BEA WebLogic Server™

WebLogic Tuxedo Connector プログラマーズ ガイド

BEA WebLogic Server バージョン 7.0
マニュアルの日付 : 2002 年 6 月
改訂 : 2002 年 6 月 28 日

著作権

限定的権利条項

商標または登録商標

WebLogic Tuxedo Connector プログラマーズ ガイド

パート番号	マニュアルの日付	ソフトウェアのバージョン
なし	2002 年 6 月 28 日	BEA WebLogic Server バージョン 7.0

目次

このマニュアルの内容

3. WebLogic Tuxedo Connector サービス EJB の開発

4. RMI/IIOP および CORBA を相互に運用する WebLogic Tuxedo Connector の使用

5. WebLogic Tuxedo Connector JATMI トランザクション

6. WebLogic Tuxedo Connector JATMI 会話

7. WebLogic Tuxedo Connector JATMI VIEW

8. アプリケーション エラーの管理

このマニュアルの内容

このマニュアルでは、BEA WebLogic Server WebLogic Tuxedo Connector アプリケーションの開発環境について説明します。また、WebLogic Server と Tuxedo オブジェクトを相互に運用できるようにする EJB の開発方法について説明します。

このマニュアルの構成は次のとおりです。

- 第 1 章「WebLogic Tuxedo Connector プログラミングの概要」では、WebLogic Server と Tuxedo 間で相互運用するアプリケーションのコードを記述するのに使用する開発環境について説明します。
- 第 2 章「WebLogic Tuxedo Connector クライアント EJB の開発」では、クライアント EJB の作成方法について説明します。
- 第 3 章「WebLogic Tuxedo Connector サービス EJB の開発」では、サービス EJB の作成方法について説明します。
- 第 4 章「RMI/IIOP および CORBA を相互に運用する WebLogic Tuxedo Connector の使用」では、WebLogic Tuxedo Connector の CORBA アプリケーションを開発する方法について説明します。
- 第 5 章「WebLogic Tuxedo Connector JATMI トランザクション」では、グローバル トランザクションの概要とそれらをアプリケーションで定義および管理する方法について説明します。
- 第 6 章「WebLogic Tuxedo Connector JATMI 会話」では、対話の概要とそれらをアプリケーションで定義および管理する方法について説明します。
- 第 7 章「WebLogic Tuxedo Connector JATMI VIEW」では、ビュー バッファの概要とそれらをアプリケーションで定義および管理する方法について説明します。
- 第 8 章「アプリケーション エラーの管理」では、エラー条件を管理および解釈するメカニズムについて説明します。

対象読者

このマニュアルは、WebLogic Server と Tuxedo 環境で相互に運用される分散 Java アプリケーションを構築するシステム管理者およびアプリケーション開発者を対象としています。WebLogic Server、Tuxedo、CORBA、および Java プログラミングに読者が精通していることを前提として書かれています。

e-docs Web サイト

BEA 製品のドキュメントは、BEA の Web サイトで入手できます。BEA のホーム ページで [製品のドキュメント] をクリックします。

このマニュアルの印刷方法

Web ブラウザの [ファイル | 印刷] オプションを使用すると、Web ブラウザからこのマニュアルを一度に 1 章ずつ印刷できます。

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

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

関連情報

BEA の Web サイトでは、WebLogic Server および Tuxedo のすべてのドキュメントを提供しています。

Java および Java CORBA アプリケーションの詳細については、以下を参照してください。

- OMG Web サイト（http://www.omg.org/）
- Sun Microsystems, Inc. の Java サイト（http://java.sun.com/）

サポート情報

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

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

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

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

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
〔Ctrl〕+〔Tab〕	複数のキーを同時に押すことを示す。
斜体	強調または書籍のタイトルを示す。

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

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

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

1 WebLogic Tuxedo Connector プログラミングの概要

注意： WebLogic Server エンタープライズ JavaBean（EJB）の開発方法の詳細については、『WebLogic Server エンタープライズ JavaBeans プログラマーズ ガイド』を参照してください。

次の節では、WebLogic Server と Tuxedo 間で相互運用するアプリケーションのコードを記述するために使用する開発環境について説明します。

- WebLogic Tuxedo Connector アプリケーションの開発
- WebLogic Tuxedo Connector JATMI プリミティブ
- WebLogic Tuxedo Connector TypedBuffers

WebLogic Tuxedo Connector アプリケーションの開発

注意： WebLogic Tuxedo Connector JATMI の詳細については、WebLogic クラスの Javadoc を参照してください。WebLogic Tuxedo Connector クラスは、`weblogic.wtc.jatmi` および `weblogic.wtc.gwt` パッケージに含まれています。

アプリケーションのロジックを表現する Java コードに加えて、WebLogic Server と Tuxedo 間のインタフェースを提供する Java Application -to-Transaction Monitor Interface（JATMI）を使用します。これによって、既存の Tuxedo サービスを修正しなくても、クライアントおよびサーバを開発できます。

WebLogic Tuxedo Connector クライアントの開発

注意： 詳細については、2-1 ページの「WebLogic Tuxedo Connector クライアント EJB の開発」を参照してください。

クライアント プロセスはユーザの入力を受け取り、リクエストされたサービスを提供するサーバ プロセスにサービス リクエストを送信します。WebLogic Tuxedo Connector JATMI クライアント クラスは、Tuxedo で検出されたサービスにアクセスするクライアントを作成するために使用します。これらのクライアント クラスは、WebLogic Tuxedo Connector WTCServer MBean を介して提供される任意のサービスに使用できます。

WebLogic Tuxedo Connector サーバの開発

注意： 詳細については、3-1 ページの「WebLogic Tuxedo Connector サービス EJB の開発」を参照してください。

サーバは、任意の数のサービスを提供するプロセスです。サーバは頻繁にサービス リクエストのメッセージ キューをチェックし、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。WebLogic Tuxedo Connector は、EJB を使用して Tuxedo クライアントが呼び出すサービスを実装します。

Tuxedo CORBA オブジェクトを相互運用する WebLogic Tuxedo Connector の使用

注意： 詳細については、4-1 ページの「RMI/IIOP および CORBA を相互に運用する WebLogic Tuxedo Connector の使用」を参照してください。

WebLogic Tuxedo Connector は、WebLogic Server および Tuxedo CORBA オブジェクト間に双方向の相互運用性を提供します。WebLogic Tuxedo Connector は次のことを可能にします。

- Tuxedo CORBA オブジェクトが、RMI/IIOP API（着信）を使用して WebLogic Server にデプロイされた EJB を呼び出すことができます。

