があります。

関連項目

- 『BEA Tuxedo アプリケーションの設定』
- 『BEA Tuxedo のファイル形式とデータ記述方法』の UBBCONFIG(5)

環境変数の設定

アプリケーション管理者は、最初にアプリケーションの実行環境を定義する変数を設定します。これらの環境変数を設定するには、UBBCONFIG ファイルの MACHINES セクションで ENVFILE パラメータに値を指定します。詳細については、『BEA Tuxedo アプリケーションの設定』を参照してください。

アプリケーションのクライアント・ルーチンとサーバ・ルーチンに対して、既存の環境変数を更新したり、新しい変数を作成することができます。次の表は、よく使用される環境変数を示しています。変数は機能別に分類されています。

表 2-2 プログラミング関連の環境変数

関数	環境変数	定義内容 . .	使用先 . .
グローバル	TUXDIR	BEA Tuxedo システムのバイナリ・ファイルの場所。	BEA Tuxedo アプリケーション・プログラム
コンフィギュレーション	TUXCONFIG	BEA Tuxedo コンフィギュレーション・ファイルの場所。	BEA Tuxedo アプリケーション・プログラム

関数	環境変数	定義内容 . .	使用先 . .
コンパイル	ALTCC[1]	COBOL コンパイラを呼び出すコマンド。デフォルト値は cobcc です。	builclient() -C および buildserver() -C コマンド
	ALTCFLAGS[1]	COBOL コンパイラに渡すリンクのフラグ。このフラグは必須ではありません。	builclient() -C および buildserver() -C コマンド
	COBOPT	コンパイルのコマンド行で使用する引数。	builclient() -C および buildserver() -C コマンド
	COBCPY	コンパイラで使用される COBOL COPY の各ファイルが置かれたディレクトリ。	builclient() -C および buildserver() -C コマンド
データ圧縮	TMCMPPRFM	圧縮レベル (1 ～ 9)。	データ圧縮を行う BEA Tuxedo アプリケーション・プログラム
ロード・バランシング	TMNETLOAD	リモート・キューの負荷値に加算される数値。この値を指定すると、リモート・キューに実際より多くの作業負荷があるように設定できます。その結果、ロード・バランシングが有効になっていても、ローカル要求がリモート・キューよりローカル・キューに送られるようになります。	ロード・バランシングを実行する BEA Tuxedo アプリケーション・プログラム

関数	環境変数	定義内容 . .	使用先 . .
レコード管理	FIELDTBLS または FIELDTBLS32	FML および FML32 型付きレコードのフィールド・テーブル・ファイル名のカンマ区切りのリスト。FML VIEW 型のみで必要です。	FML 型付きレコード、FML32 型付きレコード、および FML VIEW
	FLDTBLDIR または FLDTBLDIR32	FML および FML32 のフィールド・テーブル・ファイルが検索されるディレクトリのコロン区切りのリスト。Windows 2000 では、セミコロンで区切られます。	FML 型付きレコード、FML32 型付きレコード、および FML VIEW
	VIEWFILES または VIEWFILES32	VIEW および VIEW32 型付きレコードで使用できるファイル名のカンマ区切りのリスト。	VIEW 型付きレコード、VIEW32 型付きレコード
	VIEWDIR または VIEWDIR32	VIEW および VIEW32 ファイルが検索されるディレクトリのコロン区切りのリスト。Windows 2000 では、セミコロンで区切られます。	VIEW 型付きレコード、VIEW32 型付きレコード

1. Windows 2000 システムでは、ALTCC および ALTCFLAGS 環境変数は使用できません。これらの変数を設定すると、予想外の結果が生じます。まず COBOL コンパイラを使用してアプリケーションをコンパイルし、次に生成されたオブジェクト・ファイルを buildclient または buildserver コマンドに渡す必要があります。

UNIX 環境では、環境変数 PATH に $TUXDIR/bin を追加して、アプリケーションが BEA Tuxedo システムのコマンドに対する実行可能ファイルを見つけられるようにします。環境設定の詳細については、『BEA Tuxedo アプリケーションの設定』を参照してください。

関連項目

- 『BEA Tuxedo アプリケーションの設定』

C 言語のデータ型に相当する COBOL 言語のデータ型の定義

次の表は、C 言語のデータ型に相当する COBOL 言語の各データ型を示しています。

表 2-3C 言語のデータ型に相当する COBOL 言語のデータ型

C 言語のデータ型	相当する COBOL 言語のデータ型
float	COMP-1
double	COMP-2
long	S9(9) COMP-5[1]
short	S9(4) COMP-5[1]
dec_t	COBOL COMP-3 パック 10 進フィールド

1. COMP-5 は、MicroFocus COBOL の使用のために提供されており、COBOL の整数型フィールドを対応する C フィールドのデータ形式と一致させます。VS COBOL II のデータ型は COMP です。

記憶域を効率よく使用するために、COBOL ではパック 10 進数がサポートされています。パック 10 進数では、2 桁の 10 進数が 1 バイトにパックされ、下位バイトに符号が格納されます。パック 10 進数の長さは 1 〜 9 バイトで、1 〜 17 桁の数字と符号を格納できます。

dec_t フィールドは VIEW に定義されています。サイズは、カンマで区切った 2 つの値で指定されます。最初の値は、COBOL で 10 進数が占める総バイト数を示します。2 番目の値は、COBOL での小数点以下の桁数を示します。次の式を使用すると、dec_t フィールドを COBOL 宣言に変換できます。

```
dec_t(m,n) => S9(2*m-(n+1),n)COMP-3
```

たとえば、VIEW でサイズが 6,4 と指定されている場合、小数点以下の桁数は 4 桁、整数部分は 7 桁になり、最後のハーフ・バイトに符号が格納されます。COBOL アプリケーションのプログラマは、これを 9(7)V9(4) と定義します。V は小数点を表します。FML では dec_t 型はサポートされていません。FML に依存した VIEW を使用する場合は、VIEW ファイルで各フィールドを C 言語の型にマップする必要があります。たとえば、パック 10 進数は FML の文字列フィールドにマップでき、マッピング関数を使用して形式を変換できます。

アプリケーションの起動と停止

アプリケーションを起動するには、tmboot(1) コマンドを実行します。このコマンドは、アプリケーションに必要な IPC 資源を確保し、管理プロセスとアプリケーション・サーバを起動します。

アプリケーションを停止するには、tmshutdown(1) コマンドを実行します。このコマンドは、サーバを停止させ、アプリケーションで使用されていた IPC 資源を解放します。ただし、データベースなどのリソース・マネージャで使用されていた資源は解放されません。

関連項目

- 『BEA Tuxedo コマンド・リファレンス』の tmboot(1) および tmshutdown(1)

3 型付きレコードの管理

ここでは、次の内容について説明します。

- 型付きレコードの概要
- 型付きレコードの定義
- VIEW 型レコード
- FML 型レコード
- XML 型レコード

型付きレコードの概要

ほかのアプリケーション・プログラムにデータを送信する場合、送信元プログラムはまず送信データをレコードに格納します。BEA Tuxedo システムのクライアントは、型付きレコードを使用してサーバにメッセージを送ります。「型付きレコード」とは、データ・レコードと補助型レコードが対になった COBOL 言語のレコードです。データ・レコードは、静的領域内に定義され、別のアプリケーション・プログラムに渡すアプリケーション・データが入ります。データ・レコードには、補助型レコードが付加されます。補助型レコードは、異機種システム間で情報をやり取りする場合に、BEA

Tuxedo システムで使用されるデータ・レコードの解釈と変換の規則を指定します。型付きレコードは、BEA Tuxedo システムでサポートされる分散プログラミング環境の基本要素の 1 つです。

なぜ「型付き」レコードを使用するのでしょうか。分散環境では、アプリケーションが異機種システムにインストールされ、異なるプロトコルを使用して複数のネットワーク間で通信が行われます。レコード・タイプが異なると、初期化、メッセージの送受信、およびデータの符号化 / 復号化にそれぞれ別のルーチンが必要になります。各レコードに特定のタイプが割り当てられていると、プログラマが介在しなくても、そのタイプに対応するルーチンを自動的に呼び出すことができます。

以下に示す表は、BEA Tuxedo システムでサポートされる型付きレコードと、そのレコードが次の条件を満たしているかを示しています。

- 自己記述型であるかどうか。つまり、レコードのデータ型と長さが、タイプとサブタイプ、およびそのデータからわかるかどうか。
- サブタイプが必要かどうか。
- 型付きレコードのデータ依存型ルーティングがシステムでサポートされているかどうか。
- 型付きレコードの符号化 / 復号化がシステムでサポートされているかどうか。

ルーティング用のルーチンが必要な場合は、アプリケーション・プログラマが用意します。

表 3-1 型付きバッファ

型付きレコード	機能説明	自己記述型	サブタイプ	データ依存型ルーティング	符号化 / 復号化
CARRAY	未定義の文字配列。LOW-VALUE を含むことができます。BEA Tuxedo システムでは配列のセマンティクスは解釈されないので、この型付きレコードは曖昧なデータを処理する場合に使用します。CARRAY は自己記述型ではないので、転送時には長さを指定する必要があります。システムではバイトは解釈されないので、マシン間のメッセージ送信では符号化 / 復号化はサポートされません。	該当せず	該当せず	該当せず	該当せず
FML (フィールド操作言語)	BEA Tuxedo システム固有の自己記述型レコード・タイプ。このレコードでは、各データ・フィールドに対応する識別子、オカレンス番号、場合によっては長さを示す値が格納されています。型付きレコードでは、データからの独立性と柔軟性が確立されています。 FML 型レコードでは、フィールド識別子とフィールド長に 16 ビットが使用されます。 詳細については、3-17 ページの「FML 型レコード」を参照してください。	該当	該当せず	該当	該当

表 3-1 型付きバッファ (続き)

型付きレコード	機能説明	自己記述型	サブタイプ	データ依存型ルーティング	符号化 / 復号化
FML32	FML と同じ。ただし、フィールド識別子とフィールド長に 32 ビットが使用されます。より長いフィールドを多数使用できるので、レコード全体が大きくなります。 ただし、C プログラミング言語で FML 型レコードの操作に使用できる FML ルーチンは、COBOL 言語では使用できません。COBOL 言語で FML32 を使用する主な目的は、単に VIEW32 または FML32 型レコードが使用されている C 言語プログラムを操作することです。 詳細については、3-17 ページの「FML 型レコード」を参照してください。	該当	該当せず	該当	該当
STRING	最後が LOW-VALUE 文字で終了する文字配列。異なる文字セットを使用するマシン間でデータを交換する場合は、BEA Tuxedo システムによってデータが自動的に変換されます。	該当せず	該当せず	該当せず	該当せず

表 3-1 型付きバッファ (続き)

型付きレコード	機能説明	自己記述型	サブタイプ	データ依存型ルーティング	符号化 / 復号化
VIEW	アプリケーションで定義される COBOL データ構造体。VIEW 型には、個々のデータ構造体を示すサブタイプが必要です。VIEW 記述ファイル (データ構造体のフィールドとタイプが定義されたファイル) は、VIEW 型レコードに定義されたデータ構造体を使用するクライアント・プロセスとサーバ・プロセスがアクセスできなければなりません。異なるタイプのマシン間でレコードがやり取りされる場合は、符号化 / 復号化が自動的に行われます。詳細については、3-9 ページの「VIEW 型レコード」を参照してください。	該当せず	該当	該当	該当
VIEW32	VIEW と同じ。ただし、長さとカウントのフィールド長に 32 ビットが使用されます。より長いフィールドを多数使用できるので、レコード全体が大きくなります。 COBOL 言語で VIEW32 を使用する主な目的は、単に VIEW32 または FML32 型レコードが使用されている C 言語プログラムを操作することです。 詳細については、3-9 ページの「VIEW 型レコード」を参照してください。	該当せず	該当	該当	該当
X_COMMON	VIEW と同じ。ただし、このバッファ型は COBOL と C プログラム間の互換性を取るために使用されます。フィールド・タイプとして使用できるのは、short、long、および string だけです。	該当せず	該当	該当	該当

表 3-1 **型付きバッファ** (続き)

型付きレコード	機能説明	自己記述型	サブタイプ	データ依存型ルーティング	符号化 / 復号化
XML	XML 文書は、次の要素から構成されます。 ■ 符号化された文字の並びで構成されるテキスト ■ 文書の論理構造の記述と、その構造に関する情報 XML 文書のルーティングは、エレメントの内容、またはエレメント・タイプと属性値に基づいて行われます。使用されている文字符号化は XML パーサによって判別されます。符号化が BEA Tuxedo のコンフィギュレーション・ファイル (UBBCONFIG(5) と DMCONFIG(5)) で使用されているネイティブな文字セット (US-ASCII または EBCDIC) と異なる場合、エレメントと属性名は US-ASCII または EBCDIC に変換されます。詳細については、3-24 ページの「XML 型レコード」を参照してください。	該当せず	該当せず	該当	該当せず
X_OCTET	CARRAY と同じ。	該当せず	該当せず	該当せず	該当せず

すべてのレコード・タイプは、$TUXDIR/lib ディレクトリの tmtypesw.c ファイルに定義されています。クライアント・プログラムとサーバ・プログラムで認識されるレコード・タイプは、tmtypesw.c に定義されているものだけです。tmtypesw.c ファイルを編集して、レコード・タイプを追加したり削除できます。また、UBBCONFIG の BUFTYPE パラメータを使用して、特定のサービスで処理できるタイプとサブタイプを制限できます。

tmtypesw.c ファイルは、共用オブジェクトやダイナミック・リンク・ライブラリのビルドに使用されます。このオブジェクトは、BEA Tuxedo 管理サーバ、およびアプリケーション・クライアントとアプリケーション・サーバによって動的にロードされます。

関連項目

- 3-9 ページの「VIEW 型レコード」
- 3-17 ページの「FML 型レコード」
- 3-24 ページの「XML 型レコード」
- 『BEA Tuxedo のファイル形式とデータ記述方法』の tuxtypes(5)
- 『BEA Tuxedo のファイル形式とデータ記述方法』の UBBCONFIG(5)

型付きレコードの定義

TPTYPE-REC COBOL 構造体は、アプリケーション・データの送受信に必ず使用されます。

次の表は、TPTYPE-REC 構造体のフィールドを示しています。

フィールド	説明
REC-TYPE	アプリケーションで送信または受信するレコード・タイプを指定します。
SUB-TYPE	レコード (VIEW レコードなど) 内でレコード・タイプをさらに細かく分類する場合は、そのサブタイプを指定します。

フィールド	説明
LEN	データの送信時に、送信バイト数を指定します。データ送信が正常に行われると、LEN には送信されたバイト数が格納されます。データの受信時に、TPTYPE-REC の LEN はデータ・レコードに送られるバイト数を指定します。呼び出しが正常に行われると、LEN にはデータ・レコードに送られたバイト数が格納されます。受信メッセージのサイズが LEN で指定されているサイズより大きい場合、データは切り捨てられます。つまり、LEN を超えて受信されたデータは破棄され、TPTYPE-STATUS に TPTRUNCATE が設定されます。

次のコード例は、TPTYPE データ構造体を示しています。

```
05 REC-TYPE PIC X(8).
  88 X-OCTET VALUE "X_OCTET".
  88 X-COMMON VALUE "X_COMMON".
05 SUB-TYPE PIC X(16).
05 LEN PIC S9(9) COMP-5.
  88 NO-LENGTH VALUE 0.
05 TPTYPE-STATUS PIC S9(9) COMP-5.
  88 TPTYPEOK  VALUE 0.
  88 TPTRUNCATE VALUE 1.
```

VIEW 型レコード

VIEW 型レコードには 2 種類あります。1 つは FML VIEW で、FML レコードから生成される COBOL レコードです。もう 1 つは、単なる非依存型 COBOL レコードです。

FML レコードを COBOL レコードに変換して再び元に戻す理由 (および FML VIEW 型レコードを使用する目的) は、FML 関数が COBOL プログラミング環境では使用できないからです。

FML 型レコードの詳細については、『BEA Tuxedo FML リファレンス』を参照してください。

VIEW 型レコードを使用するには、次の手順に従います。

- 適切な環境変数を設定します。
- 各構造体を VIEW 記述ファイルに定義します。
- BEA Tuxedo VIEW コンパイラの viewc -C ÇšégópÇµÇƒÅAVIEW ãLèqÉtÉ@ÉCÉãÇšÉRÉìÉpÉCÉãÇµÇ‹Ç²ÅB このコマンドを実行すると、1 つ以上の COBOL の COPY ファイルが各 VIEW 記述ファイルに生成されます。各 COPY ファイルには、データ記述レコードが記述されます。これらのレコードは、プログラムの要求に応じて、DATA DIVISION の LINKAGE セクションまたは WORKING STORAGE セクションで使用できます。

VIEW 型レコードの環境変数の設定

アプリケーションで VIEW 型レコードを使用するには、次の環境変数を設定します。

表 3-2VIEW 型レコードの環境変数

環境変数	説明
FIELDTBLS または FIELDTBLS32	FML または FML32 型レコードのフィールド・テーブル・ファイル名のカンマ区切りのリスト。FML VIEW 型のみで必要です。
FLDTBLDIR または FLDTBLDIR32	FML または FML32 型レコードのフィールド・テーブル・ファイルが検索されるディレクトリのコロン区切りのリスト。Microsoft Windows では、セミコロンで区切られます。FML VIEW 型のみで必要です。
VIEWFILES または VIEWFILES32	VIEW または VIEW32 記述ファイルに使用されるファイル名のカンマ区切りのリスト。
VIEWDIR または VIEWDIR32	VIEW または VIEW32 ファイルが検索されるディレクトリのコロン区切りのリスト。Microsoft Windows では、セミコロンで区切られます。

VIEW 記述ファイルの作成

VIEW 型レコードを使用するには、VIEW 記述ファイルに COBOL レコードを定義する必要があります。VIEW 記述ファイルには、各エントリの VIEW、および COBOL プロシージャのマッピングと FML 変換パターンを記述した VIEW が定義されています。VIEW の名前は、COBOL プログラムに含まれる copy ファイルの名前に相当します。

VIEW 記述ファイルの各レコードは、次の形式で定義します。

```
$ /* VIEW 構造体 */
  VIEW viewname
  type    cname     fbname   count   flag    size    null
```

次の表は、VIEW 記述ファイルに指定する必要がある各 COBOL レコードのフィールドを示しています。

表 3-3VIEW 記述ファイルのフィールド

フィールド	説明
type	フィールドのデータ型。short、long、float、double、char、string、または carray を指定できます。
cname	COBOL レコードのフィールド名。
fbname	FML から VIEW、または VIEW から FML への変換ルーチンを使用する場合、対応する FML 名をこのフィールドに指定する必要があります。このフィールド名は、FML フィールド・テーブル・ファイルにも必要です。FML に依存しない VIEW には必要ありません。
count	フィールドの出現回数。
flag	次のいずれかのオプション・フラグを指定します。 ■ P － LOW-VALUE 値の解釈を変更します。 ■ S －フィールド化レコードから構造体に一方向のマッピングを行います。 ■ F －構造体からフィールド化レコードへの一方向マッピングを行います。 ■ N －ゼロ方向のマッピングを行います。 ■ C －連想カウント・メンバ (ACM) に追加フィールドを生成します。 ■ L － STRING および CARRAY に転送されるバイト数を保持します。
size	STRING または CARRAY 型レコードの最大長を指定します。それ以外のレコード・タイプでは、このフィールドは無視されます。

表 3-3VIEW 記述ファイルのフィールド (続き)

フィールド	説明
null	ユーザ定義の LOW-VALUE 値、または - の場合はフィールドのデフォルト値。VIEW 型レコードで使用される LOW-VALUE 値は、空の COBOL レコード・メンバを示します。 数値型の場合、デフォルトの LOW-VALUE 値は 0 (dec_t の場合は 0.0) になります。文字型の場合、デフォルトの LOW-VALUE 値は &slq;\0‘ になります。STRING 型と CARRAY 型の場合、デフォルトの LOW-VALUE 値は &d“drq; になります。 エスケープ文字として使用されている定数も、LOW-VALUE 値の指定に使用できます。VIEW コンパイラで認識されるエスケープ定数は、\ddd (d は 8 進数)、\0、\n、\t、\v、\r、\f、\\、\′、および \″ です。 STRING、CARRAY、および LOW-VALUE 値は、二重引用符または一重引用符で囲みます。VIEW コンパイラでは、ユーザ定義の LOW-VALUE 値でエスケープされていない引用符は使用できません。 VIEW メンバ記述の LOW-VALUE フィールドにキーワード NONE を指定することもできます。このキーワードは、そのメンバの LOW-VALUE 値がないことを示します。文字列メンバおよび文字配列メンバのデフォルト値の最大サイズは、2660 文字です。詳細については、『BEA Tuxedo FML リファレンス』を参照してください。

行頭に # または $ 文字を付けてコメント行を挿入できます。行頭に $ が挿入されたコード行は、.h ファイルに出力されます。

次のコード例は、FML レコードに基づく VIEW 記述サンプル・ファイルの一部です。このコード例では、*fbname* フィールドを指定する必要があり、この値は対応するフィールド・テーブル・ファイルの値と一致していなければなりません。CARRAY1 フィールドのオカレンス・カウントが 2 に設定されていること、C フラグが設定されて追加のカウント・エレメントの作成が定義

されていることに注目してください。また、L フラグが設定され、アプリケーションが CARRAY1 フィールドを格納するときの文字数を示す長さエレメントが定義されています。

コード リスト 0-1 FML VIEW の VIEW 記述ファイル

```
$ /* VIEW 構造体 */
  VIEW MYVIEW
  #type     cname      fbname    count   flag    size    null
  float     float1     FLOAT1     1       -       -       0.0
  double    double1    DOUBLE1    1       -       -       0.0
  long      long1      LONG1      1       -       -       0
  short     short1     SHORT1     1       -       -       0
  int       int1       INT1       1       -       -       0
  dec_t     dec1       DEC1       1       -       9,16    0
  char      char1      CHAR1      1       -       -       '\0'
  string    string1    STRING1    1       -       20      '\0'
  carray    carray1    CARRAY1    2       CL      20      '\0'
  END
```

次のコード例は、同じ VIEW 記述ファイルで非依存型 VIEW のものを示しています。

コード リスト 0-2 非依存型 VIEW の VIEW 記述ファイル

```
$ /* VIEW データ構造体 */
  VIEW MYVIEW
  #type    cname    fbname  count   flag    size    null
  float    float1   -       1       -       -       -
  double   double1  -       1       -       -       -
  long     long1    -       1       -       -       -
  short    short1   -       1       -       -       -
  int      int1     -       1       -       -       -
  dec_t    dec1     -       1       -       9,16    -
  char     char1    -       1       -       -       -
  string   string1  -       1       -       20      -
  carray   carray1  -       2       CL      20      -
  END
```

この形式は FML 依存型 VIEW と同じです。ただし、*fbname* フィールドと *null* フィールドには意味がなく、viewc コンパイラで無視されます。これらのフィールドには、プレースホルダとしてダッシュ (-) などの値を挿入する必要があります。

VIEW コンパイラの実行

VIEW 型レコードをコンパイルするには、引数として VIEW 記述ファイルの名前を指定して viewc -C コマンドを実行します。非依存型 VIEW を指定するには、-n オプションを使用します。生成される出力ファイルを書き込むディレクトリを指定することもできます (省略可能)。デフォルトでは、出力ファイルはカレント・ディレクトリに書き込まれます。

たとえば、FML 依存型 VIEW をコンパイルするには、次のようにコンパイラを実行します。

```
viewc -C myview.v
```

注記　VIEW32 型レコードをコンパイルするには、viewc32 -C コマンドを実行します。

非依存型 VIEW の場合、コマンド行で次のように -n オプションを指定します。

```
viewc -C -n myview.v
```

viewc コマンドでは、次が出力されます。

- MYVIEW.cbl など、1 つ以上の COBOL の COPY ファイル
- 同じ VIEW を共有する C 言語ルーチンに対して、アプリケーション・プログラムで使用される構造体定義が記述されたヘッダ・ファイル
- myview.V など、バイナリ・バージョンのソース記述ファイル

  注記　Microsoft Windows など、大文字と小文字が区別されないプラットフォームでは、バイナリ・バージョンのソース記述ファイル名には、拡張子 vv (myview.vv など) が使用されます。

次のコード例は、viewc によって生成される COBOL の COPY ファイルを示しています。

コード リスト 0-3 COBOL の COPY ファイルのコード例

```
*    VIEWFILE: "myview.v"
*    VIEWNAME: "MYVIEW"
  05 FLOAT1                        USAGE IS COMP-1.
  05 DOUBLE1                       USAGE IS COMP-2.
  05 LONG1                         PIC S9(9) USAGE IS COMP-5.
  05 SHORT1                        PIC S9(4) USAGE IS COMP-5.
  05 FILLER                        PIC X(02).
  05 INT1                          PIC S9(9) USAGE IS COMP-5.
  05 DEC1.
    07 DEC-EXP                     PIC S9(4) USAGE IS COMP-5.
    07 DEC-POS                     PIC S9(4) USAGE IS COMP-5.
    07 DEC-NDGTS                   PIC S9(4) USAGE IS COMP-5.
*     DEC-DGTS は実際のパック 10 進値です。
    07 DEC-DGTS                    PIC S9(1)V9(16) COMP-3.
    07 FILLER                      PIC X(07).
  05 CHAR1                         PIC X(01).
  05 STRING1                       PIC X(20).
  05 FILLER                        PIC X(01).
  05 L-CARRAY1 OCCURS 2 TIMES      PIC 9(4) USAGE IS COMP-5.
*    CARRAY1 の長さ
  05 C-CARRAY1                     PIC S9(4) USAGE IS COMP-5.
*    CARRAY1 のカウント
  05 CARRAY1 OCCURS 2 TIMES        PIC X(20).
  05 FILLER                        PIC X(02).
```

VIEW に対する COBOL の COPY ファイルは、COPY 文を使用してクライアント・プログラムとサービス・サブルーチンに含める必要があります。

このコード例では、コンパイラは FILLER ファイルを読み込んで、COBOL 言語コードでのフィールドの配置を C 言語コードでの配置と一致させています。

パック 10 進値の DEC1 は、5 つのフィールドから構成されます。このうちの DEC-EXP、DEC-POS、DEC-NDGTS、および FILLER フィールドは、C 言語だけで使用され、dec_t 型で定義されます。これらのフィールドは、COBOL レコードの配置を満たすために取り込まれます。COBOL アプリケーションでは、これらのフィールドを使用しないでください。

5 番目のフィールド (DEC-DGTS) には、システムによって実際のパック 10 進値が格納されます。COBOL プログラムではこの値を使用する必要があります。ATMI 呼び出しは、DEC-DGTS フィールドに以下の操作を行います。

- C 言語プログラムから COBOL 言語プログラムにレコードが渡される前にそのフィールドを生成します。
- COBOL 言語プログラムから C 言語プログラムにフィールドが渡される場合、フィールドを dec_t 型に変換し直します。

唯一の制約として、COBOL 言語プログラムでは、ATMI インターフェイス外から C 言語関数に直接レコードを渡すことはできません。COBOL 言語プログラムと C 言語関数では 10 進数の形式が異なるからです。

最後に、COBOL の COPY サンプル・ファイルで L-CARRAY1 長さフィールドが 2 回使用されていることに注目してください。つまり、CARRAY1 と C-CARRAY1 カウント・フィールドに対して 1 回ずつ使用されています。

viewc は、C 言語バージョンのヘッダ・フィルを生成します。このファイルを使用すると、C と COBOL 言語のサービスやクライアント・プログラムを混在させることができます。

関連項目

- 3-17 ページの「FML 型レコード」
- 3-24 ページの「XML 型レコード」
- 『BEA Tuxedo コマンド・リファレンス』の viewc、viewc32(1)

FML 型レコード

FML インターフェイスは、C 言語で使用されるように設計されたものです。COBOL 言語に対しては、ルーチンが提供されています。そのため、受信した FML 型レコードを COBOL レコードに変換して処理した後で、FML 型に再度変換できます。

FML 型レコードを使用するには、次の手順に従います。

- 適切な環境変数を設定します。
- FML フィールド・テーブルに、使用する可能性があるフィールドを記述します。
- FINIT を使用して FML レコードを初期化します。
- FML ヘッダ・ファイルを作成し、アプリケーションで同じ VIEW を共有する C 言語ルーチンの #include 文に、そのヘッダ・ファイルを指定します。

FML ルーチンは、フィールド化レコードから C 構造体への変換、またその逆の変換など、型付きレコードを操作する場合に使用します。これらの関数を使用すると、データ構造やデータの格納状態がわからなくても、データ値にアクセスしたり更新できます。FML ルーチンの詳細については、『BEA Tuxedo FML リファレンス』を参照してください。

FML 型レコードの環境変数の設定

アプリケーション・プログラムで FML 型レコードを使用するには、次の環境変数を設定する必要があります。

表 3-4FML 型レコードの環境変数

環境変数	説明
FIELDTBLS または FIELDTBLS32	FML または FML32 型付きレコードのフィールド・テーブル・ファイル名のカンマ区切りのリスト。
FLDTBLDIR または FLDTBLDIR32	FML または FML32 型バッファのフィールド・テーブル・ファイルが検索されるディレクトリのコロン区切りのリスト。Microsoft Windows では、セミコロンで区切られます。

フィールド・テーブル・ファイルの作成

FML 型レコードや FML 依存型 VIEW を使用する場合は、常にフィールド・テーブル・ファイルが必要です。フィールド・テーブル・ファイルは、FML 型レコードのフィールドの論理名をそのフィールドを一意に識別する文字列にマッピングします。

FML フィールド・テーブルの各フィールドは、次の形式で定義します。

```
$ /* FML 構造体 */
   *base value
   name     number     type   flags   comments
```

次の表は、FML フィールド・テーブルに指定する必要がある FML フィールドを示しています。

表 3-5 フィールド・テーブル・ファイルのフィールド

フィールド	説明
*base *value*	後続のフィールド番号をオフセットするためのベース値。関連するフィールドのセットを簡単にグループ分けし、番号を付け直すことができるようになります。*base オプションを使用すると、フィールド番号を再利用できます。16 ビットのレコードの場合、ベース値とそれに関連する番号を加算した値が、100 以上 8191 未満でなければなりません。 注記　BEA Tuxedo システムでは、フィールド番号 1 ～ 100 と 6,000 ～ 7,000 は、内部使用のために予約されています。FML ではフィールド番号 101 ～ 8,191、FML32 ではフィールド番号 101 ～ 33、554、および 431 がアプリケーション定義のフィールド用に使用できます。
name	フィールドの識別子。この値は 30 文字以下の文字列で、英数字と下線文字だけを指定できます。
rel-number	フィールドの相対数値。現在のベース値が指定されている場合、この値は現在のベース値に加算されて、フィールド番号が計算されます。
type	フィールドのタイプ。指定できるのは、char、string、short、long、float、double、または carray です。
flag	将来使用するために予約されたフィールド。プレースホルダとしてダッシュ (-) を挿入します。
comment	コメント (省略可能)。

すべてのフィールドは省略可能です。また、複数個使用できます。

次のコード例は、FML àÀë¹â^ VIEW の例で使用されるフィールド・テーブル・ファイルを示しています。

コード リスト 0-4 FML VIEW **のフィールド・テーブル・ファイル**

```
# name        number    type      flags    comments
  FLOAT1      110       float     -        -
  DOUBLE1     111       double    -        -
  LONG1       112       long      -        -
  SHORT1      113       short     -        -
  INT1        114       long      -        -
  DEC1        115       string    -        -
  CHAR1       116       char      -        -
  STRING1     117       string    -        -
  CARRAY1     118       carray    -        -
```

型付きレコードの初期化

FML 型レコードは、FINIT プロシージャを使用して初期化する必要があります。TPINIT プロシージャは、指定された FML レコード (できればワード境界に配置されているレコード) を使用し、FMLINFO レコードの FML-LENGTH フィールドに長さとして指定された値を使用します。

TPNOCHANGE が設定されている場合は、プログラムによって (作成されたのではなく) 受信された FML レコードが自動的に初期化されます。その場合、FINIT を呼び出す必要はありません。

次のコード例は、初期化の方法を示しています。

コード リスト 0-5 FML/VIEW **変換**

```
WORKING-STORAGE SECTION.
* レコード・タイプおよび長さ
 01 TPTYPE-REC.
COPY TPTYPE.
* 呼び出しの状態
 01 TPSTATUS-REC.
```

```
COPY TPSTATUS.
* サービス呼び出しフラグ / レコード
 01 TPSVCDEF-REC.
COPY TPSVCDEF.
* TPINIT フラグ / レコード
 01 TPINFDEF-REC.
COPY TPINFDEF.
* FML 呼び出しフラグ / レコード
 01 FML-REC.
COPY FMLINFO.
*
*
* アプリケーション FML レコード - 配置
 01 MYFML.
05 FBFR-DTA OCCURS 100 TIMES   PIC S9(9) USAGE IS COMP-5.
* アプリケーション VIEW レコード
 01 MYVIEW.
        COPY MYVIEW.

.....

* MYVIEW へのデータの移動

.....

* FML レコードの初期化
 MOVE LENGTH OF MYFML TO FML-LENGTH.
CALL "FINIT" USING MYFML FML-REC.
IF NOT FOK
        MOVE "FINIT Failed" TO LOGMSG-TEXT
        PERFORM DO-USERLOG
        PERFORM EXIT-PROGRAM
 END-IF.

* VIEW の FML レコードへの変換
 SET FUPDATE TO TRUE.
MOVE "MYVIEW" TO VIEWNAME.
CALL "FVSTOF" USING MYFML MYVIEW FML-REC.
IF NOT FOK
        MOVE "FVSTOF Failed" TO LOGMSG-TEXT
        PERFORM DO-USERLOG
        PERFORM EXIT-PROGRAM
 END-IF.
* FML レコードを使用したサービスの呼び出し
 MOVE "FML" TO REC-TYPE IN TPTYPE-REC.
MOVE SPACES TO SUB-TYPE IN TPTYPE-REC.
MOVE LENGTH OF MYFML TO LEN.
CALL "TPCALL" USING TPSVCDEF-REC
        TPTYPE-REC
```

```
        MYFML
        TPTYPE-REC
        MYFML
        TPSTATUS-REC.
IF NOT TPOK
        MOVE "TPCALL MYFML Failed" TO LOGMSG-TEXT
        PERFORM DO-USERLOG
        PERFORM EXIT-PROGRAM
 END-IF.
* FML レコードの MYVIEW への再変換
 CALL "FVFTOS" USING MYFML MYVIEW FML-REC.
IF NOT FOK
        MOVE "FVFTOS Failed" TO LOGMSG-TEXT
        PERFORM DO-USERLOG
        PERFORM EXIT-PROGRAM
 END-IF.
```

このコード例では、FVSTOF プロシージャで FML レコードを VIEW レコードに変換しています。VIEW を定義するために、VIEW コンパイラによって生成された copy ファイルが読み込まれています。FML-REC レコードには VIEWNAME と FML-MODE 転送モードがあり、転送モードには FUPDATE、FOJOIN、FJOIN、または FCONCAT を設定できます。これらのモードで行われる処理は、Fupdate、Fupdate32(3fml)、Fojoin、Fojoin32(3fml)、Fjoin、Fjoin32(3fml)、Fconcat、Fconcat32(3fml) での処理と同じです。

FVFTOS プロシージャは、VIEW レコードを FML レコードに変換しています。パラメータは FVSTOF プロシージャのパラメータと同じですが、FML-MODE を設定する必要はありません。各フィールドは、VIEW のエレメントの記述に基づいて、フィールド化レコードから構造体にコピーされます。フィールド化レコードのフィールドに対応するエレメントが COBOL レコードに存在しない場合、そのフィールドは無視されます。COBOL レコードに指定されたエレメントに対応するフィールドがフィールド化レコードに存在しない場合、そのエレメントに NULL 値がコピーされます。使用する NULL 値は、エレメントごとに VIEW 記述ファイルに定義できます。

フィールドの複数のオカレンスを COBOL レコードに格納するには、レコード・エレメントを OCCURS で定義します。レコードのフィールドのオカレンス数がエレメントのオカレンス数より少ない場合は、余分なエレメントには

NULL 値が割り当てられます。また、レコードのフィールドのオカレンス数がエレメントのオカレンス数より多い場合は、余分なオカレンスは無視されます。

FML32 および VIEW32 では、FINIT32、FVSTOF32、および FVFTOS32 プロシージャを使用する必要があります。

正常終了した場合は、FML-STATUS に FOK が設定されます。エラーが発生した場合は、FML-STATUS に 0 以外の値が設定されます。

FML ヘッダ・ファイルの作成

クライアント・プログラムやサービス・サブルーチンで FML 型レコードを使用するには、FML ヘッダ・ファイルを作成し、アプリケーションの #include 文にそのヘッダ・ファイルを指定する必要があります。

フィールド・テーブル・ファイルから FML ヘッダ・ファイルを作成するには、mkfldhdr(1) コマンドを使用します。たとえば、myview.flds.h というファイルを作成するには、次のコマンドを入力します。

```
mkfldhdr myview.flds
```

FML32 型レコードの場合は、mkfldhdr32 コマンドを使用します。

次のコード例は、mkfldhdr コマンドによって作成される myview.flds.h ヘッダ・ファイルを示しています。

コード リスト 0-6 myview.flds.h **ヘッダ・ファイル**

```
/*        fname    fldid            */
/*        -----    -----            */

#define  FLOAT1    ((FLDID)24686)   /* 番号 : 110 タイプ : float  */
#define  DOUBLE1   ((FLDID)32879)   /* 番号 : 111 タイプ : double */
#define  LONG1     ((FLDID)8304)    /* 番号 : 112 タイプ : long   */
#define  SHORT1    ((FLDID)113)     /* 番号 : 113 タイプ : short  */
#define  INT1      ((FLDID)8306)    /* 番号 : 114 タイプ : long   */
#define  DEC1      ((FLDID)41075)   /* 番号 : 115 タイプ : string */
#define  CHAR1     ((FLDID)16500)   /* 番号 : 116 タイプ : char   */
```

```
#define  STRING1   ((FLDID)41077)   /* 番号 : 117 タイプ : string */
#define  CARRAY1   ((FLDID)49270)   /* 番号 : 118 タイプ : carray */
```

アプリケーションの #include 文に新しいヘッダ・ファイルを指定します。ヘッダ・ファイルがインクルードされると、シンボリック名でフィールドを参照できるようになります。

関連項目

- 3-9 ページの「VIEW 型レコード」
- 3-24 ページの「XML 型レコード」
- 『BEA Tuxedo コマンド・リファレンス』の mkfldhdr、mkfldhdr32(1)

XML 型レコード

XML 型レコードを使用すると、BEA Tuxedo アプリケーションで XML を使用して、アプリケーション内やアプリケーション間でデータを交換できるようになります。BEA Tuxedo アプリケーションでは、単純な XML 型レコードの送受信や、それらのレコードを適切なサーバにルーティングできます。解析など、XML 文書のすべての処理ロジックはアプリケーション側にあります。

XML 文書は、次の要素から構成されます。

- 文書のテキストを符号化した文字の並び
- 文書の論理構造の記述と、その構造に関する情報

イベント処理で行われるフォーマット処理とフィルタ処理は、STRING 型レコードが使用されている場合はサポートされますが、XML 型レコードではサポートされません。そのため、XML 型レコードのレコード・タイプ・スイッチ内の _tmfilter と _tmformat のポインタは、LOW-VALUE に設定されます。

BEA Tuxedo システムの XML パーサは、次の操作を行います。

- 符号化された文字の自動検出
- 文字コードの変換
- データ・エレメントの内容と属性値の検出
- データ型の変換

XML 型レコードでは、データ依存型ルーティングがサポートされています。XML 文書のルーティングは、エレメントの内容、またはエレメントのタイプと属性値に基づいて行われます。使用される文字符号化は XML パーサによって判別されます。符号化が BEA Tuxedo のコンフィギュレーション・ファイル (UBBCONFIG と DMCONFIG) で使用されるネイティブな文字セット (US-ASCII または EBCDIC) と異なる場合、エレメントと属性名は US-ASCII または EBCDIC に変換されます。

XML 文書には、ルーティング用に設定する属性を含めなければなりません。属性がルーティング基準として設定されていても XML 文書に含まれていない場合、ルーティング処理は失敗します。

エレメントの内容と属性値は、ルーティング・フィールド値の構文とセマンティクスに従っていることが必要です。また、ルーティング・フィールド値のタイプも指定しなければなりません。XML でサポートされるのは文字データだけです。範囲フィールドが数値の場合、そのフィールドの内容や値はルーティング処理時に数値に変換されます。

関連項目

- 3-9 ページの「VIEW 型レコード」
- 3-17 ページの「FML 型レコード」

4 クライアントのコーディング

ここでは、次の内容について説明します。

- アプリケーションへの参加
- TPINFDEF-REC レコードの機能
- アプリケーションからの分離
- クライアントのビルド
- クライアント・プロセスの例

アプリケーションへの参加

クライアントがサービスを要求する場合、BEA Tuxedo アプリケーションに明示的または暗黙的に参加している必要があります。アプリケーションに参加すると、クライアントは要求を送り、その応答を受け取ることができるようになります。

クライアントが明示的にアプリケーションに参加するには、次の文法を指定して TPINITIALIZE(3cbl) を呼び出します。

```
01 TPINFDEF-REC.
   COPY TPINFDEF.
01 USER-DATA-REC        PIC X(any-length).
01 TPSTATUS-REC.
   COPY TPSTATUS.
CALL "TPINITIALIZE" USING TPINFDEF-REC USER-DATA-REC TPSTATUS-REC.
```

クライアントが TPINITIALIZE を呼び出す前にサービス要求 (または ATMI 呼び出し) を行うと、暗黙的にアプリケーションに参加したことになります。その場合、SPACES パラメータを使用して、TPINITIALIZE がクライアントではなく BEA Tuxedo システムによって呼び出されます。TPINFDEF-REC レコードは BEA Tuxedo システムの特別な型付きレコードであり、クライアントがアプリケーションに参加するときに、クライアントの ID と認証の情報をシステムに渡すためにクライアント・プログラムによって使用されます。このレコードは、次のように COBOL の COPY ファイルに定義されています。

```
05 USRNAME           PIC X(30).
05 CLTNAME           PIC X(30).
05 PASSWD            PIC X(30).
05 GRPNAME           PIC X(30).
05 NOTIFICATION-FLAG PIC S9(9) COMP-5.
  88 TPU-SIG           VALUE 1.
  88 TPU-DIP           VALUE 2.
  88 TPU-IGN           VALUE 3.
05 ACCESS-FLAG       PIC S9(9) COMP-5.
  88 TPSA-FASTPATH     VALUE 1.
  88 TPSA-PROTECTED    VALUE 2.
05 DATALEN           PIC S9(9) COMP-5.
```

次の表は、COBOL の COPY ファイルに定義されているフィールドを示しています。

表 4-1COBOL **の** COPY **ファイルのフィールド**

フィールド	**説明**
USRNAME	呼び出し元を表す名前。このフィールドには、UNIX コマンドの getuid(2) の戻り値を指定することもできます。USRNAME には、MAXTIDENT で指定された文字数 (30 文字) までの値を指定できます。
CLTNAME	アプリケーション定義のセマンティクスに従ったクライアント名。CLTNAME には、MAXTIDENT で指定された文字数 (30 文字) までの値を指定できます。
PASSWD	非暗号化形式のアプリケーション・パスワード。TUXCONFIG ファイルに格納されたアプリケーション・パスワードとの認証で、TPINITIALIZE によって使用されます。PASSWD には、MAXTIDENT の値までの文字列を指定します。
GRPNAME	クライアントを対応付けるリソース・マネージャ・グループの名前。クライアントは、グローバル・トランザクションの一部として、XA 準拠のリソース・マネージャにアクセスできるようになります。GRPNAME には、MAXTIDENT で指定された文字数 (30 文字) までの値を指定できます。ただし、現在のところ、GRPNAME は SPACES として渡す必要があります。その場合、クライアントはリソース・マネージャ・グループに対応付けられず、デフォルトのクライアント・グループに属することを指定します。
NOTIFICATION-FLAG	使用する通知メカニズム、およびシステム・アクセス・モード。有効な値については、4-6 ページの「任意通知型通知の処理」を参照してください。
ACCESS-FLAG	使用するシステム・アクセス・モード。使用できる値については、4-8 ページの「システム・アクセス・モード」を参照してください。

フィールド	説明
DATALEN	認証サービスに送信されるアプリケーション固有データの長さ。ネイティブ・クライアントの場合、このデータはシステムによって符号化されずに、クライアントによって提供される認証サービスに渡されます。ワークステーション・クライアントの場合、クライアント認証はシステムによって行われ、暗号化形式でネットワークを介して渡されます。

TPINITIALIZE が呼び出されると、USRNAME と CLTNAME フィールドはクライアント・プロセスに対応付けられます。この 2 つのフィールドは、ブロードキャスト通知と管理統計情報の取得に使用されます。

関連項目

- 『BEA Tuxedo COBOL リファレンス』の TPINITIALIZE(3cbl)

TPINFDEF-REC レコードの機能

次の TPINFDEF-REC レコードの機能を ATMI クライアントで利用するには、TPINITIALIZE を明示的に呼び出す必要があります。

- クライアント命名
- 任意通知型通知の処理
- システム・アクセス・モード
- リソース・マネージャとの対応付け
- クライアント認証

クライアント命名

クライアントがアプリケーションに参加すると、BEA Tuxedo システムによって一意なクライアント識別子が割り当てられます。識別子は、クライアントによって呼び出される各サービスに渡されます。識別子は、任意通知型通知に使用することもできます。

一意なクライアント名とユーザ名をそれぞれ 30 文字以内で割り当てることもできます。その場合、TPINFDEF-REC レコードを使用して名前を TPINITIALIZE に渡します。BEA Tuxedo システムでは、各プロセスに対応付けられているクライアント名とユーザ名、およびプロセスが実行されているマシンの論理マシン ID (LMID) を組み合わせることにより、そのプロセスに対して一意な識別子が使用されます。これらのフィールド値を取得する方法は選択することができます。

注記　プロセスがアプリケーションの管理ドメイン以外で実行されている場合 (つまり、管理ドメインに接続されたワークステーション上で実行されている場合)、アプリケーションにアクセスするためにワークステーション・クライアントで使用されているマシンの LMID が割り当てられます。

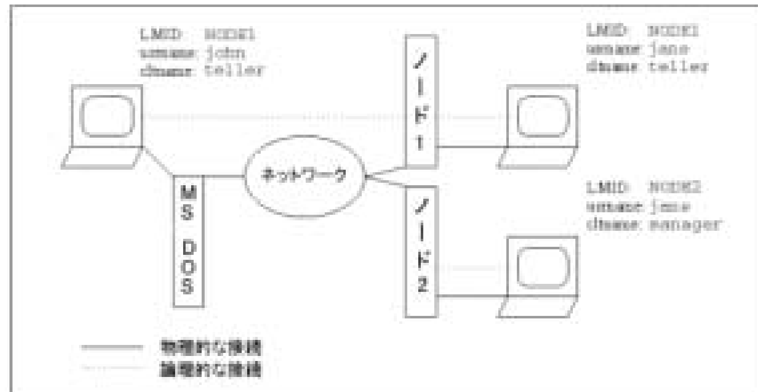
クライアント・プロセスに対して一意な識別子が作成されると、次の操作を行うことができます。

- クライアント認証をインプリメントできます。
- TPNOTIFY と TPBROADCAST を使用して、特定のクライアントまたはクライアントのグループに任意通知型メッセージを送信できます。
- tmadmin(1) を使用して、詳細な統計データを収集できます。

任意通知型メッセージの送受信の詳細については、「イベント・ベースのクライアントおよびサーバのコーディング」を参照してください。tmadmin(1) の詳細については、『BEA Tuxedo C リファレンス』を参照してください。

次の図は、アプリケーションにアクセスするクライアントに名前を割り当てる方法を示しています。この例では、ジョブ関数を示す cltname フィールドがアプリケーションで使用されています。

図 4-1 クライアントの名前付け



任意通知型通知の処理

任意通知型通知とは、クライアントが予期していないサービス要求に対する応答 (またはエラー・コード) を受け取る通信です。たとえば、管理者が 5 分後にシステムをシャットダウンすることを通知するメッセージをブロードキャストした場合などです。

クライアントに任意通知型メッセージを通知する方法は数多くあります。たとえば、オペレーティング・システムがクライアントにシグナルを送って、クライアントの現在の処理を中断させる方法があります。BEA Tuxedo システムでは、ATMI 呼び出しが行われるたびに任意通知型メッセージが到着していないかどうかがデフォルトで確認されます。これはディップ・インと呼ばれる方法で、次の利点があります。

- すべてのプラットフォームでサポートされています。
- 現在の処理が中断されません。

「ディップ・イン」では、メッセージの到着を確認するまでの間隔が長い場合があります。そのため、アプリケーションで TPCHKUNSOL を呼び出して、既に到着している任意通知型メッセージがないかどうかを確認できます。TPCHKUNSOL 呼び出しの詳細については、8-1 ページの「イベント・ベースのクライアントおよびサーバのコーディング」を参照してください。

クライアントが TPINITIALIZE を使用してアプリケーションに参加する場合、フラグを定義して任意通知型メッセージの処理方法を指定できます。クライアントへの通知では、次の表に示す値を NOTIFICATION-FLAG に指定できます。

表 4-2TPINFDEF-REC **レコードのクライアント通知でのフラグ**

フラグ	説明
TPU_SIG	シグナルによる任意通知を選択します。このフラグは、シングル・スレッドでシングル・コンテキストのアプリケーションのみで使用します。このモードの利点は、直ちに通知できることです。このモードには、次のような不都合があります。 ■ ネイティブ・クライアントを実行している場合、呼び出し元プロセスで送信元プロセスと同じ *UID* を使用する必要があります。ワークステーション・クライアントには、この制約はありません。 ■ すべてのプラットフォーム上で TPU_SIG が使用できるわけではありません。特に、MS-DOS ワークステーションでは使用できません。 システムや環境の要件を満たしていない場合にこのフラグを指定すると、フラグに TPU_DIP が設定され、ログにイベントが記録されます。
TPU_DIP (デフォルト)	ディップ・インを使用した任意通知型メッセージを指定します。クライアントは TPSETUNSOL 呼び出しを使用してメッセージ処理ルーチンの名前を指定し、TPCHKUNSOL 呼び出しを使用して待機中の任意通知型メッセージを確認できます。

フラグ	説明
TPU_THREAD	別のスレッド内の THREAD 通知を指定します。このフラグは、マルチスレッドをサポートするプラットフォーム専用です。マルチスレッドがサポートされていないプラットフォームで TPU_THREAD を指定すると、無効な引数として処理されます。その結果、エラーが返されて、TP-STATUS に TPEINVAL が設定されます。
TPU_IGN	任意通知を無視します。

TPINFDEF-REC フラグの詳細については、『BEA Tuxedo COBOL リファレンス』の TPINITIALIZE(3cbl) を参照してください。

システム・アクセス・モード

アプリケーションは、protected または fastpath のいずれかのモードで BEA Tuxedo システムにアクセスできます。クライアントは、TPINITIALIZE を使用してアプリケーションに参加するときに、モードを要求できます。モードを指定するには、TPINFDEF-REC レコードの ACCESS-FLAG フィールドに次のいずれかのモードを指定して、その値を TPINITIALIZE に渡します。

表 4-3TPINFDEF-REC レコードのシステム・アクセス・フラグ

モード	説明
TPSA-PROTECTED	アプリケーション内で ATMI 関数を呼び出し、共用メモリを使用して BEA Tuxedo システムの内部テーブルにアクセスします。BEA Tuxedo システム・ライブラリ外のアプリケーション・コードからのアクセスに対して、共用メモリを保護します。この値は、NO_OVERRIDE が指定されている場合を除き、UBBCONFIG の値に優先します。UBBCONFIG の詳細については、『BEA Tuxedo アプリケーションの設定』を参照してください。

モード	説明
TPSA-FASTPATH (デフォルト)	アプリケーション・コード内で ATMI 関数を呼び出し、共用メモリを使用して BEA Tuxedo システム内部にアクセスします。BEA Tuxedo システム・ライブラリ外のアプリケーション・コードからのアクセスに対して、共用メモリを保護しません。この値は、NO_OVERRIDE が指定されている場合を除き、UBBCONFIG の値に優先します。UBBCONFIG の詳細については、『BEA Tuxedo アプリケーションの設定』を参照してください。

リソース・マネージャとの対応付け

アプリケーション管理者は、リソース・マネージャに対応付けられたサーバ (トランザクションを調整するための管理プロセスを提供するサーバを含む) をグループ化できます。グループの定義については、『BEA Tuxedo アプリケーションの設定』を参照してください。

アプリケーションに参加している場合、クライアントは TPINFDEF-REC の *grpname* フィールドにグループ名を指定して、特定のグループに参加できます。

クライアント認証

BEA Tuxedo システムでは、オペレーティング・システムのセキュリティ、アプリケーション・パスワード、ユーザ認証、省略可能なアクセス制御リスト、必須のアクセス制御リスト、リンク・レベルの暗号化などのセキュリティ・レベルを設定できます。セキュリティ・レベル設定の詳細については、『BEA Tuxedo アプリケーションの設定』を参照してください。

セキュリティ・レベルとしてアプリケーション・パスワードが設定されている場合、アプリケーションに参加するときに、すべてのクライアントがアプリケーション・パスワードを入力する必要があります。管理者はアプリケーション・パスワードを設定したり変更できます。また、そのパスワードを有効なユーザに提供する必要があります。

このレベルのセキュリティが設定されていると、BEA Tuxedo システムで提供される ud(1) などのクライアント・プログラムでは、アプリケーション・パスワードの入力が求められます。ud、wud(1) の詳細については、『BEA Tuxedo アプリケーション実行時の管理』を参照してください。アプリケーション固有のクライアント・プログラムには、ユーザからパスワードを取得するコードが記述されていることが必要です。クライアントがアプリケーションに参加するために TPINITIALIZE を呼び出すと、非暗号化パスワードが TPINFDEF-REC レコードに格納されて評価されます。

注記　パスワードは画面には表示しません。

TPCHKAUTH(3cbl) を使用すると、次の内容を確認できます。

- アプリケーションで認証が必要かどうか。
- アプリケーションで認証が必要な場合、次のどちらの認証が行われるか。
  - アプリケーション・パスワードに基づくシステム認証
  - アプリケーション・パスワードとユーザ固有の情報に基づくアプリケーション認証

通常、クライアントは TPINITIALIZE より先に TPCHKAUTH を呼び出して、初期化時に指定する必要のあるセキュリティ情報を確認します。

セキュリティのプログラミング手法の詳細については、『BEA Tuxedo CORBA アプリケーションのセキュリティ機能』を参照してください。

アプリケーションからの分離

クライアントがすべてのサービスを要求して、それに対する応答を受信したら、TPTERM(3cbl) を使用してアプリケーションから分離できます。TPTERM の呼び出しには、次の文法を使用します。

```
01 TPSTATUS-REC.
   COPY TPSTATUS.
CALL "TPTERM" USING TPSTATUS-REC.
```

クライアントのビルド

実行可能クライアントをビルドするには、`buildclient(1)` コマンドを実行して、BEA Tuxedo システム・ライブラリとそのほかのすべての参照ファイルを使用してアプリケーションをコンパイルします。COBOL プログラムをコンパイルするには、`-C` オプションを指定します。次は、`buildclient` コマンドの構文です。

```
buildclient -C filename.cbl -o filename -f filenames -l filenames
```

次の表は、`buildclient` コマンドのオプションを示しています。

表 4-4buildclient のオプション

オプションまたは引数 . .	説明 . .
filename.cbl	コンパイルする COBOL 言語のアプリケーション。
-o *filename*	実行可能な出力ファイル。出力ファイルのデフォルト名は a.out です。
-f *filenames*	BEA Tuxedo システムのライブラリより先にリンクされるファイルのリスト。-f オプションは、コマンド行で複数回指定できます。また、各 -f に複数のファイル名を指定できます。COBOL プログラム・ファイル (*file*.cbl) を指定すると、リンクされる前にコンパイルが行われます。ほかのオブジェクト・ファイル (*file*.o) を個別に、またはアーカイブ・ファイル (*file*.a) にまとめて指定することもできます。
-l *filenames*	BEA Tuxedo システム・ライブラリの後にリンクされるファイルのリスト。-l オプションは、コマンド行で複数回指定できます。また、各 -l に複数のファイル名を指定できます。COBOL プログラム・ファイル (*file*.cbl) を指定すると、リンクされる前にコンパイルが行われます。ほかのオブジェクト・ファイル (*file*.o) を個別に、またはアーカイブ・ファイル (*file*.a) にまとめて指定することもできます。

オプションまたは引数 . .	説明 . .
-r	実行可能サーバにリンクされるリソース・マネージャのアクセス・ライブラリ。アプリケーション管理者は、buildtms(1) コマンドを使用して、すべての有効なリソース・マネージャ情報を $TUXDIR/updataobj/RM ファイルに事前に定義しておく必要があります。指定できるリソース・マネージャは 1 つだけです。詳細については、『BEA Tuxedo アプリケーションの設定』を参照してください。

注記　BEA Tuxedo ライブラリは自動的にリンクされます。コマンド行に BEA Tuxedo ライブラリを指定する必要はありません。

リンクするライブラリ・ファイルの指定順序は重要です。関数を呼び出す順序と、それらの関数への参照を含むライブラリによって、この順序が決定されます。

デフォルトでは、buildclient コマンドは UNIX の cc コマンドを呼び出します。環境変数 ALTCC を指定して別のコンパイル・コマンドを指定したり、ALTCFLAGS を指定してコンパイル・フェーズやリンク・フェーズにフラグを設定することができます。デフォルトでは、ALTCC は cobcc に設定されます。詳細については、2-7 ページの「環境変数の設定」を参照してください。

注記　Windows 2000 システムでは、環境変数 ALTCC と ALTCFLAGS は使用できません。これらを設定すると、予想しない結果が生じます。アプリケーションをコンパイルするには、最初に COBOL コンパイラを使用し、次に生成されたオブジェクト・ファイルを buildclient コマンドに渡す必要があります。次に例を示します。

```
buildclient -C -o audit -f audit.o
```

次のコマンド行の例では、COBOL プログラム audit.cbl をコンパイルして、実行可能ファイル audit を生成しています。

```
buildclient -C -o audit -f audit.cbl
```

関連項目

- 5-36 ページの「サーバのビルド」
- 『BEA Tuxedo コマンド・リファレンス』の buildclient(1)

クライアント・プロセスの例

次の擬似コードは、通常のクライアント・プロセスがアプリケーションに参加してから分離するまでの処理を示しています。

コード リスト 4-1　クライアント・プロセスのパラダイム

```
. . .
Check level of security
   CALL TPSETUNSOL to name your handler routine for TPU-DIP
   get USRNAME, CLTNAME
   prompt for application PASSWD
       SET TPU-DIP TO TRUE.
       CALL "TPINITIALIZE" USING TPINFDEF-REC
                    USER-DATA-REC
                    TPSTATUS-REC.
       IF NOT TPOK
     error processing
. . .
make service call
receive the reply
check for unsolicited messages
. . .
CALL "TPTERM" USING TPSTATUS-REC.
IF NOT TPOK
             error processing
.. .
EXIT PROGRAM.
```

この例では、TPINITIALIZE は 次の 3 つの引数を取ります。

- TPINFDEF-REC (COBOL の COPY ファイルに定義される構造体)
- ユーザ・データ (USER-DATA-REC)
- TPSTATUS-REC (COBOL の COPY ファイルに定義され、状態を格納する構造体)

TPINITIALIZE と TPTERM は、呼び出しが成功すると、TP-STATUS IN TPSTATUS-REC に [TPOK] を返します。いずれのコマンドでもエラーが検出されると、コマンドは失敗し、エラーの原因を示す値が TP-STATUS に設定されます。TPSTATUS-REC は、COBOL の COPY ファイルに定義されています。TP-STATUS の値については、11-1 ページの「エラーの管理」を参照してください。各 ATMI 呼び出しで返される全エラー・コードのリストについては、『BEA Tuxedo COBOL リファレンス』の「COBOL アプリケーション・トランザクション・モニタ・インターフェイスの紹介」を参照してください。

次の例は、TPINITIALIZE と TPTERM ルーチンの使用方法を示しています。このコード例は、BEA Tuxedo ソフトウェアに提供されている銀行業務のサンプル・アプリケーション bankapp から引用したものです。

コード リスト 4-2 アプリケーションへの参加と分離

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FIG1-3.
AUTHOR. TUXEDO DEVELOPMENT.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
*
WORKING-STORAGE SECTION.
*****************************************************
* Tuxedo の定義
*****************************************************
01 TPSTATUS-REC.
COPY TPSTATUS.
*
01 TPINFDEF-REC.
COPY TPINFDEF.
*****************************************************
* ログ・メッセージの定義
*****************************************************
01 LOGMSG.
     05 FILLER          PIC X(10) VALUE "FIG12-3 =>".
```

```
     05 LOGMSG-TEXT    PIC X(50).
01 LOGMSG-LEN          PIC S9(9) COMP-5.
*
01 USER-DATA-REC  PIC X(75).
******************************************************
PROCEDURE DIVISION.
START-HERE.
MOVE LENGTH OF LOGMSG TO LOGMSG-LEN.
*****************************************************
* システムへのクライアントの登録
*****************************************************
MOVE SPACES TO USRNAME.
MOVE SPACES TO CLTNAME.
MOVE SPACES TO PASSWD.
MOVE SPACES TO GRPNAME.
MOVE ZERO TO DATALEN.
SET TPU-DIP TO TRUE.
*
CALL "TPINITIALIZE" USING TPINFDEF-REC
         USER-DATA-REC
         TPSTATUS-REC.
IF NOT TPOK
        MOVE "TPINITIALIZE FAILED" TO LOGMSG-TEXT
        PERFORM DO-USERLOG
        PERFORM EXIT-PROGRAM.
*****************************************************
* アプリケーション固有のコード
*****************************************************
.. .
*****************************************************
* アプリケーションからの分離
*****************************************************
CALL "TPTERM" USING TPSTATUS-REC.
IF  NOT TPOK
    MOVE "TPTERM FAILED" TO LOGMSG-TEXT
    PERFORM DO-USERLOG.
EXIT-PROGRAM.
STOP RUN.
*****************************************************
* ユーザ・ログにメッセージを記録
*****************************************************
DO-USERLOG.
CALL "USERLOG" USING LOGMSG
      LOGMSG-LEN
      TPSTATUS-REC.
```

この例は、TPINITIALIZE を呼び出すことによって、アプリケーションに参加しているクライアント・プロセスを示しています。エラーが発生すると、USERLOG を呼び出して、中央イベント・ログにメッセージを書き込みます。

5 サーバのコーディング

ここでは、次の内容について説明します。

- BEA Tuxedo システムの制御プログラム
- システムで提供されるサーバおよびサービス
- サーバのコーディングのためのガイドライン
- サービスの定義
- サービス・ルーチンの終了
- サービスの宣言と宣言の取り消し
- サーバのビルド

BEA Tuxedo システムの制御プログラム

BEA Tuxedo システムには、サーバを簡単に開発できるように、サーバのロード・モジュール用に定義済み制御プログラムが提供されています。`buildserver -C` コマンドを実行すると、制御プログラムが自動的にサーバの一部として組み込まれます。

注記　制御プログラムはシステムで提供されたものなので、変更することはできません。

定義済み制御プログラムは、アプリケーションへの参加と終了のほかに、サーバに代わって次の操作を行います。

- ハングアップを無視してプロセスを実行します。つまり、SIGHUP シグナルを無視します。
- 標準オペレーティング・システム・ソフトウェアの終了シグナル (SIGTERM) を受信すると、終了処理を開始します。サーバはシャットダウンされ、必要な場合は再起動します。
- 掲示板サービスが参照できるように共用メモリを割り当てます。
- プロセスに対してメッセージ・キューを作成します。
- サーバによって提供される初期サービスを宣言します。初期サービスは、定義済み制御プログラムとリンクされたすべてのサービス、または BEA Tuxedo システムの管理者がコンフィギュレーション・ファイルに指定したサブセットです。
- コマンド行に入力された 2 つのダッシュ (--) までの引数を処理します。2 つのダッシュは、システムで認識される引数の終わりを示します。
- TPSVRINIT ルーチンを呼び出して、コマンド行で 2 つのダッシュ (--) の後に入力された引数を処理したり、リソース・マネージャをオープンします (省略可能)。このようなコマンド行の引数は、アプリケーション固有の初期化に使用されます。
- 中止が要求されるまで、要求キューにサービス要求メッセージがあるかどうかを確認します。
- サービス要求メッセージが要求キューに到着すると、中止が要求されるまで、main() は次の処理を行います。
  - -r オプションが指定されている場合、サービス要求の開始時間を記録します。
  - 掲示板を更新して、サーバが BUSY であることを示します。
  - サービスをディスパッチします。つまり、サービス・サブルーチンを呼び出します。

- サービスが入力に対する処理を終了して制御が戻ると、中止が要求されるまで、main() は次の処理を行います。
  - -r オプションが指定されている場合、サービス要求の終了時間を記録します。
  - 統計を更新します。
  - 掲示板を更新して、サーバが IDLE 状態であること、つまりサーバの準備ができたことを示します。
  - キューに次のサービス要求があるかどうかを確認します。
- サーバの中止が要求されると、TPSVRDONE を呼び出して必要なシャットダウン操作を行います。

以上からわかるように、main() ルーチンは、アプリケーションへの参加と終了、レコードやトランザクションの管理、および通信に関する詳細を扱っています。

注記　システムで提供される制御プログラムは、アプリケーションへの参加と終了を行います。そのため、TPINITIALIZE または TPTERM ルーチンへの呼び出しをコードに記述しないでください。これらのルーチンを呼び出すとエラーが発生して、TP-STATUS に TPEPROTO が返されます。TPINITIALIZE または TPTERM ルーチンの詳細については、4-1 ページの「クライアントのコーディング」を参照してください。

システムで提供されるサーバおよびサービス

制御プログラムは、システムで提供される 1 つのサーバ AUTHSVR、および 2 つのサブルーチン TPSVRINIT と TPSVRDONE を提供します。これらの 3 つのデフォルト・バージョンについては、以降の節で説明します。これらのサーバとサブルーチンは、ご使用のアプリケーションに合わせて変更することができます。

注記　TPSVRINIT と TPSVRDONE を独自にコーディングする場合、この 2 つのルーチンのデフォルト・バージョンが、それぞれ tx_open() と tx_close() を呼び出すことに注意してください。tx_open() ではなく tpopen() を呼び出す TPSVRINIT の新しいバージョンをコーディングする場合は、tpclose() を呼び出す TPSVRDONE もコーディングする必要があります。つまり、この 2 つのルーチンが呼び出すオープンとクローズのルーチンは対になっていなければなりません。

システムで提供されるサーバ : AUTHSVR( )

AUTHSVR(5) を使用すると、アプリケーションで各クライアントの認証を行うことができます。このサーバは、アプリケーションのセキュリティ・レベルが TPAPPAUTH、USER_AUTH、ACL、または MANDATORY_ACL に設定されている場合に、TPINITIALIZE ルーチンによって呼び出されます。

AUTHSVR のサービスは、USER-DATA-REC レコードでユーザ・パスワードを確認します。このユーザ・パスワードは、TPINFDEF-REC レコードの PASSWD フィールドにあるアプリケーション・パスワードとは異なります。デフォルトでは、システムによって data から文字列が取得され、それと合致する文字列が /etc/passwd ファイルで検索されます。

TPINITIALIZE がネイティブ・サイトのクライアントから呼び出された場合、受信した USER-DATA-REC レコードはそのまま転送されます。そのため、アプリケーションでパスワードの暗号化が必要な場合、それに応じてクライアント・プログラムをコーディングする必要があります。

TPINITIALIZE がワークステーション・クライアントから呼び出された場合、データは暗号化されてからネットワークに送信されます。

システムで提供されるサービス : TPSVRINIT ルーチン

サーバの起動時に、BEA Tuxedo システムの制御プログラムはその初期化時、つまりサービス要求の処理を開始する前に、TPSVRINIT(3cbl) を呼び出します。

アプリケーションがこのルーチンのカスタム・バージョンをサーバに提供していない場合、制御プログラムで提供されるデフォルトのルーチンが呼び出されます。このルーチンは、リソース・マネージャをオープンし、エントリを中央イベント・ログに記録してサーバの起動が成功したことを示します。中央ユーザ・ログは自動的に生成されるファイルで、USERLOG(3cbl) ルーチンを呼び出して、プロセスがこのファイルにメッセージを書き込みます。中央イベント・ログの詳細については、11-1 ページの「エラーの管理」を参照してください。

TPSVRINIT ルーチンを使用すると、アプリケーションで要求される次のような初期化プロセスを行うことができます。

- コマンド行オプションの取得
- データベースのオープン

以降の節では、TPSVRINIT を呼び出すことによって、これらの初期化プロセスがどのように行われるのかをコード例で示します。以下のコード例では示していませんが、このルーチン内ではメッセージ交換を行うこともできます。ただし、非同期応答が未処理のままで TPSVRINIT ルーチンが制御を戻すと、このルーチンは失敗します。その場合、BEA Tuxedo システムでは応答は無視されて、サーバが正常に終了します。

また、TPSVRINIT ルーチンで、トランザクションを開始したり終了することができます。11-1 ページの「エラーの管理」を参照してください。

TPSVRINIT ルーチンの呼び出しには、次の文法を使用します。

```
LINKAGE SECTION.
01 CMD-LINE.
   05 ARGC      PIC 9(4) COMP-5.
   05 ARGV.
      10 ARGS PIC X OCCURS 0 TO 9999 DEPENDING ON ARGC.
01 TPSTATUS-REC.
   COPY TPSTATUS.
PROCEDURE DIVISION USING CMD-LINE TPSTATUS-REC.
* ユーザ・コード
EXIT PROGRAM.
```

コマンド行オプションの取得

サーバは、起動時に最初の処理として、コンフィグレーション・ファイルに指定されているサーバ・オプションを読み取ります。これらのオプションは、引数の個数を格納する ARGC と、1 つの SPACE 文字で区切られた引数を格納する ARGV を使用して渡されます。その後、定義済み制御プログラムが TPSVRINIT を呼び出します。

次のコード例は、TPSVRINIT ルーチンでコマンド行オプションを取得する方法を示しています。

コード リスト 5-1 TPSVRINIT を使用したコマンド行オプションの取得

```
  IDENTIFICATION DIVISION.
  PROGRAM-ID. TPSVRINIT.
  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SOURCE-COMPUTER. USL-486.
  OBJECT-COMPUTER. USL-486.
*
  DATA DIVISION.
  WORKING-STORAGE SECTION.
*
  LINKAGE SECTION.
*
  01 CMD-LINE.
      05 ARGC PIC 9(4) COMP-5.
      05 ARGV.
           10 ARGS PIC X OCCURS 0 TO 9999 DEPENDING ON ARGC.
  01 SERVER-INIT-STATUS.
```

```
  COPY TPSTATUS.
*
 PROCEDURE DIVISION USING CMD-LINE SERVER-INIT-STATUS.
**********************************************************
* ARGC は引数の個数を示し
* ARGV は 1 つの SPACE で区切られた引数を格納します。
**********************************************************
  A-START.
*
   . . . INSPECT the ARGV line and process arguments
   IF arguments are invalid
         SET TPEINVAL IN SERVER-INIT-STATUS TO TRUE.
   ELSE arguments are OK continue
         SET TPOK IN SERVER-INIT-STATUS TO TRUE.
*
   EXIT PROGRAM.
```

リソース・マネージャのオープン

TPSVRINIT のもう 1 つの使用法として、リソース・マネージャをオープンすることができます。以下のコード例は、その方法を示しています。■訳文不要■ BEA Tuxedo システムには、リソース・マネージャをオープンするルーチンとして、TPOPEN(3cbl) と TXOPEN(3cbl) があります。また、これらと対のルーチンとして、TPCLOSE(3cbl) と TXCLOSE(3cbl) があります。これらのルーチンを呼び出してリソース・マネージャをオープンしたりクローズするアプリケーションは、その意味では移植性があります。これらのアプリケーションは、コンフィギュレーション・ファイルに設定されたリソース・マネージャのインスタンス固有の情報にアクセスすることによって動作します。

これらのルーチン呼び出しは省略可能です。また、リソース・マネージャがデータベースの場合、リソース・マネージャ固有の呼び出しがデータ操作言語 (DML) の一部であるときは、その呼び出しの代わりに使用できます。USERLOG(3cbl) ルーチンを使用して、中央イベント・ログに書き込んでいることに注目してください。

注記　コマンド行オプションを受け取り、データベースをオープンする初期化関数を作成するには、次のコード例と前述のコード例を組み合わせます。

コード リスト 5-2　TPSVRINIT **を使用したリソース・マネージャのオープン**

```
  IDENTIFICATION DIVISION.
  PROGRAM-ID. TPSVRINIT.
  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SOURCE-COMPUTER. USL-486.
  OBJECT-COMPUTER. USL-486.
*
  DATA DIVISION.
  WORKING-STORAGE SECTION.
    01 TPSTATUS-REC.
      COPY TPSTATUS.
    01 LOGMSG         PIC X(50).
    01 LOGMSG-LEN    PIC S9(9) COMP-5.
*
  LINKAGE SECTION.
    01 CMD-LINE.
      05 ARGC PIC 9(4) COMP-5.
      05 ARGV.
           10 ARGS PIC X OCCURS 0 TO 9999 DEPENDING ON ARGC.
    01 SERVER-INIT-STATUS.
     COPY TPSTATUS.
*
  PROCEDURE DIVISION USING CMD-LINE SERVER-INIT-STATUS.
  A-START.
    . . . INSPECT the ARGV line and process arguments
    IF arguments are invalid
               MOVE "Invalid Arguments Passed" TO LOGMSG
               PERFORM EXIT-NOW.
    ELSE arguments are OK continue

    CALL "TPOPEN" USING TPSTATUS-REC.
    IF NOT TPOK
          MOVE "TPOPEN Failed" TO LOGMSG
    ELSE IF TPESYSTEM
          MOVE "System /T error has occurred" TO LOGMSG
    ELSE IF TPEOS
          MOVE "An Operating System error has occurred" TO LOGMSG
    ELSE IF TPEPROTO
         MOVE "TPOPEN was called in an improper Context" TO LOGMSG
    ELSE IF TPERMERR
          MOVE "Resource manager Failed to Open" TO LOGMSG
          PERFORM EXIT-NOW.
    SET TPOK IN SERVER-INIT-STATUS TO TRUE.
    EXIT PROGRAM.
 EXIT-NOW.
  SET TPEINVAL IN SERVER-INIT-STATUS TO TRUE
```

```
    MOVE 50 LOGMSG-LEN.
    CALL "USERLOG" USING LOGMSG
                  LOGMSG-LEN
                  TPSTATUS-REC.
EXIT PROGRAM.
```

初期化時にエラーが発生しないように、サーバを終了してからサービス要求の処理を開始するように TPSVRINIT をコーディングできます。

システムで提供されるサービス : TPSVRDONE ルーチン

TPSVRINIT が TPOPEN を呼び出してリソース・マネージャをオープンするのと同じように、TPSVRDONE ルーチンは TPCLOSE を呼び出してリソース・マネージャをクローズします。

TPSVRDONE ルーチンの呼び出しには、次の文法を使用します。

```
  01 TPSTATUS-REC.
     COPY TPSTATUS.
  PROCEDURE DIVISION.
* ユーザ・コード
  EXIT PROGRAM.
```

次のコード例は、TPSVRDONE ルーチンを使用して、リソース・マネージャをクローズして終了する方法を示しています。

コード リスト 5-3 TPSVRDONE **を使用したリソース・マネージャのクローズ**

```
  IDENTIFICATION DIVISION.
  PROGRAM-ID. TPSVRDONE.
  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SOURCE-COMPUTER. USL-486.
  OBJECT-COMPUTER. USL-486.
*
  DATA DIVISION.
```

```
  WORKING-STORAGE SECTION.
   01 TPSTATUS-REC.
      COPY TPSTATUS.
   01 LOGMSG                PIC X(50).
   01 LOGMSG-LEN            PIC S9(9) COMP-5.
   01 SERVER-DONE-STATUS.
      COPY TPSTATUS.
  PROCEDURE DIVISION.
  A-START.
   CALL "TPCLOSE" USING TPSTATUS-REC.
   IF NOT TPOK
          MOVE "TPCLOSE Failed" TO LOGMSG
   ELSE IF TPESYSTEM
          MOVE "System /T error has occurred" TO LOGMSG
   ELSE IF TPEOS
          MOVE "An Operating System error has occurred" TO LOGMSG
   ELSE IF TPEPROTO
         MOVE "TPCLOSE was called in an improper Context" TO LOGMSG
   ELSE IF TPERMERR
          MOVE "Resource manager Failed to Open" TO LOGMSG
          PERFORM EXIT-NOW.
   SET TPOK IN SERVER-DONE-STATUS TO TRUE.
   EXIT PROGRAM.
  EXIT-NOW.
   SET TPEINVAL IN SERVER-DONE-STATUS TO TRUE
   MOVE 50 LOGMSG-LEN.
   CALL "USERLOG" USING LOGMSG
                  LOGMSG-LEN
                  TPSTATUS-REC.
EXIT PROGRAM.
```

サーバのコーディングのためのガイドライン

通信の詳細は BEA Tuxedo システムの制御プログラムによって処理されるので、プログラマは通信のインプリメントよりもアプリケーション・サービスのロジックに集中できます。ただし、システムで提供される制御プログラムと互換性を保つために、アプリケーション・サービスが特定の規約に従って

いる必要があります。これらの規約は、サービス・ルーチンをコーディングするためのサービス・テンプレートと言えます。以下に、これらの規約についてまとめます。

- 要求 / 応答サービスが一度に受信できる要求の数は 1 つだけであり、送信できる応答も 1 つだけです。
- 要求 / 応答サービスが一度に処理する要求は 1 つだけです。別の要求を受け取ることができるのは、要求元に応答を送信した後、または別の処理を行うために別のサービスに要求を転送した後だけです。
- サービス・ルーチンを終了するには、TPRETURN または TPFORWAR ルーチンのいずれかを呼び出す必要があります。
- TPACALL を使用して別のサーバと通信する場合は、サービスの開始元は未処理の応答がすべて処理されるまで待機するか、または TPCANCEL を使用して未処理の応答をすべて無効にしてから、TPRETURN または TPFORWAR を呼び出す必要があります。

サービスの定義

サービス・ルーチンをコーディングする場合、最初に TPSVCSTART(3cbl) ルーチンを呼び出す必要があります。このルーチンは、サービスのパラメータとデータを取得します。TPSVCSTART ルーチンの呼び出しには、次の文法を使用します。

```
01 TPSVCDEF-REC.
   COPY TPSVCDEF.
01 TPTYPE-REC.
   COPY TPTYPE.
01 DATA-REC.
   COPY User Data.
01 TPSTATUS-REC.
   COPY TPSTATUS.
CALL "TPSVCSTART" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

サービス情報のデータ構造体は、COBOL の COPY ファイルに TPSVCDEF として定義されます。このデータ構造体のメンバは、次のとおりです。

```
05 COMM-HANDLE          PIC S9(9) COMP-5.
05 TPBLOCK-FLAG         PIC S9(9) COMP-5.
       88 TPBLOCK       VALUE 0.
       88 TPNOBLOCK     VALUE 1.
05 TPTRAN-FLAG          PIC S9(9) COMP-5.
       88 TPTRAN        VALUE 0.
       88 TPNOTRAN      VALUE 1.
05 TPREPLY-FLAG         PIC S9(9) COMP-5.
       88 TPREPLY       VALUE 0.
       88 TPNOREPLY     VALUE 1.
05 TPACK-FLAG         PIC S9(9) COMP-5 REDEFINES TPREPLY-FLAG.
       88 TPNOACK        VALUE 0.
       88 TPACK          VALUE 1.
05 TPTIME-FLAG          PIC S9(9) COMP-5.
       88 TPTIME        VALUE 0.
       88 TPNOTIME      VALUE 1.
05 TPSIGRSTRT-FLAG      PIC S9(9) COMP-5.
       88 TPNOSIGRSTRT   VALUE 0.
       88 TPSIGRSTRT     VALUE 1.
05 TPGETANY-FLAG        PIC S9(9) COMP-5.
       88 TPGETHANDLE   VALUE 0.
       88 TPGETANY      VALUE 1.
05 TPSENDRECV-FLAG      PIC S9(9) COMP-5.
       88 TPSENDONLY     VALUE 0.
```

```
        88 TPRECVONLY     VALUE 1.
 05 TPNOCHANGE-FLAG      PIC S9(9) COMP-5.
        88 TPCHANGE      VALUE 0.
        88 TPNOCHANGE    VALUE 1.
 05 TPSERVICETYPE-FLAG   PIC S9(9) COMP-5.
        88 TPREQRSP      VALUE 0.
        88 TPCONV        VALUE 1.
*
 05 APPKEY               PIC S9(9) COMP-5.
 05 CLIENTID OCCURS 4 TIMES PIC S9(9) COMP-5.
 05 SERVICE-NAME         PIC X(15).
```

次の表は、TPSVCDEF データ構造体のメンバを示しています。

表 5-1TPSVCDEF データ構造体

フィールド	説明
COMM-HANDLE	要求元プロセスがサービスの呼び出しに使用した通信ハンドル。この通信ハンドルは、サービス・ルーチンに対して指定されます。
SETTINGS (TPBLOCK-FLAG TPTRAN-FLAG など)	サーバの特性を制御するその他の設定。設定値の詳細については、『BEA Tuxedo COBOL リファレンス』を参照してください。
APPKEY	アプリケーションで使用するために予約された値。アプリケーション固有の認証が設計に含まれている場合、認証サーバによって認証の成功または失敗を示す値、およびクライアント認証キーが返される必要があります。認証サーバは、クライアントがアプリケーションに参加するときに呼び出されます。BEA Tuxedo システムは、APPKEY をクライアントのために保持し、このフィールドに格納して以降のサービス要求に渡します。APPKEY がサービスに渡されたときは、クライアントの認証は終了しています。ただし、サービス内で APPKEY フィールドを使用して、サービスを呼び出したユーザ、またはそのユーザに関するそのほかのパラメータを識別できます。
CLIENTID	要求元クライアントの識別子。
SERVICE-NAME	要求元プロセスがサービスの呼び出しに使用したサービス・ルーチンの名前。

TPTYPE-REC データ構造体については、3-7 ページの「型付きレコードの定義」を参照してください。

DATA-REC に格納される要求データにサービスがアクセスする場合は、コンフィギュレーション・ファイルでそのサービスに定義されているレコード・タイプにデータが格納されるようにコーディングする必要があります。制御が正常に戻った場合、DATA-REC には受け取ったデータが格納され、LEN には移動した実際のバイト数が格納されます。

次のコード例は、一般的なサービス定義を示しています。

コード リスト 5-4　一般的なサービス定義

```
    IDENTIFICATION DIVISION.
    PROGRAM-ID. BUYSR.
    AUTHOR. TUXEDO DEVELOPMENT.
    ENVIRONMENT DIVISION.
    CONFIGURATION SECTION.
    SOURCE-COMPUTER. USL-486.
    OBJECT-COMPUTER. USL-486.
*
    INPUT-OUTPUT SECTION.
. . .
******************************************************
* Tuxedo の定義
******************************************************
    01 TPSVCRET-REC.
    COPY TPSVCRET.
*
    01 TPTYPE-REC.
     COPY TPTYPE.
*
    01 TPSTATUS-REC.
    COPY TPSTATUS.
*
    01 TPSVCDEF-REC.
    COPY TPSVCDEF.
******************************************************
* ログ・メッセージの定義
******************************************************
    01 LOGMSG.
            05 LOGMSG-TEXT        PIC X(50).
*
    01 LOGMSG-LEN                 PIC S9(9) COMP-5.
******************************************************
```

```
* ユーザ定義のデータ・レコード
******************************************************
    01 CUST-REC.
    COPY CUST.
*
    LINKAGE SECTION.
*
    PROCEDURE DIVISION.
*
 START-BUYSR.
    MOVE LENGTH OF LOGMSG TO LOGMSG-LEN.
    OPEN files or DATABASE
******************************************************
* クライアントから送信されたデータの取得
******************************************************
    MOVE "Server Started" TO LOGMSG-TEXT.
    PERFORM DO-USERLOG.
    MOVE LENGTH OF CUST-REC TO LEN IN TPTYPE-REC.
    CALL "TPSVCSTART" USING TPSVCDEF-REC
                      TPTYPE-REC
                      CUST-REC
                      TPSTATUS-REC.
    IF TPTRUNCATE
        MOVE "Input data exceeded CUST-REC length" TO LOGMSG-TEXT
        PERFORM DO-USERLOG
        PERFORM A-999-EXIT.
    IF NOT TPOK
        MOVE "TPSVCSTART Failed" TO LOGMSG-TEXT
        PERFORM DO-USERLOG
        PERFORM A-999-EXIT.
    IF REC-TYPE NOT = "VIEW"
        MOVE "REC-TYPE in not VIEW" TO LOGMSG-TEXT
        PERFORM DO-USERLOG
        PERFORM A-999-EXIT.
    IF SUB-TYPE NOT = "cust"
        MOVE "SUB-TYPE in not cust" TO LOGMSG-TEXT
        PERFORM DO-USERLOG
        PERFORM A-999-EXIT.
     . . .
    set consistency level of the transaction
     .. .
******************************************************
* 終了
******************************************************
  A-999-EXIT.
     MOVE "Exiting" TO LOGMSG-TEXT.
     PERFORM DO-USERLOG.
     SET TPFAIL TO TRUE.
```

```
          COPY TPRETURN REPLACING TPSVCRET-REC BY TPSVCRET-REC
          TPTYPE-REC BY TPTYPE-REC
          DATA-REC BY CUST-REC
          TPSTATUS-REC BY TPSTATUS-REC.
******************************************************
* ユーザ・ログへの書き込み

******************************************************
   DO-USERLOG.
          CALL "USERLOG" USING LOGMSG
                   LOGMSG-LEN
                   TPSTATUS-REC.
```

この例では、クライアント側の要求レコードが最初に送られたときに、REC-TYPE に VIEW、SUB-TYPE に cust が設定されています。BUYSR サービスは、VIEW 型レコードを認識するサービスとしてコンフィギュレーション・ファイルに定義されています。BUYSR サービスは、CUST-REC レコードにアクセスしてデータ・レコードを取得します。このレコードが取得された後、データベースへの最初のアクセスを行う前に、トランザクションの整合性レベルが指定されています。トランザクションの整合性レベルの詳細については、9-1 ページの「グローバル・トランザクションのコーディング」を参照してください。

注記　優先順位を取得する TPGPRIO、および優先順位を設定する TPSPRIO 関数の詳細については、6-17 ページの「メッセージの優先順位の設定および取得」を参照してください。

ここで示すコード例は、PRINTER サービスが TPGPRIO ルーチンを使用して、受信したばかりの要求の優先順位を確認する方法を示しています。その優先順位に基づいて、印刷ジョブが適切なプリンタ RNAME に送られます。

その後、INPUT-REC の内容がプリンタに送られます。アプリケーションは TPSVCDEF-REC を照会し、応答が必要かどうかを判定します。応答が必要な場合は、プリンタ名をクライアントに返します。TPRETURN ルーチンの詳細については、5-20 ページの「サービス・ルーチンの終了」を参照してください。

コード リスト 5-5　受信した要求の優先順位の確認

```
    IDENTIFICATION DIVISION.
    PROGRAM-ID. PRINTSR.
    AUTHOR. TUXEDO DEVELOPMENT.
    ENVIRONMENT DIVISION.
    CONFIGURATION SECTION.
    SOURCE-COMPUTER. USL-486.
    OBJECT-COMPUTER. USL-486.
*
    INPUT-OUTPUT SECTION.
. . .
******************************************************
* Tuxedo の定義
******************************************************
    01 TPSVCRET-REC.
    COPY TPSVCRET.
*
    01 TPTYPE-REC.
    COPY TPTYPE.
*
    01 TPSTATUS-REC.
    COPY TPSTATUS.
*
    01 TPSVCDEF-REC.
    COPY TPSVCDEF.
*
    01 TPPRIDEF-REC.
    COPY TPPRIDEF.
******************************************************
* ログ・メッセージの定義
******************************************************
    01 LOGMSG.
            05 FILLER           PIC S9(9) VALUE
                    "TP-STATUS=".
            05 LOG-TP-STATUS    PIC S9(9).
            05 LOGMSG-TEXT      PIC X(50).
*
    01 LOGMSG-LEN  PIC S9(9) COMP-5.
******************************************************
* ユーザ定義のデータ・レコード
******************************************************
    01 INPUT-REC           PIC X(1000).
    01 PRNAME              PIC X(20).
*
    LINKAGE SECTION.
*
```

```
    PROCEDURE DIVISION.
*
 START-PRINTSR.
    MOVE LENGTH OF LOGMSG TO LOGMSG-LEN.
    OPEN files or DATABASE
******************************************************
* クライアントから送信されたデータの取得
******************************************************
    MOVE ZERO to TP-STATUS.
    MOVE "Server Started" TO LOGMSG-TEXT.
    PERFORM DO-USERLOG.
    MOVE LENGTH OF INPUT-REC TO LEN.
    CALL "TPSVCSTART" USING TPSVCDEF-REC
                      TPTYPE-REC
                      INPUT-REC


                      TPSTATUS-REC.
    IF NOT TPOK
             MOVE "TPSVCSTART Failed" TO LOGMSG-TEXT
             PERFORM DO-USERLOG
             SET TPFAIL TO TRUE.
             PERFORM A-999-EXIT.
    . . .
   Check other parameters
    CALL "TPGPRIO" USING TPPRIDEF-REC
                   TPSTATUS-REC.
    IF NOT TPOK
             MOVE "TPGPRIO Failed" TO LOGMSG-TEXT
             PERFORM DO-USERLOG
             SET TPFAIL TO TRUE.
             PERFORM A-999-EXIT.
    IF PRIORITY < 20
             MOVE "BIGJOBS" TO RNAME
    ELSE IF PRIORITY < 60
             MOVE "MEDJOBS" TO RNAME
    ELSE
             MOVE "HIGHSPEED" TO RNAME.
    . . .
   Print INPUT-REC on RNAME printer
    . . .
    IF TPNOREPLY
             MOVE SPACES TO REC-TYPE
             MOVE 0 TO LEN
             SET TPSUCCESS TO TRUE
             PERFORM A-999-EXIT
    IF TPREPLY
             MOVE "STRING" TO REC-TYPE
             MOVE LENGTH OF PRNAME TO LEN
```

```
             SET TPSUCCESS TO TRUE
             PERFORM A-999-EXIT.
******************************************************
* 終了
******************************************************
 A-999-EXIT.
     MOVE "Exiting" TO LOGMSG-TEXT.
     PERFORM DO-USERLOG.
     SET TPSUCCESS TO TRUE.
     COPY TPRETURN REPLACING TPSVCRET-REC BY TPSVCRET-REC
              TPTYPE-REC buTPTYPE-REC
              DATA-REC BY PRNAME
              TPSTATUS-REC BY TPSTATUS-REC.
******************************************************
* ユーザ・ログへの書き込み
******************************************************
 DO-USERLOG.


     MOVE TP-STATUS TO LOG-TP-STATUS.
     CALL "USERLOG" USING LOGMSG
               LOGMSG-LEN
               TPSTATUS-REC.
```

サービス・ルーチンの終了

TPRETURN(3cbl)、TPCANCEL(3cbl)、および TPFORWAR(3cbl) ルーチンは、サービス・ルーチンが完了したことをそれぞれ次の方法で通知します。

- TPRETURN は、呼び出し元クライアントに応答を送信します。
- TPCANCEL は、現在の要求を取り消します。
- TPFORWAR は、別の処理を行うために別のサービスにサービス要求を転送します。

応答の送信

TPRETURN(3cbl) および TPFORWAR(3cbl) 呼び出しは、サービス・ルーチンの終了を示す EXIT 文を含んだ COBOL の copy ファイルです。これらの呼び出しは、それぞれ要求元にメッセージを送信するか、または別のサービスに要求を転送します。TPRETURN ルーチンの呼び出しには、次の文法を使用します。

```
01 TPSVCRET-REC.
   COPY TPSVCRET.
01 TPTYPE-REC.
   COPY TPTYPE.
01 DATA-REC.
   COPY User Data.
01 TPSTATUS-REC.
   COPY TPSTATUS.
COPY TPRETURN REPLACING TPSVCRET-REC BY TPSVCRET-REC
              TPTYPE-REC BY TPTYPE-REC
              DATA-REC BY DATA-REC
               TPSTATUS-REC BY TPSTATUS-REC.
```

注記　ここでは、CALL ではなく COPY を使用する必要があります。これにより、EXIT 文が正しく呼び出され、COBOL サービス・ルーチンが BEA Tuxedo システムに制御を戻します。

次のコードは、TPSVCRET-REC レコードの文法を示しています。

```
05 TPRETURN-VAL   PIC S9(9) COMP-5.
   88 TPSUCCESS      VALUE 0.
   88 TPFAIL         VALUE 1.
   88 TPFAIL         VALUE 2.
05 APPL-CODE     PIC S9(9) COMP-5.
```

次の表は、TPSVCRET-REC データ構造体のメンバを示しています。

表 5-2TPSVCRET-REC データ構造体のメンバ

メンバ	説明
TP-RETURN-VAL	サービスが正常に終了したかどうかをアプリケーション・レベルで示す値。この値は、シンボリック名で表される整数値です。有効な設定は、次のとおりです。 ■ TPSUCCESS ―ルーチン呼び出しが成功したことを示します。ルーチンは、応答メッセージを呼び出し元のレコードに格納します。つまり、応答メッセージがある場合は、呼び出し元のレコード内にあります。 ■ TPFAIL(デフォルト)―サービスが失敗したことを示します。ルーチンは、応答を待つクライアント・プロセスにエラー・メッセージを通知します。その場合、クライアントが呼び出した TPCALL または TPGETRPLY ルーチンが失敗し、変数 TP-STATUS にアプリケーション定義の失敗を示す TPESVCFAIL が設定されます。応答メッセージが要求されている場合、呼び出し元のレコードから取得できます。 ■ TPEXIT ―サービスが失敗したことを示します。ルーチンは、応答を待つクライアント・プロセスにエラー・メッセージを通知して終了します。 この引数の値がグローバル・トランザクションに与える影響については、9-1 ページの「グローバル・トランザクションのコーディング」を参照してください。
APPLC-CODE	アプリケーション定義の戻りコードを呼び出し元に返します。クライアントは、APPL-RETURN-CODE IN TPSTATUS-REC を照会して、APPLC-CODE に返された値にアクセスできます。成功または失敗に関係なく、このコードはルーチンから返されます。

TPTYPE-REC レコードについては、5-12 ページの「サービスの定義」を参照してください。

サービス・ルーチンの主なタスクは、要求を処理してクライアント・プロセスに応答を返すことです。ただし、要求されたタスクを行うために必要なすべての処理を 1 つのサービスで行う必要はありません。サービスは要求元として動作し、クライアントが元の要求を行ったときと同じように、TPCALL または TPACALL を呼び出して、要求呼び出しを別のサービスに渡すことができます。

注記　TPCALL および TPACALL ルーチンの詳細については、6-1 ページの「クライアントおよびサーバへの要求 / 応答のコーディング」を参照してください。

TPRETURN が呼び出された場合、常に制御プログラムに制御が戻ります。非同期応答でサービスが要求を送信している場合、制御プログラムに制御を戻す前にすべての応答を受信するか、または TPCANCEL を使用して既に送信した要求を無効にする必要があります。それ以外の場合、未処理の応答は BEA Tuxedo システムの制御プログラムで受信されると自動的に破棄され、呼び出し元にエラーが返されます。

クライアントが TPCALL を使用してサービスを呼び出した場合、TPRETURN の呼び出しが成功すると、O-DATA-REC レコードから応答メッセージを取得できます。TPACALL を使用して要求を送信し、TPRETURN から正常に制御が戻されると、TPGETRPLY の DATA-REC レコードに応答メッセージが格納されます。

応答が必要な場合に、TPRETURN の引数の処理時にエラーが発生すると、呼び出し元プロセスに失敗を示すメッセージが送信されます。呼び出し元は、TP-STATUS に格納されている値を調べてエラーを検出します。失敗を示すメッセージが送信された場合、TP-STATUS に TPESVCERR が設定されます。この値は、APPL-RETURN-CODE IN TPSTATUS-REC の値よりも優先されます。このようなエラーが発生した場合、応答データは返されず、呼び出し元の出力レコードの内容と長さは変更されません。