- オブジェクト（EJB または RMI オブジェクトなど）が、RMI/IIOP API（発信）を使用して Tuxedo にデプロイされた CORBA オブジェクトを呼び出すことができます。
- オブジェクト（EJB または RMI オブジェクトなど）が、CORBA Java API（発信）を使用して Tuxedo にデプロイされた CORBA オブジェクトを呼び出すことができます。

WebLogic Tuxedo Connector JATMI プリミティブ

JATMI は、トランザクションの開始と終了、バッファの割り当てと解放、およびクライアントとサーバ間の通信の提供に使用するプリミティブのセットです。

表 1-1 JATMI プリミティブ

名前	操作
tpacall	要求 / 応答通信で Tuxedo サービスの非同期呼び出しに使用する。
tpcall	要求 / 応答通信で Tuxedo サービスの同期呼び出しに使用する。
tpconnect	Tuxedo 対話サービスとの接続を確立するために使用する。
tpdiscon	会話を管理するプロセスによって実行された場合に対話接続を中止し、TPEV_DISCONIMM イベントを生成するために使用する。
tpdequeue	要求 / 応答通信で Tuxedo /Q からメッセージを受信するために使用する。
tpenqueue	要求 / 応答通信で Tuxedo /Q にメッセージを配置するために使用する。
tpgetrply	要求 / 応答通信で Tuxedo サービスから応答を取得するために使用する。

表 1-1 JATMI プリミティブ (続き)

名前	操作
tprecv	対話通信で Tuxedo アプリケーションからオープンな接続を介してデータを受信するために使用する。
tpsend	対話通信で Tuxedo アプリケーションにオープンな接続を介してデータを送信するために使用する。
tpterm	Tuxedo オブジェクトへの接続を終了するために使用する。

WebLogic Tuxedo Connector TypedBuffers

注意 : WebLogic Tuxedo Connector は、2 バイト文字セット（国際文字セット）をサポートしていません。これらの機能は、Tuxedo の今後のリリースに依存します。

WebLogic Tuxedo Connector は、Tuxedo の型付きバッファに対応する TypedBuffers というインタフェースを提供します。メッセージは、サーバの型付きバッファに渡されます。WebLogic Tuxedo Connector は、次のバッファ タイプを提供します。

表 1-2 TypedBuffers

バッファ タイプ	説明
TypedString	データが NULL 文字で終了する文字の配列である場合に使用されるバッファ タイプ。Tuxedo の等価タイプ : STRING。
TypedCArray	データが、NULL 可能な文字の未定義配列（バイト配列）である場合に使用されるバッファ タイプ。Tuxedo の等価タイプ : CARRAY。

表 1-2 TypedBuffers (続き)

バッファ タイプ	説明
TypedFML	データが自己定義である場合に使用されるバッファ タイプ。各データ フィールドは独自の識別子、オカレンス番号、および可能であれば長さインジケータを保持する。Tuxedo の等価タイプ : FML。
TypedFML32	TypeFML に似たバッファ タイプだが、より大きい文字フィールド、より多くのフィールド、およびより大きいバッファ全体に対して使用可能。Tuxedo の等価タイプ : FML32。
TypedXML	データが XML ベースのメッセージである場合に使用されるバッファ タイプ。Tuxedo の等価タイプ : XML（Tuxedo リリース 7.1 以降）。
TypedView	ビュー記述ファイルを用いてバッファ構造を定義するためにアプリケーションが Java 構造体を使用するとき、使用されるバッファ タイプ。Tuxedo の等価タイプ : View。
TypedView32	View に似たバッファ タイプだが、より大きい文字フィールド、より多くのフィールド、およびより大きいバッファ全体に対して使用可能。Tuxedo の等価タイプ : View32。

2 WebLogic Tuxedo Connector クライアント EJB の開発

注意： WebLogic Tuxedo Connector JATMI の詳細については、WebLogic クラスの Javadoc を参照してください。WebLogic Tuxedo Connector クラスは、weblogic.wtc.jatmi および weblogic.wtc.gwt パッケージに含まれています。

次の節では、ユーザの入力を受け取り、サーバ プロセスまたはリクエストされたサービスを提供する発信オブジェクトにサービス リクエストを送信するクライアント EJB の作成方法について説明します。

- アプリケーションとの接続および切断
- クライアントの基本操作
- クライアント EJB の例

WebLogic Tuxedo Connector JATMI クライアント クラスは、Tuxedo で検出されたサービスにアクセスするクライアントを作成するために使用します。

アプリケーションとの接続および切断

Tuxedo および WebLogic Tuxedo Connector は、それぞれ異なった方法でサービスに接続します。

アプリケーションへの接続

次の節では、Tuxedo および WebLogic Tuxedo Connector がアプリケーションに接続する方法を比較します。

- Tuxedo は、アプリケーションに接続するために `tpinit()` を使用します。

- WebLogic Tuxedo Connector は、Tuxedo サービスへのパスを作成するために必要とされる情報を提供するために WTCServer MBean を使用します。セキュリティおよびクライアント認証は、WTCServer MBean の Remote TDM コンポーネントおよび Imported Services MBean コンポーネントをコンフィグレーションすることで提供されます。この経路は、WebLogic Server 起動時に、WTCServer MBean が `config.xml` ファイル内に存在していて、サーバに対象として割り当てられている場合に、作成されます。
- WebLogic Tuxedo Connector は `TuxedoConnection` を使用して Tuxedo オブジェクトを取得したのち、`getTuxedoConnection()` を使用して Tuxedo オブジェクトへの接続を作成します。次の例は、WebLogic Server アプリケーションが WebLogic Tuxedo Connector を使用してどのように Tuxedo アプリケーションに接続するのかを示しています。

コード リスト 2-1 Tuxedo アプリケーションに接続するためのクライアントコード例

```
.
.
.
try {
     ctx = new InitialContext();
     tcf =
          (TuxedoConnectionFactory)
          ctx.lookup("tuxedo.services.TuxedoConnection");
     } catch (NamingException ne) {

// tuxedo オブジェクトを取得できなかった場合に TPENOENT を送出する
throw new TPException(TPException.TPENOENT,
     "Could not get TuxedoConnectionFactory :" + ne);
     }

myTux = tcf.getTuxedoConnection();
.
.
.
```

アプリケーションからの切断

次の節では、Tuxedo および WebLogic Tuxedo Connector がアプリケーションとの接続を切断する方法を比較します。

- Tuxedo は、アプリケーションとの接続を切断するために `tpterm()` を使用します。
- WebLogic Tuxedo Connector は、JATMI プリミティブ tpterm() を使用して Tuxedo オブジェクトへの接続を終了します。
- WebLogic Tuxedo Connector は、WTCServer MBean が新しい対象サーバに割り当てられる時、またはサーバの停止時に Tuxedo サービスへの経路を閉じます。

クライアントの基本操作

クライアント プロセスは、Java および JATMI プリミティブを使用して、次の基本アプリケーション タスクを提供します。

- Tuxedo オブジェクトの取得
- メッセージ バッファリングの実行
- メッセージの送信および受信
- Tuxedo オブジェクトへの接続の終了

クライアントは、アプリケーションとの接続を切断する前に、いくつでもサービス リクエストを送信および受信できます。

Tuxedo オブジェクトの取得

`TuxedoConnectionFactory` を使用して JNDI ツリーで「`tuxedo.services.TuxedoConnection`」をルックアップすることによって、リモート ドメインへの接続を確立し、`getTuxedoConnection()` を使用して `TuxedoConnection` オブジェクトを取得します。

メッセージ バッファリングの実行

アプリケーションと Tuxedo 間でメッセージの送信および受信を行う場合は、次の `TypedBuffers` を使用します。

表 2-1 TypedBuffers

バッファ タイプ	説明
TypedString	データが NULL 文字で終了する文字の配列である場合に使用されるバッファ タイプ。Tuxedo の等価タイプ : STRING。
TypedCArray	データが、NULL 可能な文字の未定義配列（バイト配列）である場合に使用されるバッファ タイプ。Tuxedo の等価タイプ : CARRAY。
TypedFML	データが自己定義である場合に使用されるバッファ タイプ。各データ フィールドは独自の識別子、オカレンス番号、および可能であれば長さインジケータを保持する。Tuxedo の等価タイプ : FML。
TypedFML32	TypeFML に似たバッファ タイプだが、より大きい文字フィールド、より多くのフィールド、およびより大きいバッファ全体に対して使用可能。Tuxedo の等価タイプ : FML32。
TypedXML	データが XML ベースのメッセージである場合に使用されるバッファ タイプ。Tuxedo の等価タイプ : XML（Tuxedo リリース 7.1 以降）。
Typed View	ビュー記述ファイルを用いてバッファ構造を定義するためにアプリケーションが Java 構造体を使用するとき、使用されるバッファ タイプ。Tuxedo の等価タイプ : View。
Typed View32	View に似たバッファ タイプだが、より大きい文字フィールド、より多くのフィールド、およびより大きいバッファ全体に対して使用可能。Tuxedo の等価タイプ : View32。

メッセージの送信および受信

WebLogic Tuxedo Connector クライアントは、Tuxedo サービス アプリケーションとの間で以下の 2 タイプの通信をサポートしています。

- 要求 / 応答通信
- 対話通信

要求 / 応答通信

WebLogic Tuxedo Connector クライアント アプリケーションと Tuxedo 間で要求を行い、応答メッセージを受信するには、次の JATMI プリミティブを使用します。

表 2-2 JATMI プリミティブ

名前	操作
`tpacall`	Tuxedo サービスの非同期呼び出しに使用する。
`tpcall`	Tuxedo サービスの同期呼び出しに使用する。
`tpdequeue`	Tuxedo /Q からのメッセージ受信に使用する。
`tpenqueue`	Tuxedo /Q にメッセージを配置するために使用する。
`tpgetrply`	Tuxedo サービスからの応答を検索するために使用する。

対話通信

注意： 対話通信の詳細については、6-1 ページの「WebLogic Tuxedo Connector JATMI 会話」を参照してください。

以下の 会話プリミティブは、Tuxedo サービスと通信する対話クライアントを作成する際に使用します。

表 2-3 WebLogic Tuxedo Connector の対話クライアントのプリミティブ

名前	操作
tpconnect	Tuxedo 対話サービスとの接続を確立するために使用する。
tpdiscon	会話を管理するプロセスによって実行された場合に接続を中止し、TPEV_DISCONIMM イベントを生成するために使用する。
tprecv	Tuxedo アプリケーションからオープンな接続を介してデータを受信するために使用する。
tpsend	Tuxedo アプリケーションにオープンな接続を介してデータを送信するために使用する。

Tuxedo オブジェクトへの接続の終了

オブジェクトへの接続を終了し、このオブジェクトでは今後操作を行わないようにするには、tpterm() を使用します。

クライアント EJB の例

次の Java コードは、サーバに文字列引数を送信し、サーバから応答文字列を受信する ToupperBean.java クライアント EJB の例です。

コード リスト 2-2 クライアント アプリケーションの例

```
.
.
.
public String Toupper(String toConvert)
   throws TPException, TPReplyException
```

```
{
            Context ctx;
            TuxedoConnectionFactory tcf;
            TuxedoConnection myTux;
            TypedString myData;
       Reply myRtn;
            int status;

            log("toupper called, converting " + toConvert);

            try {
              ctx = new InitialContext();
                        tcf = (TuxedoConnectionFactory) ctx.lookup(
                     "tuxedo.services.TuxedoConnection");
            }
            catch (NamingException ne) {
                        // tuxedo オブジェクトを取得できなかった場合に TPENOENT を送出す
る
              throw new TPException(TPException.TPENOENT, "Could not get
TuxedoConnectionFactory :" + ne);
       }

            myTux = tcf.getTuxedoConnection();

            myData = new TypedString(toConvert);

            log("About to call tpcall");
       try {
                        myRtn = myTux.tpcall("TOUPPER", myData, 0);
       }
            catch (TPReplyException tre) {
                        log("tpcall threw TPReplyExcption " + tre);
              throw tre;
            }
       catch (TPException te) {
              log("tpcall threw TPException " + te);
              throw te;
       }
       catch (Exception ee) {
              log("tpcall threw exception:" + ee);
              throw new TPException(TPException.TPESYSTEM, "Exception:" + ee);
       }
     log("tpcall successfull!");

       myData = (TypedString) myRtn.getReplyBuffer();

       myTux.tpterm();// Tuxedo との関連付けの終了

       return (myData.toString());
```

```
}
.
.
.
```

3 WebLogic Tuxedo Connector サービス EJB の開発

次の節では、WebLogic Tuxedo Connector サービス EJB の作成方法について説明します。

- サービス EJB の基本操作
- サービス EJB の例

サービス EJB の基本操作

サービス アプリケーションは、Java および JATMI プリミティブを使用して、次のタスクを提供します。

- サービス情報へのアクセス
- バッファ メッセージ
- リクエストされたサービスの実行

サービス情報へのアクセス

TPServiceInformation クラスを使用すると、サービスを実行するために Tuxedo クライアントによって送信されるサービス情報にアクセスできます。

表 3-1 JATMI TPServiceInformation プリミティブ

バッファ タイプ	説明
getServiceData()	Tuxedo クライアントから送信されるサービス データを返すために使用する。
getServiceFlags()	Tuxedo クライアントから送信されるサービス フラグを返すために使用する。
getServiceName()	呼び出されたサービス名を返すために使用する。