TPRETURN が不明なタイプのレコードにメッセージを返すか、または呼び出し元で使用できないレコードにメッセージを返した場合、つまり TPNOCHANGE が設定されて呼び出しが行われた場合、TP-STATUS に TPEOTYPE が返されます。その場合、アプリケーションの成功また失敗は判定されず、呼び出し元の出力レコードの内容と長さは変更されません。

TPRETURN ルーチンが呼び出され、呼び出し元が応答を待っている間にタイムアウトが発生した場合、APPL-RETURN-CODE IN TPSTATUS-REC に返される値は意味を持ちません。この状況は、TP-STATUS に値が返されるどの状況よりも優先します。その場合、TP-STATUS に TPETIME が設定され、応答データは送信されず、呼び出し元の応答レコードの内容と長さは変更されません。BEA Tuxedo システムにはブロッキング・タイムアウトとトランザクション・タイムアウトの 2 種類のタイムアウトがあります。詳細については、9-1 ページの「グローバル・トランザクションのコーディング」を参照してください。

ここで示すコード例は、XFER サーバの一部である TRANSFER サービスを示しています。基本的に、TRANSFER サービスは WITHDRAWAL および DEPOSIT サービスへの同期呼び出しを行います。このサービスでは、WITHDRAWAL と DEPOSIT の両サービスの呼び出しに同じ要求レコードを使用する必要があるので、応答メッセージ用に別のレコードが割り当てられます。WITHDRAWAL の呼び出しが失敗した場合、フォーム上のステータス行に「cannot withdraw」というメッセージが出力され、TPRETURN ルーチンの TP-RETURN-VAL IN TPSVCRET-REC に TPFAIL が設定されます。呼び出しが成功した場合、振替元口座の残高が応答レコードから取得されます。

注記　次のコード例では、フィールド化レコード transf 内の ACCOUNT_ID フィールドのゼロ番目のオカレンスに、アプリケーションが cr_id 変数から取得した「振替先口座」の識別子を移動しています。このような移動が必要なのは、FML レコード内のフィールドのこのオカレンスが、データ依存型ルーティングに使用されるからです。詳細については、『BEA Tuxedo アプリケーションの設定』を参照してください。

withdrawal サービス呼び出しのシナリオは、DEPOSIT サービスへの呼び出しにも適用できます。呼び出しが成功すると、このサービスによって TP-RETURN-VAL IN TPSVCRET-REC に TPSUCCESS が設定されて、適切な残高情報がステータス行に返されます。

コード リスト 5-6 TPRETURN ルーチン

```
   IDENTIFICATION DIVISION.
   PROGRAM-ID. TRANSFER.
   AUTHOR. TUXEDO DEVELOPMENT.
   ENVIRONMENT DIVISION.
   CONFIGURATION SECTION.
   SOURCE-COMPUTER. USL-486.
   OBJECT-COMPUTER. USL-486.
*
   INPUT-OUTPUT SECTION.
. . .
******************************************************
* Tuxedo の定義
******************************************************
   01  TPSVCRET-REC.
   COPY TPSVCRET.
*
   01  TPTYPE-REC.
   COPY TPTYPE.
*
   01 TPSTATUS-REC.
   COPY TPSTATUS.
*
   01  TPSVCDEF-REC.
   COPY TPSVCDEF.
******************************************************
* ユーザ定義のデータ・レコード
******************************************************
   01 TRANS-REC.
      COPY TRANS-AMOUNT.
*
    LINKAGE SECTION.
*
    PROCEDURE DIVISION.
*
   START-TRANSFER.
******************************************************
* クライアントから送信されたデータの取得
******************************************************
    MOVE LENGTH OF TRANS-REC TO LEN.
    CALL "TPSVCSTART" USING TPSVCDEF-REC
                      TPTYPE-REC
                      TRANS-REC
                      TPSTATUS-REC.
    IF NOT TPOK
             MOVE "Transaction Encountered An Error" TO STATUS-LINE
```

```
            SET TPFAIL TO TRUE.
            COPY TPRETURN REPLACING TPSVCRET-REC BY TPSVCRET-REC
                     TPTYPE-REC BY TPTYPE-REC
                     DATA-REC BY TRANS-REC
                     TPSTATUS-REC BY TPSTATUS-REC.
    ELSE
            . . . Check other parameters
******************************************************
* 振替元と振替先の口座番号が有効かどうかを確認します。
******************************************************
    CALL "FIND-ACCOUNT-FUNCTION" USING TRANS-DEBIT-ACCOUNT IN  TRANS-REC.

    IF TRANS-DEBIT-ACCOUNT is not valid
            MOVE "Invalid Debit Account Number"
                     TO STATUS-LINE IN TRANS-REC
            SET TPFAIL TO TRUE
            COPY TPRETURN REPLACING
                     DATA-REC BY TRANS-REC.

    CALL "FIND-ACCOUNT-FUNCTION" USING TRANS-CREDIT-ACCOUNT IN TRANS-REC.

    IF TRANS-CREDIT-ACCOUNT  is not valid
            MOVE "Invalid Credit Account Number"
            TO STATUS-LINE IN TRANS-REC
            SET TPFAIL TO TRUE
            COPY TPRETURN REPLACING
                      DATA-REC BY TRANS-REC.
******************************************************
*  振替金額を確認します。
******************************************************
    IF TRANS-AMOUNT IN TRANS-REC < 0


            MOVE "Invalid Transfer Amount Requested"
                      TO STATUS-LINE IN TRANS-REC
            SET TPFAIL TO TRUE
            COPY TPRETURN REPLACING
                      DATA-REC BY TRANS-REC.
******************************************************
*  別のサービスを使用して Withdrawal を実行します。
******************************************************
    MOVE "WITHDRAWAL" TO SERVICE-NAME.
    . . . set other TPCALL parameters
    CALL "TPCALL" USING . . .
    IF NOT TPOK
            MOVE "Cannot withdraw from debit account"
                      TO STATUS-LINE IN TRANS-REC
            SET TPFAIL TO TRUE
            COPY TPRETURN REPLACING
```

```
                       DATA-REC BY TRANS-REC.
******************************************************
*  別のサービスを使用して Deposit を実行します。
******************************************************
    MOVE "DEPOSIT" TO SERVICE-NAME.
    . . . set other TPCALL parameters
    CALL "TPCALL" USING . . .
    IF NOT TPOK
             MOVE "Cannot Deposit into credit account"
                       TO STATUS-LINE IN TRANS-REC
             SET TPFAIL TO TRUE
             COPY TPRETURN REPLACING
                       DATA-REC BY TRANS-REC.
    . . .
    MOVE "Transfer completed" TO STATUS-LINE IN TRANS-REC
    . . . MOVE all the data into TRANS-REC needed by the client
    SET TPSUCCESS TO TRUE
    COPY TPRETURN REPLACING
                           DATA-REC BY TRANS-REC.
```

記述子の無効化

TPGETRPLY を呼び出したサービス (6-1 ページの「クライアントおよびサーバへの要求 / 応答のコーディング」を参照) が TPETIME で失敗して要求を取り消す場合、TPCANCEL(3cbl) を呼び出して記述子を無効にできます。以降、応答が届いても自動的に破棄されます。

TPCANCEL はトランザクション応答、つまり TPNOTRAN フラグが設定されていない状態で呼び出された要求への応答には使用できません。トランザクション内では、TPABORT(3cbl) がトランザクションの呼び出し記述子を無効にします。

次のコード例は、タイムアウト後の応答を無効にする方法を示しています。

コード リスト 5-7 タイムアウト後の応答の無効化

```
. . .  Set up parameters to TPACALL
SET TPNOTRAN TO TRUE.
CALL "TPACALL" USING TPSVCDEF-REC
                   TPTYPE-REC
```

```
                   DEBIT-REC
                   TPSTATUS-REC.
IF  NOT TPOK
       error processing
. . .
CALL "TPGETRPLY" USING TPSVCDEF-REC
                   TPTYPE-REC
                   DEBIT-REC
                   TPSTATUS-REC.
IF  NOT TPOK
     error processing
IF TPETIME
        CALL "TPCANCEL" TPSVCDEF-REC
                   TPSTATUS-REC.
    . . .
    SET TPSUCCESS TO TRUE.
    COPY TPRETURN REPLACING TPSVCRET-REC BY TPSVCRET-REC
                   TPTYPE-REC BY TPTYPE-REC
                   DATA-REC BY DEBIT-REC
                   TPSTATUS-REC BY TPSTATUS-REC.
```

要求の転送

TPFORWAR(3cbl) ルーチンを使用すると、サービス要求をほかのサービスに転送して、別の処理を行うことができます。

TPFORWAR ルーチンの呼び出しには、次の文法を使用します。

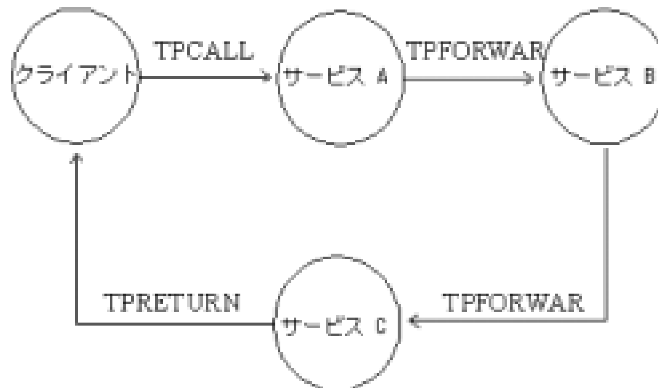
```
01 TPSVCDEF-REC.
   COPY TPSVCDEF.
01 TPTYPE-REC.
   COPY TPTYPE.
01 DATA-REC.
   COPY User Data.
01 TPSTATUS-REC.
   COPY TPSTATUS.
COPY TPFORWAR REPLACING TPSVCDEF-REC BY TPSVCDEF-REC
              TPTYPE-REC BY TPTYPE-REC
              DATA-REC BY DATA-REC
              TPSTATUS-REC BY TPSTATUS-REC.
```

TPSVCDEF-REC および TPTYPE-REC レコードについては、5-12 ページの「サービスの定義」を参照してください。

TPFORWAR は、サービス呼び出しとは異なります。つまり、要求の転送元サービスでは、応答は要求されていません。応答を返すのは、要求の転送先サービスです。このサービスを転送されたサービスが、要求の発信元プロセスに応答を返します。転送が連鎖的に行われる場合、最後のサーバが TPRETURN を呼び出して、要求の発信元であるクライアントに応答を返します。

次の図は、あるサーバから別のサーバに要求を転送したときのイベントの流れを示しています。この例では、クライアントは TPCALL ルーチンを使用して要求を開始し、連鎖の最後のサービス (サービス _C) が TPRETURN ルーチンを使用して応答を返しています。

図 5-1 要求の転送



サービス・ルーチンは TPSPRIO ルーチンを使用して、クライアント・プロセスが要求を送るのと同じように、指定された優先順位に従って要求を転送できます。

プロセスが TPFORWAR を呼び出すと、システムで提供された制御プログラムに制御が戻り、サーバ・プロセスは別の要求を処理できるようになります。

注記　クライアントとして動作するサーバ・プロセスが応答を要求する場合、このサーバが自分自身からサービスを要求することはできません。つまり、必要なサービスの唯一のインスタンスが要求を行っているサーバ・プロセスからのみ提供される場合、その呼び出しは失敗して再帰呼び出しができないことが示されます。ただし、

TPNOREPLY 通信フラグが設定された状態でサービス・ルーチンが自分宛てに要求を送信または転送した場合、サービスは自分からの応答を待機しないので、呼び出しは失敗しません。

TPFORWAR 呼び出しを使用して、その呼び出しを行った時点まで要求の処理が成功していたことを示すことができます。アプリケーション・エラーが検出されなかった場合、TPFORWAR を呼び出します。エラーが検出された場合、TP-RETURN-VAL IN TPSVCRET-REC に TPFAIL を設定して TPRETURN を呼び出します。

次のコード例は、TPFORWAR を呼び出して、サービスがそのデータ・レコードを DEPOSIT サービスに送る方法を示しています。新規口座の追加が成功した場合、支店レコードが更新されてその口座が反映され、データ・レコードが DEPOSIT サービスに転送されます。失敗した場合、TP-RETURN-VAL IN TPSVCRET-REC に TPFAIL が設定されて TPRETURN が呼び出され、失敗を示すメッセージがフォーム上のステータス行に出力されます。

コード リスト 5-8 TPFORWAR の使用方法

```
    . . .
******************************************************
* クライアントから送信されたデータの取得
******************************************************
    MOVE LENGTH OF TRANS-REC TO LEN.
    CALL "TPSVCSTART" USING TPSVCDEF-REC
                      TPTYPE-REC
                      TRANS-REC
                      TPSTATUS-REC.
    IF NOT TPOK
             MOVE "Transaction Encountered An Error" TO STATUS-LINE
             SET TPFAIL TO TRUE.
             COPY TPRETURN REPLACING
                      DATA-REC BY TRANS-REC.
    ELSE
             . . . Check other parameters
******************************************************
* 新規の口座レコードの挿入
******************************************************
    CALL "ADD-NEW-ACCOUNT-FUNCTION" USING TRANS-ACCOUNT IN TRANS-REC.
    IF Adding New Account Failed
             MOVE "Account not added" TO STATUS-LINE IN TRANS-REC
             SET TPFAIL TO TRUE
             COPY TPRETURN REPLACING
```

```
                             DATA-REC BY TRANS-REC.
******************************************************
* レコードを DEPOSIT サービスに転送して、
* 初期残高を口座に追加
******************************************************
    MOVE "DEPOSIT" TO SERVICE-NAME.
    . . . set other TPFORWAR parameters
    COPY TPFORWAR REPLACING
                             DATA-REC BY TRANS-REC.
```

サービスの宣言と宣言の取り消し

サーバは起動時に、コンフィギュレーション・ファイルの CLOPT パラメータに指定された値に基づいて、提供するサービスを宣言します。

注記　サーバが宣言するサービスは、buildserver コマンドの実行時に最初に定義されます。-s オプションを使用すると、複数のサービスをカンマ区切りで指定できます。また、宣言されたサービスと異なる名前のルーチンを呼び出して、サービス要求を処理できます。詳細については、『BEA Tuxedo コマンド・リファレンス』の buildserver(1) を参照してください。

デフォルトでは、サーバに組み込まれたすべてのサービスをそのサーバが宣言します。詳細については、『BEA Tuxedo のファイル形式とデータ記述方法』の UBBCONFIG(5) または servopts(5) リファレンス・ページを参照してください。

宣言されたサービスでは掲示板のサービス・テーブル・エントリが使用されるので、リソースが消費される場合があります。そのため、サーバの起動時には、提供されるサービスのサブセットだけを利用できるようにします。アプリケーションで利用できるサービスを制限するには、コンフィギュレーション・ファイルの SERVERS セクションで該当するエントリに CLOPT パラメータを定義し、-s オプションの後に必要なサービスをカンマで区切って指定します。また、-s オプションを使用すると、サービス要求を処理するため

に呼び出される宣言済みのサービスと異なる名前のルーチンを呼び出すこともできます。詳細については、『BEA Tuxedo のファイル形式とデータ記述方法』の servopts(5) リファレンス・ページを参照してください。

BEA Tuxedo アプリケーションの管理者は、tmadmin(1) の advertise および unadvertise コマンドを使用して、サーバで提供されるサービスを管理できます。TPADVERTISE および TPUNADVERTISE ルーチンを使用すると、要求 / 応答型サーバまたは会話型サーバでのサービスの宣言を動的に制御できます。ただし、宣言されるサービス、または宣言を取り消すサービスは、要求を行うサービスと同じサーバ内になければなりません。

サービスの宣言

TPADVERTISE(3cbl) 関数の呼び出しには、次の文法を使用します。

```
01 SERVICE-NAME          PIC X(15).
01 PROGRAM-NAME           PIC X(32).
01 TPSTATUS-REC.
    COPY TPSTATUS.
CALL "TPADVERTISE" USING SERVICE-NAME PROGRAM-NAME
TPSTATUS-REC.
```

次の表は、TPADVERTISE データ構造体のメンバを示しています。

表 5-3TPADVERTISE データ構造体のメンバ

メンバ	説明
SERVICE-NAME	宣言するサービスの名前。サービス名は 15 文字以下の文字列で指定します。15 文字を超える名前は切り捨てられます。SPACES 文字列は指定できません。NULL 文字列が指定されると、エラー (TPEINVAL) になります。
PROGRAM-NAME	サービスを実行するために呼び出される BEA Tuxedo システム・ルーチン。通常、この名前とサービス名は同じです。SPACES 文字列は指定できません。NULL 文字列が指定されると、エラーになります。

サービス宣言の取り消し

TPUNADVERTISE(3cbl) ルーチンは、掲示板のサービス・テーブルからサービス名を削除します。サービス名が削除されたサービスは、宣言されていない状態になります。

TPUNADVERTISE ルーチンの呼び出しには、次の文法を使用します。

```
01 SERVICE-NAME            PIC X(15).
01 TPSTATUS-REC.
   COPY TPSTATUS.
CALL "TPUNADVERTISE" USING SERVICE-NAME TPSTATUS-REC.
```

TPUNADVERTISE データ構造体のメンバは、次の表で説明する SERVICE-NAME だけです。

表 5-4TPUNADVERTISE データ構造体のメンバ

メンバ	説明
SERVICE-NAME	宣言するサービスの名前。サービス名は 15 文字以下の文字列で指定します。15 文字を超える名前は切り捨てられます。SPACES 文字列は指定できません。NULL 文字列が指定されると、エラー (TPEINVAL) になります。

例 : サービスの動的な宣言と宣言の取り消し

次のコード例は、TPADVERTISE ルーチンの使用方法を示しています。このコードでは、サーバ TLR が起動時に TLRINIT サービスだけを提供するようにコーディングされています。初期化後、TLRINIT は DEPOSIT と WITHDRAW という 2 つのサービスを宣言します。両サービスとも TLRFUNCS ルーチンによって実行され、サーバ TLR に組み込まれています。

DEPOSIT と WITHDRAW を宣言した後、TLRINIT は自分自身で宣言を取り消します。

コード リスト 5-9　動的な宣言と宣言の取り消し

```
    . . .
**************************************************
* TLRFUNCS ルーチンで処理する
* DEPOSIT サービスの宣言
**************************************************
    MOVE "DEPOSIT" TO SERVICE-NAME.
    MOVE "TLRFUNCS" TO PROGRAM-NAME.
    CALL "TPADVERTISE" USING SERVICE-NAME
                       PROGRAM-REC
                       TPSTATUS-REC.
    IF NOT TPOK
           error processing
**************************************************
* 同じ TLRFUNCS ルーチンで処理する
* WITHDRAW サービスの宣言
**************************************************
    MOVE "WITHDRAW" TO SERVICE-NAME.
    MOVE "TLRFUNCS" TO PROGRAM-NAME.
    CALL "TPADVERTISE" USING SERVICE-NAME
                       PROGRAM-REC
                       TPSTATUS-REC.
    IF NOT TPOK
           error processing
**************************************************
* TLRINIT サービスの宣言の取り消し
**************************************************
    MOVE "TLRINIT" TO SERVICE-NAME.
    CALL "TPUNADVERTISE" USING SERVICE-NAME
                         TPSTATUS-REC.
    IF NOT TPOK
            error processing
```

サーバのビルド

実行可能なサーバをビルドするには、`buildserver(1)` コマンドに `-C` オプションを使用して、BEA Tuxedo System サーバ・アダプタなどすべての参照ファイルでアプリケーション・サービス・サブルーチンをコンパイルします。

注記　BEA Tuxedo サーバ・アダプタは、メッセージの受信、処理のディスパッチ、トランザクションが有効な場合はトランザクションの管理を行います。

`buildserver` コマンドには、次の構文を使用します。

```
buildserver -C -o filename -f filenames -l filenames -s -v
```

次の表は、`buildserver` コマンド行オプションを示しています。

表 5-5buildserver コマンド行オプション

オプション . .	説明 . .
`-o` *filename*	実行可能な出力ファイル名。デフォルト値は `SERVER` です。
`-f` *filenames*	BEA Tuxedo システム・ライブラリより先にリンクされるファイルのリスト。`-f` オプションは複数回指定できます。また、各 `-f` オプションに複数のファイル名を指定することもできます。COBOL プログラム・ファイル (*file*.cbl) を指定すると、リンクされる前にコンパイルが行われます。ほかのオブジェクト・ファイル (*file*.o) を個別に、またはアーカイブ・ファイル (*file*.a) にまとめて指定することもできます。
`-l` *filenames*	Tuxedo システム・ライブラリの後でリンクされるファイルのリスト。`-l` オプションは複数回指定できます。また、各 `-l` に複数のファイル名を指定できます。COBOL プログラム・ファイル (*file*.cbl) を指定すると、リンクされる前にコンパイルが行われます。ほかのオブジェクト・ファイル (*file*.o) を個別に、またはアーカイブ・ファイル (*file*.a) にまとめて指定することもできます。

表 5-5buildserver コマンド行オプション

オプション . .	説明 . .
-r *filenames*	実行可能サーバにリンクされるリソース・マネージャのアクセス・ライブラリのリスト。アプリケーション管理者は、buildtms(1) コマンドを使用して、すべての有効なリソース・マネージャ情報を $TUXDIR/updataobj/RM ファイルに事前に定義しておく必要があります。リソース・マネージャは 1 つしか指定できません。詳細については、『BEA Tuxedo アプリケーションの設定』を参照してください。
-s [*service*:]*routine*	サーバに提供されるサービス名、および各サービスを実行するルーチン名。-s オプションは複数回指定できます。また、各 -s に複数のサービスを指定できます。サーバは指定されたサービス名を使用して、クライアントにサービスを宣言します。 通常、サービスとそのサービスを実行するルーチンには同じ名前を割り当てます。ただし、別の名前を指定することもできます。名前の割り当てには、*service*:*routine* という構文を使用します。

注記　BEA Tuxedo ライブラリは自動的にリンクされます。コマンド行に BEA Tuxedo ライブラリ名を指定する必要はありません。

リンクするライブラリ・ファイルの指定順序は重要です。ルーチンを呼び出す順序と、これらのルーチンへの参照を含むライブラリによって、この順序が決定されます。

デフォルトでは、buildserver コマンドは UNIX の cobcc コマンドを呼び出します。環境変数 ALTCC を指定して別のコンパイル・コマンドを指定したり、ALTCFLAGS を指定してコンパイル・フェーズやリンク・フェーズに独自のフラグを指定することができます。詳細については、2-7 ページの「環境変数の設定」を参照してください。

注記　Windows 2000 システムでは、環境変数 ALTCC および ALTCFLAGS は使用できません。これらを設定すると、予期しない結果が生じます。最初に COBOL コンパイラを使用してアプリケーションをコンパイルし、次に生成されたオブジェクト・ファイルを buildserver コマンドに渡します。

次のコマンドは、`acct.o` アプリケーション・ファイルを処理して、NEW_ACCT と CLOSE_ACCT という 2 つのサービスを含む ACCT サーバを作成しています。`NEW_ACCT` は `OPEN_ACCT` ルーチンを呼び出し、`CLOSE_ACCT` は同じ名前のルーチンを呼び出します。

```
buildserver -C -o ACCT -f acct.o -s NEW_ACCT:OPEN_ACCT -s CLOSE_ACCT
```

関連項目

- 4-11 ページの「クライアントのビルド」
- 『BEA Tuxedo コマンド・リファレンス』の `buildclient(1)`

6 クライアントおよびサーバへの要求 / 応答のコーディング

ここでは、次の内容について説明します。

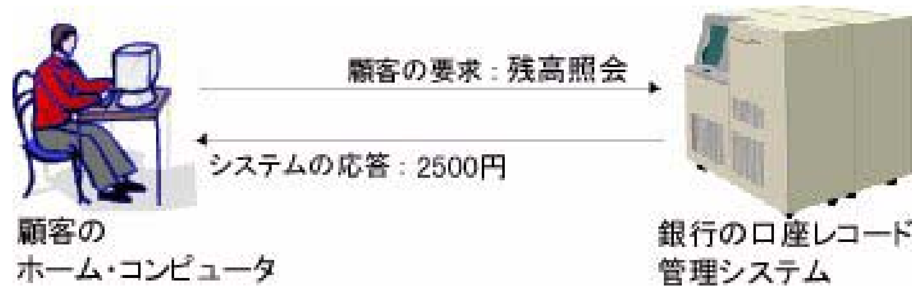
- 要求 / 応答通信の概要
- 同期メッセージの送信
- 非同期メッセージの送信
- メッセージの優先順位の設定および取得

要求 / 応答通信の概要

要求 / 応答通信モードでは、あるソフトウェア・モジュールが別のソフトウェア・モジュールに要求を送り、応答を待ちます。最初のソフトウェア・モジュールがクライアント、2 番目のソフトウェア・モジュールがサーバとして動作するので、このモードはクライアント / サーバ相互作用とも呼ばれます。オンラインの銀行業務の多くは、要求 / 応答モードでプログラミングされます。たとえば、残高照会の要求は、次のように行われます。

1. 顧客 (クライアント) は、口座レコード管理システム (サーバ) に、残高照会の要求を送信します。
2. 口座レコード管理システム (サーバ) は、指定された口座の残高を応答として顧客 (クライアント) に送ります。

図 6-1 オンライン銀行業務での応答 / 要求通信



クライアント・プロセスはアプリケーションに参加すると、要求メッセージをサブルーチンに送信して処理を行い、応答メッセージを受信できるようになります。

同期メッセージの送信

`TPCALL(3cbl)` 呼び出しは、サービス・サブルーチンに要求を送信し、同期的に応答を待ちます。`TPCALL` ルーチンの呼び出しには、次の文法を使用します。

```
01 TPSVCDEF-REC.
   COPY TPSVCDEF.
01 ITPTYPE-REC.
   COPY TPTYPE.
01 IDATA-REC.
   COPY User Data.
01 OTPYTPE-REC.
   COPY TPTYPE.
01 ODATA-REC.
   COPY User Data.
01 TPSTATUS-REC.
```

```
   COPY TPSTATUS.
CALL "TPCALL" USING TPSVCDEF-REC
       ITPTYPE-REC
       IDATA-REC
       OTPTYPE-REC
       ODATA-REC
       TPSTATUS-REC.
```

TPSVCDEF データ構造体の詳細については、『C 言語を使用した BEA Tuxedo アプリケーションのプログラミング』を参照してください。IDATA-REC および ITPTYPE-REC 構造体は、要求レコードを定義します。ODATA-REC および OTPTYPE-REC 構造体は、応答レコードを定義します。ITPTYPE-REC および OTPTYPE-REC データ構造体は、TPTYPE-REC データ構造体と似ています。

TPCALL は、応答を待ちます。

注記　TPCALL ルーチンの呼び出しは、TPACALL ルーチンを呼び出した直後に TPGETRPLY を呼び出すことと論理的には同じです。6-10 ページの「非同期メッセージの送信」を参照してください。

要求は、指定されたサービス (SERVICE-NAME) の優先順位で送信されます。ただし、TPSPRIO ルーチンの呼び出しで明示的に異なる優先順位が設定されている場合は除きます。詳細については、6-14 ページの「メッセージの優先順位の設定および取得」を参照してください。

TPCALL は整数を返します。失敗した場合、発生したエラー条件を示す値が TP-STATUS に設定されます。有効なエラー・コードの詳細については、『BEA Tuxedo COBOL リファレンス』の TPCALL(3cbl) を参照してください。

注記　通信呼び出しはいろいろな原因で失敗します。そのほとんどは、アプリケーション・レベルで修正することができます。失敗の原因としては、アプリケーション定義のエラー (TPESVCFAIL)、戻り値の処理エラー (TPESVCERR)、型付きレコードのエラー (TPEITYPE、TPEOTYPE)、タイムアウト・エラー (TPETIME)、プロトコル・エラー (TPEPROTO) などがあります。エラーの詳細については、11-1 ページの「エラーの管理」を参照してください。発生する可能性があるエラーについては、『BEA Tuxedo COBOL リファレンス』の TPCALL(3cbl) を参照してください。

BEA Tuxedo システムでは、割り当てられているレコードより大きなメッセージを受信した場合、メッセージ受信用レコードのサイズが自動的に変更されます。そのため、応答レコードのサイズが変更されたかどうかを確認する必要があります。

レコードの新しいサイズにアクセスするには、*LEN IN OTPTYPE-REC に返されたアドレスを使用します。応答レコードのサイズが変更されたかどうかを確認するには、TPCALL を呼び出す前の応答レコードのサイズと、返された応答レコードの LEN IN OTPTYPE-REC の値とを比較します。LEN IN OTPTYPE-REC が元の値より大きい場合、レコードのサイズは大きくなっています。それ以外の場合、レコードのサイズは変更されていません。

例 : 要求メッセージと応答メッセージに同じレコードを使用する

以下のコード例は、クライアント・プログラムで、要求メッセージと応答メッセージに同じレコードを使用して、同期呼び出しを行う方法を示しています。この例では、AUDV-REC メッセージ・レコードは要求情報と応答情報の両方を格納するように設定されているので、同じレコードを使用することができます。このコードでは、次の処理が行われます。

1. サービスは B_ID フィールドを照会します。ただし、このフィールドを上書きしません。
2. アプリケーションは、BALANCE フィールドをゼロに初期化して、サービスから返される値を受け取る準備をします。
3. SERVICE-NAME は要求されたサービス名を示します。この例では、口座残高と窓口残高が示されます。

コード リスト 6-1 要求メッセージと応答メッセージに同じレコードを使用する

```
WORKING-STORAGE SECTION.
*****************************************************
* Tuxedo の定義
*****************************************************
```

```
  01  TPTYPE-REC.
  COPY TPTYPE.
*
  01 TPSTATUS-REC.
  COPY TPSTATUS.
*
  01  TPSVCDEF-REC.
  COPY TPSVCDEF.
*****************************************************
* ログ・メッセージの定義
*****************************************************
  01  LOGMSG.
     05  FILLER           PIC X(6) VALUE  "FIG =>".
     05  LOGMSG-TEXT      PIC X(50).
  01  LOGMSG-LEN          PIC S9(9)  COMP-5.
*
  01  USER-DATA-REC       PIC X(75).
*****************************************************
* この VIEW レコード (audv) はサーバに送られます。
*****************************************************
  01 AUDV-REC.
  COPY AUDV.
*
******************************************************
  PROCEDURE DIVISION.
  START-FIG.
  MOVE LENGTH OF LOGMSG TO LOGMSG-LEN.
*****************************************************
*  audv レコードを準備します。
*****************************************************
  MOVE "BRANCH" TO B-ID IN AUDV-REC.
  MOVE 0 TO BALANCE IN AUDV-REC.
  MOVE LENGTH OF AUDV-REC TO LEN.
  MOVE "VIEW" TO REC-TYPE.
  MOVE "audv" TO SUB-TYPE.
  MOVE "SOMESERVICE" TO SERVICE-NAME.
  SET TPBLOCK TO TRUE.
  SET TPNOTRAN TO TRUE.
  SET TPNOTIME TO TRUE.
  SET TPSIGRSTRT TO TRUE.
  SET TPNOCHANGE TO TRUE.
  CALL "TPCALL" USING TPSVCDEF-REC
              TPTYPE-REC
              AUDV-REC
              TPTYPE-REC
              AUDV-REC
              TPSTATUS-REC.
  IF NOT TPOK
```

```
        MOVE "Service Failed" TO LOGMSG-TEXT
        PERFORM DO-USERLOG
        PERFORM EXIT-PROGRAM.
  DISPLAY BRANCH and BALANCE
  . . .
```

応答が ODATA-REC より大きい場合、ODATA-REC にはこのレコードに入るだけのメッセージが格納されます。残りのメッセージは破棄され、TPCALL は TP-STATUS IN TPSTATUS-REC に TPTRUNCATE を設定します。

例 : TPSIGRSTRT フラグを設定した同期メッセージの送信

以下のコード例は、bankapp の XFER サーバ・プロセスの一部である TRANSFER サービスに基づいています。bankapp は、BEA Tuxedo システムに提供されている銀行業務のサンプル・アプリケーションです。この例では、あるサービスがクライアントとして WITHDRAWAL および DEPOSIT サービスを呼び出します。アプリケーションはこの 2 つのサービスを呼び出すときに通信フラグを TPSIGRSTRT に設定して、トランザクションをコミットしやすいようにします。TPSIGRSTRT フラグは、シグナルの割り込みがあった場合に行う処理を指定します。通信フラグの詳細については、『BEA Tuxedo COBOL リファレンス』の TPCALL(3cbl) を参照してください。

コード リスト 6-2　TPSIGRSTRT フラグを設定した同期メッセージの送信

```
   WORKING-STORAGE SECTION.
*****************************************************
* Tuxedo の定義
*****************************************************
   01 TPTYPE-REC.
   COPY TPTYPE.
*
   01 TPSTATUS-REC.
   COPY TPSTATUS.
*
   01 TPSVCDEF-REC.
```

```
   COPY TPSVCDEF.
*****************************************************
* この VIEW レコード (audv) はサーバに送られます。
*****************************************************
   01 AUDV-REC.
   COPY AUDV.
*
******************************************************
   PROCEDURE DIVISION.
 START-FIG.
*****************************************************
* 引き出し用に audv レコードを準備します。
*****************************************************
  . . .
  MOVE "WITHDRAWAL" TO SERVICE-NAME.
  SET TPSIGRSTRT TO TRUE.
  PERFORM DO-TPCALL.
  IF NOT TPOK
         MOVE "Cannot withdraw from debit account" TO LOGMSG-TEXT
         PERFORM DO-USERLOG
         PERFORM EXIT-PROGRAM.
  MOVE "DEPOSIT" TO SERVICE-NAME.
  SET TPSIGRSTRT TO TRUE.
  PERFORM DO-TPCALL.
  IF NOT TPOK
         MOVE "Cannot deposit into credit account" TO LOGMSG-TEXT
         PERFORM DO-USERLOG
         PERFORM EXIT-PROGRAM.
. . .
*****************************************************
*  TPCALL を実行します。
*****************************************************
 DO-TPCALL.
   MOVE LENGTH OF AUDV-REC TO LEN.
   MOVE "VIEW" TO REC-TYPE.
   MOVE "audv" TO SUB-TYPE.
   SET TPBLOCK TO TRUE.
   SET TPNOTRAN TO TRUE.
   SET TPNOTIME TO TRUE.
   SET TPNOCHANGE TO TRUE.
   CALL "TPCALL" USING TPSVCDEF-REC
                 TPTYPE-REC
                 AUDV-REC
                 TPTYPE-REC
                 AUDV-REC
                 TPSTATUS-REC.
   . . .
```

例 : TPNOTRAN フラグを設定した同期メッセージの送信

以下のコード例は、トランザクション・モードではない通信呼び出しを示しています。この呼び出しは、リソース・マネージャに関連していないサービスに対して行われます。サービスがトランザクションに参加するとエラーになります。アプリケーションは、データベース ACCOUNTS から取得した情報に基づいて生成された売掛金勘定レポート ACCRV を出力します。

サービス・ルーチン REPORT は指定されたパラメータを解釈し、完了したレポートのバイト・ストリームを応答として送信します。クライアントは、TPCALL を使用して PRINTER サービスにバイト・ストリームを送信します。PRINTER は、クライアントに近いプリンタにバイト・ストリームを送信します。そして、応答が印刷されます。最後に、PRINTER サービスはハードコピーの印刷が終了したことをクライアントに通知します。

注記　6-12 ページの「TPNOTRAN または TPNOREPLY を設定した非同期メッセージの送信」では、同じ例を使用して非同期メッセージの呼び出しを行っています。

コード リスト 6-3　TPNOTRAN **フラグを設定した同期メッセージの送信**

```
    WORKING-STORAGE SECTION.
*****************************************************
* Tuxedo の定義
*****************************************************
    01  ITPTYPE-REC.
    COPY TPTYPE.
    01  OTPTYPE-REC.
    COPY TPTYPE.
*
    01 TPSTATUS-REC.
    COPY TPSTATUS.
*
    01  TPSVCDEF-REC.
    COPY TPSVCDEF.
*****************************************************
    01 REPORT-REQUEST         PIC X(100) VALUE SPACES.
    01 REPORT-OUTPUT          PIC X(50000) VALUE SPACES.
******************************************************
```

```
    PROCEDURE DIVISION.
  START-FIG.
    . . .
    join application
  start transaction
    .. .
*******************************************************
* REPORT サービスにレポート要求を送ります。
* REPORT-OUTPUT に応答を受け取ります。
*******************************************************
    MOVE "REPORT=accrcv DBNAME=accounts" TO REPORT-REQUEST.
    MOVE "STRING" TO REC-TYPE IN ITYPE-REC.
    MOVE 29 TO LEN IN ITYPE-REC.
    MOVE "STRING" TO REC-TYPE IN OITYPE-REC.
    MOVE 50000 TO LEN IN OTYPE-REC.
    MOVE "REPORT" TO SERVICE-NAME.
    SET TPTRAN TO TRUE.
    SET TPBLOCK TO TRUE.
    SET TPNOTIME TO TRUE.
    SET TPSIGRSTRT TO TRUE.
    SET TPNOCHANGE TO TRUE.
    CALL "TPCALL" USING TPSVCDEF-REC
              ITPTYPE-REC
              REPORT-REQUEST
              OTPTYPE-REC
              REPORT-OUTPUT
              TPSTATUS-REC.
    IF NOT TPOK
       error processing
    IF TPETRUNCATE
       The report was truncated
   error processing
*******************************************************
* PRINTER サービスに REPORT-OUTPUT を送ります。
*******************************************************
    MOVE "PRINTER" TO SERVICE-NAME.
    SET TPNOTRAN TO TRUE.
    MOVE "STRING" TO REC-TYPE IN ITTYPE-REC.
    MOVE LEN IN OTYPE-REC TO LEN IN ITYPE-REC.
    CALL "TPCALL" USING TPSVCDEF-REC
              ITPTYPE-REC
              REPORT-OUTPUT
              OTPTYPE-REC
              REPORT-OUTPUT
              TPSTATUS-REC.
    IF NOT TPOK
         error processing
  . . .
```

```
terminate transaction
 leave application
```

注記　この例の error routine は、エラー・メッセージの出力、トランザクションの中止、クライアントのアプリケーションからの分離、およびプログラムの終了が行われることを示しています。

また、この例では、最初に割り当てられたレコード・タイプと同じタイプで応答メッセージを返す必要があることを示して、TPNOCHANGE 通信フラグを使用して厳密なタイプ・チェックを行う方法を示しています。厳密なタイプ・チェックのフラグ TPNOCHANGE が設定されているので、STRING 型のレコードに応答が返されます。

厳密なタイプ・チェックを行うのは、REPORT サービス・サブルーチンでエラーが発生して、不適切なタイプの応答レコードが使用されることを防ぐためです。もう 1 つの理由は、依存関係にあるすべてのエリアで一貫していない変更が行わることを防ぐためです。たとえば、あるプログラマが REPORT サービスを変更してすべての応答を別の STRING 形式で標準化したが、それを反映するためにクライアント・プロセスを変更しなかった場合などがあります。

非同期メッセージの送信

この節では、次の操作を行う方法について説明します。

- TPACALL ルーチンを使用した非同期要求の送信
- TPGETRPLY ルーチンを使用した非同期応答の取得

ここで説明する非同期の処理は、ファンアウト並列処理と呼ばれます。クライアントの要求が複数のサービスに同時に分散 (つまり「ファンアウト」) されて処理が行われるからです。

このほかに、BEA Tuxedo システムでは、非同期処理としてパイプライン並列処理もサポートされています。この処理では、TPFORWAR ルーチンを使用して 1 つのサービスから別のサービスに処理が渡されます (転送されます)。TPFORWAR ルーチンについては、5-1 ページの「サーバのコーディング」を参照してください。

非同期要求の送信

TPACALL(3cbl) ルーチンは、サービス要求を送信し、直ちに制御を戻します。TPACALL ルーチンの呼び出しには、次の文法を使用します。

```
01 TPSVCDEF-REC.
   COPY TPSVCDEF.
01 TPTYPE-REC.
   COPY TPTYPE.
01 DATA-REC.
   COPY User Data.
01 TPSTATUS-REC.
   COPY TPSTATUS.
CALL "TPACALL" USING TPSVCDEF-REC TPTYPE-REC DATA-REC
TPSTATUS-REC.
```

TPSVCDEF および TPTYPE-REC データ構造体の詳細については、5-12 ページの「サービスの定義」を参照してください。

TPACALL ルーチンは、SERVICE-NAME に指定されたサービスに要求メッセージを送信し、直ちに制御を戻します。呼び出しが正常に終了すると、TPACALL ルーチンは整数値を返します。この値は、関連する要求に対する正しい応答にアクセスするための記述子として使用されます。TPACALL がトランザクション・モードで実行されている場合 (9-1 ページの「グローバル・トランザクションのコーディング」を参照)、トランザクションのコミット時に未処理の応答が存在することはありません。つまり、あるトランザクションの範囲内では、要求ごとにその応答が返されるので、最終的には対応する応答を必ず受信することになります。

TPNOREPLY が設定されると、応答が必要ないことが TPACALL に通知されます。このフラグが設定されている場合、TPACALL の処理が正常に終了すると、応答記述子として 0 が返されます。以降の処理で、この値が TPGETRPLY ルーチンに渡されると、この値は無効になります。プロセスがトランザク

ション・モードのときにこの設定を正しく使用するためのガイドラインについては、9-1 ページの「グローバル・トランザクションのコーディング」を参照してください。

エラーが発生した場合、TPACALL は発生したエラーの内容を示す値を TP-STATUS に設定します。TPACALL が返すエラー・コードの多くは、TPCALL が返すエラー・コードと同じです。この 2 つの関数のエラー・コードは、一方が同期呼び出し、もう一方が非同期呼び出しに基づいているという点が異なります。これらのエラーについては、11-1 ページの「エラーの管理」を参照してください。

以下のコード例は、TPACALL で TPNOTRAN と TPNOREPLY 設定を使用する方法を示しています。このコードは、6-8 ページの「例 : TPNOTRAN フラグを設定した同期メッセージの送信」のコードと同じです。ただし、ここで示すコードでは、PRINTER サービスからの応答は要求されていません。TPNOREPLY はクライアントが応答を要求していないこと、TPNOTRAN は PRINTER サービスが現在のトランザクションに参加しないことを示します。詳細については、11-1 ページの「エラーの管理」を参照してください。

コード リスト 6-4　TPNOTRAN **または** TPNOREPLY **を設定した非同期メッセージの送信**

```
   WORKING-STORAGE SECTION.
*****************************************************
* Tuxedo の定義
*****************************************************
   01 ITPTYPE-REC.
   COPY TPTYPE.
   01 OTPTYPE-REC.
   COPY TPTYPE.
*
   01 TPSTATUS-REC.
   COPY TPSTATUS.
*
   01 TPSVCDEF-REC.
   COPY TPSVCDEF.
*****************************************************
   01 REPORT-REQUEST        PIC X(100) VALUE SPACES.
   01 REPORT-OUTPUT         PIC X(50000) VALUE SPACES.
******************************************************
   PROCEDURE DIVISION.
 START-FIG.
```

```
    . . .
     join application
  start transaction
     .. .
*****************************************************
* REPORT サービスにレポート要求を送ります。
* REPORT-OUTPUT に応答を受け取ります。
*****************************************************
   MOVE "REPORT=accrcv DBNAME=accounts" TO REPORT-REQUEST.
   MOVE "STRING" TO REC-TYPE IN ITPTYPE-REC.
   MOVE 29 TO LEN IN ITPTYPE-REC.
   MOVE "STRING" TO REC-TYPE IN OITYPE-REC.
   MOVE 50000 TO LEN IN OTPTYPE-REC.
   MOVE "REPORT" TO SERVICE-NAME.
   SET TPTRAN TO TRUE.
   SET TPBLOCK TO TRUE.
   SET TPNOTIME TO TRUE.
   SET TPSIGRSTRT TO TRUE.
   SET TPREPLY TO TRUE.
   SET TPNOCHANGE TO TRUE.
   CALL "TPCALL" USING TPSVCDEF-REC
                 ITPTYPE-REC
                 REPORT-REQUEST
                 OTPTYPE-REC
                 REPORT-OUTPUT
                 TPSTATUS-REC.
   IF NOT TPOK
      error processing
   IF TPETRUNCATE
      The report was truncated
   error processing
*****************************************************
* PRINTER サービスに REPORT-OUTPUT を送ります。
*****************************************************
   MOVE "PRINTER" TO SERVICE-NAME.
   SET TPNOTRAN TO TRUE.


   SET TPNOREPLY TO TRUE.
   MOVE "STRING" TO REC-TYPE IN ITPTYPE-REC.
   MOVE LEN IN OTPTYPE-REC TO LEN IN ITPTYPE-REC.
   CALL "TPACALL" USING TPSVCDEF-REC
                  ITPTYPE-REC
                  REPORT-OUTPUT
                  TPSTATUS-REC.
   IF NOT TPOK
      error processing
   . . .
```

```
    commit transaction
leave application
```

非同期応答の受信

サービス呼び出しに対する応答は、TPGETRPLY(3cbl) ルーチンを呼び出すと非同期的に受信できます。TPGETRPLY ルーチンは、TPACALL が以前に送信した要求に対する応答をキューから取り出します。

TPGETRPLY ルーチンの呼び出しには、次の文法を使用します。

```
01 TPSVCDEF-REC.
   COPY TPSVCDEF.
01 TPTYPE-REC.
   COPY TPTYPE.
01 DATA-REC.
   COPY User Data.
01 TPSTATUS-REC.
   COPY TPSTATUS.
CALL "TPGETRPLY" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

TPSVCDEF および TPTYPE-REC データ構造体の詳細については、5-12 ページの「サービスの定義」を参照してください。

デフォルトでは、この関数は通信ハンドルが参照する値に対応した応答を待ちます。応答を待っている間に、ブロッキング・タイムアウトが発生する場合があります。タイムアウトが発生するのは、TPGETRPLY が失敗し、TP-STATUS に TPETIME が設定された場合です。ただし、TPNOTIME が設定されている場合は除きます。

メッセージの優先順位の設定および取得

2 つの ATMI 呼び出し、TPSPRIO(3cbl) および TPGPRIO(3cbl) を使用して、メッセージ要求の優先順位を決定したり設定したりできます。この優先順位に従って、サーバがキューから要求を取り出します。つまり、最も優先順位の高い要求が最初に取り出されます。

この節では、次の内容について説明します。

- メッセージの優先順位の設定
- メッセージの優先順位の取得

メッセージの優先順位の設定

TPSPRIO(3cbl) ルーチンを使用すると、メッセージ要求の優先順位を設定できます。

TPSPRIO ルーチンで優先順位を設定できるのは、1 つの要求だけです。つまり、TPCALL または TPACALL によって次に送信される要求、またはサービス・サブルーチンによって次に転送される要求だけです。

TPSPRIO ルーチンの呼び出しには、次の文法を使用します。

```
01 TPPRIDEF-REC.
   COPY TPPRIDEF.
01 TPSTATUS-REC.
   COPY TPSTATUS.
CALL "TPSPRIO" USING TPPRIDEF-REC TPSTATUS-REC.
```

TPPRIDEF-REC データ構造体の呼び出しには、次の文法を使用します。

```
05 PRIORITY    PIC S9(9) COMP-5.
05 PRIO-FLAG   PIC S9(9) COMP-5.
  88 TPABSOLUTE     VALUE 0.
  88 TPRELATIVE     VALUE 1.
```

次の表は、TPSPRIO ルーチンのフィールドを示しています。

表 6-1TPSPRIO ルーチンのフィールド

フィールド	**説明**
PRIORITY	新しい優先順位を示す整数値。この引数の持つ意味は、PRIO-FLAG によって異なります。PRIO-FLAG にゼロが設定されている場合、PRIORITY は相対値を示し、値の符号は現在の優先順位を上げることまたは下げることを示します。ほかの値が設定されている場合、指定された値は絶対値を示し、PRIORITY には 0 ～ 100 の範囲の値を設定する必要があります。この範囲外の値を設定すると、50 に設定されます。
PRIO-FLAG	PRIORITY の値を相対値 (0) または絶対値 (TPABSOLUTE) のどちらの値として処理するのかを示す値。デフォルトは相対値です。

以下のコード例は、TRANSFER サービスから引用したものです。このコードでは、TRANSFER サービスはクライアントとして動作し、TPCALL を使用して WITHDRAWAL サービスに同期要求を送信しています。TRANSFER は TPSPRIO を呼び出して WITHDRAWAL に対する要求メッセージの優先順位を上げます。また、TRANSFER のキューを待機した後で、WITHDRAWAL サービス (その後 DEPOSIT サービス) に対する要求がキューに格納されないようにします。

コード リスト 6-5　要求メッセージの優先順位の設定