バッファ メッセージ

アプリケーションと Tuxedo 間でメッセージの送信および受信を行う場合は、次の TypedBuffers を使用します。

表 3-2 TypedBuffer s

バッファ タイプ	説明
TypedString	データが NULL 文字で終了する文字の配列である場合に使用されるバッファ タイプ。Tuxedo の等価タイプ : STRING。
TypedCArray	データが、NULL 可能な文字の未定義配列（バイト配列）である場合に使用されるバッファ タイプ。Tuxedo の等価タイプ : CARRAY。
TypedFML	データが自己定義である場合に使用されるバッファ タイプ。各データ フィールドは独自の識別子、オカレンス番号、および可能であれば長さインジケータを保持する。Tuxedo の等価タイプ : FML。

表 3-2 TypedBuffer (続き)s

バッファ タイプ	説明
TypedFML32	TypeFML に似たバッファ タイプだが、より大きい文字フィールド、より多くのフィールド、およびより大きいバッファ全体に対して使用可能。Tuxedo の等価タイプ : FML32。
TypedXML	データが XML ベースのメッセージである場合に使用されるバッファ タイプ。Tuxedo の等価タイプ : XML（Tuxedo リリース 7.1 以降）。
Typed View	ビュー記述ファイルを用いてバッファ構造を定義するためにアプリケーションが Java 構造体を使用するとき、使用されるバッファ タイプ。Tuxedo の等価タイプ : View。
Typed View32	View に似たバッファ タイプだが、より大きい文字フィールド、より多くのフィールド、およびより大きいバッファ全体に対して使用可能。Tuxedo の等価タイプ : View32。

リクエストされたサービスの実行

サービスを提供するために必要なロジックを表現するには、Java コードを使用します。

要求 / 応答通信でクライアント メッセージを返す

クライアント リクエストに応答するには、`TuxedoReply` クラスの `setReplyBuffer()` メソッドを使用します。

対話通信での tpsend および tprecv の使用

注意 : 対話通信の詳細については、6-1 ページの「WebLogic Tuxedo Connector JATMI 会話」を参照してください。

以下の JATMI プリミティブは、Tuxedo クライアントと通信する対話サーバを作成する際に使用します。

表 3-3 WebLogic Tuxedo Connector の対話クライアントのプリミティブ

名前	操作
tpconnect	Tuxedo 対話サービスとの接続を確立するために使用する。
tpdiscon	会話を管理するプロセスによって実行された場合に接続を中止し、TPEV_DISCONIMM イベントを生成するために使用する。
tprecv	Tuxedo アプリケーションからオープンな接続を介してデータを受信するために使用する。
tpsend	Tuxedo アプリケーションにオープンな接続を介してデータを送信するために使用する。

サービス EJB の例

文字列引数を受信し、その文字列をすべて小文字に変換して、変換された文字列をクライアントに返す `TolowerBean.java` サービス EJB の例を次に示します。

コード リスト 3-1 サービス EJB の例

```
.
.
.

public Reply service(TPServiceInformation mydata) throws TPException {
     TypedString data;
     String lowered;
     TypedString return_data;

     log("service tolower called");

     data = (TypedString) mydata.getServiceData();
     lowered = data.toString().toLowerCase();
```

```
        return_data = new TypedString(lowered);

        mydata.setReplyBuffer(return_data);
        return (mydata);
}
.
.
.
```

4 RMI/IIOP および CORBA を相互に運用する WebLogic Tuxedo Connector の使用

注意： CORBA を相互運用する WebLogic Tuxedo Connector をコンフィグレーションするには、いくつかの管理タスクを実行する必要があります。CORBA を相互運用する WebLogic Tuxedo Connector を管理する方法については、「CORBA アプリケーションの管理」を参照してください。

Tuxedo CORBA アプリケーションを開発する方法については、「CORBA プログラミング」ページを参照してください。

次の節では、WebLogic Server および Tuxedo CORBA オブジェクト間の相互運用性をサポートする WebLogic Tuxedo Connector を使用するために、アプリケーションを変更する方法について説明します。

- CORBA Java API を用いて WebLogic Tuxedo Connector クライアント Bean を開発する方法
- WebLogic Tuxedo Connector 用の RMI/IIOP アプリケーションを開発する方法
- FederationURL 形式の使い方
- Tuxedo CORBA アプリケーションに対するトランザクションを管理する方法

CORBA Java API を用いて WebLogic Tuxedo Connector クライアント Bean を開発する方法

WebLogic Tuxedo Connector を使うと、オブジェクト（たとえば EJB や RMI オブジェクト）は、CORBA Java API（発信）を用いて、Tuxedo にデプロイされた CORBA オブジェクトを呼び出すことができます。

オブジェクトが Tuxedo にデプロイされた CORBA オブジェクトを呼び出せるようにするには、以下の手順を行います。

- WTC ORB の使用
- オブジェクト参照の取得
- オブジェクトの呼び出し

WTC ORB の使用

CORBA Java API を使用するには、WTC ORB を使用する必要があります。Bean で WTC ORB をインスタンス化するには、次の文を使用します。

```
Prop.put("org.omg.CORBA.ORBClass",
         "weblogic.wtc.corba.ORB");
```

オブジェクト参照の取得

注意： オブジェクト参照の詳細については、4-11 ページの「FederationURL 形式の使い方」を参照してください。

WebLogic Tuxedo Connector は、リモート Tuxedo CORBA ドメインにあるオブジェクトへの参照を取得するために、CosNaming サービスを使用します。これは、オブジェクト参照 `corbaloc:tgiop` または `corbaname::tgiop` を使用して行います。次の文は、CosNaming サービスを使用して Tuxedo CORBA オブジェクトへの参照を取得します。

```
// シンプル ファクトリを取得する
     org.omg.CORBA.Object simple_fact_oref =
          orb.string_to_object("corbaname:tgiop:simpapp#simple_factory");
```

各値の説明は次のとおりです。

- `simpapp` は、Tuxedo UBB に指定された Tuxedo ドメインのドメイン ID。
- `simple_factory` は、Tuxedo CORBA CosNaming サーバにおいてオブジェクト参照がバインドされていた名前。

オブジェクトの呼び出し

タスクは、CORBA Java API を使用して Tuxedo にデプロイされた CORBA オブジェクトを呼び出すことによって実行します。

ToupperCorbaBean.java コードの例

注意： 発信 Tuxedo CORBA オブジェクト用のクライアント Bean を開発する方法の例については、WebLogic Server サンプル配布キットの `examples/wtc/corba/simpappcns` パッケージを参照してください。

次の `ToupperCorbaBean.java` コードは、WTC ORB を呼び出し、COSNaming サービスを使用してオブジェクト参照を取得する方法の例です。