```
    WORKING-STORAGE SECTION.
*****************************************************
* Tuxedo の定義
*****************************************************
    01 TPTYPE-REC.
       COPY TPTYPE.
*
    01 TPSTATUS-REC.
       COPY TPSTATUS.
*
    01 TPSVCDEF-REC.
       COPY TPSVCDEF.
*
    01 TPPRIDEF-REC.
       COPY TPPRIDEF.
*****************************************************
    01 DATA-REC               PIC X(100) VALUE SPACES.
```

```
*******************************************************
    PROCEDURE DIVISION.
 START-FIG.
    . . .
    join application
    . . .
    MOVE 30 TO PRIORITY.
    SET TPRELATIVE TO TRUE.
    CALL "TPSPRIO" USING TPPRIDEF-REC TPSTATUS-REC
    IF NOT TPOK
       error processing
    MOVE "CARRAY" TO REC-TYPE.
    MOVE 100 TO LEN.
    MOVE "WITHDRAWAL" TO SERVICE-NAME.
    SET TPTRAN TO TRUE .
    SET TPBLOCK TO TRUE .
    SET TPNOTIME TO TRUE .
    SET TPSIGRSTRT TO TRUE .
    SET TPREPLY TO TRUE .
    CALL "TPACALL" USING TPSVCDEF-REC
                   TPTYPE-REC
                   DATA-REC
                   TPSTATUS-REC.
    IF NOT TPOK
       error processing
    .. .
    leave application
```

メッセージの優先順位の取得

TPGPRIO(3cbl) ルーチンを使用すると、メッセージ要求の優先順位を取得できます。

TPGPRIO ルーチンの呼び出しには、次の文法を使用します。

```
01 TPPRIDEF-REC.
   COPY TPPRIDEF.
01 TPSTATUS-REC.
   COPY TPSTATUS.
CALL "TPGPRIO" USING TPPRIDEF-REC TPSTATUS-REC.
```

要求元は、TPCALL または TPACALL ルーチンを呼び出した後に TPGPRIO ルーチンを呼び出して、要求メッセージの優先順位を取得できます。要求元が関数を呼び出したが要求が送信されていない場合、ルーチンは失敗し、TP-STATUS に TPENOENT が設定されます。TPGPRIO の処理が成功すると、TP-STATUS に TPOK が設定され、1 〜 100 の範囲内の整数値が返されます。100 が最も高い優先順位です。

TPSPRIO ルーチンを使用して優先順位が明示的に設定されていない場合、要求を処理するサービス・ルーチンの優先順位がメッセージの優先順位として設定されます。アプリケーション内では、要求を処理するサービスの優先順位にデフォルト値の 50 が設定されます。ただし、システム管理者が別の値を指定している場合は除きます。

次のコード例は、非同期呼び出しによって送信されたメッセージの優先順位を確認する方法を示しています。

コード リスト 6-6　送信後の要求の優先順位の確認

```
   WORKING-STORAGE SECTION.
*****************************************************
* Tuxedo の定義
*****************************************************
   01 TPTYPE-REC-1.
   COPY TPTYPE.
   01 TPTYPE-REC-2.
   COPY TPTYPE.
*
   01 TPSTATUS-REC.
   COPY TPSTATUS.
*
   01 TPSVCDEF-REC-1.
   COPY TPSVCDEF.
   01 TPSVCDEF-REC-2.
   COPY TPSVCDEF.
*
   01 TPPRIDEF-REC-1.
   COPY TPPRIDEF.
   01 TPPRIDEF-REC-2.
   COPY TPPRIDEF.
*****************************************************
   01 DATA-REC-1     PIC X(100) VALUE SPACES.
   01 DATA-REC-2     PIC X(100) VALUE SPACES.
******************************************************
   PROCEDURE DIVISION.
```

```
START-FIG.
    . . .
   join application
  populate DATA-REC1 and DATA-REC2 with send request
    . . .
    MOVE "CARRAY" TO REC-TYPE IN TYPE-REC-1.
    MOVE 100 TO LEN IN TYPE-REC-1.
    MOVE "SERVICE1" TO SERVICE-NAME IN TPSVCDEV-REC-1.
    SET TPTRAN TO TRUE IN TPSVCDEV-REC-1.
    SET TPBLOCK TO TRUE IN TPSVCDEV-REC-1.
    SET TPNOTIME TO TRUE IN TPSVCDEV-REC-1.
    SET TPSIGRSTRT TO TRUE IN TPSVCDEV-REC-1.
    SET TPREPLY TO TRUE IN TPSVCDEV-REC-1.
    CALL "TPACALL" USING TPSVCDEF-REC-1
                   TPTYPE-REC-1
                   DATA-REC-1
                   TPSTATUS-REC.
    IF NOT TPOK
       error processing
    CALL "TPGPRIO" USING TPPRIDEF-REC-1 TPSTATUS-REC
    IF NOT TPOK
       error processing
    MOVE "CARRAY" TO REC-TYPE IN TYPE-REC-2.
    MOVE 100 TO LEN IN TYPE-REC-2.
    MOVE "SERVICE2" TO SERVICE-NAME IN TPSVCDEV-REC-2.
    SET TPTRAN TO TRUE IN TPSVCDEV-REC-2.
    SET TPBLOCK TO TRUE IN TPSVCDEV-REC-2.
    SET TPNOTIME TO TRUE IN TPSVCDEV-REC-2.
    SET TPSIGRSTRT TO TRUE IN TPSVCDEV-REC-2.
    SET TPREPLY TO TRUE IN TPSVCDEV-REC-2.
    CALL "TPACALL" USING TPSVCDEF-REC-2
                   TPTYPE-REC-2
                   DATA-REC-2
                   TPSTATUS-REC.
    IF NOT TPOK
       error processing
    CALL "TPGPRIO" USING TPPRIDEF-REC-2 TPSTATUS-REC
    IF NOT TPOK
       error processing
    IF PRIORITY IN TPSVCDEF-REC-1 >= PRIORITY IN TPSVCDEF-REC-2
                PERFORM DO-GETREPLY1
                PERFORM DO-GETREPLY2
    ELSE
                     PERFORM DO-GETREPLY2
                     PERFORM DO-GETREPLY1
    END-IF.
    . . .
   leave application
```

```
DO-GETRPLY1.
   SET TPGETHANDLE TO TRUE IN TPSVCDEV-REC-1.
   SET TPCHANGE TO TRUE IN TPSVCDEV-REC-1.
   SET TPBLOCK TO TRUE IN TPSVCDEV-REC-1.
   SET TPNOTIME TO TRUE IN TPSVCDEV-REC-1.
   SET TPSIGRSTRT TO TRUE IN TPSVCDEV-REC-1.
   CALL "TPGETRPLY" USING TPSVCDEF-REC-1
                    TPTYPE-REC-1
                    DATA-REC-1
                    TPSTATUS-REC.
   IF NOT TPOK


            error processing
 DO-GETRPLY2
   SET TPGETHANDLE TO TRUE IN TPSVCDEV-REC-2.
   SET TPCHANGE TO TRUE IN TPSVCDEV-REC-2.
   SET TPBLOCK TO TRUE IN TPSVCDEV-REC-2.
   SET TPNOTIME TO TRUE IN TPSVCDEV-REC-2.
   SET TPSIGRSTRT TO TRUE IN TPSVCDEV-REC-2.
   CALL "TPGETRPLY" USING TPSVCDEF-REC-2
                    TPTYPE-REC-2
                    DATA-REC-2
                    TPSTATUS-REC.
   IF NOT TPOK
          error processing
```

7 会話型クライアントおよびサーバのコーディング

ここでは、次の内容について説明します。

- 会話型通信の概要
- アプリケーションへの参加
- 接続の確立
- メッセージの送受信
- 会話の終了
- 会話型のクライアントおよびサーバのビルド
- 会話型通信イベント

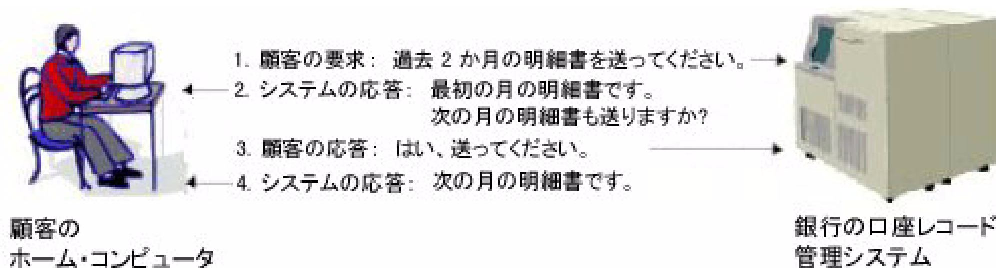
会話型通信の概要

会話型通信は BEA Tuxedo システムのメッセージ交換のパラダイムで、人の会話に似た通信がクライアントとサーバ間でインプリメントされています。この通信方法では、クライアント (イニシエータ) とサーバ (下位サーバ) 間で仮想接続が行われて、双方で会話の状態に関する情報が保持されます。この接続は、接続を終了するイベントが発生するまで継続します。

会話型通信では、クライアントとサーバ間に半二重接続が確立されます。半二重接続では、メッセージが 1 方向だけに送信されます。接続に関する制御は、イニシエータから下位サーバへ、またはその逆に移ります。制御を持つプロセスがメッセージを送信でき、持たないプロセスは受信しかできません。

以下に銀行業務のオンライン・アプリケーションを例に、BEA Tuxedo アプリケーションで行われる会話型通信について説明します。この例では、銀行の顧客が過去 2 か月間の当座預金の明細書を要求しています。

図 7-1 銀行業務オンライン・アプリケーションでの会話型通信



1. 顧客が過去 2 か月間の当座預金口座の明細書を要求します。
2. 口座レコード管理システムは、この要求に対する応答として、当座預金口座の最初の月の明細書を送信します。次に、次の月の明細書を送信するかどうかを `More` プロンプトで確認します。
3. 顧客は `More` プロンプトを選択して、次の月の明細書を要求します。

注記　口座レコード管理システムでは、状態情報を保持して、顧客が More プロンプトを選択した場合にどの明細書を送るのか認識できるようにする必要があります。

4. 口座レコード管理システムは、次の月の明細書を送信します。

要求 / 応答型通信の場合と同じように、BEA Tuxedo システムでは型付きレコードを使用してデータが渡されます。アプリケーションがレコード・タイプを認識できることが必要です。レコード・タイプの詳細については、3-1 ページの「型付きレコードの概要」を参照してください。

会話型のクライアントおよびサーバには、次の特徴があります。

- 会話型のクライアントとサーバ間の論理接続は、接続が終了するまで継続します。
- 会話型のクライアントとサーバ間の接続で転送できるメッセージの数には制限はありません。
- クライアントとサーバとの会話では、データの送受信に TPSEND および TPRECV ルーチンが使用されます。

会話型通信は、次の点で要求 / 応答型通信と異なります。

- 会話型クライアントは、サービスを要求するときに TPCALL または TPACALL ではなく、TPCONNECT を使用します。
- 会話型クライアントは、会話型サーバにサービス要求を送信します。
- 会話型サービスを定義するために、コンフィギュレーション・ファイルに会話型サーバの一部が予約されています。
- 会話型サーバは、TPFORWAR を使用して呼び出しを行うことはできません。

アプリケーションへの参加

会話型クライアントは、サービスへの接続を確立する前に、TPINITIALIZE を呼び出してアプリケーションに参加する必要があります。詳細については、4-1 ページの「クライアントのコーディング」を参照してください。

接続の確立

TPCONNECT(3cbl) ルーチンは、会話を行うための接続を確立します。

TPCONNECT ルーチンの呼び出しには、次の文法を使用します。

```
01 TPSVCDEF-REC.
     COPY TPSVCDEF.

01 TPTYPE-REC.
     COPY TPTYPE.

01 DATA-REC.
     COPY User Data.

01 TPSTATUS-REC.
     COPY TPSTATUS.

CALL "TPCONNECT" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

TPSVCDEF-REC レコードの詳細については、5-12 ページの「サービスの定義」を参照してください。TPTYPE-REC レコードの詳細については、3-7 ページの「型付きレコードの定義」を参照してください。

接続が確立されると同時に、DATA-REC を使用して、LEN IN TPTYPE-REC に指定されたデータ長でデータを送信できます。DATA-REC のデータの REC-TYPE と SUB-TYPE は、呼び出されたサービスで認識できるタイプであることが必要です。データが送信されていない場合は、REC-TYPE の値は SPACES であり、DATA-REC と LEN は無視されます。

TPCONNECT または TPSVCSTART によって接続が確立されると、BEA Tuxedo システムから通信ハンドル (COMM-HANDLE IN TPSVCDEF-REC) が返されます。この COMM-HANDLE は、特定の会話で以降に送られるメッセージを識別するために使用されます。クライアントまたは会話型サービスは、複数の会話に同時に参加できます。最大 64 個の会話を同時に行うことができます。

TPCONNECT の呼び出しが失敗すると、対応するエラー・コードが TP-STATUS に設定されます。エラー・コードについては、『BEA Tuxedo COBOL リファレンス』の TPCONNECT(3cbl) を参照してください。

次のコード例は、TPCONNECT ルーチンの使用方法を示しています。

コード リスト 7-1　会話型接続の確立

```
    . . .
* 送信するレコードを準備
  MOVE "HELLO" TO DATA-REC.
  MOVE 5 TO LEN.
  MOVE "STRING" TO REC-TYPE.
*
  SET TPBLOCK TO TRUE.
  SET TPNOTRAN TO TRUE.
  SET TPNOTIME TO TRUE.
  SET TPSIGRSTRT TO TRUE.
  SET TPSENDONLY TO TRUE.
*
  CALL "TPCONNECT" USING TPSVCDEF-REC
                   TPTYPE-REC
                   DATA-REC
                   TPSTATUS-REC.
  IF NOT TPOK
           error processing ...
  ELSE
      COMM-HANDLE is valid.
```

メッセージの送受信

BEA Tuxedo システムで会話型接続が確立されると、イニシエータと下位サーバ間の通信は送信呼び出しと受信呼び出しによって行われます。接続の制御を持つプロセスは、TPSEND(3cbl) ルーチンを使用してメッセージを送信できます。制御がないプロセスは、TPRECV(3cbl) ルーチンを使用してメッセージを受信できます。

注記　発信元 (クライアント) は、最初に TPCONNECT 呼び出しの TPSENDONLY または TPRECVONLY フラグを使用して、どのプロセスが制御を持っているのかを判別します。TPSENDONLY は、発信元が制御を持つことを示します。TPRECVONLY は、呼び出されたサービスに制御が渡されたことを示します。

メッセージの送信

メッセージを送信するには TPSEND(3cbl) ルーチンを使用します。このルーチンには、次の文法を使用します。

```
01 TPSVCDEF-REC.
     COPY TPSVCDEF.

01 TPTYPE-REC.
     COPY TPTYPE.

01 DATA-REC.
     COPY User Data.

01 TPSTATUS-REC.
     COPY TPSTATUS.

CALL "TPSEND" USING TPSVCDEF-REC TPTYPE-REC USER-DATA-REC TPSTATUS-REC.
```

TPSVCDEF-REC レコードの詳細については、5-12 ページの「サービスの定義」を参照してください。TPTYPE-REC レコードの詳細については、3-7 ページの「型付きレコードの定義」を参照してください。

TPSEND ルーチンが失敗すると、対応するエラー・コードが TP-STATUS に設定されます。エラー・コードについては、『BEA Tuxedo COBOL リファレンス』の TPSEND(3cbl) を参照してください。

TPSEND ルーチンを呼び出すたびに、制御を渡す必要はありません。一部のアプリケーションでは、TPSEND の呼び出しを認められているプロセスが、制御をほかのプロセスに渡すまで、現在のタスクで必要な回数だけ呼び出しを実行できます。ただし、プログラムのロジックによっては、会話が継続する間は常に 1 つのプロセスが接続の制御を持たなければならないアプリケーションもあります。

次のコード例は、TPSEND ルーチンの呼び出し方法を示しています。

コード リスト 7-2　会話モードでのデータ送信

```
   . . .
   SET TPNOBLOCK TO TRUE.
   SET TPNOTIME TO TRUE.
   SET TPSIGRSTRT TO TRUE.
   SET TPRECVONLY TO TRUE.
*
   CALL "TPSEND" USING TPSVCDEF-REC
                 TPTYPE-REC
                 DATA-REC
                 TPSTATUS-REC.
   IF NOT TPOK
       error processing . . .
```

メッセージの受信

オープン接続を介してデータを受信するには、TPRECV(3cbl) ルーチンを使用します。この関数には、次の文法を使用します。

```
01 TPSVCDEF-REC.
     COPY TPSVCDEF.

01 TPTYPE-REC.
     COPY TPTYPE.
```

```
01 DATA-REC.
     COPY User Data.

01 TPSTATUS-REC.
     COPY TPSTATUS.

CALL "TPRECV" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

TPSVCDEF-REC レコードの詳細については、5-12 ページの「サービスの定義」を参照してください。TPTYPE-REC レコードの詳細については、3-7 ページの「型付きレコードの定義」を参照してください。

次のコード例は、TPRECV ルーチンの使用方法を示しています。

コード リスト 7-3 会話型でのデータ受信

```
  . . .
  SET TPNOCHANGE TO TRUE.
  SET TPBLOCK TO TRUE.
  SET TPNOTIME TO TRUE.
  SET TPSIGRSTRT TO TRUE.
*
  MOVE LENGTH OF DATA-REC TO LEN.
*
  CALL "TPRECV" USING TPSVCDEF-REC
                   TPTYPE-REC
                   DATA-REC
                   TPSTATUS-REC.
  IF NOT TPOK
      error processing . . .
```

会話の終了

次の場合、接続が切断されて会話が正常に終了します。

- 単純な会話で、TPRETURN の呼び出しが成功した場合
- 接続が階層構造になった複雑な会話で、一連の TPRETURN の呼び出しが成功した場合

- グローバル・トランザクションの場合 (9-1 ページの「グローバル・トランザクションのコーディング」を参照)

注記 TPRETURN ルーチンについては、6-1 ページの「クライアントおよびサーバへの要求 / 応答のコーディング」を参照してください。

以下の節では、会話を正常に終了する方法について、2 つの例を挙げて説明します。これらの会話には、TPRETURN 関数を使用するグローバル・トランザクションは含まれていません。

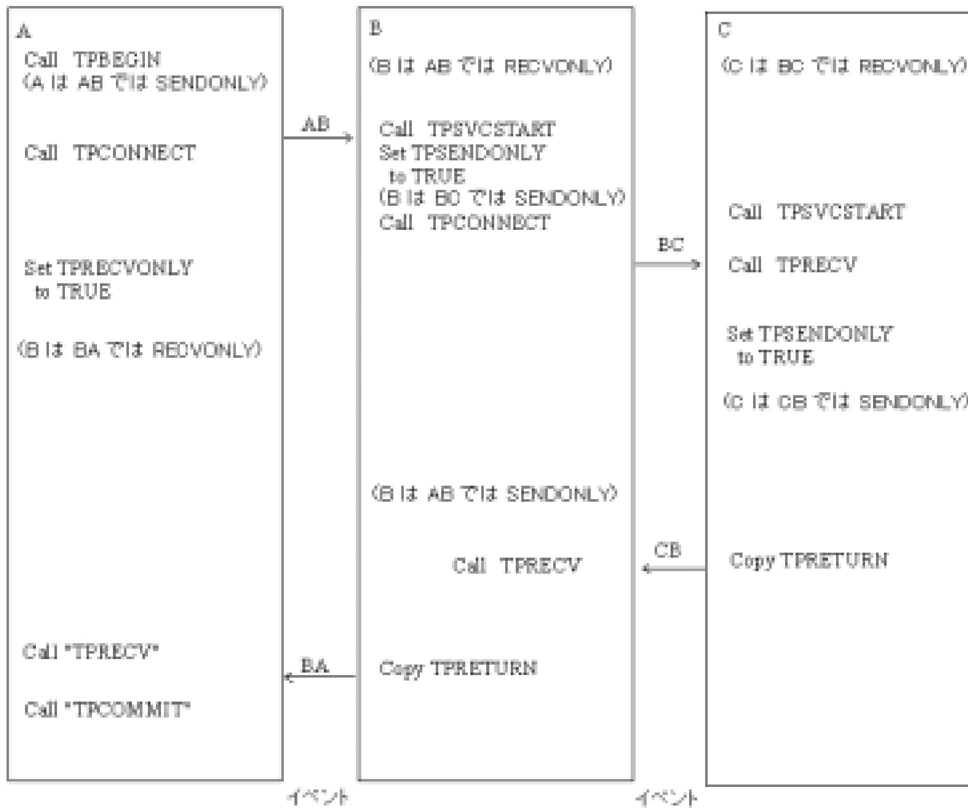
最初の例では、2 つのコンポーネント間の単純な会話を終了する方法を示します。2 番目の例では、会話が階層構造になっている複雑な会話を終了する方法を示します。

接続がオープンになっているときに会話を終了すると、エラーが返されます。その場合、TPCOMMIT または TPRETURN は失敗します。

例 : 単純な会話の終了

次の図は、正常に終了する A と B 間の単純な会話を示しています。

図 7-2 正常に終了する単純な会話



次の順序で処理が行われます。

1. A は、TPSENDONLY フラグで TPCONNECT を呼び出して接続を設定します。このフラグは、B が会話の受信側であることを示します。
2. A は TPRECVONLY フラグで TPSEND を呼び出して、接続の制御を B に移します。その結果、TPEV_SENDONLY イベントが生成されます。

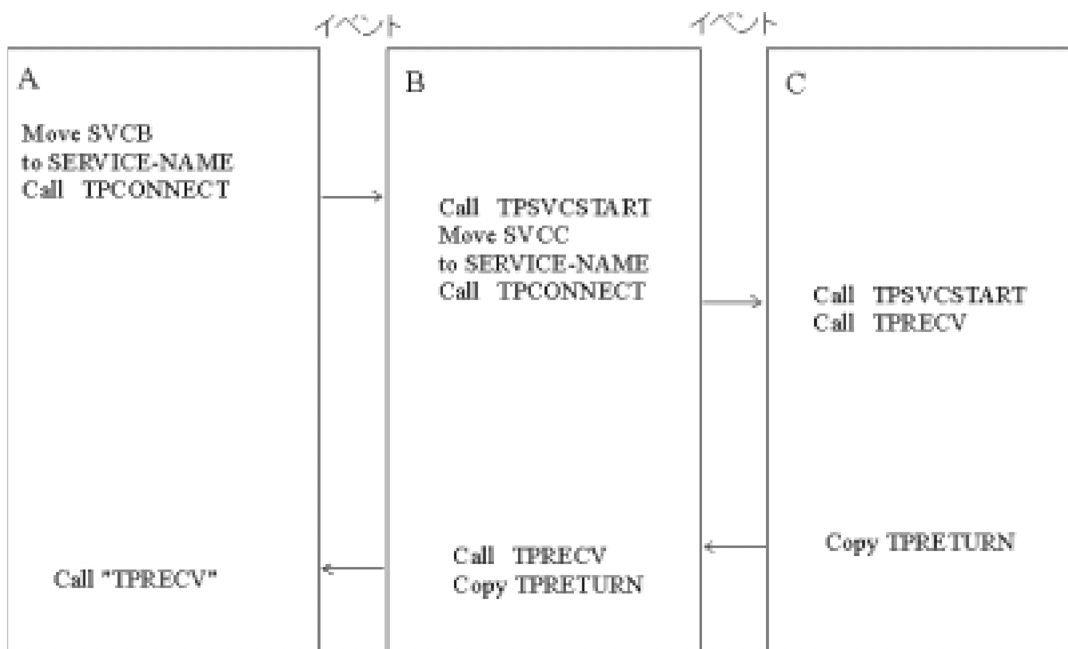
3. B が次に TPRECV を呼び出すと TP-STATUS に TPEEVENT が設定されます。TPEVENT に TPEV_SENDONLY が返されて、制御が B に移ったことが示されます。
4. B は、TPRETURN-VAL IN TPSVCRET に TPSUCCESS を設定して TPRETURN を呼び出します。この呼び出しにより、A に対して TPEV_SVCSUCC イベントが生成され、接続が正常に切断されます。
5. A は、TPRECV を呼び出して、イベントが発生したことと会話が終了したことを認識します。イベント TPEV_SVCFAIL が発生した場合でも、この TPRECV への呼び出しでデータを受信できます。

注記　この例では、A はクライアントまたはサーバのどちらでもかまいませんが、B はサーバでなければなりません。

例 : 階層構造の会話の終了

次の図は、正常に終了する階層構造の会話を示しています。

図 7-3 接続の階層構造



この例では、サービス B は会話のメンバで、2 番目のサービス C との接続を開始しています。つまり、A - B 間 と B - C 間という 2 つのアクティブな接続が存在しています。B がこの両方の接続を制御している場合に TPRETURN の呼び出しを行うと、呼び出しは失敗し、すべてのオープン接続に TPEV_SVCERR イベントが通知され、接続が切断されます。

両方の接続を正常に終了するには、次の処理を順に行います。

1. B は、C との接続に TPRECVONLY フラグを設定して TPSEND を呼び出し、B - C 間の接続の制御を C に渡します。
2. C は、状況に応じて TPRETURN-VAL IN TPSVCRET に TPSUCCESS、TPFAIL、または TPEXIT を設定して、TPRETURN を呼び出します。
3. B は、TPRETURN を呼び出し、A にイベント (TPEV_SVCSUCC または TPEV_SVCFAIL) を通知します。

注記　会話型サービスは、別のサービスと通信するために要求 / 応答型の呼び出しを行うことができます。そのため、前述の例では、B から C への呼び出しに TPCONNECT ではなく TPCALL または TPACALL を使用することもできます。ただし、会話型サービスが TPFORWAR を呼び出すことはできません。

会話の切断

エラーの発生により接続を終了する唯一の方法は、TPDISCON(3cbl) ルーチンを呼び出すことです。これは、「プラグを抜くこと」と同じです。このルーチンを呼び出すことができるのは、会話のイニシエータ (クライアント) だけです。

注記　この方法で会話を終了することはお勧めしません。アプリケーションを正常に終了するには、下位サーバで TPRETURN ルーチンを呼び出します。

TPDISCON ルーチンの呼び出しには、次の文法を使用します。

```
01 TPSVCDEF-REC.
     COPY TPSVCDEF.

01 TPSTATUS-REC.
     COPY TPSTATUS.

CALL "TPDISCON" USING TPSVCDEF-REC TPSTATUS-REC.
```

COMM-HANDLE 引数は、接続が確立したときに TPCONNECT ルーチンによって返される通信ハンドルを指定します。

TPDISCON ルーチンは、接続の相手側のサービスに対して TPEV_DISCONIMM イベントを生成し、COMM-HANDLE を無効にします。トランザクションが実行中の場合、そのトランザクションはアボートし、データは失われます。

COMM-HANDLE で接続の開始側と識別されていないサービスから TPDISCON が呼び出されると、そのルーチンは失敗し、エラー・コード TPEBADDESC が生成されます。

イベントおよびエラー・コードの全リストとその説明については、『BEA Tuxedo COBOL リファレンス』の TPDISCON(3cbl) を参照してください。

会話型のクライアントおよびサーバのビルド

次のコマンドを使用して、会話型のクライアントおよびサーバをビルドします。

- buildclient() (『C 言語を使用した BEA Tuxedo アプリケーションのプログラミング』の「クライアントのビルド」を参照)
- buildserver() (『C 言語を使用した BEA Tuxedo アプリケーションのプログラミング』の「サーバのビルド」を参照)

会話型サービスと要求 / 応答型サービスでは、次の操作を行うことはできません。

- 同じサーバにクライアントとサーバを作成すること
- クライアントとサーバに同じ名前を付けること

会話型通信イベント

BEA Tuxedo システムの会話型通信では、5 つのイベントが認識されます。これらのイベントはすべて TPRECV に通知でき、そのうちの 3 つは TPSEND にも通知できます。

次の表は、イベント、そのイベントを受け取るルーチン、および各イベントの簡単な説明を示しています。

表 7-1 会話型通信のイベント

イベント	イベントを受け取る関数	説明
TPEV_SENDONLY	TPRECV	接続の制御が渡されました。この時点で、このプロセスは TPSEND を呼び出すことができます。

表 7-1 会話型通信のイベント

イベント	**イベントを受け取る関数**	**説明**
TPEV_DISCONIMM	TPSEND、TPRECV、TPRETURN	接続は既に切断され、通信を継続することはできません。TPDISCON ルーチンはこのイベントを接続の開始側に通知します。下位サービスとの接続がオープンしたままになっている場合に、TPRETURN が呼び出されたときは、このイベントをすべてのオープン接続に送信します。接続はエラーが原因で切断されます。トランザクションが存在している場合は、アボートします。
TPEV_SVCERR	TPSEND	接続の開始側が受信します。通常は、下位プログラムが接続の制御を持たない場合に、TPRETURN を呼び出したことを示します。
	TPRECV	接続の開始側が受信します。下位プログラムが TPSUCCESS または TPFAIL、および妥当なデータ・バッファを使用して TPRETURN を呼び出したが、エラーが発生して呼び出しが完了しなかったことを示します。
TPEV_SVCFAIL	TPSEND	接続の開始側が受信します。下位プログラムが接続の制御を持たない場合に TPRETURN を呼び出し、TPFAIL または TPEXIT、およびデータなしで TPRETURN が呼び出されたことを示します。
	TPRECV	接続の開始側が受信します。下位サービスの処理が正常に終了しなかったこと (つまり、TPRETURN が TPFAIL または TPEXIT で呼び出されたこと) を示します。
TPEV_SVCSUCC	TPRECV	接続の開始側が受信します。下位サービスの処理が正常に終了したこと (つまり、TPRETURN が TPSUCCESS で呼び出されたこと) を示します。

8 イベント・ベースのクライアントおよびサーバのコーディング

ここでは、次の内容について説明します。

- イベントの概要
- 任意通知型メッセージ・ハンドラの定義
- 任意通知型メッセージの送信
- 任意通知型メッセージの確認
- 任意通知型メッセージの取得
- イベントのサブスクライブ
- イベントに対するサブスクリプションの削除
- イベントのポスト

イベントの概要

イベント・ベースの通信では、特定の状況 (イベント) が発生すると、BEA Tuxedo システムのプロセスに通知されます。

BEA Tuxedo システムには、次の 2 種類のイベント・ベースの通信があります。

- 任意通知型イベント
- ブローカ・イベント

任意通知型イベント

任意通知型イベントは、メッセージを待たないクライアント・プログラム、またはメッセージを要求しないクライアント・プログラムとの通信に使用されるメッセージです。

ブローカ・イベント

ブローカ・イベントを使用すると、メッセージの受信と配信を行う「無名」ブローカを介して、クライアントとサーバが透過的に通信できるようになります。このブローカを使用した通信は、BEA Tuxedo システムの基本要素であるクライアント / サーバ通信パラダイムの 1 つです。

イベント・ブローカは、イベント・ポスト・メッセージを受信してフィルタ処理し、それらのメッセージをサブスクライバに配布する BEA Tuxedo のサブシステムです。ポスト元とは、特定のイベントが発生したときにそれをイベント・ブローカに報告 (ポスト) する BEA Tuxedo システムのプロセスです。サブスクライバとは、特定のイベントがポストされたときに常に通知する必要がある BEA Tuxedo システムのプロセスです。

BEA Tuxedo システムでは、サービスの要求側と提供側の数の比率が一定である必要はなく、任意の数のポスト元が任意の数のサブスクライバに対してメッセージをポストできます。ポスト元は単にイベントをポストするだけ

で、情報を受信するプロセスや、情報の処理方法については関知しません。サブスクライバには指定されたイベントが通知されますが、その情報のポスト元は通知されません。このように、イベント・ブローカでは位置透過性が実現されます。

通常、イベント・ブローカ・アプリケーションは、例外イベントを処理します。アプリケーションの設計者は、アプリケーション内でどのイベントを例外イベントとして定義して監視する必要があるのかを決定しなければなりません。たとえば、銀行業務アプリケーションでは、高額な引き出しがあったときにイベントがポストされるように設定し、すべての引き出しに対してイベントがポストされる必要はありません。また、すべてのユーザがそのイベントをサブスクライブする必要はありません。支店長だけに通知すれば十分です。

通知処理

イベントがポストされると、イベント・ブローカはイベントをサブスクライブしているクライアントまたはサーバに対して、1 つ以上の通知処理を行います。次の表は、イベント・ブローカが行う通知処理の種類を示しています。

表 8-1 イベント・ブローカの通知処理

通知処理	**説明**
任意通知型通知メッセージ	クライアントは、TPNOTIFY ルーチンで送信された場合と同じように、クライアントの任意通知型メッセージ処理ルーチンでイベント通知メッセージを受信します。
サービス呼び出し	サーバは、TPACALL によって送信された場合と同じように、サービス・ルーチンに対する入力としてイベント通知メッセージを受け取ります。

通知処理	**説明**
信頼性の高いキュー	イベント通知メッセージは、TPDEQUEUE(3cbl) を使用して、BEA Tuxedo システムの高信頼性キューに格納されます。イベント通知レコードは、内容が要求されるまで格納されます。BEA Tuxedo システムのクライアントまたはサーバ・プロセスで TPDEQUEUE(3cbl) を呼び出して、この通知レコードを取り出すことができます。または、TMQFORWARD(5) を設定して、通知レコードを取り出す BEA Tuxedo システムのサービス・ルーチンを自動的にディスパッチすることもできます。 /Q の詳細については、『BEA Tuxedo /Q コンポーネント』を参照してください。

アプリケーション管理者は、BEA Tuxedo の管理用 API を使用して、次の通知処理を行う EVENT_MIB(5) に関する追加情報 エントリを作成できます。

- システム・コマンドの呼び出し
- ディスク上のシステムのログ・ファイルへのメッセージの書き込み

注記　EVENT_MIB(5) に関する追加情報 エントリを作成できるのは、BEA Tuxedo アプリケーション管理者だけです。

EVENT_MIB(5) に関する追加情報 の詳細については、『BEA Tuxedo のファイル形式とデータ記述方法』を参照してください。

イベント・ブローカ・サーバ

TMUSREVT は BEA Tuxedo システムで提供されるサーバで、ユーザ定義のイベントに対するイベント・ブローカとして動作します。TMUSREVT はイベント・レポート用のメッセージ・レコードを処理し、それらのレコードをフィルタ処理して配信します。イベントのブローカ処理を行うには、BEA Tuxedo アプリケーション管理者がこれらのサーバを 1 つ以上起動する必要があります。

TMSYSEVT は BEA Tuxedo システムで提供されるサーバで、システム定義のイベントに対するイベント・ブローカとして動作します。TMSYSEVT と TMUSREVT は似ています。ただし、別個のサーバが提供されているので、ア

プリケーション管理者はこの 2 種類のイベント通知に対して異なる処理方法を取り入れることができます。詳細については、『BEA Tuxedo アプリケーションの設定』を参照してください。

システム定義のイベント

BEA Tuxedo システムでは、システムの警告と障害に関連する定義済みの特定のイベントが検出されてポストされます。これらの処理はイベント・ブローカによって行われます。たとえば、システム定義のイベントには、設定の変更、状態の変更、接続の障害、マシンの分断などがあります。イベント・ブローカによって検出されるシステム定義のイベントの全リストについては、『BEA Tuxedo のファイル形式とデータ記述方法』の `EVENTS(5)` を参照してください。

システム定義のイベントは、BEA Tuxedo システムで定義されているので、ポストする必要はありません。システム定義のイベント名は、アプリケーション定義のイベント名とは異なり、必ずドット (&dlq;.&drq;) で始まります。アプ “ ン定義のイ ” ドットで始めることはできません。

クライアントとサーバは、システム定義のイベントをサブスクライブできます。ただし、システム定義のイベントは、アプリケーション内のすべてのクライアントが使用するのではなく、主にアプリケーション管理者が使用します。

イベント・ブローカをアプリケーションに組み込む場合、イベント・ブローカが多数のサブスクライバに大量の配信を行うためのシステムではないことを考慮してください。発生するすべての動作に対してイベントをポストしないでください。また、すべてのクライアントとサーバがイベントをサブスクライブする必要はありません。イベント・ブローカに負荷がかかると、システムのパフォーマンスに影響し、通知が行われなくなる場合があります。負荷を最小限にするには、『Tuxedo システムのインストール』で説明するように、アプリケーション管理者がオペレーティング・システムの IPC 資源を慎重に調整する必要があります。

イベント・ブローカ・プログラミング・インターフェイス

イベント・ブローカ・プログラミング・インターフェイスは、ワークステーションを含むすべての BEA Tuxedo システムのサーバ・プロセスおよびクライアント・プロセスに対して C 言語および COBOL 言語で使用できます。

プログラマは、次の処理をコーディングします。

1. クライアントまたはサーバは、アプリケーション定義のイベント名にレコードをポストします。
2. ポストされたレコードは、そのイベントをサブスクライブしている任意の数のプロセスに転送されます。

サブスクライバへの通知方法にはいろいろな方法 (「通知処理」を参照) があり、イベントはフィルタ処理されます。通知処理とフィルタ処理は、プログラミング・インターフェイスと BEA Tuxedo システムの管理用 API を使用して設定します。

任意通知型メッセージ・ハンドラの定義

任意通知型メッセージ・ハンドラを定義するには、次の文法を使用して TPSETUNSOL(3cbl) ルーチンを呼び出します。

```
01 CURR-ROUTINE    PIC S9(9) COMP-5.
01 PREV-ROUTINE    PIC S9(9) COMP-5.
01 TPSTATUS-REC.
   COPY TPSTATUS.
CALL "TPSETUNSOL" USING CURR-ROUTINE PREV-ROUTINE TPSTATUS-REC.
```

TPSETUNSOL を使用すると、任意通知型メッセージが BEA Tuxedo システム・ライブラリによって受信されたときに呼び出されるルーチンをクライアントが識別できるようになります。TPSETUNSOL を初めて呼び出す前に、クライアントに代わって BEA Tuxedo システム・ライブラリが受信した任意通知型メッセージはログに記録されるだけで無視されます。システムが通知や検出に使用する処理方法は、アプリケーションのデフォルト設定によって決まります。このデフォルト設定は、クライアントごとに変更できます。詳細については、『BEA Tuxedo COBOL リファレンス』の TPINITIALIZE(3cbl) を参照してください。

CURR-ROUTINE パラメータは、任意通知型メッセージを処理する 16 種類の定義済みルーチンを識別します。これらのルーチンには、C ルーチンとしては tm_displatch1 ～ _tm_dispatch8 の 8 種類、COBOL ルーチンとしては TMDISPATCH9 ～ TMDISPATCH16 の 8 種類があります。また、CURR-ROUTINE に

0 を設定した場合、BEA Tuxedo システム・ライブラリがクライアントに代わって受信した任意通知型メッセージはログに記録されるだけで無視されます。C ルーチンは、TPSETUNSOL(3cbl) のパラメータ定義に準拠している必要があります。COBOL ルーチンを使用する場合は、TPGETUNSOL を呼び出してデータを受信する必要があります。

次のコード例は、COBOL プログラムでの任意通知型ルーチンの設定方法を示しています。

コード リスト 8-1　任意通知型ルーチンの設定

```
*
* TPSETUNSOL を呼び出します。- COBOL 任意通知型メッセージ・ハンドラを設定します。
*   TMDISPATCH9 ルーチンが呼び出されます。
*
   MOVE 9 to CURR-ROUTINE.
   CALL "TPSETUNSOL" USING
                      CURR-ROUTINE
                      PREV-ROUTINE
                      TPSTATUS-REC.
   IF NOT TPOK
            Routine TMDISPATCH9 will receive unsolicited messages
   ELSE
              Process error condition
```

任意通知型メッセージの送信

BEA Tuxedo システムでは、要求 / 応答型呼び出しまたは会話型通信の処理を妨げずに、クライアント・プロセスに任意通知型メッセージを送信できます。

任意通知型メッセージは、名前、または以前に処理されたメッセージと共に受信した識別子を使用して、クライアント・プロセスに送信できます。名前による送信には TPBROADCAST(3cbl)、識別子による送信には

TPNOTIFY(3cbl) を使用します。TPBROADCAST で送信されるメッセージの発信元は、サービスまたは別のクライアントです。TPNOTIFY で送信されるメッセージの発信元は、サービスだけです。

名前によるメッセージのブロードキャスト

TPBROADCAST(3cbl) ルーチンを使用すると、アプリケーションの登録されたクライアントにメッセージが送信されます。TPBROADCAST ルーチンは、サービスまたは別のクライアントから呼び出すことができます。登録されたクライアントとは、TPINITIALIZE を呼び出したが、まだ TPTERM を呼び出してはいないクライアントのことです。

TPBROADCAST ルーチンの呼び出しには、次の文法を使用します。

```
01 TPBCTDEF-REC.
   COPY TPBCTDEF.
01 TPTYPE-REC.
   COPY TPTYPE.
01 DATA-REC.
   COPY User Data.
01 TPSTATUS-REC.
   COPY TPSTATUS.
CALL "TPBROADCAST" USING TPBCTDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

次の表は、TPBCTDEF-REC データ構造体のメンバを示しています。

表 8-2TPBCTDEF-REC データ構造体のメンバ

メンバ	説明
LMID	クライアントの論理マシン識別子を指すポインタ。SPACES 値をワイルドカードとして使用できるので、複数のクライアントにメッセージを送信できます。
USRNAME	クライアント・プロセスのユーザ名が存在する場合、そのユーザ名。SPACES 値をワイルドカードとして使用できるので、複数のクライアントにメッセージを送信できます。

メンバ	説明
CLTNAME	クライアント・プロセスのクライアント名が存在する場合、そのクライアント名。NULL 値をワイルドカードとして使用できるので、複数のクライアントにメッセージを送信できます。
設定値（TPBLOCK-FLAG など）	TPBROADCAST コマンドの設定値。使用できる設定値については、『BEA Tuxedo COBOL リファレンス』の TPBROADCAST(3cbl) を参照してください。

TPTYPE-REC レコードについては、5-12 ページの「サービスの定義」を参照してください。

次のコード例は、すべてのクライアントを送信先として TPBROADCAST を呼び出す方法を示しています。送信メッセージは、STRING 型レコード内にあります。

コード リスト 8-2　TPBROADCAST の使用

```
    . . .
**************************************************
* ブロードキャストにレコードを用意します。
**************************************************
    MOVE "HELLO, WORLD" TO DATA-REC.
    MOVE 11 TO LEN.
    MOVE "STRING" TO REC-TYPE.
*
    SET TPNOBLOCK TO TRUE.
    SET TPNOTIME TO TRUE.
    SET TPSIGRSTRT TO TRUE.
*
    MOVE SPACES TO LMID.
    MOVE SPACES TO USRNAME.
    MOVE SPACES TO CLTNAME.
    CALL "TPBROADCAST" USING TPBCTDEF-REC
                   TPTYPE-REC
                   DATA-REC
                   TPSTATUS-REC.
    IF NOT TPOK
        error processing
```

識別子によるメッセージのブロードキャスト

TPNOTIFY(3cbl) ルーチンを使用すると、以前に処理されたメッセージと共に受信した識別子を使用してメッセージがブロードキャストされます。TPNOTIFY ルーチンは、サービスからのみ呼び出すことができます。

TPNOTIFY ルーチンの呼び出しには、次の文法を使用します。

```
01 TPSVCDEF-REC.
   COPY TPSVCDEF.
01 TPTYPE-REC.
   COPY TPTYPE.
01 DATA-REC.
   COPY User Data.
01 TPSTATUS-REC.
   COPY TPSTATUS.
CALL "TPNOTIFY" USING TPSVCDEF-REC TPTYPE-REC DATA-REC TPSTATUS-REC.
```

TPSVCDEF-REC データ構造体については、9-1 ページの「グローバル・トランザクションのコーディング」を参照してください。TPTYPE-REC レコードについては、5-12 ページの「サービスの定義」を参照してください。

任意通知型メッセージの確認

クライアントが「ディップ・イン」通知モードで実行されている場合に任意通知型メッセージがあるかどうかを確認するには、次の文法を使用して TPCHKUNSOL(3cbl) ルーチンを呼び出します。

```
01 MSG-NUM           PIC S9(9)  COMP-5.
01 TPSTATUS-REC.
   COPY TPSTATUS.
CALL "TPCHKUNSOL" USING MSG-NUM TPSTATUS-REC.
```

未処理のメッセージがある場合、TPSETUNSOL で指定された任意通知型メッセージ処理ルーチンが呼び出されます。処理が終了すると、このルーチンは処理した任意通知型メッセージの数を返し、[TPOK] に TP-STATUS を設定します。

クライアントがシグナル・ベースまたはスレッド・ベースの通知モードで実行されている場合、またはクライアントが任意通知型メッセージを無視している場合にこのルーチンを呼び出すと、ルーチンは何も処理を行わずにすぐに制御を戻します。

次の例は、任意通知型メッセージが到着しているかどうかを確認する方法を示しています。

コード リスト 8-3　任意通知型メッセージの到着

```
*
* 任意通知型メッセージを確認します。
*
    CALL "TPCHKUNSOL" USING MESS-NUM
                      TPSTATUS-REC.
*
    IF TPOK
         IF MESS-NUM IS = 0
            No messages were processed by the
      unsolicited routine
    ELSE
         MESS-NUM    number of messages were
      processed by the unsolicited routine
    END-IF
    ELSE
        process error
   END-IF
```

任意通知型メッセージの取得

任意通知型メッセージを取得するには、TPGETUNSOL(3cbl) ルーチンを呼び出す必要があります。このルーチンは、任意通知型メッセージ・ハンドラからのみ呼び出すことができます。TPGETUNSOL ルーチンの呼び出しには、次の文法を使用します。

```
01 TPTYPE-REC.
   COPY TPTYPE.
01 DATA-REC.
```

```
   COPY User data.
01 TPSTATUS-REC.
   COPY TPSTATUS.
CALL "TPGETUNSOL" USING TPTYPE-REC DATA-REC TPSTATUS-REC.
```

TPTYPE-REC レコードについては、5-12 ページの「サービスの定義」を参照してください。

次の例は、任意通知型メッセージの取得方法を示しています。

コード リスト 8-4　任意通知型メッセージの取得

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  TMDISPATCH9.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  USL-486.
 OBJECT-COMPUTER.  USL-486.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
 01 TPTYPE-REC.
    COPY TPTYPE.
*
 01 TPSTATUS-REC.
    COPY TPSTATUS.
*
 01  DATA-REC                 PIC X(1000).
*
 PROCEDURE DIVISION.
*
 A-000.
*
   MOVE "CARRAY" TO REC-TYPE.
   MOVE 1000  TO LEN.
   CALL "TPGETUNSOL" USING TPTYPE-REC
                     DATA-REC
                     TPSTATUS-REC.
   IF  NOT TPOK
             error processing
*
   Process message
   DISPLAY "TPGETUNSOL IS TPOK".
   DISPLAY "MESSAGE IS" DATA-REC.
   DISPLAY "LENGTH IS" LEN.
```

```
   EXIT PROGRAM.
*
```

イベントのサブスクライブ

TPSUBSCRIBE(3cbl) ルーチンを使用すると、BEA Tuxedo システムのクライアントまたはサーバがイベントをサブスクライブできるようになります。

サブスクライバは、任意通知型通知メッセージ、サービス呼び出し、高い信頼性のキュー、またはアプリケーション管理者が設定する別の通知方法によって、通知を受け取ります。別の通知方法の設定については、『BEA Tuxedo アプリケーションの設定』を参照してください。

TPSUBSCRIBE ルーチンの呼び出しには、次の文法を使用します。

```
01  TPEVTDEF-REC.
  COPY TPEVTDEF.