コード リスト 4-1 サービス アプリケーションの例

```
.
.
.
public String Toupper(String toConvert)
```

```
throws RemoteException
{
     log("toupper called, converting " + toConvert);

     try {
        // ORB を初期化する
        String args[] = null;
         Properties Prop;

        Prop = new Properties();
        Prop.put("org.omg.CORBA.ORBClass",
                "weblogic.wtc.corba.ORB");

        ORB orb = ORB.init(args, Prop);

        // シンプル ファクトリを取得する
        org.omg.CORBA.Object simple_fact_oref =
        orb.string_to_object("corbaname:tgiop:simpapp#simple_factory");

        // シンプル ファクトリをナロー変換する
        SimpleFactory simple_factory_ref =
        SimpleFactoryHelper.narrow(simple_fact_oref);

        // シンプル ファクトリを見つける
        Simple simple = simple_factory_ref.find_simple();

       // 文字列を大文字に変換する
        org.omg.CORBA.StringHolder buf =
          new org.omg.CORBA.StringHolder(toConvert);
        simple.to_upper(buf);
        return buf.value;
     }
     catch (Exception e) {
        throw new RemoteException("Can't call TUXEDO CORBA server: " +e);
     }
}
.
.
.
```

WebLogic Tuxedo Connector 用の RMI/IIOP アプリケーションを開発する方法

注意： RMI/IIOP アプリケーションを開発する方法に関する詳細については、『WebLogic RMI over IIOP プログラマーズ ガイド』を参照してください。

WebLogic Tuxedo Connector 用の RMI/IIOP アプリケーションを開発する方法の例については、WebLogic Server 配布キットの `examples/iiop/ejb/stateless/server/tux` パッケージを参照してください。

RMI over IIOP（Internet Inter-ORB Protocol）は、Java プログラムが CORBA（Common Object Request Broker Architecture）クライアントと対話して CORBA オブジェクトを実行できるように、RMI を拡張しています。WebLogic Tuxedo Connector は次のことを可能にします。

- Tuxedo CORBA オブジェクトが、WebLogic Server（着信）にデプロイされた EJB を呼び出すことができます。
- オブジェクト（EJB または RMI オブジェクトなど）が、Tuxedo（発信）にデプロイされた CORBA オブジェクトを呼び出すことができます。

次の節では、Tuxedo CORBA アプリケーションと対話する WebLogic Tuxedo Connector を使用するために、RMI/IIOP アプリケーションを修正する方法に関する情報を提供します。

- WebLogic Tuxedo Connector を使用するために着信 RMI/IIOP アプリケーションを変更する方法
- WebLogic Tuxedo Connector を使用するために発信 RMI/IIOP アプリケーションを開発する方法

WebLogic Tuxedo Connector を使用するために着信 RMI/IIOP アプリケーションを変更する方法

クライアントは、COSNaming Service にバインドされている WebLogic Server のネーム サービスの正しいバインド名を COSNaming Service に渡す必要があります。

次のコードは、ネーミング コンテキストを取得する例です。「WLS」は、「CORBA アプリケーションの管理」で説明されている、cnsbind コマンドで指定されたバインド名です。

コード リスト 4-2 ネーミング コンテキストを取得するコードの例

```
.
.
.
// ネーミング コンテキストを取得
     TP::userlog("Narrowing to a naming context");
     CosNaming::NamingContext_var context =
          CosNaming::NamingContext::_narrow(o);
     CosNaming::Name name;
     name.length(1);
     name[0].id = CORBA::string_dup("WLS");
     name[0].kind = CORBA::string_dup("");
.
.
.
```

WebLogic Tuxedo Connector を使用するために発信 RMI/IIOP アプリケーションを開発する方法

EJB は、リモート Tuxedo CORBA オブジェクトにアクセスする目的で使用される初期コンテキストを取得するために FederationURL を使用する必要があります。WebLogic Tuxedo Connector を使用するために発信 RMI/IIOP アプリケーションを修正する方法を次の節で説明します。

- FederationURL を EJB に渡すために ejb-jar.xml ファイルを修正する方法

- オブジェクトにアクセスするための FederationURL を使用するために EJB を変更する方法

FederationURL を EJB に渡すために ejb-jar.xml ファイルを修正する方法

次のコードは、FederationURL 形式を EJB に実行時に渡すために `ejb-jar.xml` ファイルをコンフィグレーションする方法の例です。

コード リスト 4-3 FederationURL を EJB に渡す ejb-jar.xml ファイルの例

```
<?xml version="1.0"?>

<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
1.1//EN' 'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>

<ejb-jar>
     <small-icon>images/green-cube.gif</small-icon>
     <enterprise-beans>
     <session>
          <small-icon>images/orange-cube.gif</small-icon>
          <ejb-name>IIOPStatelessSession</ejb-name>
          <home>examples.iiop.ejb.stateless.TraderHome</home>
          <remote>examples.iiop.ejb.stateless.Trader</remote>
          <ejb-class>examples.iiop.ejb.stateless.TraderBean</ejb-class>
          <session-type>Stateless</session-type>
        <transaction-type>Container</transaction-type>
          <env-entry>
               <env-entry-name>foreignOrb</env-entry-name>
               <env-entry-type>java.lang.String </env-entry-type>
               <env-entry-value>corbaloc:tgiop:simpapp</env-entry-value>
          </env-entry>
          <env-entry>
               <env-entry-name>WEBL</env-entry-name>
               <env-entry-type>java.lang.Double </env-entry-type>
               <env-entry-value>10.0</env-entry-value>
          </env-entry>
          <env-entry>
               <env-entry-name>INTL</env-entry-name>
               <env-entry-type>java.lang.Double </env-entry-type>
               <env-entry-value>15.0</env-entry-value>
          </env-entry>
          <env-entry>
```

```
                <env-entry-name>tradeLimit</env-entry-name>
                <env-entry-type>java.lang.Integer </env-entry-type>
                <env-entry-value>500</env-entry-value>
           </env-entry>
     </session>
     </enterprise-beans>
     <assembly-descriptor>
          <container-transaction>
               <method>
                    <ejb-name>IIOPStatelessSession</ejb-name>
                    <method-intf>Remote</method-intf>
                    <method-name>*</method-method-name>
               </method>
          <trans-attribute>NotSupported</trans-attribute>
          </container-transaction>
     </assembly-descriptor>
</ejb-jar>
```

実行時に FederationURL を EJB に渡すには、アプリケーションの `ejb-jar.xml` ファイルにその EJB 用の `env-entry` を追加します。次の `env-entry` 下位要素を割り当てる必要があります。