01  TPQUEDEF-REC.
  COPY TPQUEDEF.

01 TPSTATUS-REC.
  COPY TPSTATUS.

CALL "TPSUBSCRIBE" USING TPEVTDEF-REC TPQUEDEF-REC TPSTATUS-REC
```

TPEVTDEF-REC データ構造体の文法は、次のとおりです。

```
05 TPBLOCK-FLAG      PIC S9(9) COMP-5.
  88 TPBLOCK        VALUE 0.
  88 TPNOBLOCK       VALUE 1.
05 TPTRAN-FLAG      PIC S9(9) COMP-5.
  88 TPTRAN         VALUE 0.
  88 TPNOTRAN        VALUE 1.
05 TPREPLY-FLAG      PIC S9(9) COMP-5.
  88 TPREPLY        VALUE 0.
  88 TPNOREPLY       VALUE 1.
05 TPTIME-FLAG      PIC S9(9) COMP-5.
  88 TPTIME         VALUE 0.
  88 TPNOTIME        VALUE 1.
05 TPSIGRSTRT-FLAG   PIC S9(9) COMP-5.
```

```
  88 TPNOSIGRSTRT     VALUE 0.
  88 TPSIGRSTRT      VALUE 1.
05 TPEV-METHOD-FLAG   PIC S9(9) COMP-5.
  88 TPEVNOTIFY      VALUE 0.
  88 TPEVSERVICE     VALUE 1.
  88 TPEVQUEUE      VALUE 2.
05 TPEV-PERSIST-FLAG  PIC S9(9) COMP-5.
  88 TPEVNOPERSIST    VALUE 0.
  88 TPEVPERSIST     VALUE 1.
05 TPEV-TRAN-FLAG    PIC S9(9) COMP-5.
  88 TPEVNOTRAN      VALUE 0.
  88 TPEVTRAN       VALUE 1.
*
05 EVENT-COUNT     PIC S9(9) COMP-5.
05 SUBSCRIPTION-HANDLE PIC S9(9) COMP-5.
05 NAME-1        PIC X(31).
05 NAME-2        PIC X(31).
05 EVENT-NAME      PIC X(31).
05 EVENT-EXPR      PIC X(255).
05 EVENT-FILTER     PIC X(255).
```

次の表は、TPEVTDEF-REC データ構造体のメンバを示しています。

メンバ	説明
EVENT-COUNT	イベント・カウント。
SUBSCRIPTION-HANDLE	サブスクリプション・ハンドル。
NAME-1、NAME-2	キューに登録された空間の名前。サブスクライバが TPEVQUEUE を設定した場合、イベント通知は NAME-1 で指定されるキュー・スペースと、NAME-2 で指定されるキューに登録されます。
EVENT-NAME	イベント名。

メンバ	説明
EVENT-EXPR	サブスクライブするイベント。正規表現を含み、NULL 文字で終了する最大 255 文字の文字列を指定します。正規表現は、tpsubscribe(3c) で指定された形式です (『C 言語を使用した BEA Tuxedo アプリケーションのプログラミング』を参照)。たとえば、次のように eventexpr を設定します。 ■ "\\..*" ー呼び出し元は、すべてのシステム定義のイベントをサブスクライブします。 ■ "\\.SysServer.*" ー呼び出し元は、サーバに関連するすべてのシステム定義のイベントをサブスクライブします。 ■ "[A-Z].*" ー呼び出し元は、A ～ Z の大文字で始まるすべてのユーザ定義のイベントをサブスクライブします。 ■ ".*(ERR|err).*" ー呼び出し元は、account_error または ERROR_STATE など、イベント名に ERR または err を含むすべてのユーザ定義のイベントをサブスクライブします。

メンバ	説明
EVENT-FILTER	ブール型のフィルタ規則を含む文字列。イベント・ブローカがイベントをポストする前に、この規則を評価する必要があります。ポストするイベントを受け取ると、イベント・ブローカはそのイベントのデータにフィルタ規則 (定義されている場合) を適用します。データが正しく評価された場合、イベント・ブローカは指定された通知方法を呼び出します。正しく評価されなかった場合、イベント・ブローカは指定された通知方法を無視します。呼び出し元は、異なるフィルタ・ルールを利用して同じイベントを何度でもサブスクライブすることができます。 イベント・フィルタ機能を使用すると、サブスクライバは通知されるイベントを限定できます。たとえば、100 万円を超える額の引き出しがあった場合に、イベントがポストされるとします。その場合、サブスクライバが 100 万円より高い額 (たとえば 500 万円) の通知だけを必要とすることがあります。または、特定の顧客による高額の引き出しの通知だけを必要とする場合があります。 フィルタ・ルールは、フィルタ・ルールの適用対象となる型付きレコードによって異なります。フィルタ規則の詳細については、『BEA Tuxedo COBOL リファレンス』の TPSUBSCRIBE(3cbl) リファレンス・ページを参照してください。
SETTINGS (TPBLOCK-FLAG、TPTRAN-FLAG、など)	サーバの特性を制御するその他の設定。設定値の詳細については、『BEA Tuxedo COBOL リファレンス』を参照してください。

TPQUEDEF-REC データ構造体の詳細については、『BEA Tuxedo /Q コンポーネント』を参照してください。

システム定義のイベントとアプリケーション定義のイベントは、TPSUBSCRIBE ルーチンを使用してサブスクライブできます。

サブスクリプション、および MIB を更新するために BEA Tuxedo システムのサーバ・プロセスで実行されるサービス・ルーチンは、信頼されたコードと見なされます。

このルーチンの詳細については、『BEA Tuxedo COBOL リファレンス』の TPSUBSCRIBE(3cbl) を参照してください。

イベントに対するサブスクリプションの削除

TPUNSUBSCRIBE(3cbl) ルーチンを使用すると、BEA Tuxedo システムのクライアントまたはサーバがイベントに対するサブスクリプションを削除できます。

TPUNSUBSCRIBE ルーチンの呼び出しには、次の文法を使用します。

```
01  TPEVTDEF-REC.
  COPY TPEVTDEF.

01 TPSTATUS-REC.
  COPY TPSTATUS.

CALL "TPUNSUBSCRIBE" USING TPEVTDEF-REC TPSTATUS-REC
```

TPEVTDEF-REC データ構造体の詳細については、8-13 ページの「イベントのサブスクライブ」を参照してください。TPQUEDEF-REC データ構造体の詳細については、『BEA Tuxedo /Q コンポーネント』を参照してください。

イベントのポスト

TPPOST(3cbl) ルーチンを使用すると、BEA Tuxedo のクライアントまたはサーバがイベントをポストできます。

TPPOST ルーチンの呼び出しには、次の文法を使用します。

```
01  TPEVTDEF-REC.
  COPY TPEVTDEF.
```

```
01 TPTYPE-REC.
  COPY TPSTATUS.

01 TPDATA-REC.
  COPY TPSTATUS.

01 TPSTATUS-REC.
  COPY TPSTATUS.

CALL "TPPST" USING TPEVTDEF-REC TPTYPE-REC TPDATA-REC TPSTATUS-REC
```

TPEVTDEF-REC データ構造体の詳細については、8-13 ページの「イベントのサブスクライブ」を参照してください。TPTYPE-REC レコードについては、5-12 ページの「サービスの定義」を参照してください。

9 グローバル・トランザクションのコーディング

ここでは、次の内容について説明します。

- グローバル・トランザクションとは
- トランザクションの開始
- トランザクションの終了
- トランザクションの終了
- グローバル・トランザクションの暗黙的な定義
- XA 準拠のサーバ・グループに対するグローバル・トランザクションの定義
- トランザクションが開始されたことの確認

グローバル・トランザクションとは

グローバル・トランザクションとは、複数のリソース・マネージャを使用し、複数のサーバ上で行われる複数の操作を 1 つの論理単位として処理できるようにするメカニズムです。

プロセスがトランザクション・モードになると、サーバに要求されたサービスが現在のトランザクションに代わって処理されます。呼び出されてトランザクションに参加したサービスは、「トランザクションのパーティシパント」と呼ばれます。パーティシパントから返される値によって、トランザクションの結果が変わる場合があります。

グローバル・トランザクションは複数のローカル・トランザクションから構成され、各トランザクションは同じリソース・マネージャにアクセスします。リソース・マネージャは、同時実行制御とデータ更新の原子性を実現します。ローカル・トランザクションでは、アクセスが正常に終了するか、または全体が失敗します。つまり、一部だけが成功することはありません。

1 つのトランザクションに参加可能なサーバ・グループは最大 16 個です。

BEA Tuxedo システムでは、グローバル・トランザクションが参加しているリソース・マネージャと共に管理され、原子性、一貫性、独立性、および持続性という特徴を持つ特定シーケンスの操作として処理されます。つまり、グローバル・トランザクションは、以下のような特徴を持つ論理的な作業単位と言えます。

- すべての部分が成功するか、または何も効果が発生しません。
- 操作が実行されて、リソースがある一貫した状態から別の状態に正しく移行します。
- ほかのトランザクションから中間の結果にアクセスすることはできません。ただし、トランザクションに含まれるプロセスには、別のプロセスに対応付けられたデータにアクセスできるものもあります。
- シーケンスが完了すると、その結果はどのような失敗による影響も受けません。

BEA Tuxedo システムでは、個々のグローバル・トランザクションの状態がトラッキングされ、そのトランザクションをコミットするかロールバックするかが決定されます。

トランザクションの開始

グローバル・トランザクションを開始するには、次の文法を使用して TPBEGIN(3cbl) ルーチンを呼び出します。

```
*
  01 TPTRXDEF-REC.
     COPY TPTRXDEF.
*
  01 TPSTATUS-REC.
     COPY TPSTATUS.
*
   CALL "TPBEGIN" USING TPTRXDEF-REC TPSTATUS-REC.
```

次の表は、TPTRXDEF-REC 構造体のフィールドを示しています。

表 9-1TPTRXDEF 構造体のフィールド

フィールド	説明
T-OUT	トランザクションがタイムアウトになるまでの時間 (秒単位)。この引数に 0 を指定すると、システムで可能な最長時間 (秒単位) に設定されます。つまり、*timeout* には、システムで定義された符号なし long 型の最大値が設定されます。 T-OUT パラメータに 0 または非現実的な大きな値を指定すると、システムによるエラー検出と報告が遅れる原因となります。T-OUT パラメータを使用すると、サービス要求に対する応答が妥当な時間内に確実に返されるようになります。また、ネットワーク障害などの問題が発生した場合に、コミットされる前にトランザクションを終了できます。 応答を人が待っている場合、このパラメータには小さな値 (可能な場合は 30 秒未満) を設定します。 生産システムの場合、T-OUT に大きな値を設定して、システムの負荷やデータベースの競合に起因する遅延に対応できるようにします。予測される平均応答時間を 2、3 倍した時間が最適です。 注記　T-OUT パラメータに設定する値は、BEA Tuxedo のアプリケーション管理者がコンフィギュレーション・ファイルに設定した SCANUNIT パラメータの値と一致していなければなりません。SCANUNIT パラメータには、タイムアウトになったトランザクションとサービス要求でブロックされた呼び出しがないかどうかを確認、つまりスキャンする頻度を指定します。このパラメータの値は、定期的なスキャンの間隔 (走査を行う間隔) を表します。 T-OUT パラメータには、走査を行う間隔より大きな値を設定します。T-OUT パラメータに設定された値が走査を行う間隔より小さいと、トランザクションがタイムアウトになる時間と、そのタイムアウトが検出される時間にずれが生じます。SCANUNIT のデフォルト値は 10 秒です。T-OUT パラメータの設定値についてはアプリケーション管理者と検討し、T-OUT パラメータに設定した値がシステム・パラメータの値と矛盾しないようにします。
TRANID	トランザクション識別子。

TPBEGIN は、どのプロセスからも呼び出すことができます。ただし、既にトランザクション・モードになっているプロセスからは呼び出すことはできません。トランザクション・モードで TPBEGIN が呼び出されると、プロトコル・エラーになって呼び出しが失敗し、TP-STATUS に TPEPROTO が設定されます。プロセスがトランザクション・モードの場合でも、この失敗はトランザクションには影響しません。

次のコード例は、グローバル・トランザクションの定義方法を簡単に示しています。

コード リスト 9-1　トランザクションの定義

```
. . .
MOVE 0 TO T-OUT.
CALL "TPBEGIN" USING
TPTRXDEF-REC
TPSTATUS-REC.
IF  NOT TPOK
    error processing
. . .
    program statements
. . .
CALL "TPCOMMIT" USING
                TPTRXDEF-REC
                TPSTATUS-REC.
IF  NOT TPOK
    error processing
```

次のコード例は、未処理の応答に起因するエラーの発生を示しています。

コード リスト 9-2　エラー - 未処理の応答があるトランザクションの開始

```
  . . .
  MOVE "BUY" TO SERVICE-NAME.
  SET TPBLOCK TO TRUE.
  SET TPNOTRAN TO TRUE.
  SET TPREPLY TO TRUE.
  SET TPNOTIME TO TRUE.
  SET TPSIGRSTRT TO TRUE.
  CALL "TPACALL" USING
```

```
                  TPSVCDEF-REC
                  TPTYPE-REC
                  BUY-REC
                  TPSTATUS-REC.
  IF  NOT TPOK
      error processing
  . . .
  MOVE 0 TO T-OUT.
  CALL "TPBEGIN" USING
                  TPTRXDEF-REC
                  TPSTATUS-REC.
  IF  NOT TPOK
      error processing
* ERROR TP-STATUS に TPEPROTO が設定されます。
  . . .
      program statements
  . . .
  SET TPBLOCK TO TRUE.
  SET TPNOTRAN TO TRUE.
  SET TPCHANGE TO TRUE.
  SET TPNOTIME TO TRUE.
  SET TPSIGRSTRT TO TRUE.
  SET TPGETANY TO TRUE.
  CALL "TPGETRPLY" USING
                    TPSVCDEF-REC
                    TPTYPE-REC
                    WK-AREA
                    TPSTATUS-REC.
  IF  NOT TPOK
      error processing
```

トランザクションがタイムアウトになった場合、TPCOMMIT を呼び出すとトランザクションがアボートします。その結果、TPCOMMIT が失敗し、TP-STATUS に TPEABORT が設定されます。

次のコード例は、トランザクションのタイムアウトを確認する方法を示しています。T-OUT の値が 30 秒に設定されていることに注目してください。

コード リスト 9-3　トランザクションのタイムアウトの確認

```
. . .
MOVE 30 TO T-OUT.
CALL "TPBEGIN" USING TPTRXDEF-REC TPSTATUS-REC.
```

```
IF   NOT TPOK
    MOVE "Failed to BEGIN a transaction" TO LOG-REC-TEXT.
    MOVE 29 to LOG-REC-LEN
    CALL "USERLOG" USING
                   LOG-REC-TEXT
                   LOG-REC-LEN
                   TPSTATUS-REC
    CALL "TPTERM" USING
                  TPSTATUS-REC
    PERFORM A-999-EXIT.
. . .
      communication CALL statements
. . .
IF   TPETIME
    CALL "TPABORT" USING
                   TPTRXDEF-REC
                   TPSTATUS-REC
IF   NOT TPOK
     error processing
ELSE
   CALL "TPCOMMIT" USING
                   TPTRXDEF-REC
                   TPSTATUS-REC
   IF   NOT TPOK
          error processing
```

注記　トランザクション・モードのプロセスで、TPNOTRAN を設定して通信呼び出しを行うと、呼び出されたサービスは現在のトランザクションに参加できません。サービス要求の成功や失敗は、トランザクションの結果に影響しません。トランザクションは、サービスから応答が返されるのを待つ間にタイムアウトになる場合もあります。これは、そのサービスがトランザションに参加しているかどうかには関係ありません。TPNOTRAN フラグの影響については、11-1 ページの「エラーの管理」を参照してください。

次のコード例は、トランザクションの定義方法を示しています。

コード リスト 9-4　トランザクションの定義

```
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
```

```
 01  TPTYPE-REC.
 COPY TPTYPE.
*
 01 TPSTATUS-REC.
 COPY TPSTATUS.
*
 01 TPINFDEF-REC.
 COPY TPINFDEF.
*
 01  TPSVCDEF-REC.
 COPY TPSVCDEF.
*
 01  TPTRXDEF-REC.
 COPY TPTRXDEF.
*
 01 LOG-REC                  PIC X(30) VALUE " ".
 01 LOG-REC-LEN              PIC S9(9)  COMP-5.
*
 01 USR-DATA-REC             PIC X(16).
*
 01 AUDV-REC.
     05  AUDV-BRANCH-ID          PIC S9(9) COMP-5.
     05  AUDV-BALANCE            PIC S9(9) COMP-5.
     05  AUDV-ERRMSG             PIC X(60).
*
 PROCEDURE DIVISION.
*
 A-000.
. . .
* コマンド行オプションを取得します。変数 (Q-BRANCH) を設定します。
  MOVE SPACES TO USRNAME.
  MOVE SPACES TO CLTNAME.
  MOVE SPACES TO PASSWD.
  MOVE SPACES TO GRPNAME.
  CALL "TPINITIALIZE" USING TPINFDEF-REC
                      USR-DATA-REC
                      TPSTATUS-REC.
  IF  NOT TPOK
      MOVE "Failed to join application" TO LOG-REC
      MOVE 26 TO LOG-REC-LEN
       CALL "USERLOG" USING LOG-REC
                      LOG-REC-LEN
                      TPSTATUS-REC
       PERFORM A-999-EXIT.
* グローバル・トランザクションを開始します。
  MOVE 30 TO T-OUT.
  CALL "TPBEGIN" USING TPTRXDEF-REC TPSTATUS-REC.
  IF  NOT TPOK
      MOVE 29 to LOG-REC-LEN
```

```
        MOVE "Failed to begin a transaction" TO LOG-REC
        CALL "USERLOG" USING LOG-REC
                       LOG-REC-LEN
                       TPSTATUS-REC
        PERFORM DO-TPTERM.
  * レコードを設定します。
    MOVE Q-BRANCH TO AUDV-BRANCH-ID.


    MOVE ZEROS TO AUDV-BALANCE.
    MOVE SPACES TO AUDV-ERRMSG.
  * TPCALL レコードを設定します。
    MOVE "GETBALANCE" TO SERVICE-NAME.
    MOVE "VIEW" TO REC-TYPE.
    MOVE LENGTH OF AUDV-REC TO LEN.
    SET TPBLOCK TO TRUE.
    SET TPTRAN IN TPSVCDEF-REC TO TRUE.
    SET TPNOTIME TO TRUE.
    SET TPSIGRSTRT TO TRUE.
    SET TPCHANGE TO TRUE.
  *
    CALL "TPCALL" USING TPSVCDEF-REC
                  TPTYPE-REC
                  AUDV-REC
                  TPTYPE-REC
                  AUDV-REC
                  TPSTATUS-REC.
    IF  NOT TPOK
        MOVE 19 to LOG-REC-LEN
        MOVE "Service call failed" TO LOG-REC
        CALL "USERLOG" USING LOG-REC
                       LOG-REC-LEN
                       TPSTATUS-REC
        PERFORM DO-TPABORT
        PERFORM DO-TPTERM.
  * グローバル・トランザクションをコミットします。
    CALL "TPCOMMIT" USING TPTRXDEF-REC
                    TPSTATUS-REC
    IF  NOT TPOK
        MOVE 16 to LOG-REC-LEN
        MOVE "Failed to commit" TO LOG-REC
        CALL "USERLOG" USING LOG-REC
                  LOG-REC-LEN
                  TPSTATUS-REC
        PERFORM DO-TPTERM.
  * トランザクションが成功した場合のみ、結果を表示します。
    DISPLAY "BRANCH=" Q-BRANCH.
    DISPLAY "BALANCE=" AUDV-BALANCE.
    PERFORM DO-TPTERM.
```

```
* トランザクションをアボートします。
 DO-TPABORT.
  CALL "TPABORT" USING TPTRXDEF-REC
               TPSTATUS-REC
  IF  NOT TPOK
      MOVE 26 to LOG-REC-LEN
      MOVE "Failed to abort transaction" TO LOG-REC
      CALL "USERLOG" USING LOG-REC
                   LOG-REC-LEN


                   TPSTATUS-REC.
* アプリケーションを分離します。
 DO-TPTERM.
  CALL "TPTERM" USING TPSTATUS-REC.
  IF  NOT TPOK
      MOVE 27 to LOG-REC-LEN
      MOVE "Failed to leave application" TO LOG-REC
      CALL "USERLOG" USING LOG-REC
                   LOG-REC-LEN
                   TPSTATUS-REC.
                   EXIT PROGRAM.
*
 A-999-EXIT.
*
 EXIT PROGRAM.
```

トランザクションの終了

グローバル・トランザクションを終了するには、TPCOMMIT(3cbl) を呼び出して現在のトランザクションをコミットするか、または TPABORT(3cbl) を呼び出して処理をアボートして、すべての操作をロールバックします。

注記　TPCALL、TPACALL、または TPCONNECT を呼び出すときに明示的に TPNOTRAN が設定されている場合、呼び出されたサービスによって実行される操作は、トランザクションに含まれません。つまり、このようなサービスによって実行される操作は、TPABORT ルーチンを呼び出したときにロールバックされません。

現在のトランザクションのコミット

TPCOMMIT(3cbl) ルーチンは、現在のトランザクションをコミットします。TPCOMMIT から正常に制御が戻ると、現在のトランザクションの結果としてリソースに加えられた変更は永続的なものとなります。

TPCOMMIT ルーチンの呼び出しには、次の文法を使用します。

```
*
  01 TPTRXDEF-REC.
     COPY TPTRXDEF.
*
  01 TPSTATUS-REC.
     COPY TPSTATUS.
*
   CALL "TPCOMMIT" USING TPTRXDEF-REC TPSTATUS-REC.
```

TPTRXDEF-REC 構造体については、9-3 ページの「トランザクションの開始」を参照してください。

トランザクションをコミットするための条件

TPCOMMIT を正常に実行するには、次の条件を満たしていることが必要です。

- 呼び出し元プロセスは、TPBEGIN を呼び出してトランザクションを開始したプロセスと同じでなければなりません。
- TPNOTRAN フラグを設定しないで呼び出した場合、呼び出し元プロセスに未処理のトランザクション応答が存在することはできません。
- トランザクションの状態が「ロールバックのみ」ではなく、またタイムアウトになっていないことが必要です。

最初の条件を満たしていない場合、呼び出しは失敗し、プロトコル・エラーを示す TPEPROTO が TP-STATUS に設定されます。2 番目または 3 番目の条件を満たしていない場合、呼び出しは失敗し、トランザクションがロールバックされたことを示す TPEABORT が TP-STATUS に設定されます。トランザクションに未処理の応答があるときに TPCOMMIT がイニシエータによって呼び出されると、トランザクションはアボートされ、トランザクションに関連する応答記述子が無効になります。パーティシパントが TPCOMMIT または TPABORT を呼び出しても、トランザクションには影響しません。

サービス呼び出しで TPFAIL が返されるか、またはサービス・エラーが発生すると、トランザクションは「ロールバックのみ」の状態になります。「ロールバックのみ」のトランザクションに対して TPCOMMIT が呼び出されると、このルーチンはトランザクションを取り消し、-1 を返して TP-STATUS に TPEABORT を設定します。既にタイムアウトになっているトランザクションに対して TPCOMMIT を呼び出した場合も同じ結果になり、これにより、TPCOMMIT は -1 を返し、TP-STATUS に TPEABORT が設定されます。トランザクション・エラーの詳細については、11-1 ページの「エラーの管理」を参照してください。

2 フェーズ・コミット・プロトコル

TPCOMMIT ルーチンが呼び出されると、2 フェーズ・コミット・プロトコルによる通信が開始されます。このプロトコルは、その名前が示すように、次の 2 段階の処理に分かれています。

1. 参加する各リソース・マネージャがコミットの準備ができたことを示します。
2. トランザクションのイニシエータが、参加する各リソース・マネージャにコミット許可を与えます。

トランザクションのイニシエータが TPCOMMIT ルーチンを呼び出すと、コミット・シーケンスが開始されます。指定されたコーディネータ・グループ内の BEA Tuxedo TMS サーバ・プロセスは、コミット・プロトコルの最初のフェーズを実行する各パーティシパント・グループの TMS と通信を行います。次に、各グループの TMS は、そのグループのリソース・マネージャ (RM) に、トランザクション・マネージャ と RM 間の通信用に定義されている XA プロトコルを使用してコミットするように指示します。RM は、安定記憶域にコミット・シーケンスの前後のトランザクションの状態を書き込み、TMS に成功か失敗かを通知します。その後、TMS はトランザクション・コーディネータの TMS に応答を渡します。

トランザクション・コーディネータの TMS は、すべてのグループから成功の通知を受け取ると、トランザクションのコミット中であることをログに記録し、第 2 フェーズのコミット通知をすべてのパーティシパント・グループに送信します。その後、各グループの RM はトランザクションの更新を完了します。

トランザクション・コーディネータの TMS が、グループから第 1 フェーズのコミットの失敗の通知を受けた場合、またはグループからの応答の受信に失敗した場合、各 RM にロールバック通知を送信し、RM はすべてのトランザクション更新を以前の状態に戻します。これにより、TPCOMMIT は失敗し、TP-STATUS に TPEABORT が設定されます。

コミットの成功条件の選択

1 つのトランザクションに複数のグループが関係している場合、TPCOMMIT が正常に制御を戻すための条件として、次のいずれかを指定できます。

- すべてのパーティシパントからコミットの準備が完了したことが通知された場合。つまり、すべてのパーティシパントが 2 フェーズ・コミットの第 1 フェーズが完了したことを報告し、トランザクション・コーディネータの TMS が安定記憶域にコミットの決定を書き込んだ場合です。
- すべてのパーティシパントで 2 フェーズ・コミットの第 2 フェーズが完了した場合。

この 2 つの条件のいずれかを指定するには、コンフィギュレーション・ファイルの RESOURCES セクションの CMTRET パラメータに、次のいずれかの値を設定します。

- LOGGED ー第 1 フェーズの完了が必須であることを示します。
- COMPLETE ー第 2 フェーズの完了が必須であることを示します。

デフォルトでは、CMTRET は COMPLETE に設定されます。

コミット条件での妥協点

ほとんどの場合、グローバル・トランザクションのすべてのパーティシパントが第 1 フェーズの正常終了を記録した場合、第 2 フェーズも正常終了します。CMTRET に LOGGED を設定すると、TCOMMIT の呼び出しから制御が多少早く戻るようになります。ただし、パーティシパントが、コミットの決定と矛盾する方法で、トランザクションの担当部分をヒューリスティックで終了する危険性があります。

このようなリスクを負うべきかどうかの選択は、アプリケーションの性質に左右されます。たとえば、財務アプリケーションなど正確さが要求されるアプリケーションでは、すべてのパーティシパントが 2 フェーズ・コミットを完了するまでは、制御を戻さないようにします。時間的な条件を重視するアプリケーションでは、正確さを犠牲にしても実行速度を上げます。

現在のトランザクションのアボート

TPABORT(3cbl) ルーチンを使用すると、異常な状態を通知して、明示的にトランザクションをアボートできます。この関数は、トランザクションの応答に未処理のものがあると、その呼び出し記述子を無効にします。その場合、トランザクションで行われた変更はリソースには適用されません。TPABORT ルーチンの呼び出しには、次の文法を使用します。

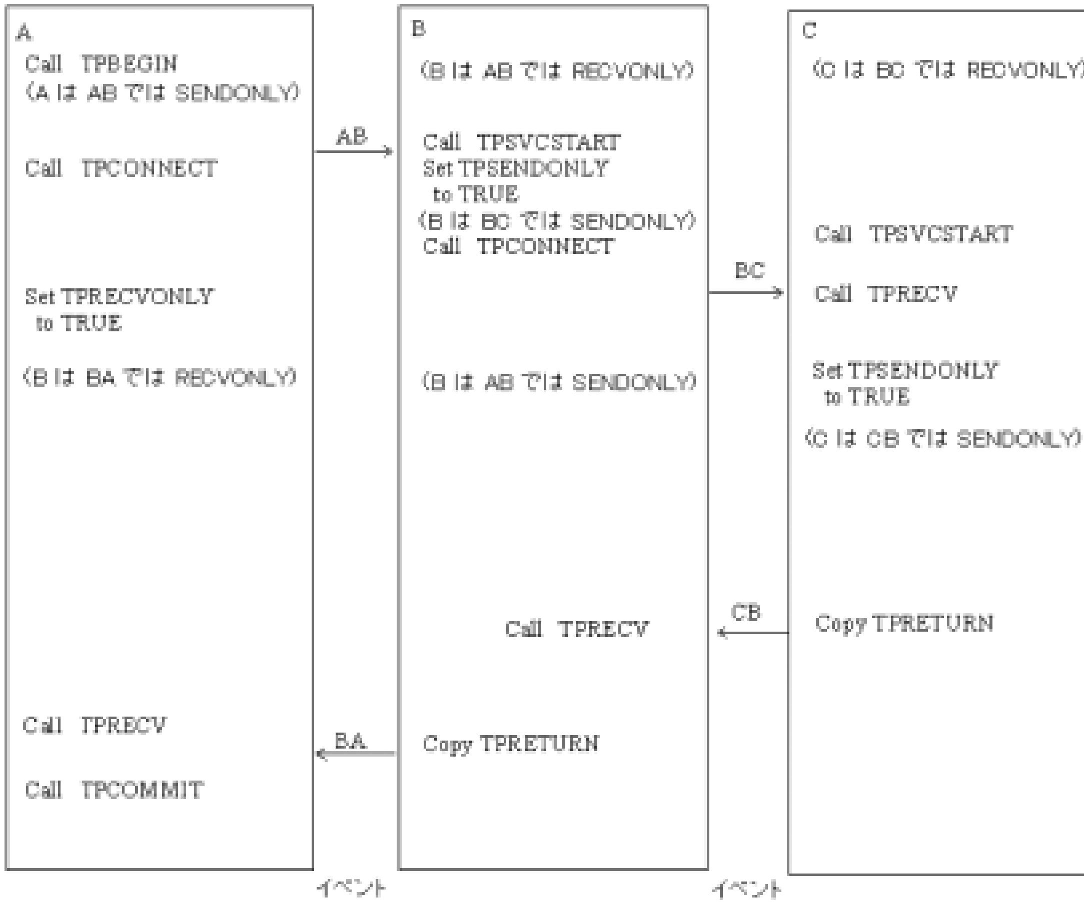
```
*
   01 TPTRXDEF-REC.
      COPY TPTRXDEF.
*
   01 TPSTATUS-REC.
      COPY TPSTATUS.
*
    CALL "TPABORT" USING TPTRXDEF-REC TPSTATUS-REC.
```

TPTRXDEF-REC 構造体の詳細については、9-3 ページの「トランザクションの開始」を参照してください。

例 : 会話モードによるトランザクションのコミット

次の図は、グローバル・トランザクションを行う階層構造の会話型接続を示しています。

図 9-1 トランザクション・モードにおける接続階層構造



接続階層構造は、次の処理が行われることで構築されます。

1. クライアント (プロセス A) は、TPBEGIN と TPCONNECT を呼び出して、トランザクション・モードで接続を開始します。
2. クライアントは、実行する下位サービスを呼び出します。

3. 各下位サービスは、処理が完了すると、処理が成功したか失敗したか (TPEV_SVCSUCC または TPEV_SVCFAIL) を示す応答を階層構造を通じてトランザクションを開始したプロセスに送信します。この例では、トランザクションを開始したプロセスはクライアント (プロセス A) です。下位サービスは、応答の送信が終了すると、つまり未処理の応答がなくなると、TPRETURN を呼び出します。
4. クライアント (プロセス A) は、すべての下位サービスが正常終了したかどうかを確認します。
   - すべての下位サービスが正常終了した場合、クライアントは TPCOMMIT を呼び出して、それらのサービスが実行した変更をコミットし、トランザクションを終了します。
   - 正常終了していない下位サービスがある場合、TPCOMMIT は成功しないので、クライアントは TPABORT を呼び出します。

例 : パーティシパントのエラーの確認

次のコード例では、クライアントは REPORT サービスへの同期呼び出し (24 行目) を行います。次に、通信呼び出しで返される可能性があるエラーを調べて (30 ～ 42 行目)、パーティシパントの失敗を確認します。

コード リスト 9-5 パーティシバントの成功 / 失敗の確認

```
01    . . .
02    CALL "TPINITIALIZE" USING TPINFDEF-REC
03    USR-DATA-REC
04    TPSTATUS-REC.
05    IF  NOT TPOK
06        error message,
07        EXIT PROGRAM .
08    MOVE 30 TO T-OUT.
09    CALL "TPBEGIN" USING TPTRXDEF-REC TPSTATUS-REC.
10    IF  NOT TPOK
11        error message,
12        PERFORM DO-TPTERM.
13  * レコードを設定します。
14    MOVE "REPORT=accrcv DBNAME=accounts" TP-RECORD.
```

```
15   MOVE 27 TO LEN.
16   MOVE "REPORTS" TO SERVICE-NAME.
17   MOVE "STRING" TO REC-TYPE.
18   SET TPBLOCK TO TRUE.
19   SET TPTRAN IN TPSVCDEF-REC TO TRUE.
20   SET TPNOTIME TO TRUE.
21   SET TPSIGRSTRT TO TRUE.
22 SET TPCHANGE TO TRUE.
23  *
24   CALL "TPCALL" USING TPSVCDEF-REC
25                TPTYPE-REC
26                TP-RECORD
27                TPTYPE-REC
28                TP-RECORD
29                TPSTATUS-REC.
30   IF  TPOK
31        PERFORM DO-TPCOMMIT
32        PERFORM DO-TPTERM.
33  * 戻り値を確認します。
34    IF  TPESVCERR
35    DISPLAY "REPORT service's TPRETURN encountered problems"
36    ELSE IF  TPESVCFAIL
37    DISPLAY "REPORT service FAILED with return code=" APPL-RETURN-CODE
38    ELSE IF  TPEOTYPE
39    DISPLAY "REPORT service's reply is not of any known REC-TYPE"
40  *
41    PERFORM DO-TPABORT
42    PERFORM DO-TPTERM.
43  * グローバル・トランザクションをコミットします。
44   DO-TPCOMMIT.
45    CALL "TPCOMMIT" USING TPTRXDEF-REC
46                    TPSTATUS-REC
47    IF  NOT TPOK
48        error message
49  * トランザクションをアボートします。
50   DO-TPABORT.
51    CALL "TPABORT" USING TPTRXDEF-REC
52                   TPSTATUS-REC
53    IF  NOT TPOK
54        error message
55  * アプリケーションを分離します。
56   DO-TPTERM.
57    CALL "TPTERM" USING TPSTATUS-REC.
58    IF  NOT TPOK
59        error message
60    EXIT PROGRAM.
```

グローバル・トランザクションの暗黙的な定義

アプリケーションでは、次のいずれかの方法でグローバル・トランザクションを開始できます。

- ATMI 呼び出しにより明示的に行います。9-3 ページの「トランザクションの開始」を参照してください。
- サービス・ルーチン内から暗黙的に開始します。

この節では、2 番目の方法について説明します。

コンフィギュレーション・ファイルのシステム・パラメータ AUTOTRAN を設定すると、サービス・ルーチンがトランザクション・モードになります。AUTOTRAN に Y を設定すると、別のプロセスから要求を受信したときに、サービス・サブルーチン内でトランザクションが自動的に開始されます。

暗黙的にトランザクションを定義する場合は、以下の規則に従います。

- 呼び出し元プロセスがトランザクション・モードになっていない場合に、システム・パラメータ AUTOTRAN がトランザクションを開始するように設定されていると、プロセスが別のプロセスのサービスを要求したときにトランザクションが開始されます。
- 既にトランザクション・モードになっているプロセスが別のプロセスのサービスを要求した場合、システムはまず、呼び出し元で TPNOTRAN が設定されているかどうかを確認します。

  TPNOTRAN が設定されていない場合、呼び出されたプロセスは「伝達の規則」によってトランザクション・モードになります。システムによって AUTOTRAN パラメータは確認されません。

  TPTRN-FLAG IN TPSVCDEF-REC に TPNOTRAN が設定されている場合、呼び出されたプロセスによって実行されるサービスは、現在のトランザクションに含まれません。つまり、「伝達の規則」は適用されません。システムによって AUTOTRAN パラメータが確認されます。

- AUTOTRAN に N が設定されている場合 (つまり AUTOTRAN が有効になっていない場合)、呼び出されたプロセスはトランザクション・モードになりません。
- AUTOTRAN に Y が設定されている場合、呼び出されたプロセスはトランザクション・モードになります。ただし、新しいトランザクションとして処理されます。

注記　サービスは自動的にトランザクション・モードにできるので、TPNOTRAN フラグが設定されたサービスから、AUTOTRAN パラメータが設定されたサービスを呼び出すことができます。そのようなサービスが別のサービスを要求した場合、サービスのデータ構造体のメンバは照会を行ったときに TPTRAN を返します。たとえば、TPNOTRAN | TPNOREPLY を設定して呼び出しを行い、サービスが呼び出されたときに (そのサービスによって) トランザクションが自動的に開始された場合、データ構造体は、TPTRAN | TPNOREPLY に設定されます。

XA 準拠のサーバ・グループに対するグローバル・トランザクションの定義

アプリケーション・プログラマが XA 準拠のサーバ・グループのサービスをコーディングする場合、グループのリソース・マネージャを介して操作を行うようにするのが一般的です。通常、サービスは 1 つのトランザクション内ですべての操作を行います。それに対して、TPNOTRAN を設定してサービスを呼び出すと、データベース操作の実行時に予期しない結果を受け取る場合があります。

予測不能な動作を防ぐには、XA 準拠のリソース・マネージャに対応付けられているグループのサービスが、常にトランザクション・モードで呼び出されるようにアプリケーションを設計します。または、コンフィギュレーション・ファイルの AUTOTRAN に Y を設定します。また、サービス・コードの早い段階で、トランザクション・レベルを確認します。

トランザクションが開始されたことの確認

特定のエラー条件を回避したり正しく解釈するには、プロセスがトランザクション・モードかどうかを確認することが大切です。たとえば、既にトランザクション・モードになっているプロセスが TPBEGIN を呼び出すとエラーになります。そのようなプロセスが TPBEGIN を呼び出すと、呼び出しは失敗し、TP-STATUS に TPEPROTO が設定されて、呼び出し元が既にトランザクションに参加しているにもかかわらず呼び出されたことが示されます。トランザクションに影響はありません。

サービス・サブルーチンがトランザクション・モードかどうかを確認した後で、TPBEGIN を呼び出すようにアプリケーションを設計できます。次のいずれかの方法で、トランザクション・レベルを確認できます。

- サービス・サブルーチンに渡されるサービスのデータ構造体の設定値を照会します。TPTRAN に設定されていると、サービスはトランザクション・モードになっています。
- TPGETLEV(3cbl) ルーチンを呼び出します。

TPGETLEV ルーチンの呼び出しには、次の文法を使用します。

```
01 TPTRXLEV-REC.
   COPY TPTRXLEV.
01 TPSTATUS-REC.
   COPY TPSTATUS.
CALL "TPGETLEV" USING TPTRXLEV-REC TPSTATUS-REC.
```

TPGETLEV は、呼び出し元がトランザクション・モードになっていない場合は TP-NOT-IN-TRAN を返し、トランザクション・モードになっている場合は TP-IN-TRAN を返します。

次のコード例は、TPGETLEV ルーチン (3 行目) を使用して、トランザクション・レベルを確認する方法を示しています。プロセスがトランザクション・モードになっていない場合、アプリケーションでトランザクションを開始し

ます (5 行目)。TPBEGIN が失敗した場合、メッセージがステータス行に返され (9 行目)、TPRETURN の APPL-CODE IN TPSVCRET-REC に APL-RETURN-CODE IN TPSTATUS-REC で取得できるコードが設定されます (1 行目と 11 行目)。

コード リスト 9-6 トランザクション・レベルの確認

```
  . . . Application defined codes
001       77 BEG-FAILED    PIC S9(9) VALUE 3.
     . . .
002  PROCEDURE DIVISION.
     . . .
003  CALL "TPGETLEV" USING TPTRCLEV-REC
                     TPSTATUS-REC.
004  IF NOT TPOK
        error processing   EXIT PROGRAM
005  IF TP-NOT-IN-TRAN
006         MOVE 30 TO T-OUT.
007         CALL "TPBEGIN" USING
                           TPTRXDEF-REC
                           TPSTATUS-REC.
008         IF  NOT TPOK
009              MOVE "Attempt to TPBEGIN within service failed"
                           TO USER-MESSAGE.
010              SET TPFAIL TO TRUE.
011              MOVE BEG-FAILED TO APPL-CODE.
012              COPY TPRETURN REPLACING
013                        DATA-REC BY USER-MESSAGE.
     . . .
```

AUTOTRAN に Y が設定されている場合、トランザクション・ルーチンの TPBEGIN、TPCOMMIT、TPABORT を明示的に呼び出す必要はありません。その結果、トランザクション・レベルを確認するオーバーヘッドを減らすことができます。また、TRANTIME パラメータを設定して、タイムアウト間隔を指定することもできます。タイムアウト間隔は、サービスに対するトランザクションが開始されてからの経過時間です。また、トランザクションが完了しなかった場合は、トランザクションがロールバックされるまでの時間です。

たとえば、前述のコードの OPEN_ACCT サービスを変更するとします。現在のコードでは、OPEN_ACCT にトランザクションが明示的に定義され、そのトランザクションの有無を確認しています。これらの処理のオーバーヘッドを減らすには、そのコードを削除します。その場合、OPEN_ACCT は常にトランザ

クション・モードで呼び出す必要があります。この要件を指定するには、コンフィギュレーション・ファイルの AUTOTRAN と TRANTIME システム・パラメータを有効にします。

関連項目

- 『BEA Tuxedo アプリケーションの設定』の 9-18 ページの「グローバル・トランザクションの暗黙的な定義」の AUTOTRAN コンフィギュレーション・パラメータ
- 『BEA Tuxedo アプリケーションの設定』の TRANTIME コンフィギュレーション・パラメータ

10 マルチスレッドおよびマルチコンテキスト・アプリケーションのプログラミング

ここでは、次の内容について説明します。

- マルチスレッドおよびマルチコンテキスト・アプリケーションのプログラミングに対するサポート
- マルチスレッドおよびマルチコンテキスト・アプリケーションの計画と設計
- マルチスレッドおよびマルチコンテキスト・アプリケーションのインプリメント
- マルチスレッドおよびマルチコンテキスト・アプリケーションのテスト

マルチスレッドおよびマルチコンテキスト・アプリケーションのプログラミングに対するサポート

BEA Tuxedo システムでは、次のアプリケーションがサポートされています。

- カーネルレベルのスレッド・パッケージ (ユーザレベルのスレッド・パッケージはサポートされていません)。
- C 言語で記述されたマルチスレッド・アプリケーション (COBOL でのマルチスレッド・アプリケーションはサポートされていません)。
- C 言語または COBOL 言語で記述されたマルチコンテキスト・アプリケーション。

お使いのオペレーティング・システムで POSIX スレッド関数と共にほかのスレッド関数がサポートされている場合は、POSIX スレッド関数を使用することをお勧めします。この関数を使用すると、後でコードをほかのプラットフォームに簡単に移植できます。

お使いのプラットフォームでカーネルレベルのスレッド・パッケージ、C 言語の関数、または POSIX 関数がサポートされているかどうかを確認するには、『BEA Tuxedo システムのインストール』で、使用しているオペレーティング・システムのデータ・シートを参照してください。

マルチスレッドおよびマルチコンテキスト・アプリケーションに関するプラットフォーム固有の検討事項

多くのプラットフォームには、マルチスレッドおよびマルチコンテキスト・アプリケーション固有の要件があります。プラットフォーム固有の要件については、『BEA Tuxedo システムのインストール』に説明があります。お使いのプラットフォームの要件については、該当するデータシートを参照してください。

関連項目

- 10-5 ページの「マルチスレッドおよびマルチコンテキストとは」
- 10-10 ページの「マルチスレッドおよびマルチコンテキスト・アプリケーションの利点と問題点」
- 10-13 ページの「クライアントでのマルチスレッドとマルチコンテキストの動作」
- 10-20 ページの「サーバでのマルチスレッドとマルチコンテキストの動作」

マルチスレッドおよびマルチコンテキスト・アプリケーションの計画と設計

ここでは、次の内容について説明します。

- マルチスレッドおよびマルチコンテキストとは
- マルチスレッドおよびマルチコンテキスト・アプリケーションの利点と問題点
- クライアントでのマルチスレッドとマルチコンテキストの動作
- サーバでのマルチスレッドとマルチコンテキストの動作
- マルチスレッドおよびマルチコンテキスト・アプリケーションの設計上の検討事項

マルチスレッドおよびマルチコンテキストとは

BEA Tuxedo システムでは、単一のプロセスで複数のタスクを同時に実行できます。このようなプロセスをインプリメントするプログラミング手法はマルチスレッドおよびマルチコンテキストと呼ばれます。この節では、これらの手法に関する基本事項について説明します。

- マルチスレッドとは
- マルチコンテキストとは

マルチスレッドとは

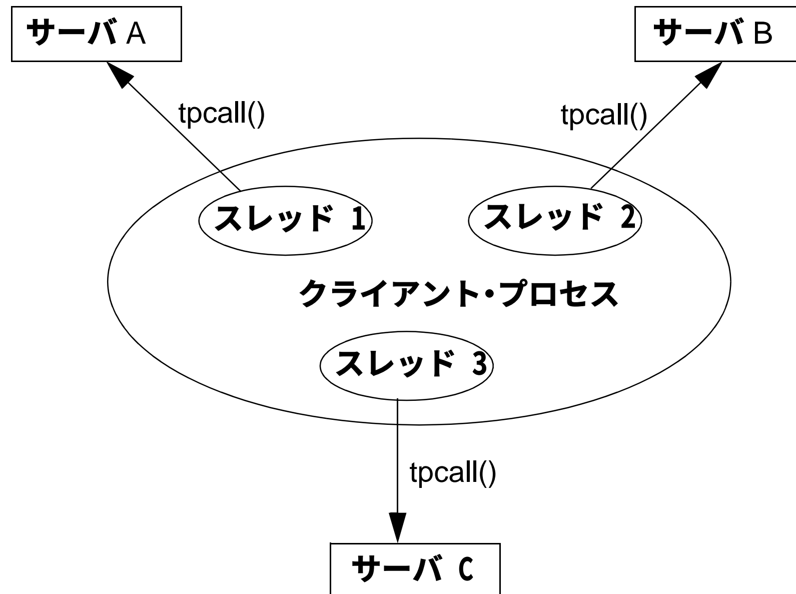
マルチスレッドとは、1 つのプロセスに複数の実行単位が含まれている処理方法です。マルチスレッド・アプリケーションでは、同じプロセスから同時に複数の呼び出しを行うことができます。たとえば、個々のプロセスが 1 つの未終了の `tpcall(3c)` に制限されることはありません。

サーバのマルチスレッドでは、アプリケーション生成のスレッドがシングルコンテキスト・サーバで使用される場合を除き、マルチコンテキストが必要です。マルチスレッドのシングルコンテキスト・アプリケーションを作成する唯一の方法は、アプリケーション生成のスレッドを使用することです。

BEA Tuxedo システムでは、C 言語で記述されたマルチスレッド・アプリケーションがサポートされています。COBOL 言語のマルチスレッド・アプリケーションはサポートされていません。

次の図は、マルチスレッド・クライアントが 3 つのサーバに対して同時に呼び出しを行う方法を示しています。

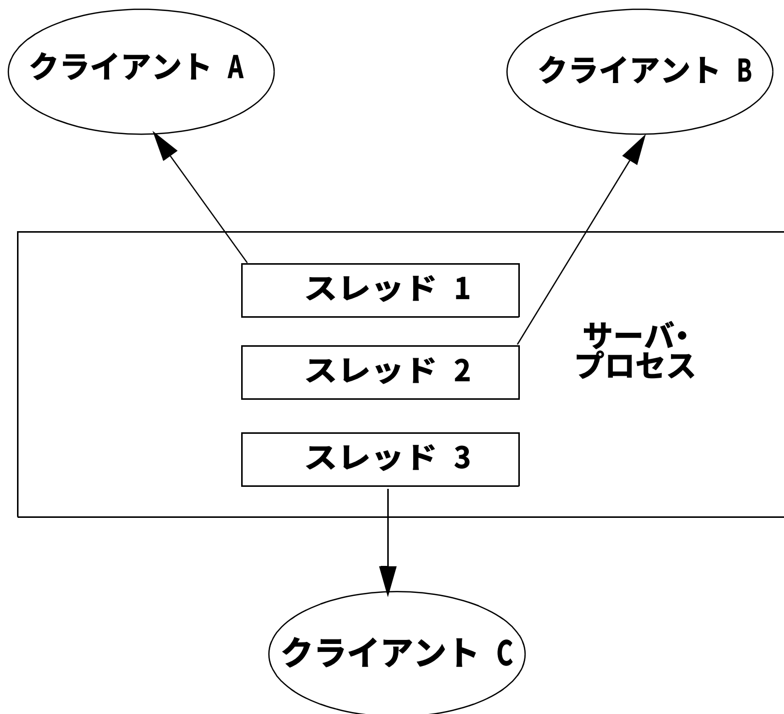
図 10-1 マルチスレッド・プロセスの例



マルチスレッド・アプリケーションでは、同じサーバで複数のサービス・ディスパッチ・スレッドを使用できるので、アプリケーションに対して起動するサーバ数が少なくて済みます。

次の図は、異なるクライアントに対して、サーバ・プロセスが同時に複数のスレッドをディスパッチする方法を示しています。

図 10-21 つのサーバ・プロセスによる複数のサービス・スレッドのディスパッチ



マルチコンテキストとは

コンテキストはドメインへの対応付けです。マルチコンテキストを使用すると、1 つのプロセスで次のいずれかが可能になります。

- ドメイン内での複数接続
- 複数ドメインへの接続

マルチコンテキストは、クライアントとサーバの両方で使用できます。サーバでマルチコンテキストを使用すると、マルチスレッドも使用することになります。

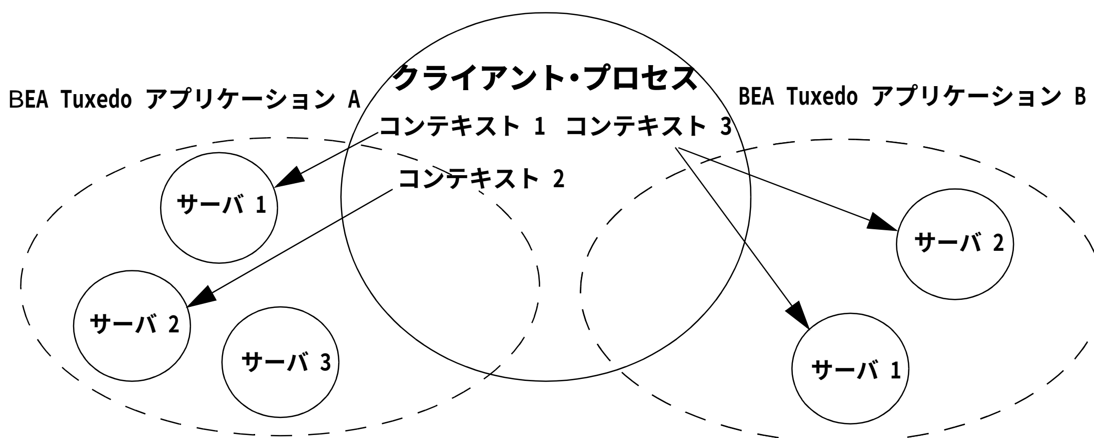
コンテキストの特徴の詳細については、次のいずれかの節で「コンテキストの属性」を参照してください。

- 10-35 ページの「クライアントでマルチコンテキストを使用するためのコーディング」
- 10-44 ページの「サーバでマルチコンテキストとマルチスレッドを使用するためのコーディング」

BEA Tuxedo システムでは、C 言語または COBOL 言語のいずれかで記述されたマルチコンテキスト・アプリケーションがサポートされています。ただし、サポートされるマルチスレッド・アプリケーションは、C 言語で記述されたものだけです。

次の図は、ドメイン内でのマルチコンテキスト・クライアント・プロセスの動作を示しています。矢印はサーバへの未終了の呼び出しを表します。

図 10-32 つのドメイン内でのマルチコンテキスト・プロセス



マルチスレッド・アプリケーションまたはマルチコンテキスト・アプリケーションのライセンス

ライセンスの関係で、各コンテキストは 1 人のユーザとしてカウントされます。1 つのコンテキストで複数のスレッドを使用するために、ライセンスを追加する必要はありません。次に例を示します。

- アプリケーション A に対応する 2 つのコンテキストと、アプリケーション B に対応する 1 つのコンテキストを持つプロセスの場合、アプリケーション A に 2 人、アプリケーション B に 1 人の合計 3 人のユーザとしてカウントされます。
- 1 つのアプリケーションにアクセスする複数のスレッドが同じコンテキスト内にあるプロセスの場合、ユーザは 1 人としてカウントされます。

関連項目

- 10-10 ページの「マルチスレッドおよびマルチコンテキスト・アプリケーションの利点と問題点」
- 10-13 ページの「クライアントでのマルチスレッドとマルチコンテキストの動作」
- 10-20 ページの「サーバでのマルチスレッドとマルチコンテキストの動作」

マルチスレッドおよびマルチコンテキスト・アプリケーションの利点と問題点

マルチスレッドとマルチコンテキストを適切な状況で使用すると、BEA Tuxedo アプリケーションのパフォーマンスを向上できます。ただし、これらの手法を取り入れる前に、潜在的な利点と問題点について理解しておくことが大切です。

マルチスレッドおよびマルチコンテキスト・アプリケーションの利点

マルチスレッドおよびマルチコンテキスト・アプリケーションには、以下の利点があります。

- パフォーマンスと並列性の向上

  マルチスレッドとマルチコンテキストを併用すると、一部のアプリケーションではパフォーマンスと並列性が向上します。また、一部のアプリケーションでは、パフォーマンスが変わらないか、逆に低下する場合があります。パフォーマンスへの影響は、使用しているアプリケーションによって異なります。

- リモート・プロシージャ・コールと会話のコーディングの単純化

  アプリケーションによっては、異なるリモート・プロシージャ・コールと会話を別々のスレッドでコーディングした方が、同じスレッドで管理するより簡単な場合があります。

- 複数アプリケーションへの同時アクセス

  BEA Tuxedo クライアントを同時に複数のアプリケーションに接続できます。

- 必要最低限の数のサーバの使用

  1 つのサーバで複数のサービス・スレッドをディスパッチできるので、アプリケーションに対して起動するサーバの数を減らすことができます。

このように複数のスレッドをディスパッチできる機能は、特に会話型サーバの場合に有用です。会話型サーバにこの機能がない場合、会話が終了するまで 1 つのクライアントしか使用できなくなります。

アプリケーションで、クライアント・スレッドが Microsoft Internet Information Server API または Netscape Enterprise Server インターフェイス (NSAPI) によって生成される場合、これらのツールの機能を最大限に利用するにはマルチスレッドが不可欠です。ほかのツールについても同じことが言えます。

マルチスレッドおよびマルチコンテキスト・アプリケーションの問題点

マルチスレッドおよびマルチコンテキスト・アプリケーションには、以下の問題点があります。

- コーディングの難しさ

  マルチスレッドおよびマルチコンテキスト・アプリケーションのコーディングは簡単ではありません。このようなアプリケーションのコーディングは、十分な経験を持つプログラマだけが行うことができます。

- デバッグの難しさ

  マルチスレッド・アプリケーションまたはマルチコンテキスト・アプリケーションで発生したエラーを再現することは、シングルスレッドおよびシングルコンテキスト・アプリケーションで再現するより難しい作業です。そのため、エラー発生時にその根本的な原因を特定して検証することがさらに難しくなります。

- 並列性の管理の難しさ

  スレッド間の並列性の管理は難しい作業であり、アプリケーションで新たに問題を引き起こす可能性があります。

- テストの難しさ

  マルチスレッド・アプリケーションのテストは、シングルスレッド・アプリケーションのテストより難しい作業です。問題がタイミングに関連していることが多く、再現が困難だからです。

- 既存コードの移植の難しさ

  既存のコードでマルチスレッドとマルチコンテキストを使用するには、大部分のコードを再構築しなければなりません。プログラマは、以下の作業を行う必要があります。

  - 静的変数を削除します。
  - スレッド・セーフではないすべての関数呼び出しを置き換えます。
  - スレッド・セーフではないその他のコードを置き換えます。

移植が完了したら何度もテストを繰り返す必要があり、マルチスレッドおよびマルチコンテキスト・アプリケーションの移植には多くの作業が必要になります。

関連項目

- 10-5 ページの「マルチスレッドおよびマルチコンテキストとは」
- 10-13 ページの「クライアントでのマルチスレッドとマルチコンテキストの動作」
- 10-20 ページの「サーバでのマルチスレッドとマルチコンテキストの動作」
- 10-26 ページの「マルチスレッドおよびマルチコンテキスト・アプリケーションの設計上の検討事項」

クライアントでのマルチスレッドとマルチコンテキストの動作

マルチスレッドおよびマルチコンテキスト・アプリケーションがアクティブの場合、クライアントのライフ・サイクルは次の 3 つのフェーズから構成されます。

- 起動フェーズ
- 作業フェーズ
- 完了フェーズ

起動フェーズ

起動フェーズでは、次の操作が行われます。

- 一部のクライアント・スレッドは `tpinit(3c)` を呼び出して、1 つ以上の BEA Tuxedo アプリケーションに参加します。
- その他のクライアント・スレッドは `tpsetctxt(3c)` を呼び出して、上記のスレッドによって生成されるコンテキストを共有します。
- 一部のクライアント・スレッドは複数のコンテキストに参加します。
- 一部のクライアント・スレッドは既存のコンテキストに切り替えます。

注記　BEA Tuxedo システムから独立して動作するスレッドが存在する場合もあります。ここでは、そのようなスレッドについては説明していません。

クライアント・スレッドの複数コンテキストへの参加

BEA Tuxedo マルチコンテキスト・アプリケーションのクライアントは、次の規則に従う限り、複数のアプリケーションに対応付けることができます。

- すべての対応付けは、BEA Tuxedo システムの同じインストールに対して行う必要があります。
- すべてのアプリケーションとの対応付けは、同じタイプのクライアントから行う必要があります。つまり、次のいずれかの条件を満たしている必要があります。
  - すべてのアプリケーションとの対応付けは、ネイテイブ・クライアントから行う必要があります。
  - すべてのアプリケーションとの対応付けは、ワークステーション・クライアントから行う必要があります。

クライアントが複数のコンテキストに参加するには、`TPINFO` データ型の *`flags`* に `TPMULTICONTEXTS` フラグを設定して、`tpinit(3c)` 関数を呼び出します。

TPMULTICONTEXTS フラグを設定して tpinit() 関数が呼び出されると、アプリケーションとの新しい対応付けが生成され、スレッドに対するカレントの対応付けが指定されます。新しい対応付けが生成される BEA Tuxedo ドメインは、TUXCONFIG または WSENVFILE/WSNADDR 環境変数の値で決定されます。

クライアント・スレッドの既存のコンテキストへの切り替わり

多くの ATMI 関数はコンテキスト単位で動作します。このような ATMI 関数のリストについては、10-58 ページの「マルチスレッド・クライアントでのコンテキスト単位の関数とデータ構造体」を参照してください。コンテキスト単位で動作するには、ターゲット・コンテキストが現在のコンテキストであることが必要です。クライアントは複数のコンテキストに参加できますが、状況とスレッドにかかわらず現在のコンテキストになることができるコンテキストは 1 つだけです。

アプリケーション内でタスクの優先順位が移り、ほかの BEA Tuxedo ドメインと通信する必要が発生した場合、あるコンテキストから別のコンテキストにスレッドを再割り当てした方がよい場合があります。

そのような場合、あるクライアント・スレッドが tpgetctxt(3c) を呼び出し、返された現在のコンテキストを値として持つハンドルを別のクライアント・スレッドに渡します。2 番目のスレッドは tpsetctxt(3c) を呼び出し、最初のスレッドで tpgetctxt(3c) から受け取ったハンドルを指定して、現在のコンテキストとの対応付けを確立します。

目的のコンテキストとの対応付けが確立されると、2 番目のスレッドはコンテキスト単位で動作する ATMI 関数を使用してタスクを実行できるようになります。詳細については、10-58 ページの「マルチスレッド・クライアントでのコンテキスト単位の関数とデータ構造体」を参照してください。

作業フェーズ

このフェーズでは、各スレッドによって処理が行われます。次は、行われる処理の例です。

- サービスを要求します。
- サービス要求に対する応答を受け取ります。
- 会話を開始して会話に参加します。
- トランザクションの開始、コミット、またはロールバックを行います。

サービス要求

スレッドは、同期要求の場合は `tpcall(3c)`、非同期要求の場合は `tpacall(3c)` を呼び出して、サーバに要求を送ります。`tpcall()` で要求を送った場合、以降操作を行わなくても応答を受け取ることができます。

サービス要求に対する応答

`tpcall(3c)` でサービスの非同期要求を送った場合、同じコンテキスト内のスレッドは `tpgetrply(3c)` を呼び出して応答を受け取ります。このスレッドは、要求を送ったスレッドと同じスレッドではない場合もあります。

トランザクション

あるスレッドがトランザクションを開始すると、そのスレッドのコンテキストを共有するすべてのスレッドでそのトランザクションが共有されます。

コンテキスト内の多くのスレッドでトランザクションに関する処理が行われますが、トランザクションをコミットまたはアボートできるのは 1 つのスレッドだけです。トランザクションをコミットまたはアボートするスレッドは、トランザクションを開始したスレッドである必要はなく、トランザクションを処理しているどのスレッドでもかまいません。スレッド・アプリケーションでは、通常のトランザクション規則に従うために、適切に同期を行う必要があります。たとえば、未終了の RPC 呼び出しや会話がある場合に、トランザクションをコミットすることはできません。また、トランザクションがコミットまたはアボートされた後で、そのトランザクションに対す

る呼び出しを行うことはできません。プロセスは、アプリケーションの各対応付けに対して、1 つのトランザクションの一部にだけなることができます。

アプリケーションの 1 つのスレッドが tpcommit(3c) を呼び出し、それと同時に別のスレッドが RPC 呼び出しまたは会話型呼び出しを行うと、これらの呼び出しは特定の順序で呼び出されたものとして処理されます。アプリケーション・コンテキストは、シングルスレッド・プログラムとシングルコンテキスト・プログラムに対する制約と同じ制約に従って、tpsuspend(3c) を呼び出してトランザクションを一時的に中断し、別のトランザクションを開始します。

任意通知型メッセージ

マルチスレッド・アプリケーションまたはマルチコンテキスト・アプリケーションの各コンテキストでは、任意通知型メッセージを次の 3 種類のいずれかの方法で処理できます。

処理方法 . .	**設定** . .
任意通知型メッセージの無視	TPU_IGN
ディップ・イン通知	TPU_DIP
専用のスレッド通知 (C 言語のアプリケーションのみで利用可能です)	TPU_THREAD

以下の事柄に注意してください。

- シグナル・ベースの通知は、マルチスレッド・プロセスまたはマルチコンテキスト・プロセスでは使用できません。
- アプリケーションを実行しているプラットフォームで、マルチコンテキストがサポートされていてもマルチスレッドがサポートされていない場合、TPU_THREAD を使用した任意通知型通知の処理を行うことはできません。そのため、イベントの即時通知を受け取ることはできません。

  イベントの即時通知を受け取る必要がある場合は、そのプラットフォームでマルチコンテキストを使用するかどうかを慎重に検討します。

- 専用のスレッド通知は、次のものに対してだけ使用できます。
  - C 言語で記述されたアプリケーション
  - BEA Tuxedo システムでサポートされているマルチスレッド・プラットフォーム

専用のスレッド通知の場合、任意通知型メッセージの受信と、任意通知型メッセージ・ハンドラのディスパッチに別々のスレッドが使用されます。あるコンテキストで一度に実行できる任意通知型メッセージ・ハンドラは 1 つだけです。

スレッドがサポートされていない BEA Tuxedo システム用プラットフォームで `tpinit(3c)` が呼び出された場合に、スレッドがサポートされていないプラットフォーム上で `TPU_THREAD` 通知が要求されたことを示すパラメータが指定されていると、`tpinit()` は `-1` を返して `tperrno` に `TPEINVAL` を設定します。`UBBCONFIG(5)` のデフォルトの `NOTIFY` オプションが `THREAD` に設定されている場合に、特定のマシンでスレッドを利用できないと、そのマシンのデフォルトの機能は `DIPIN` になります。このような動作の相違があるので、スレッドがサポートされているマシンとサポートされていないマシンが混在する環境では、管理者はすべてのマシンにデフォルトを指定できます。ただし、そのマシンで利用できない機能をクライアントが明示的に要求することはできません。

`tpsetunsol(3c)` がコンテキストに対応付けされていないスレッドから呼び出されると、新しく生成されるすべての `tpinit(3c)` コンテキストに対して、プロセス単位のデフォルトの任意通知型メッセージ・ハンドラが作成されます。特定のコンテキストは、コンテキストがアクティブのときに `tpsetunsol()` を再度呼び出して、そのコンテキストの任意通知型メッセージ・ハンドラを変更することができます。プロセス単位のデフォルトの任意通知型メッセージ・ハンドラは、コンテキストに現在対応付けされていないスレッドで `tpsetunsol()` を再度呼び出すと、変更できます。

プロセスが同じアプリケーションと複数の対応付けを持つ場合、各対応付けに異なる `CLIENTID` を割り当てられ、任意通知型メッセージを特定のアプリケーションとの対応付けに送信できるようになります。プロセスが同じアプリケーションと複数の対応付けを持つ場合、ブロードキャスト基準を満たすアプリケーションの各対応付けに任意の `tpbroadcast(3c)` が別々に送信されます。任意通知型メッセージを受信する場合のディップ・イン・チェックでは、カレントのアプリケーションとの対応付けに送信されるメッセージだけが対象となります。

任意通知型メッセージ・ハンドラでは ATMI 関数を利用できるほか、任意通知型メッセージ・ハンドラ内で tpgetctxt(3c) を呼び出すことができます。そのため、任意通知型メッセージ・ハンドラは別のスレッドを生成して、同じコンテキスト内で必要となる実質的な ATMI 作業を行うことができるようになります。

ユーザ・ログで保持されるスレッド固有の情報

userlog(3c) を使用すると、各アプリケーション内の各スレッドに対して次の識別情報が記録されます。

process_ID.thread_ID.context_ID

スレッドがサポートされていないプラットフォームやシングルコンテキスト・アプリケーションに対しては、*thread_ID* フィールドと *context_ID* フィールドにプレースホルダが出力されます。

TM_MIB(5) では、この機能は T_ULOG クラスの TA_THREADID フィールドと TA_CONTEXTID フィールドでサポートされています。

完了フェーズ

このフェーズでは、クライアント・プロセスの終了時に、現在のコンテキストおよび対応付けられたすべてのスレッドに代わって 1 つのスレッドが tpterm(3c) を呼び出してそのアプリケーションとの対応付けを終了します。ほかの ATMI 関数と同じように、tpterm() は現在のコンテキストに対して処理を行います。tpterm() は、終了するコンテキストに対応付けされたすべてのスレッドに影響し、これらのスレッドで共有されるすべてのコンテキストを終了します。

アプリケーションの設計が適切であれば、特定のコンテキスト内のすべての処理が完了してから tpterm() が呼び出されます。tpterm() が呼び出される前に、すべてのスレッドが同期していなければなりません。

関連項目

- 10-5 ページの「マルチスレッドおよびマルチコンテキストとは」
- 10-26 ページの「マルチスレッドおよびマルチコンテキスト・アプリケーションの設計上の検討事項」
- 10-35 ページの「クライアントでマルチコンテキストを使用するためのコーディング」
- 10-50 ページの「マルチスレッド・クライアントのコーディング」
- 10-38 ページの「クライアント終了前のスレッドの同期」

サーバでのマルチスレッドとマルチコンテキストの動作

マルチスレッドおよびマルチコンテキスト・アプリケーションがアクティブの場合、サーバで行われる処理は次の 3 つのフェーズに分類できます。

- 起動フェーズ
- 作業フェーズ
- 完了フェーズ

起動フェーズ

起動フェーズで行われる処理は、コンフィギュレーション・ファイルの MINDISPATCHTHREADS と MAXDISPATCHTHREADS パラメータの値によって異なります。

MINDISPATCHTHREADS **の値** . .	MAXDISPATCHTHREADS **の値** . .	**処理内容** . .
0	> 1	1. BEA Tuxedo システムがスレッド・ディスパッチャを生成します。 2. ディスパッチャが tpsvrinit(3c) を呼び出して、アプリケーションに参加します。
> 0	> 1	1. BEA Tuxedo システムがスレッド・ディスパッチャを生成します。 2. ディスパッチャが tpsvrinit(3c) を呼び出して、アプリケーションに参加します。 3. BEA Tuxedo システムがサービス要求を処理する新しいスレッドと、そのスレッドに対するコンテキストを生成します。 4. システムで生成された新しいスレッドがそれぞれ tpsvrthrinit(3c) を呼び出して、アプリケーションに参加します。

作業フェーズ

このフェーズでは、次の処理が行われます。

- 1 つのサーバに対する複数のクライアント要求が、複数のコンテキストで同時に処理されます。要求ごとに別のスレッドが割り当てられます。
- 必要に応じて、MAXDISPATCHTHREADS で指定された値まで新しいスレッドが生成されます。
- サーバ・スレッドの統計がシステムで保持されます。

サーバ・ディスパッチ・スレッド

クライアントのサービス要求への応答として、サーバ・ディスパッチャは設定可能な最大数まで複数のスレッドを 1 つのサーバに生成します。このサーバは、各種のクライアント要求に同時に割り当てることができます。サーバが tpinit(3c) を呼び出してクライアントになることはできません。

各ディスパッチ・スレッドは、別々のコンテキストと対応付けられています。この機能は会話型サーバと RPC サーバの両方で有用です。特に、会話型サーバではこの機能を利用できないと、ほかの会話接続がサービスを待っている間、クライアント側の会話をアイドル状態で待つことになります。

この機能は、UBBCONFIG(5) ファイルの SERVERS セクションと TM_MIB(5) の次のパラメータで制御されます。

UBBCONFIG **パラメータ**	MIB **パラメータ**	**デフォルト値**
MINDISPATCHTHREADS	TA_MINDISPATCHTHREADS	0
MAXDISPATCHTHREADS	TA_MAXDISPATCHTHREADS	1
THREADSTACKSIZE	TA_THREADSTACKSIZE	0 (オペレーティング・システムのデフォルト値)

- 各ディスパッチ・スレッドは、THREADSTACKSIZE または TA_THREADSTACKSIZE で指定されるスタック・サイズで生成されます。このパラメータが指定されていない場合、または 0 の場合、オペレーティング・システムのデフォルト値が使用されます。オペレーティング・システムのデフォルト値が小さすぎて BEA Tuxedo システムで使用できない場合、その値より大きなデフォルト値が使用されます。
- このパラメータが指定されていない (0) の場合、あるいはオペレーティング・システムで THREADSTACKSIZE 設定がサポートされていない場合は、オペレーティング・システムのデフォルト値が使用されます。
- MINDISPATCHTHREADS または TA_MINDISPATCHTHREADS は、MAXDISPATCHTHREADS または TA_MAXDISPATCHTHREADS 以下でなければなりません。

- MAXDISPATCHTHREADS または TA_MAXDISPATCHTHREADS が 1 の場合、ディスパッチャ・スレッドとサービス関数スレッドは同じスレッドです。
- MAXDISPATCHTHREADS または TA_MAXDISPATCHTHREADS が 1 より大きい場合、ほかのスレッドのディスパッチに使用されるスレッドは、ディスパッチ・スレッドとしてカウントされません。
- システムは最初に MINDISPATCHTHREADS または TA_MINDISPATCHTHREADS のサーバ・スレッドを起動します。
- システムが MAXDISPATCHTHREADS または TA_MAXDISPATCHTHREADS を超えるサーバ・スレッドを起動することはありません。

アプリケーション生成のスレッド

オペレーティング・システム関数を使用して、アプリケーション・サーバ内に新しいスレッドを追加できます。アプリケーション生成のスレッドは、次のように動作します。

- BEA Tuxedo システムから独立して動作します。
- 既存のサーバ・ディスパッチ・スレッドと同じコンテキストで動作します。
- サーバ・ディスパッチ・コンテキストに代わって作業を行います。

アプリケーション内にスレッドを生成する場合、次の制約があります。

- サーバは tpinit(3c) を呼び出してクライアントになることはできません。
- 最初、アプリケーション生成サーバ・スレッドは、どのサーバ・ディスパッチ・コンテキストにも関連していません。アプリケーション生成のサーバ・スレッドは tpsetctxt(3c) を呼び出し、サーバ・ディスパッチ・スレッド内の以前の tpgetctxt(3c) 呼び出しによって返される値を渡して、サーバ・ディスパッチ・コンテキストとの対応付けを確立します。
- アプリケーション生成のサーバ・スレッドは、tpreturn(3c) または tpforward(3c) を呼び出すことはできません。アプリケーション生成のサーバ・スレッドは処理が終了したら、元のディスパッチ・スレッドが

tpreturn() を呼び出す前に、TPNULLCONTEXT に設定されたコンテキストで tpsetctxt(3c) を呼び出す必要があります。

BBL によるシステム・プロセスの正常性チェック

BBL は定期的にサーバを検証します。特定のサービス要求の実行に時間がかかりすぎている場合、BBL はそのサーバを強制終了します。そして、指定されている場合は、そのサーバを再起動します。BBL がマルチコンテキスト・サーバを強制終了した場合、プロセスを強制終了した結果として、実行中のそのほかのサービス呼び出しも終了します。

また、BBL はタイムアウト値を超えてメッセージの受信を待機しているプロセスまたはスレッドにメッセージを送信します。すると、ブロッキング・メッセージ受信への呼び出しが、タイムアウトを示すエラーを返します。

システムで保持されるサーバ・スレッドの統計

BEA Tuxedo システムでは、各サーバに対して次の統計情報が保持されます。

- 使用できるサーバ・ディスパッチ・スレッドの最大数
- 現在使用中のサーバ・ディスパッチ・スレッドの数 (TA_CURDISPATCHTHREADS)
- サーバ起動後の同時実行サーバ・ディスパッチ・スレッドの最大数 (TA_HWDISPATCHTHREADS)
- これまでに起動したサーバ・ディスパッチ・スレッドの数 (TA_NUMDISPATCHTHREADS)

ユーザ・ログで保持されるスレッド固有の情報

userlog(3c) を使用すると、各アプリケーション内の各スレッドに対して次の識別情報が記録されます。

process_ID.thread_ID.context_ID

スレッドがサポートされていないプラットフォームやシングルコンテキスト・アプリケーションに対しては、*thread_ID* フィールドと *context_ID* フィールドにプレースホルダが出力されます。

TM_MIB(5) では、この機能は T_ULOG クラスの TA_THREADID フィールドと TA_CONTEXTID フィールドでサポートされています。

完了フェーズ

アプリケーションをシャットダウンすると、tpsvrthrdone(3c) と tpsvrdone(3c) が呼び出されて、リソース・マネージャのクローズなど、必要な終了処理が行われます。

関連項目

- 10-5 ページの「マルチスレッドおよびマルチコンテキストとは」
- 10-26 ページの「マルチスレッドおよびマルチコンテキスト・アプリケーションの設計上の検討事項」
- 10-44 ページの「サーバでマルチコンテキストとマルチスレッドを使用するためのコーディング」
- 10-65 ページの「マルチスレッド・サーバのコーディング」

マルチスレッドおよびマルチコンテキスト・アプリケーションの設計上の検討事項

マルチスレッドおよびマルチコンテキスト・アプリケーションは、一部の BEA Tuxedo ドメインでは正しく動作しますが、すべてのドメインで正しく動作するとは限りません。そのようなアプリケーションを作成するかどうかは、次の基本事項を検討してから決定します。

- 開発環境と実行時環境
- アプリケーションの設計上の要件
- 使用するスレッド・モデルのタイプ
- ワークステーション・クライアントに対する相互運用性での制約

環境の要件

マルチスレッド・アプリケーションまたはマルチコンテキスト・アプリケーションの開発では、開発環境と実行時環境に関して次の内容を検討します。

- 十分な経験を持つプログラマ・チームがあり、並列性と同期を正しく管理するマルチスレッドおよびマルチコンテキスト・プログラムのコーディングとデバッグを行うことができるかどうか。
- アプリケーションを開発するプラットフォームで、BEA Tuxedo システムのマルチスレッド機能がサポートされているかどうか。マルチスレッド機能は、オペレーティング・システムで提供されるスレッド・パッケージがインストールされ、適切なレベルの機能性が提供されたプラットフォームだけでサポートされます。
- サーバで使用されるリソース・マネージャ (RM) でマルチスレッドがサポートされているかどうか。サポートされている場合は、次の内容も検討します。
  - サーバでマルチスレッド・アクセスを使用するために RM に必要なパラメータを設定する必要があるかどうか。たとえば、マルチスレッド・アプリケーションで Oracle データベースを使用する場合、Oracle に渡される `OPENINFO` 文字列の一部として `THREADS=true` パラメータを設定する必要があります。この設定により、各スレッドが Oracle との別々の対応付けとして動作するようになります。
  - RM で処理の混在モードがサポートされているかどうか。処理の混在モードは、あるプロセスの複数のスレッドが、1 つの RM との対応付けにマップでき、また同じプロセスのほかのスレッドが別の RM との対応付けにマップできるアクセス方法です。たとえば、1 つのプロセス内でスレッド A と B が RM との対応付け X にマップし、同時にスレッド C が RM との対応付け Y にマップできます。

    すべての RM で混在モードがサポートされているわけではありません。プロセス内のすべてのスレッドが同じ RM との対応付けにマップされなければならない場合もあります。アプリケーション生成のスレッド内でトランザクションに関与する RM アクセスを使用するアプリケーションを設計する場合は、RM で混在モードがサポートされていることを確認します。

設計の要件

マルチスレッド・アプリケーションやマルチコンテキスト・アプリケーションの設計では、次の内容を検討します。

- アプリケーションによって実行されるタスクが、マルチスレッドやマルチコンテキストに適しているかどうか。
- 複数の BEA Tuxedo アプリケーションに接続するかどうか。各ターゲット・アプリケーションに必要な接続数はいくつか。
- アプリケーションで考慮すべき同期に関する検討事項は何か。
- 初期アプリケーションの完成後、アプリケーションを別のプラットフォームに移植する必要があるかどうか。

マルチスレッドやマルチコンテキストに適するアプリケーションのタスク

次の表は、アプリケーションをマルチスレッドまたはマルチコンテキストにすべきかどうかを判断するための参考となる検討事項を示しています。この表だけでは十分ではないので、個々の要件に基づいてほかの事項も検討してください。

そのほかの検討事項については、マルチスレッド・アプリケーションやマルチコンテキスト・アプリケーションのプログラミングに関する書籍を参照してください。

検討事項	検討事項に該当する場合に使用する機能
ドメイン機能を使用せずに、クライアントが複数のアプリケーションに接続する必要があるか	マルチコンテキスト
アプリケーション内でクライアントが多重化の機能を果たすか。たとえば、アプリケーション内の 1 つのマシンがそのほかの 100 台のマシンの「代理」に指定されているか	マルチコンテキスト

検討事項	検討事項に該当する場合に使用する機能
クライアントでマルチコンテキストが使用されるか	マルチスレッド。各コンテキストにスレッドを 1 つずつ割り当てると、コードを簡略化できます。
クライアントが 2 つ以上のタスクを長時間個別に実行することで、並列処理によるパフォーマンスがスレッド同期のコストと複雑さを上回るか	マルチスレッド
1 つのサーバで複数の要求を同時に処理するか	マルチスレッド。`MAXDISPATCHTHREADS` に 1 より大きな値を割り当てます。このように設定すると、1 つのサーバで複数のクライアントをそのクライアントのスレッドで処理できるようになります。
クライアントまたはサーバに複数のスレッドがある場合、各スレッドでわずかな処理を行った後でもスレッドを同期する必要があるか	マルチスレッドを使用しない

必要なアプリケーションと接続の数

アクセスするアプリケーションの数と、確立する接続の数を決定します。

- 複数のアプリケーションに接続する場合は、次のいずれかを使用します。
  - シングルスレッドおよびマルチコンテキスト・アプリケーション
  - マルチスレッドおよびマルチコンテキスト・アプリケーション
- 1 つのアプリケーションに複数の接続を確立する場合は、マルチスレッドおよびマルチコンテキスト・アプリケーションを使用します。
- 1 つのアプリケーションに 1 つだけ接続を確立する場合は、次のいずれかを使用します。
  - マルチスレッドおよびシングルコンテキスト・クライアント
  - シングルスレッドおよびシングルコンテキスト・クライアント

いずれの場合も、マルチスレッドおよびマルチコンテキスト・サーバを使用できます。

同期に関する検討事項

これは設計段階での重要な検討事項です。このマニュアルでは、この内容について取り上げていません。マルチスレッド・アプリケーションやマルチコンテキスト・アプリケーションのプログラミングに関する書籍を参照してください。

アプリケーションの移植

後でアプリケーションを移植する必要がある場合、オペレーティング・システムに応じて異なる関数が使用されていることに注目してください。あるプラットフォームで作成した初期バージョンのアプリケーションを後で移植する場合、異なる関数でコードを書き直すためにどれだけのリソース時間が必要なのかを考慮する必要があります。

最適なスレッド・モデル

現在使用されているマルチスレッド・プログラムには、次のようなモデルがあります。

- 従属 / 被従属モデル
- 兄弟モデル
- ワークフロー・モデル

スレッド・モデルについては、このマニュアルでは取り上げていません。アプリケーションのプログラミング・モデルを選択する場合は、利用できるすべてのモデルを調べて、設計の要件を慎重に検討してください。

ワークステーション・クライアントの相互運用性に関する制約

リリース 7.1 の Workstation クライアントと 7.1 以前の BEA Tuxedo システムに基づくアプリケーションとの相互運用性は、次のどの場合でもサポートされています。

- マルチスレッド化またはマルチコンテキスト化されたクライアントではない。
- マルチコンテキスト化されたクライアントである。
- クライアントがマルチスレッド化されており、各スレッドが異なるコンテキストにある。

BEA Tuxedo リリース 7.1 のワークステーション・クライアントで、1 つのコンテキストに複数のスレッドがある場合は、リリース 7.1 より前の BEA Tuxedo システムとは相互運用しません。

関連項目

- 10-10 ページの「マルチスレッドおよびマルチコンテキスト・アプリケーションの利点と問題点」
- 10-32 ページの「マルチスレッドおよびマルチコンテキスト・アプリケーションのプログラミング開始前のガイドライン」

マルチスレッドおよびマルチコンテキスト・アプリケーションのインプリメント

- 10-32 ページの「マルチスレッドおよびマルチコンテキスト・アプリケーションのプログラミング開始前のガイドライン」
- 10-35 ページの「クライアントでマルチコンテキストを使用するためのコーディング」
- 10-44 ページの「サーバでマルチコンテキストとマルチスレッドを使用するためのコーディング」
- 10-50 ページの「マルチスレッド・クライアントのコーディング」
- 10-65 ページの「マルチスレッド・サーバのコーディング」
- 10-65 ページの「マルチスレッドおよびマルチコンテキスト・アプリケーションのコードのコンパイル」

マルチスレッドおよびマルチコンテキスト・アプリケーションのプログラミング開始前のガイドライン

コーディングを開始する前に、次の内容や条件を満たしていることを確認してください。

- 10-33 ページの「マルチスレッド・アプリケーションに必要な条件」
- 10-33 ページの「マルチスレッド・アプリケーションのプログラミングでの一般的な検討事項」
- 10-34 ページの「並列性に関する検討事項」

マルチスレッド・アプリケーションに必要な条件

開発プロジェクトを開始する前に、開発環境が次の条件を満たしていることを確認します。

- BEA Tuxedo システムでサポートされる適切なスレッド・パッケージが、オペレーティング・システムで提供されていることが必要です。

  BEA Tuxedo システムには、スレッド生成用のツールは提供されていません。ただし、ほかのオペレーティング・システムで提供される各種のスレッド・パッケージがサポートされています。スレッドを生成して同期するには、オペレーティング・システム固有の関数を使用する必要があります。オペレーティング・システムでサポートされているスレッド・パッケージを確認するには、『BEA Tuxedo システムのインストール』を参照してください。

- マルチスレッド・サーバを使用している場合は、これらのサーバで使用されるリソース・マネージャでスレッドがサポートされていることが必要です。

マルチスレッド・アプリケーションのプログラミングでの一般的な検討事項

マルチスレッド・プログラムは、十分に経験を持つプログラマがコーディングします。特に、次のようなマルチスレッド固有の設計に関する基本的な知識があることが必要です。

- 複数スレッド間の同時実行制御の必要性
- ほどんどのインスタンスで静的変数の使用を避ける必要性
- マルチスレッド・プログラムでシグナルを使用することにより発生する可能性のある問題

これらは検討事項の一部にすぎず、ここに記せないほど多くの検討事項があります。マルチスレッド・プログラムをコーディングするプログラマは、それらの検討事項を熟知していることが前提となります。これらの検討事項については、マルチスレッド・アプリケーションのプログラミングに関する書籍を参照してください。

並列性に関する検討事項

マルチスレッドを使用すると、1 つのアプリケーションの異なるスレッドが同じ会話で並列処理を行うことができるようになります。この方法はお勧めしませんが、BEA Tuxedo システムで禁止されているわけではありません。異なるスレッドによって同じ会話で並列処理が行われると、システムは同時呼び出しが任意の順序で行われたように動作します。

複数のスレッドを使ってプログラミングする場合、ミューテックスなどの同時実行制御関数を使用して、スレッド間の並列処理を管理する必要があります。以下は、同時実行制御が必要になる 3 つの例です。

- マルチスレッドのスレッドが同じコンテキストで動作する場合、プログラマは関数が必要な順序で実行されるようにします。たとえば、すべての RPC 呼び出しと会話が完了した後でのみ、`tpcommit(3c)` を呼び出せるようにします。これらすべての RPC 呼び出しまたは会話型呼び出しが行われるスレッドとは別のスレッドから `tpcommit()` が呼び出される場合、アプリケーションで何らかの同時実行制御が必要になります。
- 同じように、`tpacall(3c)` と `tpgetrply(3c)` の呼び出しを別々のスレッドで行うことはできますが、アプリケーションが次のいずれかの条件を満たしている必要があります。
  - `tpacall()` が呼び出された後で `tpgetrply()` が呼び出されます。
  - `tpacall()` を呼び出す前に `tpgetrply()` が呼び出される場合、結果が管理されます。
- 複数のスレッドが同じ会話で処理を行うことは可能です。ただし、異なるスレッドが `tpsend(3c)` をほとんど同時に呼び出した場合、システムはそれらの `tpsend()` 呼び出しが任意の順序で行われたように動作します。アプリケーション・プログラマは、それを認識しておかなければなりません。

  ほとんどのアプリケーションで最良の方法は、1 つの会話のすべての処理を 1 つのスレッドにまとめてコーディングすることです。また、同時実行制御を使用して、これらの処理を連続して行う方法もあります。

関連項目

- 10-26 ページの「マルチスレッドおよびマルチコンテキスト・アプリケーションの設計上の検討事項」
- 10-35 ページの「クライアントでマルチコンテキストを使用するためのコーディング」
- 10-44 ページの「サーバでマルチコンテキストとマルチスレッドを使用するためのコーディング」
- 10-50 ページの「マルチスレッド・クライアントのコーディング」
- 10-65 ページの「マルチスレッド・サーバのコーディング」

クライアントでマルチコンテキストを使用するためのコーディング

クライアントでマルチコンテキストを使用するには、次の内容をコーディングします。

- 初期化時にマルチコンテキストを設定します。
- セキュリティをインプリメントします。
- マルチスレッドも使用する場合は、スレッドを同期します。
- コンテキストを切り替えます。
- 各コンテキストの任意通知型メッセージを処理します。

アプリケーションでトランザクションを使用する場合、トランザクションのマルチコンテキストの結果についても注目します。詳細については、10-43 ページの「マルチスレッドおよびマルチコンテキスト・アプリケーションにおけるトランザクションのコーディング規則」を参照してください。

注記　この節で示す手順とコード例は、BEA Tuxedo システムで提供される C 言語のライブラリ関数を参照します。それらに相当する COBOL ライブラリ関数も利用できます。詳細については、『BEA Tuxedo COBOL リファレンス』を参照してください。

コンテキストの属性

コンテキストを使用する場合、コーディングで以下の事柄に注意してください。

- 元のディスパッチ・スレッドが終了する前に、アプリケーション生成のサーバ・スレッドがコンテキストを変更せずに終了すると、`tpreturn(3c)` または `tpforward(3c)` が失敗します。スレッドを終了しても、自動的に `tpsetctxt(3c)` が呼び出されてコンテキストが `TPNULLCONTEXT` に変更されることはありません。
- プロセス内のすべてのコンテキストに、同じバッファ・タイプ・スイッチを使用します。
- ほかのタイプのデータ構造体と同じように、マルチスレッド・アプリケーションでは BEA Tuxedo バッファを適切に使用する必要があります。つまり、次のいずれかが該当する場合、バッファを 2 つの呼び出しで同時に使用することはできません。
    - 両方の呼び出しでバッファが使用される場合
    - 両方の呼び出しでバッファを解放する場合
    - 1 つの呼び出しでバッファを使用し、もう 1 つの呼び出しでバッファを解放する場合
- `tpinit(3c)` を複数回呼び出して、複数アプリケーションに参加するか、または単一アプリケーションに複数の接続を行う場合、`tpinit()` を呼び出すたびに、確立されているセキュリティ・メカニズムを調整する必要があります。

初期化時のマルチコンテキストの設定

クライアントがアプリケーションに参加する準備ができたら、次のコード例に示すように、TPMULTICONTEXTS フラグを設定して tpinit(3c) を指定します。

コード リスト 10-1　クライアントのマルチコンテキスト・アプリケーションへの参加

```
#include <stdio.h>
#include <atmi.h>

TPINIT   *   tpinitbuf;

main()
{
       tpinitbuf = tpalloc(TPINIT, NULL, TPINITNEED(0));

       tpinitbuf->flags = TPMULTICONTEXTS;
              .
              .
              .
       if (tpinit (tpinitbuf) == -1) {

              ERROR_PROCESSING_CODE
       }
                     .
                     .
                     .