- env-entry-name の割り当て
- env-entry-type の割り当て
- env-entry-value の割り当て

env-entry-name の割り当て

`env-entry-name` 要素は、`env-entry-value` 要素内の値を EJB に渡すために使われる変数の名前を指定するために使用されます。コード リスト 4-3 で示したコード例では、`env-entry-name` を `foreignOrb` と指定しています。

env-entry-type の割り当て

`env-entry-type` 要素は、EJB に渡される `env-entry-value` 要素のデータ型（たとえば String、Integer、Double）を指定するために使用されます。コード リスト 4-3 で示したコード例では、`foreignOrb` 変数が `String` データを EJB に渡しています。

env-entry-value の割り当て

env-entry-value 要素は、EJB に渡されるデータを指定するために使用されます。コード リスト 4-3 で示したコード例では、foreignOrb 変数が次の FederationURL 形式で EJB に渡すということを指定しています。

```
corbaloc:tgiop:simpapp
```

ここで、simpapp とは、Tuxedo UBB で指定されている Tuxedo リモート サービスの DOMAINID です。

オブジェクトにアクセスするための FederationURL を使用するために EJB を変更する方法

この節では、リモート Tuxedo CORBA オブジェクトにアクセスするために使用される InitialContext を取得するための FederationURL を使用する方法に関する情報を提供します。

次のコードは InitialContext を取得するための FederationURL の使い方の例です。

コード リスト 4-4 InitialContext を取得するための TraderBean.java コードの例

```
.
.
.
public void createRemote() throws CreateException {
     log("createRemote() called");

     try {
          InitialContext ic = new InitialContext();

     // リモート CORBA ドメインで EJB に似た CORBA サービスをルックアップ
する
        Hashtable env = new Hashtable();
          env.put(Context.PROVIDER_URL, (String)
             ic.lookup("java:/comp/env/foreignOrb")
             + "/NameService");

          InitialContext cos = new InitialContext(env);
          TraderHome thome =
             (TraderHome)PortableRemoteObject.narrow(
             cos.lookup("TraderHome_iiop"),TraderHome.class);
             remoteTrader = thome.create();
}
```

```
      catch (NamingException ne) {
      throw new CreateException("Failed to find value "+ne);
}
      catch (RemoteException re) {
      throw new CreateException("Error creating remote ejb "+re);
}
}
.
.
.
```

リモート Tuxedo CORBA オブジェクトの InitinalContext を取得する FederationURL を使用するための手順を以下に示します。

1. `ejb-jar.xml` ファイルで定義されている FederationURL 形式を取り出します。

   例：

   `"ic.lookup("java:/comp/env/foreignOrb")`

   コード リスト 4-3 で示したコード例では、`foreignOrb` 変数が次の FederationURL 形式で EJB に渡すということを指定しています。

   `corbaloc:tgiop:simpapp`

2. 「`/NameService`」という FederationURL 形式を、先の FederationURL に連結します。

   例：

   `"ic.lookup("java:/comp/env/foreignOrb") + "/NameService"`

   連結結果の FederationURL は次のとおりです。

   `corbaloc:tgiop:simpapp/NameService`

3. InitialContext を取得します。

   例：

   ```
   env.put(Context.PROVIDER_URL, (String)
        ic.lookup("java:/comp/env/foreignOrb") + "/NameService");
   InitialContext cos = new InitialContext(env);
   ```

   実行結果は Tuxedo CORBA オブジェクトの InitialContext です。

FederationURL 形式の使い方

この節では、以下の FederationURL 形式の構文に関する情報を提供します。

- CORBA URL 構文は CORBA 仕様で記述されます。詳細については、http://www.omg.org/ にある OMG の Web サイトを参照してください。
- corbaloc:tgiop フォームは、BEA tgiop プロトコル専用です。

corbaloc URL フォーマットの使用

この節では、corbaloc URL フォーマットの構文を示します。

```
<corbaloc> = "corbaloc:tgiop":[<version>] <domain>["/"<key_string>]

<version> = <major> "." <minor> "@" | empty_string

<domain> = TUXEDO CORBA domain name

<major> = number

<minor> = number

<key_string> = <string> | empty_string
```

corbaloc:tgiop の例

この節では、corbaloc:tgiop の使用例を示します。

```
orb.string_to_object("corbaloc:tgiop:simpapp/NameService");

orb.string_to_object("corbaloc:tgiop:simpapp/FactoryFinder");

orb.string_to_object("corbaloc:tgiop:simpapp/InterfaceRepository");

orb.string_to_object("corbaloc:tgiop:simpapp/Tobj_SimpleEventsService");

orb.string_to_object("corbaloc:tgiop:simpapp/NotificationService");

orb.string_to_object("corbaloc:tgiop:1.1@simpapp/NotificationService);
```

-ORBInitRef の使用例

orb.init および resolve_initial_reference に -ORBInitRef オプションを使用することもできます。

次の -ORBInitRef 定義を指定します。

```
-ORBInitRef FactoryFinder=corbaloc:tgiop:simp/FactoryFinder
-ORBInitRef InterfaceRepository=corbaloc:tgiop:simp/InterfaceRepository
-ORBInitRef Tobj_SimpleEventService=corbaloc:tgiop:simp/Tobj_SimpleEventsService
-ORBInitRef NotificationService=corbaloc:tgiop:simp/NotificationService
```

次に、

```
orb.resolve_initial_references("NameService");
orb.resolve_initial_references("FactoryFinder");
orb.resolve_initial_references("InterfaceRepository");
orb.resolve_initial_references("Tobj_SimpleEventService");
orb.resolve_initial_references("NotificationService");
```

-ORBDefaultInitRef の使用例

-ORBDefaultInitRef および resolve_initial_reference を使用できます。

次の -ORBDefaultInitRef 定義を指定します。

```
-ORBDefaultInitRef corbaloc:tgiop:simpapp
```

次に、

```
orb.resolve_initial_references("NameService");
```

corbaname URL フォーマットの使用

corbaloc フォーマットの代わりに corbaname フォーマットを使用することもできます。

-ORBInitRef の使用例

次の -ORBInitRef 定義を指定します。

```
-ORBInitRef NameService=corbaloc:tgiop:simpapp/NameService
```

次に、

```
orb.string_to_object("corbaname:rir:#simple_factory");
orb.string_to_object("corbaname:tgiop:simpapp#simple_factory");
orb.string_to_object("corbaname:tgiop:1.1@simpapp#simple_factory");
orb.string_to_object("corbaname:tgiop:simpapp#simple/simple_factory");
```

Tuxedo CORBA アプリケーションに対するトランザクションを管理する方法

注意： Tuxedo CORBA アプリケーションでトランザクションを管理する方法の詳細については、「BEA Tuxedo CORBA アプリケーションのトランザクションの概要」を参照してください。