}
```

新しいアプリケーションとの対応付けが生成され、TUXCONFIG または WSENVFILE/WSNADDR 環境変数で指定された BEA Tuxedo ドメインに割り当てられます。

注記　1 つのプロセスでは、tpinit(3c) へのすべての呼び出しに TPMULTICONTEXTS フラグを含めるか、または tpinit() へのすべての呼び出しにこのフラグを含めません。この規則には、例外が 1 つあります。つまり、tpterm(3c) への呼び出しが正常に終了して、クライ

アントのすべてのアプリケーション対応付けが終了した場合、次に tpinit() を呼び出すときに必ずしも TPMULTICONTEXTS フラグを含む必要のない状態にプロセスが復元されます。

マルチコンテキスト・クライアントのセキュリティのインプリメント

同じプロセス内の各アプリケーションとの対応付けには、別個にセキュリティ検査を行う必要があります。検査の内容は、アプリケーションで使用されているセキュリティ・メカニズムのタイプによって異なります。たとえば、BEA Tuxedo アプリケーションでは、システム・レベルのパスワードまたはアプリケーション・パスワードを使用します。

マルチコンテキスト・アプリケーションのプログラマは、アプリケーションで使用するセキュリティのタイプを決定し、そのセキュリティをプロセス内の各アプリケーションとの対応付けにインプリメントします。

クライアント終了前のスレッドの同期

クライアントをアプリケーションから切断する準備ができたら、tpterm(3c) を呼び出します。ただし、マルチコンテキスト・アプリケーションでは、tpterm() を呼び出すと現在のコンテキストが破棄されることに注目してください。その場合、現在のコンテキストで動作しているすべてのスレッドが影響を受けます。アプリケーション・プログラマは、tpterm() が不意に呼び出されることがないように、複数のスレッドを使用する場合は注意してください。

まだ処理を行っているスレッドがあるコンテキストでは、tpterm(3c) を呼び出さないようにします。そのような状況で tpterm() を呼び出すと、そのコンテキストと対応付けられていたほかのスレッドが特別な無効コンテキスト状態になります。無効コンテキスト状態では、大部分の ATMI 関数を使用できなくなります。無効コンテキスト状態からスレッドを解放するには、tpsetctxt(3c) または tpterm() を呼び出します。良く設計されたアプリケーションでは、無効コンテキスト状態が生じることはありません。

注記　BEA Tuxedo システムでは、COBOL アプリケーションのマルチスレッドはサポートされていません。

コンテキストの切り替え

次は、2 つのコンテキストからサービスを呼び出すクライアントで行われる処理の手順をまとめたものです。

1. TUXCONFIG 環境変数に firstapp で必要な値を設定します。
2. TPMULTICONTEXTS フラグを設定して tpinit(3c) を呼び出し、最初のアプリケーションに参加します。
3. tpgetctxt(3c) を呼び出して、現在のコンテキストへのハンドルを取得します。
4. tuxputenv() を呼び出して、TUXCONFIG 環境変数の値を secondapp コンテキストに必要な値に切り替えます。
5. TPMULTICONTEXTS フラグを設定して tpinit(3c) を呼び出して、2 番目のアプリケーションに参加します。
6. tpgetctxt(3c) を呼び出して、現在のコンテキストへのハンドルを取得します。
7. tpsetctxt(3c) を呼び出して、firstapp コンテキストからコンテキストの切り替えを開始します。
8. firstapp サービスを呼び出します。
9. tpsetctxt(3c) を呼び出してクライアントを secondapp コンテキストに切り替え、secondapp サービスを呼び出します。
10. tpsetctxt(3c) を呼び出してクライアントを firstapp コンテキストに切り替え、firstapp サービスを呼び出します。
11. tpterm(3c) を呼び出して、firstapp コンテキストを終了します。
12. tpsetctxt(3c) を呼び出してクライアントを secondapp コンテキストに切り替え、secondapp サービスを呼び出します。

13. tpterm(3c) を呼び出して、secondapp コンテキストを終了します。

次のコード例は、この手順を示しています。

注記　コードを簡単にするために、エラー・チェックは省略してあります。

コード リスト 10-2　クライアントでのコンテキストの切り替え

```
#include <stdio.h>
#include "atmi.h"/* BEA Tuxedo ヘッダ・ファイル */

#if defined(__STDC__) || defined(__cplusplus)
main(int argc, char *argv[])
#else
main(argc, argv)
int argc;
char *argv[];
#endif
{

   TPINIT * tpinitbuf;
   TPCONTEXT_T firstapp_contextID, secondapp_contextID;
   /* TUXCONFIG が /home/firstapp/TUXCONFIG に設定されていることを前提とします。*/
   /*
    * BEA Tuxedo システムにマルチコンテキスト・モードで接続します。
 */
   tpinitbuf=tpalloc(TPINIT, NULL, TPINITNEED(0));
   tpinitbuf->flags = TPMULTICONTEXTS;

   if (tpinit((TPINIT *) tpinitbuf) == -1) {
      (void) fprintf(stderr, "Tpinit failed\n");
      exit(1);
}

   /*
    * 現在のコンテキストへのハンドルを取得します。
 */
   tpgetctxt(&firstapp_contextID, 0);

   /*
    * tuxputenv を使用して TUXCONFIG の値を変更し、
    * 別のアプリケーションに参加 (tpinit) します。
 */
   tuxputenv("TUXCONFIG=/home/second_app/TUXCONFIG");

   /*
```

```
    * secondapp に参加 (tpinit) します。
 */
   if (tpinit((TPINIT *) tpinitbuf) == -1) {
      (void) fprintf(stderr, "Tpinit failed\n");
      exit(1);
   }

   /*
    * secondapp のコンテキストへのハンドルを取得します。
*/
   tpgetctxt(&secondapp_contextID, 0);

   /*
    * tpgetctxt から取得したハンドルと tpsetctxt を使用して、
    * 2 つのコンテキスト間で切り替えが
    * できます。firstapp から開始します。
 */

   tpsetctxt(firstapp_contextID, 0);

   /*
    * firstapp で提供されるサービスを呼び出し、
    * 次に secondapp に切り替えます。
 */

   tpsetctxt(secondapp_contextID, 0);

   /*
    * secondapp で提供されるサービスを呼び出します。
 * 次に firstapp に戻ります。
    */

   tpsetctxt(firstapp_contextID, 0);

   /*
    * firstapp で提供されるサービスを呼び出します。操作が終了したら
    * firstapp のコンテキストを終了します。
 */

   tpterm();

   /*
    * secondapp に戻ります。
 */

   tpsetctxt(secondapp_contextID, 0);
   /*
```

```
     * secondapp で提供されるサービスを呼び出します。操作が終了したら
       secondapp のコンテキストを終了し、
       プログラムを終了します。
 */

   tpterm();

   return(0);
}
```

任意通知型メッセージの処理

任意通知型メッセージを処理する各コンテキストでは、任意通知型メッセージ・ハンドラを設定するか、またはプロセス・ハンドラのデフォルトが設定されている場合はそれを使用する必要があります。

tpsetunsol(3c) がコンテキストに対応付けされていないスレッドから呼び出されると、新しく生成されるすべての tpinit(3c) コンテキストに対して、プロセス単位のデフォルトの任意通知型メッセージ・ハンドラが作成されます。特定のコンテキストは、コンテキストがアクティブのときに tpsetunsol() を再度呼び出して、そのコンテキストの任意通知型メッセージ・ハンドラを変更することができます。プロセス単位のデフォルトの任意通知型メッセージ・ハンドラは、コンテキストに現在対応付けされていないスレッドで tpsetunsol() を再度呼び出すと、変更できます。

ハンドラの設定は、シングルスレッド・アプリケーションまたはシングルコンテキスト・アプリケーションのハンドラを設定する場合と同じように行います。詳細については、tpsetunsol(3c) を参照してください。

現在処理を行っているコンテキストを識別するには、任意通知型メッセージ・ハンドラ内で tpgetctxt(3c) を使用します。

マルチスレッドおよびマルチコンテキスト・アプリケーションにおけるトランザクションのコーディング規則

トランザクションを使用する場合、コーディングで以下の事柄に注意してください。

- 1 つのコンテキストで所有できるトランザクションは 1 つだけです。
- 各コンテキストで異なるトランザクションを使用できます。
- ある時点で任意のコンテキストに対応付けされているすべてのスレッドは、そのコンテキストの同じトランザクション状態を共有します。
- スレッドを同期して、すべての会話と RPC 呼び出しが完了してから `tpcommit(3c)` が呼び出されるようにします。
- `tpcommit(3c)` の呼び出しは、特定のトランザクションの 1 つのスレッドからのみ行うことができます。

関連項目

- 10-13 ページの「クライアントでのマルチスレッドとマルチコンテキストの動作」
- 10-50 ページの「マルチスレッド・クライアントのコーディング」

サーバでマルチコンテキストとマルチスレッドを使用するためのコーディング

ここでは、次の内容について説明します。

- マルチコンテキスト・サーバのコーディング規則
- サーバおよびサーバ・スレッドの初期化と終了
- スレッドを生成するためのサーバのプログラミング
- マルチコンテキスト・サーバでアプリケーション・スレッドを生成するためのコード例

注記　この節で示す手順とコード例は、BEA Tuxedo システムで提供される C 言語のライブラリ関数を参照します。詳細については、『BEA Tuxedo C 言語リファレンス』を参照してください。COBOL アプリケーションではマルチコンテキスト・サーバの生成に必要なマルチスレッドがサポートされていないので、C 言語の関数に相当する COBOL ルーチンは利用できません。

コンテキストの属性

コンテキストを使用する場合、コーディングで以下の事柄に注意してください。

- 元のディスパッチ・スレッドが終了する前に、アプリケーション生成のサーバ・スレッドがコンテキストを変更せずに終了すると、`tpreturn(3c)` または `tpforward(3c)` が失敗します。スレッドを終了しても、自動的に `tpsetctxt(3c)` が呼び出されてコンテキストが `TPNULLCONTEXT` に変更されることはありません。
- プロセス内のすべてのコンテキストに、同じバッファ・タイプ・スイッチを使用します。

- ほかのタイプのデータ構造体と同じように、マルチスレッド・アプリケーションでは BEA Tuxedo バッファを適切に使用する必要があります。つまり、次のいずれかが該当する場合、バッファを 2 つの呼び出しで同時に使用することはできません。
  - 両方の呼び出しでバッファを使用する場合
  - 両方の呼び出しでバッファを解放する場合
  - 1 つの呼び出しでバッファを使用し、もう 1 つの呼び出しでバッファを解放する場合

マルチコンテキスト・サーバのコーディング規則

マルチコンテキスト・サーバ使用する場合、コーディングで以下の規則に注意してください。

- サーバ上の BEA Tuxedo ディスパッチャは、同じサービスまたは異なるサービスを複数回ディスパッチでき、ディスパッチされるサービスごとに異なるディスパッチ・コンテキストを生成します。
- サーバは、`tpinit(3c)` を呼び出したり、クライアントとして動作することはできません。サーバ・プロセスで `tpinit()` が呼び出されると、`tpinit()` は `-1` を返して `tperrno(5)` に `TPEPROTO` を設定します。アプリケーション生成のサーバ・スレッドは、`tpsetctxt(3c)` を呼び出す前に ATMI を呼び出すことはできません。
- サーバ・ディスパッチ・スレッドだけが `tpreturn(3c)` または `tpforward(3c)` を呼び出すことができます。
- アプリケーション生成のスレッドが任意のアプリケーションのコンテキストに対応付けされている場合、サーバは `tpreturn(3c)` または `tpforward(3c)` を実行できません。そのため、サーバ・ディスパッチ・スレッドが `tpreturn()` を呼び出す前に、そのコンテキストに対応するアプリケーション生成の各スレッドがコンテキストを `TPNULLCONTEXT` または別の有効なコンテキストに設定して `tpsetctxt(3c)` を呼び出す必要があります。

  この規則に違反すると、`tpreturn(3c)` または `tpforward(3c)` はユーザ・ログにメッセージを書き込み、呼び出し元に `TPESVCERR` を示して、メイ

ン・サーバのディスパッチ・ループに制御を戻します。無効 tpreturn() が実行されたコンテキスト内のスレッドは、無効コンテキスト状態になります。

- tpreturn(3c) または tpforward(3c) が呼び出された時点で未終了の ATMI 呼び出し、RPC 呼び出し、または会話があると、tpreturn() または tpforward() はユーザ・ログにメッセージを書き込み、呼び出し元に TPESVCERR を示して、メイン・サーバのディスパッチ・ループに制御を戻します。
- サーバ・ディスパッチ・スレッドで tpsetctxt(3c) を呼び出すことはできません。
- シングルコンテキストのサーバとは異なり、マルチコンテキスト・サーバ・スレッドでは、同じサーバ・プロセスだけで提供されるサービスを呼び出すことができます。

サーバおよびサーバ・スレッドの初期化と終了

サーバとサーバ・スレッドの初期化と終了には、BEA Tuxedo システムで提供されるデフォルトの関数や独自の関数を使用できます。

表 10-1 初期化と終了を行うデフォルトの関数

目的 . .	使用するデフォルトの関数
サーバの初期化	tpsvrinit(3c)
サーバ・スレッドの初期化	tpsvrthrinit(3c)
サーバの終了	tpsvrdone(3c)
サーバ・スレッドの終了	tpsvrthrdone(3c)

スレッドを生成するためのサーバのプログラミング

マルチコンテキスト・サーバを使用するほとんどのアプリケーションでは、システム生成のディスパッチ・サーバ・スレッドだけが使用されます。ただし、アプリケーション・サーバに新しいスレッドを生成することもできます。この節では、その方法について説明します。

スレッドの生成

オペレーティング・システムのスレッド関数を使用して、アプリケーション・サーバに新しいスレッドを生成できます。これらの新しいスレッドは、BEA Tuxedo システムから独立して動作できます。また、いずれかのサーバ・ディスパッチ・スレッドと同じコンテキストで動作することもできます。

コンテキストへのスレッドの対応付け

アプリケーション生成のサーバ・スレッドは、当初どのサーバ・ディスパッチ・コンテキストにも対応付けられていません。ただし、初期化される前に呼び出された場合、ほとんどの ATMI 関数は暗黙的に `tpinit(3c)` を実行します。サーバで `tpinit()` を呼び出すことは禁止されているので、そのような呼び出しを行うと問題が発生します。サーバ・プロセスが `tpinit()` を呼び出すと、`tpinit()` は -1 を返して `tperrno(5)` に `TPEPROTO` を設定します。

そのため、アプリケーション生成のサーバ・スレッドは、既存のコンテキストと対応付けを行ってから ATMI 関数を呼び出す必要があります。アプリケーション生成のサーバ・スレッドを既存のコンテキストに対応付けるには、以下の手順をコーディングします。

1. サーバ・ディスパッチ・スレッド _A は、`tpgetctxt(3c)` を呼び出して現在のコンテキストへのハンドルを取得します。
2. サーバ・ディスパッチ・スレッド _A は、`tpgetctxt(3c)` が返すハンドルをアプリケーション・スレッド _B に渡します。
3. アプリケーション・スレッド _B は、`tpsetctxt(3c)` を呼び出してサーバ・ディスパッチ・スレッド _A から受け取ったハンドルを指定して、現在のコンテキストとの対応付けを作成します。

4. アプリケーション生成のサーバ・スレッドは、tpreturn(3c) または tpforward(3c) を呼び出すことはできません。元のディスパッチ・スレッドが tpreturn() または tpforward() を呼び出す前に、そのコンテキストにあったすべてのアプリケーション生成のサーバ・スレッドは TPNULLCONTEXT または別の有効なコンテキストに切り替える必要があります。

   この規則に違反すると、tpforward(3c) または tpreturn(3c) が失敗し、呼び出し元にサービス・エラーが示されます。

マルチコンテキスト・サーバでアプリケーション・スレッドを生成するためのコード例

次のコード例は、サービスが別のスレッドを生成してそのサービスの作業を行うマルチコンテキスト・サーバを示しています。このコードは、サーバでアプリケーション・スレッドを生成する必要があるアプリケーションで使用します。オペレーティング・システムのスレッド関数は、オペレーティング・システムによって異なります。このコード例では、POSIX 関数と ATMI 関数が使用されています。

注記　コードを簡単にするために、エラー・チェックは省略してあります。また、BEA Tuxedo システムによってディスパッチされたスレッドだけを使用するマルチコンテキスト・サーバも省略してあります。そのようなサーバのコーディングは、スレッド・セーフのプログラミング方法が使用されている場合、シングルコンテキスト・サーバのコーディングとまったく同じです。

コード リスト 10-3　マルチコンテキスト・サーバでのスレッドの生成

```
#include <pthread.h>
#include <atmi.h>

void *withdrawalthread(void *);

struct sdata {
             TPCONTEXT_T   ctxt;
             TPSVCINFO     *svcinfoptr;
};
```

```
void
TRANSFER(TPSVCINFO *svcinfo)
{
    struct sdata     transferdata;
    pthread_t        withdrawalthreadid;

    tpgetctxt(&transferdata.ctxt, 0);
    transferdata.svcinfoptr = svcinfo;
    pthread_create(&withdrawalthreadid, NULL, withdrawalthread, &transferdata);
    tpcall("DEPOSIT", ...);
    pthread_join(withdrawalthreadid, NULL);
    tpreturn(TPSUCCESS, ...);
}


void *
withdrawalthread(void *arg)
{
    tpsetctxt(arg->ctxt, 0);
    tpopen();
    tpcall("WITHDRAWAL", ...);
    tpclose();
    return(NULL);
}
```

このコードでは、元のディスパッチ・スレッドで DEPOSIT サービスを呼び出し、アプリケーション生成のスレッドで WITHDRAWAL を呼び出して、口座振り替えを行っています。この例では、リソース・マネージャで混在モデルがサポートされていることを前提としています。つまり、サーバのすべてのスレッドが特定のインスタンスと対応付けされていなくても、そのサーバの複数のスレッドが特定のデータベース接続と対応付けられます。ただし、そのようなモデルがサポートされたリソース・マネージャはほとんどありません。

アプリケーション生成のスレッドを使用しないようにすると、このコードはさらに簡単になります。このコード例で tpcall(3c) を 2 回呼び出して行っている並列処理を実現するには、サーバ・ディスパッチ・スレッド内で tpacall(3c) と tpgetrply(3c) をそれぞれ 2 回呼び出します。

関連項目

- 10-20 ページの「サーバでのマルチスレッドとマルチコンテキストの動作」

マルチスレッド・クライアントのコーディング

ここでは、次の内容について説明します。

- マルチスレッド・クライアントのコーディング規則
- クライアントの複数のコンテキストへの初期化
- マルチスレッド環境での応答の取得
- マルチスレッド・マルチコンテキスト環境の環境変数
- マルチスレッド・クライアントでのコンテキスト単位の関数とデータ構造体
- マルチスレッド・クライアントでのプロセス単位の関数とデータ構造体
- マルチスレッド・クライアントでのスレッド単位の関数とデータ構造体
- マルチスレッド・クライアントのコード例

注記　BEA Tuxedo システムでは、COBOL アプリケーションのマルチスレッドはサポートされていません。

マルチスレッド・クライアントのコーディング規則

マルチスレッド・クライアントを使用する場合、コーディングで以下の規則に注意してください。

- 会話が開始されると、同じプロセス内のすべてのスレッドでその会話を操作できます。ハンドルと呼び出し記述子は、同じプロセスの同じコンテキスト内で移植できます。ただし、ほかのコンテキストやプロセスには移植できません。ハンドルと呼び出し記述子は、それらが元々割り当てられていたアプリケーション・コンテキストだけで使用できます。
- 同じプロセスの同じコンテキストで動作するスレッドの場合、そのスレッドが `tpacall()` の呼び出し元かどうかに関係なく、`tpgetrply(3c)` を呼び出して以前の `tpacall(3c)` 呼び出しの応答を受け取ることができます。
- トランザクションをコミットまたはアボートできるのは、1 つのスレッドだけです。トランザクションを開始したスレッドである必要はありません。
- すべての RPC 呼び出しとすべての会話は、トランザクションをコミットする前に完了していなければなりません。未終了の RPC 呼び出しや会話があるときに `tpcommit(3c)` が呼び出されると、`tpcommit()` はトランザクションをアボートし、`-1` を返して `tperrno(5)` に `TPEABORT` を設定します。
- トランザクションがまだコミットまたはアボートしていないことを確認できない場合に、`tpcall(3c)`、`tpacall(3c)`、`tpgetrply(3c)`、`tpconnect(3c)`、`tpsend(3c)`、`tprecv(3c)`、`tpdiscon(3c)` などの関数をトランザクション・モードで呼び出すことはできません。
- 同じコンテキストに 2 つの `tpbegin(3c)` 呼び出しを同時に行うことはできません。
- 既にトランザクション・モードにあるコンテキストに `tpbegin(3c)` を呼び出すことはできません。
- クライアントを使用している場合に複数のドメインに接続するときは、`TUXCONFIG` または `WSNADDR` の値を手動で変更してから `tpinit(3c)` を呼び出します。このような処理が複数のスレッドで行われる場合、環境変数の設定と `tpinit()` 呼び出しを同期する必要があります。クライアントのすべてのアプリケーション対応付けは、以下の規則に従う必要があります。
  - すべての対応付けは、同じリリースの BEA Tuxedo システムに生成します。

- 特定のクライアントのすべてのアプリケーション対応付けはネイティブ・クライアントとして生成するか、またはワークステーション・クライアントとして生成します。

- アプリケーションに参加するには、マルチスレッドのワークステーション・クライアントが、シングルコンテキスト・モードで実行している場合でも、常に TPMULTICONTEXTS フラグを設定して tpinit(3c) 関数を呼び出す必要があります。

クライアントの複数のコンテキストへの初期化

クライアントを複数のコンテキストに参加させるには、TPINIT データ構造体の *flags* 要素に TPMULTICONTEXTS フラグを設定して tpinit(3c) 関数を呼び出します。

1 つのプロセスでは、tpinit(3c) へのすべての呼び出しに TPMULTICONTEXTS フラグを含めます。または、tpinit() へのすべての呼び出しにこのフラグを含めません。この規則には、例外が 1 つあります。つまり、tpterm(3c) への呼び出しが正常に終了して、クライアントのすべてのアプリケーション対応付けが終了した場合、次に tpinit() を呼び出すときに必ずしも TPMULTICONTEXTS フラグを含む必要のない状態にプロセスが復元されます。

TPMULTICONTEXTS フラグを設定して tpinit(3c) 関数が呼び出されると、アプリケーションとの新しい対応付けが生成され、スレッドに対するカレントの対応付けが設定されます。新しい対応付けが生成される BEA Tuxedo ドメインは、TUXCONFIG または WSENVFILE/WSNADDR 環境変数の値で決定されます。

クライアント・スレッドが TPMULTICONTEXTS フラグを設定せずに tpinit(3c) を正常に実行した場合、クライアントのすべてのスレッドがシングルコンテキスト状態 (TPSINGLECONTEXT) になります。

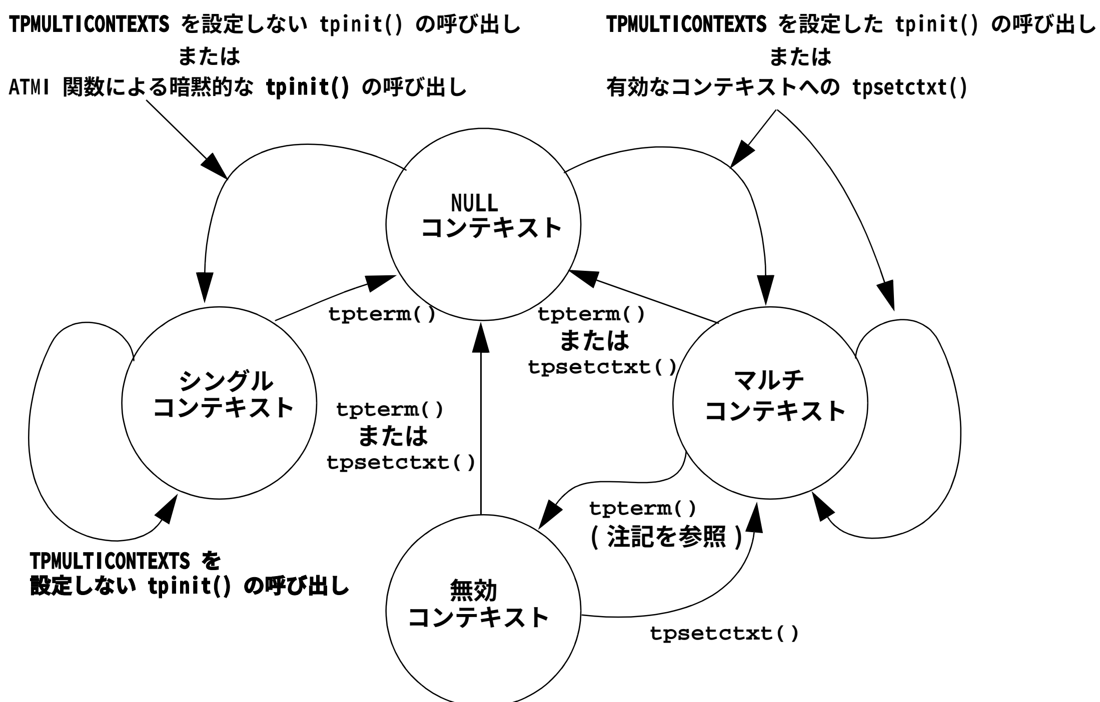
tpinit(3c) が失敗した場合、呼び出し元スレッドは元のコンテキスト、つまり tpinit() 呼び出しの前に操作していたコンテキスト状態のままになります。

まだ動作中のスレッドがあるコンテキストから tpterm(3c) を呼び出すことはできません。このような状況やそれ以外の状況で tpterm() を呼び出した結果生じるコンテキスト状態については、10-54 ページの「マルチコンテキスト状態の遷移」を参照してください。

クライアント・スレッドのコンテキスト状態の変化

マルチコンテキストのアプリケーションでは、いろいろな関数を呼び出すと、呼び出し元スレッド、および呼び出し元プロセスと同じコンテキストでアクティブなその他のスレッドのコンテキスト状態が変化します。次の図は、tpinit(3c)、tpsetctxt(3c)、および tpterm(3c) を呼び出した結果、変化したコンテキスト状態を示しています。tpgetctxt(3c) 関数を呼び出しても、コンテキスト状態は変化しません。

図 10-4 マルチコンテキスト状態の遷移



注記　`tpterm(3c)` がマルチコンテキスト状態 (TPMULTICONTEXTS) で実行しているスレッドによって呼び出されると、呼び出し元スレッドは NULL コンテキスト状態 (TPNULLCONTEXT) になります。終了するコンテキストに関連するその他すべてのスレッドは、無効コンテキスト状態 (TPINVALIDCONTEXT) に切り替わります。

次の表は、`tpinit(3c)`、`tpsetctxt(3c)`、および `tpterm(3c)` を呼び出した場合のコンテキスト状態の変化を示しています。

表 10-2 クライアント・スレッドのコンテキスト状態の変化

実行する関数 . .	**実行後のスレッドのコンテキスト状態 . .**			
	NULL **コンテキスト**	**シングルコンテキスト**	**マルチコンテキスト**	**無効コンテキスト**
TPMULTICONTEXTS が設定されていない tpinit(3c)	シングルコンテキスト	シングルコンテキスト	エラー	エラー
TPMULTICONTEXTS が設定された tpinit(3c)	マルチコンテキスト	エラー	マルチコンテキスト	エラー
TPNULLCONTEXT への tpsetctxt(3c)	NULL	エラー	NULL	NULL
コンテキスト 0 への tpsetctxt(3c)	エラー	シングルコンテキスト	エラー	エラー
コンテキスト > 0 への tpsetctxt(3c)	マルチコンテキスト	エラー	マルチコンテキスト	マルチコンテキスト
暗黙の tpinit(3c)	シングルコンテキスト	該当せず	該当せず	エラー
このスレッドでの tpterm(3c)	NULL	NULL	NULL	NULL
このコンテキストの異なるスレッドでの tpterm(3c)	該当せず	NULL	無効	該当せず

マルチスレッド環境での応答の取得

tpgetrply(3c) は、tpacall(3c) からの要求に対する応答だけを受け取ります。tpcall(3c) からの要求は、マルチスレッドまたはマルチコンテキストのレベルに関係なく、tpgetrply() で取得することはできません。

tpgetrply(3c) は、1 つのコンテキスト、つまり呼び出し元コンテキストだけで動作します。そのため、TPGETANY フラグを設定して tpgetrply() を呼び出すと、同じコンテキストで生成されたハンドルだけが考慮されます。同じように、あるコンテキストで生成されたハンドルを別のコンテキストで使用することはできません。ただし、同じコンテキストで動作するスレッドには、そのハンドルを使用できます。

tpgetrply(3c) をマルチスレッド環境で呼び出す場合、以下の制約があります。

- 1 つのスレッドが既に tpgetrply(3c) で特定のハンドルを待機しているときに、同じコンテキストの別のスレッドが tpgetrply() を呼び出して同じハンドルの取得を試みると、tpgetrply() は -1 を返し、tperrno に TPEPROTO を設定します。
- 1 つのスレッドが既に TPGETANY フラグを設定して tpgetrply(3c) の応答を待機しているときに、同じコンテキストの別のスレッドが tpgetrply() を呼び出して特定のハンドルの取得を試みると、tpgetrply() は -1 を返し、tperrno(5) に TPEPROTO を設定します。

  これは、1 つのスレッドが既に tpgetrply(3c) で特定のハンドルを待機しているときに、同じコンテキストの別のスレッドが TPGETANY フラグを設定して tpgetrply() を呼び出した場合も同じです。これらの制約により、特定のハンドルを待機しているスレッドがある場合、その応答が別のスレッドに渡されることがなくなります。
- ある時点で、TPGETANY フラグを設定して tpgetrply(3c) の応答を待機できるのは、特定のコンテキスト内で 1 つのスレッドだけです。TPGETANY フラグを設定して呼び出した tpgetrply() がまだ処理されていない場合に、同じコンテキストの別のスレッドが同じ呼び出しを行うと、この 2 番目の呼び出しは -1 を返し、tperrno(5) に TPEPROTO を設定します。

マルチスレッド・マルチコンテキスト環境の環境変数

BEA Tuxedo アプリケーションをマルチコンテキスト・マルチスレッド環境で実行する場合、環境変数に関して以下の事柄に注意してください。

- プロセスは、初期状態ではその環境をオペレーティング・システム環境から継承します。環境変数がサポートされているプラットフォームでは、そのような変数はプロセス単位で動作するエンティティを構成します。そのため、コンテキスト単位の環境設定に依存するアプリケーションでは、オペレーティング・システム関数ではなく `tuxgetenv(3c)` 関数を使用する必要があります。

  注記　オペレーティング・システム環境が認識されないオペレーティング・システムの場合、初期状態では空の環境になっています。

- 多くの環境変数は、BEA Tuxedo システムでプロセスごとに 1 回、またはコンテキストごとに 1 回だけ読み取られ、BEA Tuxedo システム内にキャッシュされます。プロセスに一度キャッシュされた変数を変更しても影響はありません。

キャッシュの実行単位 ..	**環境変数** ..
コンテキスト単位	TUXCONFIG
	FIELDTBLS と FIELDTBLS32
	FLDTBLDIR と FLDTBLDIR32
	ULOGPFX
	VIEWDIR と VIEWDIR32
	VIEWFILES と VIEWFILES32
	WSNADDR
	WSDEVICE
	WSENV
プロセス単位	TMTRACE
	TUXDIR
	ULOGDEBUG

- `tuxputenv(3c)` 関数はプロセス全体の環境に影響します。

- tuxreadenv(3c) 関数を呼び出すと、環境変数を含むファイルが読み取られ、それらの変数がプロセス全体の環境に追加されます。
- tuxgetenv(3c) 関数は、現在のコンテキストで要求された環境変数の現在の値を返します。初期設定では、すべてのコンテキストが同じ環境になります。ただし、特定のコンテキストに固有の環境ファイルを使用すると、コンテキストごとに異なる環境設定を持つことができます。
- クライアントが複数のドメインに初期化する場合、tpinit(3c) を呼び出す前に、毎回 TUXCONFIG、WSNADDR、または WSENVFILE 環境変数の値を適切な値に変更する必要があります。そのようなアプリケーションがマルチスレッドの場合、以下の処理を確実に行うために、ミューテックスなどのアプリケーション定義の同時実行制御が必要になります。
  - 適切な環境変数が再設定されること
  - ほかのスレッドによって環境変数が再設定されずに tpinit(3c) が呼び出されること
- クライアントがシステムに初期化する場合、WSENVFILE やマシン環境ファイルが読み取られ、そのコンテキストの環境だけが影響を受けます。環境ファイルで上書きされないコンテキスト部分には、プロセス全体に対する以前の環境が適用されます。

マルチスレッド・クライアントでのコンテキスト単位の関数とデータ構造体

以下に示す ATMI 関数は、呼び出し元のアプリケーション・コンテキストだけに影響します。

- tpabort(3c)
- tpacall(3c)
- tpadmcall(3c)
- tpbegin(3c)
- tpbroadcast(3c)
- tpcall(3c)
- tpcancel(3c)

- `tpchkauth(3c)`
- `tpchkunsol(3c)`
- `tpclose(3c)`
- `tpcommit(3c)`
- `tpconnect(3c)`
- `tpdequeue(3c)`
- `tpdiscon(3c)`
- `tpenqueue(3c)`
- `tpforward(3c)`
- `tpgetlev(3c)`
- `tpgetrply(3c)`
- `tpinit(3c)`
- `tpnotify(3c)`
- tpopen(3c)
- `tppost(3c)`
- `tprecv(3c)`
- `tpresume(3c)`
- `tpreturn(3c)`
- `tpscmt(3c)`
- `tpsend(3c)`
- `tpservice(3c)`
- `tpsetunsol(3c)`
- `tpsubscribe(3c)`
- `tpsuspend(3c)`
- `tpterm(3c)`
- `tpsubscribe(3c)`
- `tx_begin(3c)`
- `tx_close(3c)`
- `tx_commit(3c)`

- tx_info(3c)
- tx_open(3c)
- tx_rollback(3c)
- tx_set_commit_return(3c)
- tx_set_transaction_control(3c)
- tx_set_transaction_timeout(3c)
- userlog(3c)

注記 tpbroadcast(3c) の場合、ブロードキャスト・メッセージは特定のアプリケーションとの対応付けから送られたものとして識別されます。tpnotify(3c) の場合、通知は特定のアプリケーションとの対応付けから送られたものとして識別されます。tpinit(3c) の注記については、「マルチスレッド・クライアントでのプロセス単位の関数とデータ構造体」を参照してください。

tpsetunsol(3c) がコンテキストに対応付けされていないスレッドから呼び出されると、新しく生成されるすべての tpinit(3c) コンテキストに対して、プロセス単位のデフォルトの任意通知型メッセージ・ハンドラが作成されます。特定のコンテキストは、コンテキストがアクティブのときに tpsetunsol() を再度呼び出して、そのコンテキストの任意通知型メッセージ・ハンドラを変更することができます。プロセス単位のデフォルトの任意通知型メッセージ・ハンドラは、コンテキストに現在対応付けされていないスレッドで tpsetunsol() を再度呼び出すと、変更できます。

- CLIENTID、クライアント名、ユーザ名、トランザクション ID、および TPSVCINFO データ構造体の内容は、同じプロセス内のコンテキストによって異なる場合があります。
- 非同期呼び出しハンドルと接続記述子は、その生成元コンテキスト内で有効です。任意通知のタイプは、コンテキストごとに固有です。シグナル・ベースの通知はマルチコンテキストでは使用できませんが、各コンテキストでは次のいずれかのオプションを使用できます。
  - 任意通知型メッセージの無視
  - ディップ・イン通知
  - 専用のスレッド通知

マルチスレッド・クライアントでのプロセス単位の関数とデータ構造体

以下に示す BEA Tuxedo 関数は、呼び出し元のプロセス全体に影響します。

- `tpadvertise(3c)`
- `tpalloc(3c)`
- `tpconvert(3c)`— 要求された構造体が変換されます。ただし、プロセスのサブセットだけに対応します。
- `tpfree(3c)`
- `tpinit(3c)` ープロセス単位の TPMULTICONTEXTS モードまたはシングルコンテキスト・モードに応じて適用されます。10-58 ページの「マルチスレッド・クライアントでのコンテキスト単位の関数とデータ構造体」も参照してください。
- `tprealloc(3c)`
- `tpsvrdone(3c)`
- `tpsvrinit(3c)`
- `tptypes(3c)`
- `tpunadvertise(3c)`
- `tuxgetenv(3c)`— オペレーティング・システム環境がプロセス単位の場合
- `tuxputenv(3c)`— オペレーティング・システム環境がプロセス単位の場合
- `tuxreadenv(3c)`— オペレーティング・システム環境がプロセス単位の場合
- `Usignal(3c)`

シングルコンテキスト・モード、マルチコンテキスト・モード、または非初期化モードのどれを使用するかは、プロセス全体に影響します。また、バッファ・タイプ・スイッチ、ビュー・キャッシュ、および環境変数の値も、プロセス単位の関数です。

マルチスレッド・クライアントでのスレッド単位の関数とデータ構造体

以下に示す関数は、呼び出し元のスレッドだけに影響します。

- CATCH
- tperrordetail(3c)
- tpgetctxt(3c)
- tpgprio(3c)
- tpsetctxt(3c)
- tpsprio(3c)
- tpstrerror(3c)
- tpstrerrordetail(3c)
- TRY(3c)
- Uunix_err(3c)

Ferror、Ferror32(5)、tperrno(5)、tpurcode(5)、および Uunix_err 変数は、各スレッドに固有です。

現在のコンテキストの ID は各スレッドに固有です。

マルチスレッド・クライアントのコード例

次のコード例は、ATMI 呼び出しを使用するマルチスレッド・クライアントを示しています。スレッド関数は、オペレーティング・システムによって異なります。この例では、POSIX 関数が使用されています。

注記　コードを簡単にするために、エラー・チェックは省略してあります。

コード リスト 10-4　マルチスレッド・クライアントのコード例

```
#include <stdio.h>
#include <pthread.h>
#include <atmi.h>

TPINIT   *   tpinitbuf;
int          timeout=60;
pthread_t    withdrawalthreadid, stockthreadid;
TPCONTEXT_T  ctxt;
void * stackthread(void *);
void * withdrawalthread(void *);

main()
{
tpinitbuf = tpalloc(TPINIT, NULL, TPINITNEED(0));
/*
 * このコードでは、withdrawal スレッドと deposit スレッドという別のスレッドを
 * 使用して振り込みを行っています。また、BEA 株の現在の価格を
 * 別のアプリケーションから取得し、送金した金額で
 * 購入できる株数を計算しています。
*/

tpinitbuf->flags = TPMULTICONTEXTS;

/* 残りの tpinitbuf を設定します。*/
tpinit(tpinitbuf);

tpgetctxt(&ctxt, 0);
tpbegin(timeout, 0);
pthread_create(&withdrawalthreadid, NULL, withdrawalthread, NULL);
tpcall("DEPOSIT", ...);

/* withdrawal スレッドの完了を待機します。*/
pthread_join(withdrawalthreadid, NULL);

tpcommit(0);
tpterm();

/* stock スレッドの完了を待機します。*/
pthread_join(stockthreadid, NULL);

/* 結果を出力します。*/
printf("$%9.2f has been transferred \
from your savings account to your checking account.\n", ...);

printf("At the current BEA stock price of $%8.3f, \
```

```
you could purchase %d shares.\n", ...);

exit(0);
}




void *
stockthread(void *arg)
{

     /* ほかのスレッドが tpinit() を呼び出しているので、
      * TUXCONFIG を再設定してもスレッドに影響しません。
*/

     tuxputenv("TUXCONFIG=/home/users/xyz/stockconf");
     tpinitbuf->flags = TPMULTICONTEXTS;
     /* 残りの tpinitbuf を設定します。*/
     tpinit(tpinitbuf);
     tpcall("GETSTOCKPRICE", ...);
      /* main() でもアクセスできる変数に株価を格納します。*/
     tpterm();
     return(NULL);
}



void *
withdrawalthread(void *arg)
{
/* 別のアプリケーションから株価を取得するために新しいスレッドを
 * 生成します。
*/



     pthread_create(&stockthreadid, NULL, stockthread, NULL);
     tpsetctxt(ctxt, 0);
     tpcall("WITHDRAWAL", ...);
     return(NULL);
}
```

関連項目

- 10-13 ページの「クライアントでのマルチスレッドとマルチコンテキストの動作」
- 10-32 ページの「マルチスレッドおよびマルチコンテキスト・アプリケーションのプログラミング開始前のガイドライン」
- 10-35 ページの「クライアントでマルチコンテキストを使用するためのコーディング」

マルチスレッド・サーバのコーディング

ほとんどの場合、マルチスレッド・サーバはマルチコンテキストでもあります。マルチスレッド・サーバのコーディングについては、10-44 ページの「サーバでマルチコンテキストとマルチスレッドを使用するためのコーディング」を参照してください。

マルチスレッドおよびマルチコンテキスト・アプリケーションのコードのコンパイル

`buildserver(1)` や `buildclient(1)` など、コンパイルまたはビルドの実行可能ファイル用に BEA Tuxedo システムで提供されるプログラムには、必要なコンパイラ・フラグが自動的に設定されます。これらのツールを使用すると、コンパイル時にフラグを設定する必要がありません。

ただし、最終的なコンパイルの前に .c ファイルを .o ファイルにコンパイルする場合は、プラットフォーム固有のコンパイラ・フラグを設定する必要があります。そのようなフラグは、単一のプロセスにリンクするすべてのコードに一貫して設定しなければなりません。

マルチスレッド・サーバを生成する場合、-t オプションを指定して buildserver(1) コマンドを実行する必要があります。これはマルチスレッド・サーバの場合に必須のオプションです。ビルド時にこのオプションが指定されておらず、その後、MAXDISPATCHTHREADS の値が 1 を超えるコンフィギュレーション・ファイルを使用して新しいサーバを起動すると、警告メッセージがユーザ・ログに記録され、サーバはシングルスレッドの動作に戻ります。

マルチスレッド環境で .c ファイルを .o ファイルにコンパイルする場合に必要なオペレーティング・システム固有のコンパイラ・パラメータを識別するには、-v オプションを指定して buildclient(1) または buildserver(1) をテスト・ファイルで実行します。

関連項目

- 10-35 ページの「クライアントでマルチコンテキストを使用するためのコーディング」
- 10-44 ページの「サーバでマルチコンテキストとマルチスレッドを使用するためのコーディング」
- 10-50 ページの「マルチスレッド・クライアントのコーディング」

マルチスレッドおよびマルチコンテキスト・アプリケーションのテスト

ここでは、次の内容について説明します。

- マルチスレッドおよびマルチコンテキスト・アプリケーションのテスト時の推奨事項
- マルチスレッドおよびマルチコンテキスト・アプリケーションのトラブル・シューティング
- マルチスレッドおよびマルチコンテキスト・アプリケーションのエラー処理

マルチスレッドおよびマルチコンテキスト・アプリケーションのテスト時の推奨事項

マルチスレッドやマルチコンテキストのコードをテストする場合、以下を行うことをお勧めします。

- マルチプロセッサの使用
- マルチスレッド・デバッガの使用 (オペレーティング・システムのベンダから提供されている場合)
- いろいろなタイミング条件でのストレス・テスト

マルチスレッドおよびマルチコンテキスト・アプリケーションのトラブル・シューティング

エラーの原因を調べる場合、まず TPMULTICONTEXTS フラグが設定されているかどうか、またその設定内容を確認します。このフラグが設定されていないこと、または正しく設定されていないことが原因でよくエラーが起こります。

tpinit( ) の TPMULTICONTEXTS フラグの間違った使用

TPMULTICONTEXTS フラグを使用できない場合にこのフラグがプロセスに設定されているとき、または TPMULTICONTEXTS を設定する必要がある場合に設定されていないとき、tpinit(3c) は -1 を返し、tperrno に TPEPROTO を設定します。

TPMULTICONTEXTS が設定されていない場合の tpinit() の呼び出し

TPMULTICONTEXTS が設定されていない場合に tpinit(3c) が呼び出されると、この関数はシングルコンテキスト・アプリケーションで呼び出されたときと同じように動作します。tpinit() が既に 1 回呼び出されている場合、それ以降の tpinit() 呼び出しで TPMULTICONTEXTS フラグが設定されていなくても、この関数は正常に終了します。これは、アプリケーション内の TUXCONFIG または WSNADDR 環境変数の値が変更されている場合にも当てはまります。TPMULTICONTEXTS フラグを設定せずに tpinit() を呼び出すことは、マルチコンテキスト・モードではできません。

クライアントがアプリケーションに参加していない場合に、tpinit(3c) を呼び出す別の関数の呼び出しの結果として、暗黙的に tpinit() が呼び出されると、BEA Tuxedo システムでは TPMULTICONTEXTS フラグが設定されずに tpinit() が呼び出されたと解釈されます。これは、以降の tpinit() 呼び出しでどのフラグが使用されるかを判断するためです。

ほとんどの ATMI 関数は、既にマルチコンテキスト・モードで動作しているプロセスのコンテキストに対応付けされていないスレッドに呼び出された場合、tperrno(5)=TPEPROTO が設定されて失敗します。

スレッドのスタック・サイズの不足

一部のオペレーティング・システムでは、オペレーティング・システムのデフォルトのスレッド・スタック・サイズが BEA Tuxedo システムで使用するには十分ではありません。Compaq Tru64 UNIX と UnixWare の 2 つのオペレーティング・システムは、サイズが小さいことが認識されています。デフォルトのスレッド・スタック・サイズのパラメータが使用されている場合に、スタックを多用する関数がメイン・スレッド以外のスレッドで呼び出されると、これらのプラットフォーム上のアプリケーションはコア・ダンプします。通常、生成されるコア・ファイルから、スタック・サイズの不足が問題の原因であることはわかりません。

サーバ・ディスパッチ・スレッドやクライアントの任意通知型メッセージ・スレッドなど、BEA Tuxedo システムで独自のスレッドが生成される場合、これらのプラットフォームのデフォルトのスタック・サイズのパラメータを適切な値に調整できます。ただし、アプリケーションで独自のスレッドが生成される場合、アプリケーションで十分なスタック・サイズを指定する必要があります。BEA Tuxedo システムにアクセスするスレッドには、最低 128 K を指定してください。

Compaq Tru64 UNIX および POSIX スレッドが使用されるそのほかのシステムでは、スレッドのスタック・サイズは、`pthread_create()` を呼び出す前に `pthread_attr_setstacksize()` を呼び出して指定します。UnixWare では、スレッドのスタック・サイズは `thr_create()` の引数として指定されます。この問題の詳細については、お使いのオペレーティング・システムのマニュアルを参照してください。

マルチスレッドおよびマルチコンテキスト・アプリケーションのエラー処理

エラーはユーザ・ログに記録されます。シングルコンテキスト・モードでもマルチコンテキスト・モードでも、各エラーに対して次の情報が記録されます。

process_ID.thread_ID.context_ID

関連項目

- 10-13 ページの「クライアントでのマルチスレッドとマルチコンテキストの動作」
- 10-20 ページの「サーバでのマルチスレッドとマルチコンテキストの動作」
- 10-32 ページの「マルチスレッドおよびマルチコンテキスト・アプリケーションのプログラミング開始前のガイドライン」

11 エラーの管理

ここでは、次の内容について説明します。

- システム・エラー
- アプリケーション・エラー
- エラー処理
- トランザクションについて
- 中央イベント・ログ

システム・エラー

BEA Tuxedo システムでは、TP-STATUS IN TPSTATUS-REC を使用して、ルーチンが失敗した場合にプロセスに情報が渡されます。すべての ATMI 呼び出しは、TP-STATUS にエラーの内容を示す値を設定します。サービス・ルーチンを終了させるために使用する TPRETURN や TPFORWAR など、呼び出し元に戻らない関数の場合、成功か失敗かを確認する唯一の方法は要求元の TP-STATUS 変数です。

APPL-RETURN-CODE は、ユーザ定義の条件だけを通知します。APPL-RETURN-CODE には、TPRETURN 時に APPL-CODE IN TPSVCRET-REC の値が設定されます。TPRETURN エラーまたはトランザクション・タイムアウトが発生しない限り、APPL-RETURN-CODE IN TPSTATUS-REC の値に関係なく、APPL-RETURN-CODE に値が設定されます。

TP-STATUS に返されるコードは、エラーの種類を示します。次の表は、そのエラーの種類を示しています。

表 11-1TP-STATUS エラーの種類

エラーの種類	TP-STATUS の値
アボート	TPEABORT[2]
BEA Tuxedo システム [1]	TPESYSTEM
通信ハンドル	TPELIMIT と TPEBADDESC
会話	TPEVENT
複製操作	TPEMATCH
一般的な通信	TPESVCFAIL、TPESVCERR、TPEBLOCK、および TPGOTSIG
ヒューリスティックな判断	TPEHAZARD[2] と TPEHEURISTIC[2]
無効な引数 [1]	TPEINVAL
MIB	TPEMIB
エントリなし	TPENOENT
オペレーティング・システム [1]	TPEOS
パーミッション	TPEPERM
プロトコル[1]	TPEPROTO
キューへの登録、取り出し	TPEDIAGNOSTIC
リリース間の互換性	TPERELEASE
リソース・マネージャ	TPERMERR
タイムアウト	TPETIME
トランザクション	TPETRAN[2]
型付きレコードの不一致	TPEITYPE と TPEOTYPE

1. TP-STATUS で返される値によって失敗が通知されるすべての ATMI 呼び出しに適用されます。
2. このエラーの詳細については、11-19 ページの「致命的なトランザクション・エラー」を参照してください。

脚注 1 にあるように、TP-STATUS によって通知される 4 種類のエラーは、すべての ATMI 関数で発生するエラーです。それ以外のエラーの種類は、特定の ATMI 呼び出しだけで発生します。以下に、一部のエラーの種類について詳しく説明します。

アボート・エラー

アボートの原因となるエラーについては、11-19 ページの「致命的なトランザクション・エラー」を参照してください。

BEA Tuxedo のシステム・エラー

BEA Tuxedo のシステム・エラーは、問題がアプリケーション・レベルではなくシステム・レベルで発生していることを示します。BEA Tuxedo のシステム・エラーが発生すると、エラーの原因を示すメッセージが中央イベント・ログに書き込まれ、TP-STATUS に TPESYSTEM が返されます。詳細については、11-30 ページの「中央イベント・ログ」を参照してください。これらのエラーは、アプリケーションではなくシステムで発生するので、エラーの修正についてはシステム管理者に問い合わせてください。

通信ハンドルのエラー

通信ハンドルのエラーは、通信ハンドルの数が上限値を超えている場合、または無効な値を参照している場合に発生します。非同期呼び出しや会話型呼び出しでは、未処理の通信ハンドルの数が上限値を超えると、TPELIMIT が返されます。操作に対して無効な通信ハンドルの値が指定されている場合は、TPEBADDESC が返されます。

通信ハンドルのエラーが発生するのは、非同期呼び出しまたは会話型呼び出しを行った場合だけです。同期呼び出しでは、呼び出し記述子は使用されません。非同期呼び出しでは、通信ハンドルを使用して対応する要求に応答が対応付けられます。会話型送受信用のルーチンは、通信ハンドルを使用して接続を識別します。つまり、接続を開始する呼び出しでは、通信ハンドルを使用できることが大切です。

通信ハンドルのエラーのトラブル・シューティングは、アプリケーション・レベルで特定のエラーを調べて行います。

上限値に関するエラー

システムでは、コンテキスト (または BEA Tuxedo アプリケーションへの対応付け) ごとに未処理の通信ハンドル (応答) を 50 個まで使用できます。この上限値はシステムで定義されているので、アプリケーションで再定義することはできません。

会話型接続を同時に行う場合の通信ハンドルに関する制限は、応答時の制限ほど厳しくありません。上限値は、アプリケーション管理者がコンフィギュレーション・ファイルに定義します。アプリケーションが実行中ではない場合、管理者はコンフィギュレーション・ファイルの RESOURCES セクションの MAXCONV パラメータを変更できます。アプリケーションが実行中の場合も、MACHINES セクションは動的に変更できます。詳細については、『BEA Tuxedo コマンド・リファレンス』の tmconfig、wtmconfig(1) を参照してください。

無効な記述子によるエラー

通信ハンドルは無効になることがあります。無効な通信ハンドルが参照されると、次の場合に TP-STATUS にエラーが返されます。

- 通信ハンドルを使用してメッセージを取得したが、それがエラー・メッセージの場合 (TPEBADDESC)
- 無効になった通信ハンドルの再利用を試みた場合 (TPEBADDESC)

通信ハンドルが無効になるのは、以下のような場合です。

- アプリケーションで TPABORT または TPCOMMIT を呼び出したとき、TPNOTRAN フラグを設定せずに送信されたトランザクション応答が取得されずに残っている場合。
- トランザクションがタイムアウトになった場合。TPGETRPLY の呼び出しでタイムアウトが通知された場合、指定されたハンドルを使用してメッセージを取得することはできず、ハンドルは無効になります。

会話に関するエラー

会話型サービスで不明なハンドルが指定されると、TPSEND、TPRECV、および TPDISCON ルーチンは TPEBADDESC を返します。

会話型接続の確立後に TPSEND と TPRECV が TPEEVENT エラーで失敗した場合、イベントが発生します。TPSEND でデータを送信できるかどうかは、発生したイベントによって決まります。システムは、TPSTATUS-REC の TPEVENT メンバの TPEEVENT を返します。行われる処理は、発生したイベントによって異なります。

会話型イベントの詳細については、7-14 ページの「会話型通信イベント」を参照してください。

複製オブジェクトに関するエラー

処理の結果として複製オブジェクトが生成されるような操作が試みられると、TP-STATUS に TPEMATCH エラー・コードが返されます。次の表は、TPEMATCH エラー・コードを返すルーチンとその原因を示しています。

ルーチン	原因
TPADVERTISE	指定された *svcname* は、既にサーバに対して宣言されています。ただし、その処理は *func* 以外の関数で行われています。この関数は失敗しても、*svcname* は現在の関数で宣言されたままになります。つまり、*func* は現在の関数名を置き換えません。
TPRESUME	*tranid* が、別のプロセスが既に再開したトランザクション識別子を指しています。その場合、呼び出し元のトランザクションの状態は変化しません。
TPSUBSCRIBE	指定されたサブスクリプション情報は、既にイベント・ブローカに登録されています。

これらのルーチンの詳細については、『BEA Tuxedo COBOL リファレンス』を参照してください。

一般的な通信呼び出しのエラー

一般的な通信呼び出しのエラーは、呼び出しが同期または非同期で行われたかどうかに関係なく、どのような通信呼び出しでも発生する可能性があります。TP-STATUS には、TPESVCFAIL、TPESVCERR、TPEBLOCK、または TPGOTSIG が返されます。

TPESVCFAIL および TPESVCERR エラー

TPCALL または TPGETRPLY を呼び出した結果、通信の応答部分が失敗すると、TP-STATUS に TPESVCERR または TPSEVCFAIL が返されます。TPRETURN に渡された引数でエラーが判別され、この呼び出しで実行する処理が決定されます。

引数の処理中に TPRETURN でエラーが発生すると、システムはエラーを元の要求元に返し、TP-STATUS に TPESVCERR を設定します。受信側では、TP-STATUS の値を調べてエラーの発生を確認します。システムでは、TPRETURN 呼び出しからのデータ送信は行われず、TPGETRPLY でエラーが発生した場合は、呼び出しハンドルが無効なものと見なされます。

TPRETURN で TPESVCERR エラーが発生していない場合、TP-RETURN-VAL に返された値で呼び出しが成功したか失敗したかを判断できます。アプリケーションで、TP-RETURN-VAL に TPFAIL が指定されると、システムは TP-STATUS に TPESVCFAIL を返し、呼び出し元にデータ・メッセージを送信します。TP-RETURN-VAL に TPSUCCESS が設定されると、呼び出し元に制御が正常に戻り、TP-STATUS は設定されず、呼び出し元がデータを受信します。

TPEBLOCK および TPGOTSIG エラー

TPEBLOCK および TPGOTSIG エラー・コードは、メッセージの要求側に返される場合も応答側に返される場合もあるので、すべての通信呼び出しに対して返される可能性があります。

ブロッキング状態が発生している場合に、要求を同期または非同期に送信するプロセスでブロッキング状態を無視するように TPPNOBLOCK が設定されていると、システムは TPEBLOCK を返します。たとえば、システムのキューがすべていっぱいになっている場合、要求が送信されるとブロッキング状態になります。

TPCALL がブロッキング状態を示していない場合は、通信の送信部分だけに影響します。要求の送信に成功すると、その呼び出しが応答を待っている間にブロッキング状態が存在したとしても、TPEBLOCK は返されません。

TPNOBLOCK を設定して呼び出しを行った場合、TPGETRPLY が応答を待っている間にブロッキング状態が発生すると、TPGETRPLY に TPEBLOCK が返されます。この状況は、メッセージがその時点で使用できない場合などに発生します。

TPGOTSIG エラーは、シグナルによってシステム・コールに割り込みが発生したことを示します。このような状況は、実際にはエラーではありません。TPSIGRSTRT が設定されていると、呼び出しは失敗せず、TP-STATUS に TPGOTSIG エラー・コードが返されます。

無効な引数によるエラー

無効な引数によるエラーは、ルーチンに渡された引数が無効であることを示しています。引数を取る ATMI 呼び出しは、無効な引数が渡されると失敗します。呼び出し元に制御が戻る呼び出しの場合、関数は失敗して、TP-STATUS に TPEINVAL が設定されます。TPRETURN または TPFORWAR の場合、要求を開始して結果を待っている TPCALL または TPGETRPLY に対して、TP-STATUS に TPESVCERR が設定されます。

ルーチンに有効な引数だけを渡すようにすると、無効な引数によるアプリケーション・レベルでのエラーを修正できます。

エントリがないために発生するエラー

レコード・タイプを識別するためのデータ構造体やシステム・テーブルにエントリがないと、エラーが発生します。エントリ・タイプのエラーを示す TPENOENT の意味は、そのエラーを返す呼び出しによって異なります。次の表は、このエラーを返す呼び出しとエラーのさまざまな原因を示しています。

表 11-2 エントリがないために発生するエラー

呼び出し	原因
TPINITIALIZE	エントリ用の領域が掲示板に残っていないため、呼び出し元プロセスがアプリケーションに参加できません。システム管理者に問い合わせてください。
TPCALL TPACALL	呼び出し元プロセスが参照しているサービス SERVICE-NAME IN TPSVCDEF-REC は、掲示板にエントリがないため、システムに認識されせん。アプリケーション・レベルでサービスを正しく参照しなければなりません。正しく参照していない場合は、システム管理者に問い合わせてください。
TPCONNECT	指定されたサービスに接続できません。そのようなサービス名が存在していないか、または会話型サービスではありません。
TPGPRIO	要求が行われていないにもかかわらず、呼び出し元プロセスが要求の優先度を調べています。これは、アプリケーション・レベルのエラーです。
TPUNADVERTISE	SERVICE-NAME IN TPSVCDEF-REC の宣言を取り消すことができません。この名前は、呼び出し元プロセスによって現在宣言されていません。

オペレーティング・システムのエラー

オペレーティング・システムのエラーは、オペレーティング・システム・コールが失敗したことを示します。TP-STATUS に TPEOS が返されます。UNIX システムの場合、失敗したシステム・コールを識別する整数値がグローバル変数 Uunixerr に返されます。オペレーティング・システム・エラーを修正するには、システム管理者に問い合わせてください。

パーミッション・エラー

呼び出し元プロセスに、アプリケーションに参加するために必要なパーミッションが設定されていない場合、TPINITIALIZE 呼び出しは失敗して、TP-STATUS に TPEPERM が返されます。パーミッションは、コンフィギュレーション・ファイルに設定されるもので、アプリケーションには設定されません。このエラーが発生した場合は、アプリケーション管理者に問い合わせて、必要なパーミッションがコンフィギュレーション・ファイルに設定されていることを確認してください。

プロトコル・エラー

ATMI 呼び出しが間違った順序で行われた場合、または間違ったプロセスを使用して行われた場合、プロトコル・エラーが発生します。たとえば、アプリケーションに参加する前に、クライアントがサーバとの通信の開始を試みると、このエラーが発生します。また、イニシエータではなくトランザクションのパーティシパントによって TPCOMMIT が呼び出された場合も、このエラーが発生します。

ATMI 呼び出しを正しい順序で正しく使用すると、アプリケーション・レベルでプロトコル・エラーを修正できます。

プロトコル・エラーの原因を特定するには、次の事柄を確認してください。

- 正しい順序で呼び出しが行われているかどうか
- 正しいプロセスによって呼び出しが行われているかどうか

プロトコル・エラーでは、TP-STATUS に TPEPROTO 値が返されます。

詳細については、『BEA Tuxedo COBOL リファレンス』の「COBOL アプリケーション・トランザクション・モニタ・インターフェイスの紹介」を参照してください。

キューに関するエラー

特定のキューへの登録またはキューからの取り出しに失敗した場合、TPENQUEUE(3cbl) または TPDEQUEUE(3cbl) ルーチンは TP-STATUS に TPEDIAGNOSTIC を返します。処理が失敗した原因は、*ctl* レコードを介して返される診断値によって判別できます。有効な *ctl* フラグについては、『BEA Tuxedo COBOL リファレンス』の TPENQUEUE(3cbl) または TPDEQUEUE(3cbl) を参照してください。

リリース間の互換性に関するエラー

アプリケーション・ドメインに参加する BEA Tuxedo システムのリリース間で互換性に問題がある場合、BEA Tuxedo システムは TP-STATUS に TPERELEASE を返します。

たとえば、TPNOTIFY(3cbl) ルーチンを呼び出す際に、呼び出し元がターゲット・クライアントから承認メッセージを受け取るまでブロックすることを示す TPACK フラグが設定されている場合、ターゲット・クライアントが TPACK 承認プロトコルがサポートされていない旧バージョンの BEA Tuxedo システムを使用していると、TPERELEASE エラーが返されます。

リソース・マネージャ・エラー

リソース・マネージャ・エラーは、TPOPEN(3cbl) および TPCLOSE(3cbl) を呼び出したときに発生し、TP-STATUS に TPERMERR が返されます。リソース・マネージャを正しくオープンできなかった場合、TPOPEN でこのエラー・コードが返されます。同じように、リソース・マネージャを正しくクローズできなかった場合、TPCLOSE でこのエラー・コードが返されます。BEA

Tuxedo システムでは、移植性を保つために、この種類のエラーでは詳細な情報は返されません。リソース・マネージャ・エラーの正確な内容を判断するには、リソース・マネージャに問い合わせる必要があります。

タイムアウト・エラー

BEA Tuxedo システムでは、タイムアウト・エラーがサポートされており、アプリケーションがサービス要求またはトランザクションを待つ時間に制限があります。BEA Tuxedo システムでサポートされている設定可能なタイムアウト機構は、ブロッキング・タイムアウトとトランザクション・タイムアウトの 2 種類です。

ブロッキング・タイムアウトは、アプリケーションがサービス要求に対する応答を待つ時間の上限値を指定します。アプリケーション管理者は、コンフィギュレーション・ファイルにシステムのブロッキング・タイムアウトを設定します。

トランザクション・タイムアウトは、トランザクション (サービス要求が行われる場合もあり) の有効期間を定義します。アプリケーションのトランザクション・タイムアウトを定義するには、TPBEGIN に T-OUT 引数を渡します。

通信呼び出しでは、ブロッキング・タイムアウトまたはトランザクション・タイムアウトのいずれかが返され、TPCOMMIT ではトランザクション・タイムアウトだけが返されます。いずれの場合も、トランザクション・モードのプロセスで呼び出しが失敗して TPETIME が返された場合は、トランザクション・タイムアウトが発生しています。

デフォルトでは、プロセスがトランザクション・モードではない場合、ブロッキング・タイムアウトが実行されます。

プロセスがトランザクション・モードではない場合に、非同期呼び出しでブロッキング・タイムアウトが発生すると、ブロックされている通信呼び出しは失敗します。ただし、呼び出し記述子は有効なままであり、再度呼び出しを行う場合に使用できます。ほかの通信には影響ありません。

トランザクション・タイムアウトが発生すると、非同期トランザクション応答の通信ハンドル (TPNOTRAN フラグが指定されていないもの) は無効になり、参照できなくなります。

呼び出しがトランザクション・モードで行われていない場合、または TPNOBLOCK が設定されていない場合、TPETIME は通信呼び出しでブロッキング・タイムアウトが発生したことを示します。

注記　TPNOBLOCK が設定されている場合、ブロッキング状態が存在すると呼び出しは直ちに戻るので、ブロッキング・タイムアウトは発生しません。

タイムアウト・エラーの処理の詳細については、11-16 ページの「トランザクションについて」を参照してください。

トランザクション・エラー

トランザクション、および致命的なエラーと致命的ではないエラーについては、11-16 ページの「トランザクションについて」を参照してください。

型付きレコードのエラー

プロセスに対する要求または応答が不明なタイプのレコードで送信された場合、型付きレコードのエラーが返されます。要求データ・レコードの送信先のサービスでレコード・タイプが認識されない場合、TPCALL および TPACALL 呼び出しは TPEITYPE を返します。

プロセスで認識されるレコード・タイプは、コンフィギュレーション・ファイルとプロセスにリンクされている BEA Tuxedo システム・ライブラリの両方で識別されるものです。これらのライブラリは、プロセスで認識される型付きレコードを識別するデータ構造体を定義および初期化します。プロセスごとにライブラリを作成するか、またはレコード・タイプが定義されたアプリケーション固有のファイルのコピーをアプリケーションで用意することが

できます。アプリケーションでは、レコード・タイプ・スイッチと呼ばれるレコード・タイプ・データ構造体をプロセスごとに設定できます。詳細については、『BEA Tuxedo のファイル形式とデータ記述方法』の tuxtypes(5) および typesw(5) を参照してください。

呼び出し元で認識されないか、または使用できないレコードで応答メッセージが送信されると、TPCALL および TPGETRPLY 呼び出しは TPEOTYPE を返します。呼び出し元で使用できないレコードの場合、そのレコード・タイプはタイプ・スイッチに含まれています。ただし、返されたタイプは応答の受信用に割り当てられたレコードと一致せず、また呼び出し元は異なるレコード・タイプを使用することはできません。呼び出し元は、TPNOCHANGE を設定して、このような状況を示します。その場合、厳密なタイプ・チェックが行われ、違反が見つかると TPEOTYPE が返されます。デフォルトでは、緩やかなタイプ・チェックが行われます。その場合、呼び出し元で認識される限り、最初に割り当てられたタイプ以外のレコード・タイプが返されることもあります。応答の送信では、応答レコードは呼び出し元で認識できるものでなければなりません。また、厳密なタイプ・チェックが指定されている場合は、それに従う必要があります。

アプリケーション・エラー

アプリケーション内では、TPRETURN の *rcode* 引数を使用して、呼び出し元のプログラムにユーザ定義のエラーに関する情報を渡すことができます。また、APPL-RETURN-CODE には、TPRETURN 時に APPL-CODE IN TPSVCRET-REC の値が設定されます。TPRETURN(3cbl) の詳細については、『BEA Tuxedo COBOL リファレンス』を参照してください。

エラー処理

アプリケーションのロジックは、戻り値がある呼び出しのエラー条件を調べ、エラー発生時に適切な処理を行うように設計します。

次のコード例は、エラーの一般的な処理方法を示しています。この例では、ATMICALL(3) は、一般的な ATMI 呼び出しを表しています。

コード リスト 11-1　エラー処理