WebLogic Tuxedo Connector は、Java Transaction API（JTA）を使って、Tuxedo CORBA アプリケーションでのトランザクションを管理します。詳細については、以下を参照してください。

- 『WebLogic JTA プログラマーズ ガイド』
- 「トランザクション管理」

5 WebLogic Tuxedo Connector JATMI トランザクション

次の節では、グローバル トランザクションの概要とそれらをアプリケーションで定義および管理する方法について説明します。

- グローバル トランザクション
- JTA Transaction API
- トランザクションの定義
- WebLogic Tuxedo Connector トランザクションのルール
- トランザクション コードの例

グローバル トランザクション

グローバル トランザクションは、2 つ以上のリソース マネージャのある作業、および 1 つの論理単位として扱われる 2 つ以上の物理サイトにまたがる作業を実行するトランザクションです。グローバル トランザクションは常に、次の 4 つのプロパティによって分類された特定のシーケンスの操作として扱われます。

- 原子性 : すべての部分が正常に実行されるか、またはまったく影響しないかのいずれか。
- 一貫性 : 操作は、リソースを 1 つの一貫性のある状態から別の一貫性のある状態へと正確に変換するよう実行される。
- 隔離性 : 同じトランザクションの他のプロセスがそのデータにアクセス可能であっても、中間結果は他のトランザクションにアクセスできない。
- 永続性 : 完了したシーケンスのすべての結果は、いかなる障害によっても変更できない。

JTA Transaction API

注意： 詳細については、「JTA API」を参照してください。

WebLogic Tuxedo Connector は、トランザクションの管理に Java Transaction API（JTA）を使用します。

JTA インタフェースのタイプ

JTA には、3 つのタイプのトランザクション インタフェースがあります。

- Transaction
- TransactionManager
- UserTransaction

Transaction

`Transaction` インタフェースは、ターゲット Transaction オブジェクトのトランザクションに対して実行される操作を許可します。トランザクション オブジェクトは、作成された各グローバル トランザクションに対応して作成されます。`Transaction` インタフェースは、リソースの取得、登録の同期、およびトランザクション完了操作やステータス クエリ操作を実行するために使用します。

TransactionManager

`TransactionManager` インタフェースによって、アプリケーション サーバはアプリケーションの代わりにトランザクション境界区分のトランザクション マネージャと通信できるようになります。`TransactionManager` インタフェースは、コンテナ管理の EJB コンポーネントの代わりにトランザクション マネージャと通信するために使用します。

UserTransaction

UserTransaction インタフェースは、TransactionManager インタフェースのサブセットです。UserTransaction インタフェースは、Transaction オブジェクトへのアクセスを制限する必要があるときに使用します。

JTA トランザクション プリミティブ

Tuxedo トランザクション プリミティブの機能を、等価の JTA トランザクション プリミティブにマップした表を次に示します。

表 5-1 Tuxedo トランザクション プリミティブと JTA トランザクション プリミティブのマッピング

Tuxedo	Tuxedo 機能	同等の JTA 機能
tpabort	トランザクションの終了に使用。	setRollbackOnly
tpcommit	トランザクションの完了に使用。	commit
tpgetlev	サービス ルーチンがトランザクション モードであるかどうかの判断に使用。	getStatus
tpbegin	トランザクションの開始に使用。	setTransactionTimeout begin

トランザクションの定義

トランザクションは、クライアントまたはサーバ プロセスのいずれかで定義できます。トランザクションは、3 つの部分で構成されます。3 つの部分とは、開始ポイント、トランザクション モードのプログラム文、および終了ポイントです。

トランザクションを明示的に定義するには、begin() メソッドを呼び出します。呼び出しを行う同じプロセスであるイニシエータは、commit() または setRollbackOnly() を呼び出すことによって終了を行うプロセスにもなる必要があります。トランザクション区切り記号間で呼び出されるサービス サブルーチンは、現在のトランザクションの一部となります。

トランザクションの開始

注意： setTransactionTimeout() をあまり大きい値に設定すると、システム検出やエラーのレポートが遅延します。タイムアウト値は、適切な時間内でサービス リクエストへの応答を行ったり、ネットワーク障害などの問題が発生したトランザクションを終了したりするために使用します。プロダクション環境では、システム ロードおよびデータベース競合によって起こると予想される遅延に対応するよう、タイムアウト値を調整します。

トランザクションは、begin() の呼び出しによって開始されます。タイムアウト値を指定するには、begin() 文の前に setTransactionTimeout(int seconds) 文を指定します。

Tuxedo にトランザクションを伝播するには、次の操作を実行する必要があります。

- JNDI で TuxedoConnectionFactory オブジェクトをルックアップします。
- getTuxedoConnection() を使用して TuxedoConnection オブジェクトを取得します。

TPNOTRAN の使用

トランザクション区切り記号内で呼び出されるサービス ルーチンは、現在のトランザクションの一部となります。ただし、TPNOTRAN に設定された flags パラメータが tpcall() または tpacall() に含まれている場合、呼び出されたサービスによって実行される操作は、そのトランザクションの一部にはなりません。結果として、呼び出されたプロセスによって実行されたサービスは、現在のトランザクションの結果からの影響を受けません。

トランザクションの終了

トランザクションは、commit() または setRollbackOnly() の呼び出しによって終了されます。commit() が正常に復帰した場合、現在のトランザクションの結果としてのリソースに対する変更は、すべて永久なものとなります。setRollbackOnly() は異常な状況を示し、呼び出し記述子を元の状態にロールバックするために使用します。

commit() が正常に実行されるには、次の 2 つの条件が満たされている必要があります。

- 呼び出しプロセスは、begin() でトランザクションを開始した同じプロセスでなければならない。
- 呼び出しプロセスに、未処理のトランザクション応答が存在してはならない。

いずれかの条件が真でない場合、この呼び出しは失敗し、例外が送出されます。

WebLogic Tuxedo Connector トランザクションのルール

トランザクション モード中は、トランザクションを正常に完了させるために特定のルールに従う必要があります。トランザクション モード中に注意すべき基本的なルールは、次のとおりです。

- begin() でトランザクションを開始した「後」に、TuxedoConnection オブジェクトを使用して Tuxedo にトランザクションを伝播する必要があります。
- tpterm() はオブジェクトへの接続を終了し、このオブジェクトでは今後操作を行わないようにします。
- 同じトランザクションの参加コンポーネントであるプロセスは、それらのリクエストへの応答を要求する必要があります。
- 応答を必要としないリクエストは、tpacall() の flags パラメータが TPNOREPLY に設定されている場合のみ実行できます。