```
. . .
CALL "TPINITIALIZE" USING TPINFDEF-REC
                    USR-DATA-REC
                    TPSTATUS-REC.
IF NOT TPOK
   error message, EXIT PROGRAM
CALL "TPBEGIN" USING TPTRXDEF-REC
               TPSTATUS-REC.
IF NOT TPOK
   error message, EXIT PROGRAM

   Make atmi calls
  Check return values

IF  TPEINVAL
      DISPLAY "Invalid arguments were given."
IF  TPEPROTO
      DISPLAY "A call was made in an improper context."
```

```
. . .
    ATMICALL(3) リファレンス・ページに説明されている
    すべての場合のエラーを含めます。ほかの戻りコードは使用できません。
    そのため、それらをテストする必要はありません。
. . .
(続く)
```

TP-STATUS の値は、各問題の詳細を示し、どのレベルで問題の解決が可能かを示しています。アプリケーションで、ある処理に特定のエラー条件が定義されている場合、APPL-RETURN-CODE IN TPSTATUS-REC の値にも同じことが言えます。

トランザクションについて

以下の節では、各種のプログラミング機能がトランザクション・モードでどのように動作するかについて説明します。まず、トランザクション・モードのコーディングで従うべき基本的な通信規則について説明します。

通信規則

トランザクション・モードで実行するコードを記述する場合は、以下の基本的な通信規則に従います。

- 同じトランザクションに参加するプロセスでは、すべての要求で応答が必要です。応答を必要としない要求を含めるには、TPACALL に TPNOTRAN または TPNOREPLY を設定します。
- サービスは、TPRETURN または TPFORWAR を呼び出す前に、すべての非同期トランザクション応答を取得する必要があります。この規則には、コードをトランザクション・モードで実行するかどうかに関係なく従います。

- イニシエータは、TPCOMMIT を呼び出す前に、すべての非同期トランザクション応答 (TPNOTRAN が指定されていないもの) を取得する必要があります。
- トランザクションのパーティシパント以外からの応答を必要とする非同期呼び出しには、応答を取得する必要があります。つまり、応答が抑制されたのではなくトランザクションが抑制された TPACALL で行った要求に対する応答を取得する必要があります。
- トランザクションがタイムアウトになっていなくても、「アボートのみ」としてマークされている場合、以降の通信では TPNOTRAN を設定して、トランザクションがロールバックされた後でも通信の結果が保持されるようにします。
- トランザクションがタイムアウトになると、以下のようになります。
  - タイムアウトになった呼び出しのハンドルは無効になり、以降このハンドルを参照すると、TPEBADDESC が返されます。
  - 未処理のハンドルの TPGETRPLY または TPRECV を呼び出すと、トランザクション・タイムアウトについてのグローバル状態が返され、TP-STATUS に TPETIME が設定されます。
  - 非同期呼び出しは、TPACALL に TPNOREPLY、TPNOBLOCK、または TPNOTRAN を設定して行うことができます。
- タイムアウト以外の理由でトランザクションが一度「アボートのみ」とマークされると、TPGETRPLY の呼び出しでは、呼び出しのローカル状態を表す値が返されます。つまり、ローカル条件を反映する成功コードまたはエラー・コードのいずれかが返されます。
- 応答を取得するために TPGETRPLY で一度ハンドルを使用した場合、またはエラー条件を通知するために TPSEND または TPRECV で一度ハンドルを使用した場合、そのハンドルは無効になり、以降このハンドルを参照すると TPEBADDESC が返されます。この規則には、コードをトランザクション・モードで実行するかどうかに関係なく従います。
- トランザクションが一度アボートされると、未処理のトランザクションの呼び出しハンドル (TPNOTRAN フラグが設定されていないもの) はすべて無効になり、以降このハンドルを参照すると TPEBADDESC が返されます。

トランザクション・エラー

以下の節では、トランザクションに関連するエラーについて説明します。

致命的ではないトランザクション・エラー

トランザクション・エラーが発生すると、TP-STATUS に TPETRAN が返されます。ただし、このようなエラーの意味は、そのエラーを返す呼び出しによって異なります。次の表は、トランザクション・エラーを返す呼び出しと、考えられるエラーの原因を示しています。

表 11-3 トランザクション・エラー

呼び出し	原因
TPBEGIN	通常は、トランザクションの開始を試みたときに発生する一時的なシステム・エラーが原因で起こります。呼び出しを繰り返し行うと、問題が解決します。
TPCANCEL	トランザクションから呼び出された場合に、TPETRAN を返します。
TPRESUME	呼び出し元が、1 つ以上のリソース・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、BEA Tuxedo システムがグローバル・トランザクションを再開できません。そのような作業はすべて、グローバル・トランザクションを再開する前に完了していなければなりません。ローカル・トランザクションについての呼び出し元の状態は、変更されません。

呼び出し	原因
TPCONNECT、TPCALL、および TPACALL	トランザクションがサポートされていないサービスに対して、トランザクション・モードで呼び出しが行われました。サービスには、データベース管理システム (DBMS) にアクセスし、その結果トランザクションがサポートされるサーバ・グループに属するものがあります。そのようなグループに属さないサービスもあります。また、トランザクションがサポートされたサービスには、トランザクションがサポートされていないソフトウェアとの相互運用を必要とするものがあります。たとえば、フォームを出力するサービスの処理が、トランザクションがサポートされていないプリンタで行われる場合があります。トランザクションがサポートされていないサービスは、トランザクションのパーティシパントとして動作できない場合があります。 サービスをサーバやサーバ・グループにグループ分けする作業は、管理タスクの 1 つです。どのサービスでトランザクションがサポートされているかを確認するには、アプリケーション管理者に問い合わせてください。 トランザクション・レベルのエラーをアプリケーション・レベルで修正するには、TPSVCDEF-REF を有効にするか、またはトランザクション外でエラーが返されたサービスにアクセスします。

致命的なトランザクション・エラー

致命的なトランザクション・エラーが発生した場合、アプリケーションでは、イニシエータで TPABORT を呼び出してトランザクションを明示的にアボートしなければなりません。そのため、トランザクションにとって致命的なエラーを認識することが大切です。次の 3 つの場合、トランザクションは失敗します。

- トランザクションのイニシエータまたはパーティシパントが、次のいずれかの理由により「アボートのみ」にマークされました。
  - TPRETURN の引数の処理でエラーが発生しました。TP-STATUS に TPESVCERR が設定されます。

- TPRETURN の TP-RETURN-VAL に TPFAIL が設定されました。つまり、TP-STATUS に TPESVCFAIL が設定されました。
- 応答レコードのタイプが不明であるか、または呼び出し元で使用できないので、成功したか失敗したかを判断できません。TP-STATUS に TPEOTYPE が設定されます。

- トランザクションがタイムアウトになりました。TP-STATUS に TPETIME が設定されます。
- TPCOMMIT が、トランザクションの開始元ではなくパーティシパントによって呼び出されています。TP-STATUS に TPEPROTO が設定されます。

トランザクションにとって致命的なプロトコル・エラーが発生するのは、トランザクションの不正なパーティシパントから TPCOMMIT が呼び出された場合だけです。このエラーは、アプリケーション内で開発段階に修正できます。

イニシエータまたはパーティシパントで障害が発生した後、またはトランザクションがタイムアウトになった後で、TPCOMMIT が呼び出されると、暗黙的なアボート・エラーになります。その場合、コミットは失敗するので、トランザクションをアボートする必要があります。

通信呼び出しで TPESVCERR、TPESVCFAIL、TPEOTYPE、または TPETIME が返された場合、TPABORT を呼び出してトランザクションを明示的にアボートしなければなりません。トランザクションを明示的にアボートする前に、未処理の通信ハンドルを待つ必要はありません。ただし、これらの通信ハンドルは、呼び出しがアボートされた後は無効と見なされるので、トランザクション終了後にこれらのハンドルへのアクセスを試みると、TPEBADDESC が返されます。

TPESVCERR、TPESVCFAIL、および TPEOTYPE の場合、トランザクションがタイムアウトにならない限り、引き続き通信呼び出しを行うことができます。これらのエラーが返された場合、トランザクションは「アボートのみ」にマークされます。これ以降の処理の結果を保持するには、TPNOTRAN を設定して通信用の関数を呼び出します。このフラグを設定すると、「アボートのみ」にマークされたトランザクションで実行された処理は、トランザクションがアボートしてもロールバックされません。

トランザクション・タイムアウトが発生しても通信を続けることはできますが、次のような通信要求を行うことはできません。

- 応答を要求すること
- ブロックすること
- 呼び出し元のトランザクションに対して実行すること

したがって、非同期呼び出しを行うには、TPNOREPLY、TPNOBLOCK、または TPNOTRAN を設定する必要があります。

ヒューリスティックな判断に関するエラー

TPCOMMIT 呼び出しは、TP-COMMIT-CONTROL の設定に応じて、TPEHAZARD または TPEHEURISTIC を返します。

TP-COMMIT-CONTROL に TP-CMT-LOGGED を設定すると、2 フェーズ・コミットの第 2 フェーズの実行前にアプリケーションに制御が移ります。その場合、第 2 フェーズ中に発生したヒューリスティックな判断がアプリケーションで認識されないことがあります。

TPEHAZARD または TPEHEURISTIC は 1 フェーズ・コミットで返すことができます。ただし、これが可能なのは、トランザクションに関与しているリソース・マネージャが 1 つだけで、1 フェーズ・コミットでこのリソース・マネージャがヒューリスティックな判断を返すか、なんらかの障害の発生を示す場合です。

TP_COMMIT_CONTROL に TP_CMT_COMPLETE を設定すると、リソース・マネージャがヒューリスティックな判断を通知する場合は TPEHEURISTIC が返され、リソース・マネージャがなんらかの障害を通知する場合は TPEHAZARD が返されます。TPEHAZARD は、コミットの第 2 フェーズ (または 1 フェーズ・コミット) でパーティシパントになんらかの障害が発生し、トランザクションが正常終了したかどうかがわからない状況を示します。

トランザクション・タイムアウト

11-18 ページの「トランザクション・エラー」で説明したように、BEA Tuxedo アプリケーションでは、ブロッキング・タイムアウトとトランザクション・タイムアウトの 2 種類のタイムアウトが発生します。以下の節では、各種のプログラミング機能へのトランザクション・タイムアウトの影響について説明します。タイムアウトの詳細については、11-18 ページの「トランザクション・エラー」を参照してください。

TPCOMMIT 呼び出し

TPCOMMIT を呼び出した後でタイムアウトが発生した場合、トランザクションはどのような状態になるでしょうか。トランザクションがタイムアウトになり、そのトランザクションがアボートされたことがシステムで認識されると、システムは TP-STATUS に TPEABORT を設定して、そのような状況の発生を通知します。トランザクションの状態が不明な場合は、エラー・コードに TPETIME を設定します。

トランザクションの状態が明確ではない場合、リソース・マネージャに問い合わせる必要があります。まず、トランザクションによって行われた変更が適用されたかどうかを確認します。これにより、トランザクションがコミットされたか、またはアボートされたかを判断できます。

TPNOTRAN

トランザクション・モードのプロセスで、TPNOTRAN を設定して通信呼び出しを行うと、呼び出されたサービスは現在のトランザクションに参加できません。サービス要求の成功や失敗は、トランザクションの結果に影響しません。トランザクションは、サービスから応答が返されるのを待つ間にタイムアウトになる場合もあります。これは、そのサービスがトランザションに参加しているかどうかには関係ありません。

TPNOTRAN の使用方法の詳細については、11-23 ページの「TPRETURN および TPFORWAR 呼び出し」を参照してください。

TPRETURN および TPFORWAR 呼び出し

トランザクション・モードで実行中にプロセスを呼び出すと、TPRETURN および TPFORWAR は、トランザクションのサービス部分をそのトランザクションの完了時にコミットまたはアボートできる状態にします。同じトランザクションでサービスを何度も呼び出すことができます。システムは、トランザクションのイニシエータによって TPCOMMIT または TPABORT が呼び出されない限り、トランザクションを完全にはコミットまたはアボートしません。

サービス内で行われた通信呼び出しのすべての未処理のハンドルが取得されるまで、TPRETURN または TPFORWAR を呼び出すことはできません。TP-RETURN-VAL に TPSUCCESS を設定して、未処理のハンドルで TPRETURN を呼び出すと、プロトコル・エラーが発生し、TPGETRPLY を待機中のプロセスに TPESVCERR が返されます。そのプロセスがトランザクション・モードになっている場合、呼び出し元は「アボートのみ」にマークされます。トランザクションのイニシエータが TPCOMMIT を呼び出した場合も、トランザクションが暗黙的にアボートされます。TP-RETURN-VAL に TPFAIL を設定して、未処理のハンドルで TPRETURN を呼び出すと、TPGETRPLY を待機中のプロセスに TPESVCFAIL が返されます。トランザクションへの影響は同じです。

トランザクション・モードで実行中に TPRETURN を呼び出すと、TPRETURN で発生したプロセス・エラー、またはアプリケーションによって TP-RETURN-VAL に設定された値で示されるエラーにより、トランザクションの結果に影響することがあります。

TPFORWAR を使用すると、ある時点までは要求が正しく処理されていることを示すことができます。アプリケーション・エラーが検出されない場合、システムは TPFORWAR を呼び出します。アプリケーション・エラーが検出された場合、システムは TPFAIL を設定して TPRETURN を呼び出します。TPFORWAR を正しく呼び出さないと、システムはその呼び出しをプロセス・エラーと見なし、エラー・メッセージを要求元に返します。

tpterm( ) 関数

TPTERM 呼び出しは、アプリケーションからクライアント・コンテキストを削除します。

クライアント・コンテキストがトランザクション・モードになっている場合、呼び出しは失敗して、TP-STATUS に TPEPROTO が返されます。クライアント・コンテキストは、トランザクション・モードでアプリケーションの一部として残ります。

呼び出しが成功すると、現在の実行スレッドはアプリケーション内に存在しなくなるため、クライアント・コンテキストは、トランザクションと通信したりトランザクションに参加できなくなります。

リソース・マネージャ

ATMI 呼び出しを使ってトランザクションを定義すると、BEA Tuxedo システムによって内部呼び出しが実行され、トランザクションに参加している各リソース・マネージャにグローバル・トランザクション情報が渡されます。TPCOMMIT や TPABORT を呼び出すと、各リソース・マネージャに対して内部呼び出しが行われ、呼び出し元のグローバル・トランザクションのために行われていた作業がコミットまたはアボートされます。

グローバル・トランザクションが開始された場合 (明示的でも暗黙的でも)、アプリケーション・コードでリソース・マネージャのトランザクション呼び出しを明示的に呼び出すことはできません。このトランザクション規則に従わないと、不安定な結果が生じます。TPGETLEV 呼び出しを使用すると、リソース・マネージャのトランザクション呼び出しを呼び出す前に、グローバル・トランザクション内に既にプロセスがあるかどうかを確認できます。

リソース・マネージャによっては、トランザクションの整合性レベルなど、特定のパラメータをプログラマが設定できるものがあります。その場合、リソース・マネージャ間のインターフェイスで使用可能なオプションを指定します。そのようなオプションは、次の 2 つの方法で使用できるようになります。

- リソース・マネージャ固有の関数呼び出し。分散アプリケーションのプログラマは、これらの関数を使用してオプションを設定することができます。
- ハードコーディングされたオプション。リソース・マネージャのプロバイダで提供されるトランザクション・インターフェイスに組み込まれています。

詳細については、リソース・マネージャのマニュアルを参照してください。

オプションの設定方法はリソース・マネージャによって異なります。たとえば、BEA Tuxedo システムの SQL リソース・マネージャの場合、`set transaction` 文を使用して、BEA Tuxedo システムによって既に開始されているトランザクションに対する特定のオプション (整合性レベルとアクセス・モード) が決まります。

トランザクションのサンプル・シナリオ

以降の節では、次のトランザクションについて説明します。

- 呼び出し元と同じトランザクションでのサービス呼び出し
- AUTOTRAN が設定された別のトランザクションでのサービス呼び出し
- 新しい明示的なトランザクションを開始するサービスの呼び出し

呼び出し元と同じトランザクションでのサービス呼び出し

トランザクション・モードになっている呼び出し元が、現在のトランザクションに参加するために別のサービスを呼び出した場合、次のようになります。

- TPRETURN と TPFORWAR は、トランザクションに参加しているサービスから呼び出されると、そのトランザクションのサービス部分をイニシエータによってアボートまたはコミットできる状態にします。
- 呼び出されたプロセスの成功や失敗は、現在のトランザクションに影響します。パーティシパントで致命的なトランザクション・エラーが発生すると、現在のトランザクションは「アボートのみ」にマークされます。
- 正常終了したパーティシパントによって行われた処理が適用されるかどうかは、トランザクションの結果に依存します。つまり、トランザクションがアボートされた場合は、すべてのパーティシパントの処理は適用されません。
- 現在のトランザクションに参加するために別のサービスを呼び出すときに、TPNOREPLY を使用することはできません。

AUTOTRAN が設定された別のトランザクションでのサービス呼び出し

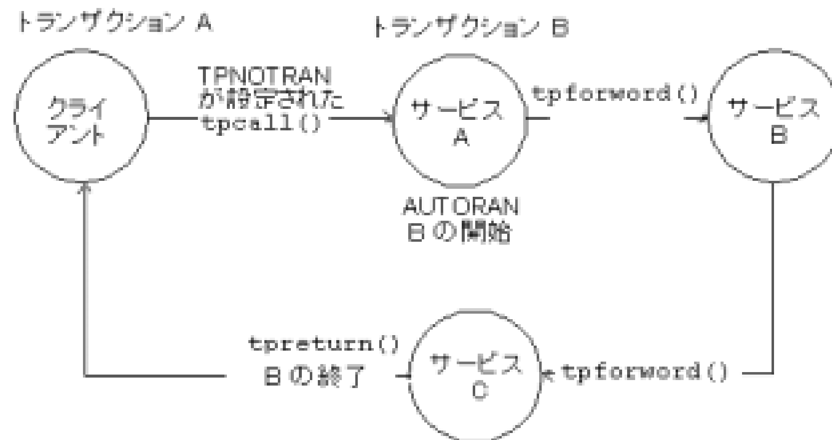
TPNOTRAN を設定して通信呼び出しを行い、呼び出されたときにトランザクションが自動的に開始するようにサービスが設定されている場合、呼び出し元プロセスと呼び出されたプロセスはどちらもトランザクション・モードになります。ただし、この 2 つは別々のトランザクションを構成します。この状況では、次の処理が行われます。

- TPRETURN は、トランザクションのイニシエータの役割を果たします。つまり、トランザクションが自動的に開始されたサービスで、トランザクションを終了します。または、TPFORWAR によって終了するサービスでトランザクションが自動的に開始された場合、転送チェインの最後のサー

ビスで発行された TPRETURN 呼び出しが、トランザクションのイニシエータの役割を果たします。つまり、トランザクションを終了します。例については、11-27 ページの「AUTOTRAN が設定された TPFORWAR と TPRETURN のトランザクションでの役割」を参照してください。

- TPRETURN はトランザクション・モードなので、トランザクションのパーティシパントの障害やトランザクション・タイムアウトなどの影響を受けます。その場合、エラー・メッセージが返されます。
- 呼び出し元のトランザクションの状態は、呼び出し元に返されるエラー・メッセージやアプリケーション障害の影響を受けません。
- 呼び出し元のトランザクションは、呼び出し元が応答を待っている間にタイムアウトになることがあります。
- 応答が必要ない場合、呼び出し元のトランザクションは、通信呼び出しによる影響を受けません。

図 11-1AUTOTRAN が設定された TPFORWAR と TPRETURN のトランザクションでの役割



新しい明示的なトランザクションを開始するサービスの呼び出し

TPNOTRAN を設定して通信呼び出しを行い、呼び出されたサービスが自動的にトランザクション・モードにならないように設定されている場合、TPBEGIN、TPCOMMIT、および TPABORT を明示的に呼び出すと、サービスで複数のトランザクションを定義できます。その結果、TPRETURN が呼び出される前にトランザクションを完了できます。

この状況では、次の処理が行われます。

- TPRETURN はトランザクションの役割は果たしません。つまり、サービス・ルーチンにトランザクションが明示的に定義されているかどうかにかかわらず、TPRETURN の役割は常に同じです。
- トランザクションの結果に関係なく、TPRETURN は TP-RETURN-VAL に任意の値を返します。
- 通常、システムはプロセス・エラー、レコード・タイプ・エラー、またはアプリケーション障害を返し、TPESVCFAIL、TPEITYPE/TPEOTYPE、および TPESVCERR の一般的な規則に従います。
- 呼び出し元のトランザクションの状態は、呼び出し元に返されるエラー・メッセージやアプリケーション障害の影響を受けません。
- 呼び出し元は、応答を待っている間にトランザクションがタイムアウトになることがあります。
- 応答が必要ない場合、呼び出し元のトランザクションは、通信呼び出しによる影響を受けません。

BEA Tuxedo システムで提供されるサブルーチン

BEA Tuxedo システムで提供されるサブルーチン TPSVRINIT および TPSVRDONE は、トランザクションで使用される場合は特定の規則に従う必要があります。

BEA Tuxedo システム・サーバは、初期化時に TPSVRINIT を呼び出します。特に、TPSVRINIT は、呼び出し元プロセスがサーバになった後、サービス要求の処理を開始する前に呼び出されます。TPSVRINIT で非同期通信を実行した場合、関数が戻る前にすべての応答が取得されなければなりません。この処理が行われなかった場合、システムは保留中の応答があっても無視して、サーバを終了します。TPSVRINIT でトランザクションを定義した場合、関数が戻る前にすべての非同期応答を取得して、トランザクションを終了しなければなりません。この処理が行われなかった場合、システムは未処理の応答が残っていてもトランザクションをアボートし、それらの応答をすべて無視します。その場合、サーバは正常に終了します。

BEA Tuxedo システム・サーバ用ルーチンは、サービス要求の処理が完了した後、ルーチンを終了する前に TPSVRDONE を呼び出します。この時点で、サーバのサービス宣言は取り消されますが、サーバ自体はアプリケーションから分離していません。TPSVRDONE で通信を開始した場合、この関数は未処理の応答をすべて取得してから戻る必要があります。この処理が行われなかった場合、システムは保留中の応答があっても無視して、サーバを終了します。TPSVRDONE 内でトランザクションを開始した場合、トランザクションはすべての応答を取得してから終了しなければなりません。この処理が行われなかった場合、システムは応答が残っていてもトランザクションをアボートし、応答を無視します。この場合もサーバは終了します。

中央イベント・ログ

中央イベント・ログには、BEA Tuxedo アプリケーションで発生する重要なイベントが記録されます。これらのイベントに関するメッセージは、アプリケーション・クライアントとサービスが USERLOG(3cbl) ルーチンを介してログに出力されます。

中央イベント・ログの分析は、アプリケーションで行う必要があります。USERLOG(3cbl) に記録するイベントに関しては、厳密なガイドラインを定義しておきます。ほとんど問題にならないようなメッセージを記録しないようにすると、アプリケーションのデバッグが簡単になります。

Windows 2000 プラットフォームの中央イベント・ログの設定の詳細については、『Windows NT での BEA Tuxedo システムの使用』を参照してください。

ログの名前

アプリケーション管理者は、コンフィギュレーション・ファイルに、各マシン上のエラー・メッセージ・ファイル名の接頭辞として使用する絶対パス名を定義します。USERLOG(3cbl) ルーチンは、月、日、年を表す *mmddyy* の形式で日付を生成し、この日付をパス名の接頭辞に付加して中央イベント・ログの完全なファイル名を構成します。毎日、新しいファイルが作成されます。そのため、中央イベント・ログに数日間にわたってメッセージが送信された場合、メッセージはそれぞれ異なるファイルに書き込まれます。

ログ・エントリの形式

ログ・エントリは、次の要素から構成されます。

- タグ。タグは次の要素から構成されます。
  - 時刻 (*hhmmss*)。
  - マシン名。たとえば、UNIX システムでは、uname(1) コマンドから返される名前が使用されます。
  - USERLOG(3cbl) を呼び出したスレッドのスレッド ID (スレッドがサポートされていないプラットフォーム上では 0)、プロセス ID、および名前。
  - USERLOG(3cbl) を呼び出したスレッドのコンテキスト ID。
- メッセージの本文。

  各メッセージの本文の前には、そのメッセージのカタログ名と番号が付きます。

たとえば、mach1 (uname コマンドから返される名前) という UNIX マシン上で、セキュリティ・プログラムが午後 4:22:14 に次のような呼び出しを行ったとします。

```
01 LOG-REC PIC X(15) VALUE "UNKNOWN USER ".
01 LOGREC-LEN PIC S9(9) VALUES IS 13.
CALL "USERLOG" USING LOG-REC LOGREC-LEN TPSTATUS-REC.
```

このログ・エントリは、次のようになります。

```
162214.mach1!security.23451:UNKNOWN USER
```

この例では、セキュリティのプロセス ID は 23451 です。

前述のメッセージが、アプリケーションではなく BEA Tuxedo システムによって生成された場合は、次のようになります。

```
162214.mach1!security.23451:COBAPI_CAT:999: UNKNOWN USER
```

この例では、メッセージのカタログ名は COBAPI_CAT、メッセージ番号は 999 です。

プロセスがトランザクション・モードのときにメッセージが中央イベント・ログに送られると、ユーザ・ログ・エントリのタグにはそのほかの要素が付加されます。これらの要素は、リテラル文字列の gtrid と、それに続く 3 桁の long 型の 16 進数で構成されます。これらの整数はグローバル・トランザクションを一意に識別するもので、グローバル・トランザクション識別子、つまり gtrid と呼ばれます。この識別子は主に管理上の目的で使用されます。また、中央イベント・ログでメッセージの前に付加されるタグの中に挿入されます。システムがトランザクション・モードで中央イベント・ログにメッセージを書き込むと、ログ・エントリは次のようになります。

```
162214.mach1!security.23451: gtrid x2 x24e1b803 x239:
UNKNOWN USER
```

イベント・ログへの書き込み

イベント・ログにメッセージを書き込むには、次の手順に従います。

- ログに書き込むエラー・メッセージをレコードに割り当て、そのレコード名を呼び出しの引数として使用します。
- メッセージのリテラル文字列を二重引用符で囲み、次の例のように USERLOG(3cbl) 呼び出しの引数として指定します。

```
01 TPSTATUS-REC.
   COPY TPSTATUS.
01 LOGMSG     PIC X(50).
01 LOGMSG-LEN PIC S9(9) COMP-5.
. . .
CALL "TPOPEN" USING TPSTSTUS-REC.
IF NOT TPOK
        MOVE "TPSVRINIT: Cannot Open Data Base" TO LOGMSG
        MOVE 43 LOGMSG-LEN
        CALL "USERLOG" USING LOGMSG
                             LOGMSG-LEN
                             TPSTATUS-REC.
. . .
```

この例では、TPOPEN(3cbl) が -1 を返した場合、メッセージが中央イベント・ログに送られます。

12 Workstation コンポーネントに対する COBOL 言語のバインディング

ここでは、次の内容について説明します。

- UNIX のバインディング
- Microsoft Windows のバインディング

Workstation プラットフォームの詳細については、『BEA Tuxedo Workstation コンポーネント』を参照してください。

UNIX のバインディング

以下に、COBOL 言語を使用して UNIX 上で BEA Tuxedo アプリケーションを開発する場合に、クライアント・プログラムを作成してビルドする方法、および正しい環境変数の設定方法について説明します。

クライアント・プログラムの作成

UNIX プラットフォーム用 COBOL クライアント・プログラムは、BEA Tuxedo 管理ドメインで COBOL クライアントを開発する場合と同じように開発できます。すべての ATMI 呼び出しを使用することができます。

クライアント・プログラムを作成する

ワークステーション・クライアント・プログラムをコンパイルしてリンクするには、`buildclient(1)` コマンドを使用します。ネイティブ・ノードで UNIX ワークステーション・クライアントをビルドする場合、`-w` オプションを指定して、ワークステーション・ライブラリを使用してクライアントがビルドされるようにします。

ネイティブ・ノードでクライアントをビルドする場合に、ネイティブ・ライブラリとワークステーション・ライブラリの両方が存在しているときは、デフォルトでネイティブ・ライブラリが使用されます。その場合、`-w` オプションを指定すると、ワークステーション・クライアントに対応するライブラリが必ず使用されます。

ワークステーション・ライブラリのみが存在する場合は、`-w` を指定する必要はありません。

次のコード例は、ネイティブ・ノード上で `buildclient` コマンドを使用する方法を示しています。

コード リスト 12-1 UNIX **プラットフォームでの** buildclient **の実行**

```
ALTCC=cobcc ALTCFLAGS="-I /APPDIR/include"
COBCPY=$TUXDIR/cobinclude
COBOPT="-C ANS85 -C ALIGN=8 -C NOIBMCOMP -C TRUNC=ANSI -C OSEXT=cbl"
export COBOPT COBCPY ALTCC ALTCFLAGS
buildclient -C -w -o empclient -f name.cbl -f "userlib1.a userlib2.a"
```

-o オプションは、出力ファイルに名前を指定する場合に使用します。-f オプションが指定された入力ファイルは、システム・ライブラリの前にリンクされます。

上記のコード例に示してあるように、TUXDIR 環境変数を使用して、buildclient コマンドがシステム・ライブラリの場所を認識できるようにします。TUXDIR は必ず指定してください。CC 環境変数には、デフォルトで cc が設定されます。ただし、ALTCC を使用して、別のコンパイラを設定することもできます。

環境変数を設定する

ワークステーション・クライアントでは、いくつかの環境変数を使用します。

次の表は、ワークステーション・クライアントがアプリケーションに参加する場合に、TPINITIALIZE で確認される環境変数です。

表 12-1UNIX プラットフォーム上の TPINITIALIZE で確認される環境変数

環境変数	**機能説明**
WSENVFILE	クライアントの環境で使用される環境変数が定義されたファイルの名前。
WSNADDR	クライアントがアプリケーションにアクセスするためのワークステーション・リスナ・プロセスのネットワーク・アドレス。ワークステーション・リスナを呼び出すには、アプリケーションのコンフィギュレーション・ファイルに指定されている値を使用してください。「0x」で始まる値は 16 進値を表す文字列と見なされ、それ以外の値は、ASCII 文字列と見なされます。
WSDEVICE	ネットワークにアクセスするために使用されるデバイスの名前。すべてのトランスポート層インターフェイスで必要なわけではありません。

環境変数	**機能説明**
WSTYPE	ワークステーションの種類。ワークステーション・クライアントが TPINITIALIZE を呼び出して、ネイティブ・サイトと符号化 / 復号化処理に関して調整する場合に、TPINITIALIZE で使用されます。WSTYPE が指定されていないと、ネイティブ・サイトでも WSTYPE が指定されていない場合でも、符号化が行われます。ネイティブ・サイトとワークステーション・クライアント・サイトの両方で明示的に同じ WSTYPE 値を指定して、符号化 / 復号化機能が必ず無効になるようにします。
WSRPLYMAX	アプリケーション応答をディスクにダンプする前にバッファに格納するために、ATMI で使用されるコア・メモリの最大サイズ。TPINITIALIZE で使用されます。デフォルトでは、システムの上限値は 256,000 バイトです。WSRPLYMAX に低い値を設定するかどうかは、使用しているマシンで利用可能なメモリ容量で判断します。応答をディスクに書き込むと、パフォーマンスが大幅に低下します。
WSFADDR	ワークステーション・クライアントがワークステーション・リスナまたはワークステーション・ハンドラに接続するのに使用するネットワーク・アドレス。この変数は、WSFRANGE 変数とともに、ワークステーション・クライアントがアウトバウンド接続を行う前にバインドしようとする TCP/IP ポートの範囲を決定します。このアドレスには、TCP/IP アドレスを指定する必要があります。
WSFRANGE	ワークステーション・クライアントのプロセスが、アウトバウンド接続を確立する前にバインドする TCP/IP ポートの範囲。範囲のベースとなるアドレスは、WSFADDR パラメータで指定します。デフォルト値は 1 です。

使用されている BEA Tuxedo システムのコンポーネントによっては、UNIX ワークステーション上の Workstation COBOL クライアントでほかの環境変数が必要になる場合もあります。

注記　MicroFocus では、共用オブジェクトとして `LIBNSL.a` が提供されています。このオブジェクトは、ワークステーション・クライアントのリンク時に `buildclient` で必要になります。MicroFocus COBOL では、UNIX 3.2 上で共用オブジェクトがサポートされていません。そのため、UNIX 3.2 版 Workstation はサポートされていません。

Microsoft Windows のバインディング

以下に、COBOL 言語を使用して Microsoft Windows プラットフォーム上で BEA Tuxedo アプリケーションを開発する場合に、クライアント・プログラムを作成してビルドする方法、ACCEPT/DISPLAY クライアントをビルドする方法、ネットワークの動作をブロックする方法、そしてネットワーク環境を復元する方法について説明します。

クライアント・プログラムの作成

プログラム固有のすべての ATMI 呼び出しを使用することができます。

クライアント・プログラムのビルド

ATMI を呼び出す COBOL のソース・コードをコンパイルするには、`LITLINK` オプションを指定して、COBOL コンパイラを使用する必要があります。ワークステーション・クライアントのオブジェクト・ファイルをリンクするには、`buildclient(1)` コマンドを使用します。コマンドの構文はわかりやすいものですが、その使い方は使用しているコンパイラによって異なります。

次のコード例は、`buildclient` コマンドの使用方法を示しています。

コード リスト 12-2 Windows **プラットフォームでの** buildclient **の実行**

```
COBCPY=C:\TUXEDO\COBINC
COBDIR=C:\COBOL\LBR;C:\COBOL\EXEDLL
PATH=C:\COBOL\EXEDLL;...
TUXDIR=C:\tuxedo
LIB=C:\NET\TOOLKIT\LIB;C:\MSVC\LIB;C:\TUXEDO\LIB;C:\COBOL\LIB
buildclient -C -o EMP.EXE -f EMP -f "/NOD/NOI/NOE/CO/SE:300" -l WLIBSOCK

For Windows NT:

buildclient -C -o EMP.EXE  -f empobj
```

次の表は、前述のコード例で使用されている buildclient コマンドのオプションを示しています。

表 12-2Windows **プラットフォーム用** buildclient **コマンドのオプション**

オプション	機能説明
-o *name*	作成している実行ファイルの名前。デフォルトは client.exe です。
-f *firstfiles*	BEA Tuxedo ライブラリの前にインクルードされる 1 つ以上のオブジェクト・ファイル。-f を使用して、コンパイラまたはリンカにオプションを渡すことができます。複数のファイル名を指定するには、-f の後に各ファイル名を入力します。その場合、各ファイル名はスペースで区切り、そのファイル・リストをまとめて二重引用符で囲みます。または、コマンド行に -f オプションを複数回指定して、複数のファイル名を指定することもできます。
-l *libfiles*	BEA Tuxedo ライブラリの後にインクルードされるライブラリ。複数のファイル名を指定するには、各ファイル名を入力します。その場合、各ファイル名はスペースで区切り、そのファイル・リストをまとめて二重引用符で囲みます。または、コマンド行に -l オプションを複数回指定して、複数のファイル名を指定することもできます。

ACCEPT/DISPLAY クライアントのビルド

次のコード例は、CSIMPAPP などの ACCEPT/DISPLAY アプリケーションに対する実行可能クライアントをビルドする方法を示しています。

コード リスト 12-3 ACCEPT/DISPLAY **クライアントのビルド**

```
a) compile the COBOL module and create a file.obj
        cobol file.cbl omf(obj) litlink;
b) use the following link statement
        link FILE+cblwinaf,,,\
        wcobatmi+cobws+wtuxws+ \
        lcobol+lcoboldw+cobw+cobfp87w+  \
        wlibsock,FILE.def  /nod/noe;
   For Windows NT the link statement is:
        cbllink -oEMP.exe EMP.obj  \
        cobws.lib ncobatmi.lib wtuxws32.lib  \
        libcmt.lib user32.lib
```