- サービスは、commit() 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。
- イニシエータは、begin() 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。
- 取り出す必要のある非同期応答には、トランザクションの非参加コンポーネントから期待される応答（つまり応答ではなくトランザクションを抑制する tpacall() で行われたリクエストに対して期待される応答）が含まれます。
- トランザクションがタイムアウトされず、「アボートのみ」にマーク付けされている場合は、通信の結果として実行された作業がトランザクションのロールバック後に永久の結果を保持するよう、今後の通信を TPNOTRAN フラグ セットで実行する必要があります。
- トランザクションがタイムアウトされた場合は次のとおりです。
  - タイムアウトされた呼び出しの記述子は無効となり、それへの今後の参照は TPEBADDESC を返します。
  - 未処理の記述子の tpgetrply() または tprecv() の以後の呼び出しは、tperrono を TPETIME に設定することによってグローバルな状態のトランザクション タイムアウトを返します。
  - 非同期呼び出しは、TPNOREPLY | TPNOBLOCK | TPNOTRAN に設定された tpacall() の *flags* パラメータによって実行されます。
- タイムアウト以外の理由でトランザクションが「アボートのみ」にマーク付けされると、tpgetrply() の呼び出しは、その呼び出しのローカルな状態を示すものはすべて返します。つまり、ローカルな状態を示す成功コードまたはエラー コードのいずれかを返すことができます。
- tpgetrply() で応答を取得するために使用された記述子は無効になり、それ以後の参照では TPEBADDESC が返されます。
- tpsend() または tprecv() でエラー状態を報告するために使用された記述子は無効になり、それ以後の参照では TPEV_DISCONIMM が返されます。
- トランザクションがアボートされると、未処理のすべてのトランザクション呼び出し記述子（TPNOTRAN フラグなしで実行）が停滞状態となり、それらに対する今後の参照は TPEBADDESC を返します。

トランザクション コードの例

次に、トランザクションのコード例を示します。

コード リスト 5-1 トランザクション コードの例

```
public class TransactionSampleBean implements SessionBean {

.....

public int transaction_sample () {

     int ret = 0;
     try {
          javax.naming.Context myContext = new InitialContext();
          TransactionManager tm = (javax.transaction.TransactionManager)
          myContext.lookup("javax.transaction.TransactionManager");

// トランザクションを開始する
          tm.begin ();

          TuxedoConnectionFactory tuxConFactory = (TuxedoConnectionFactory)
          ctxt.lookup("tuxedo.services.TuxedoConnection");

// このトランザクションの一部になるローカルの
// JDBC/XA データベース操作をここで実行できる
.....

// 注意 1: Tuxedo の呼び出しをトランザクションの一部
// にする必要がある場合は、Tuxedo 接続をトランザクション
// の開始後にのみ取得する

// 注意 2: トランザクションの開始前に Tuxedo
// 接続を取得すると、その Tuxedo 接続からの
// すべての呼び出しがトランザクションのスコープ外
// になる

          TuxedoConnection myTux = tuxConFactory.getTuxedoConnection();

// tpcall を実行する。この tpcall はトランザクションの一部
          TypedString depositData = new TypedString("somecharacters,5000.00");
```

```
          Reply depositReply = myTux.tpcall("DEPOSIT", depositData, 0);

// TPNOTRAN フラグを設定することによって、トランザクション
// の一部ではない tpcall を実行することもできる（たとえば
// 試行されたすべての操作のロギングなど）
          TypedString logData =
          new TypedString("DEPOSIT:somecharacters,5000.00");

          Reply logReply = myTux.tpcall("LOGTRAN", logData,
          ApplicationToMonitorInterface.TPNOTRAN);

// Tuxedo 接続が用済みになり、tpterm を実行する
          myTux.tpterm ();

// Transaction をコミットする
          tm.commit ();

// 注意： このトランザクションで使用されている TuxedoConnection
// オブジェクトは、TPNOTRAN フラグが設定されている場合のみ
// トランザクション後も使用できる
}
          catch (NamingException ne) {
          System.out.println ("ERROR:Naming Exception looking up JNDI:" + ne);
          ret = -1;
}
          catch (RollbackException re) {
          System.out.println("ERROR:TRANSACTION ROLLED BACK: " + re);
          ret = 0;
}
          catch (TPException te) {
          System.out.println("ERROR:tpcall failed:TpException: " + te);
          ret = -1;
}
          catch (Exception e) {
          log ("ERROR:Exception:" + e);
          ret = -1;
}

          return ret;
}
```

6 WebLogic Tuxedo Connector JATMI 会話

注意： BEA Tuxedo の対話通信の詳細については、「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。

以下の節では、会話の概要とそれらをアプリケーションで定義および管理する方法について説明します。

- WebLogic Tuxedo Connector の対話通信の概要
- WebLogic Tuxedo Connector の会話の特性
- WebLogic Tuxedo Connector JATMI 会話プリミティブ
- WebLogic Tuxedo Connector の対話クライアントおよび対話サーバの作成
- メッセージの送受信
- 会話の終了
- 無秩序な切断の実行
- 対話通信のイベントについて
- WebLogic Tuxedo Connector の会話ガイドライン

WebLogic Tuxedo Connector の対話通信の概要

WebLogic Tuxedo Connector は、BEA Tuxedo の会話を WebLogic Server アプリケーションと Tuxedo アプリケーションの間でメッセージを交換する手段としてサポートしています。この形態の通信では、仮想接続がクライアントとサーバの間で維持され、各サイドで会話の状態に関する情報が維持されます。接続を開

き、会話を開始するプロセスが、会話の開始側です。接続の制御権を持つプロセスが開始プロセスで、制御権のないプロセスが従属プロセスです。接続は、それを終了するイベントが発生するまでアクティブなままです。

対話通信では、開始プロセスと従属プロセスの間で Half-Duplex 接続が確立されます。接続の制御権は、開始プロセスと従属プロセスの間で渡されます。制御権を持つプロセスがメッセージを送信でき（開始プロセス）、制御権のないプロセスはメッセージの受信のみを行えます（従属プロセス）。

WebLogic Tuxedo Connector の会話の特性

WebLogic Tuxedo Connector JATMI 会話には以下の特性があります。

- データが `TypedBuffers` を使用して渡されます。データのタイプおよびサブタイプは、サービスで認識されるタイプおよびサブタイプと一致していなければなりません。
- 対話クライアントと対話サーバの論理接続は、それが終了されるまでアクティブなままです。
- 対話クライアントと対話サーバの接続では、任意数のメッセージを転送できます。
- WebLogic Tuxedo Connector の対話クライアントは、`tpcall` または `tpacall` ではなく `tpconnect` を使用してサービスの要求を開始します。
- WebLogic Tuxedo Connector の対話クライアントと対話サーバは、JATMI プリミティブの `tpsend` を使用してデータを送信し、`tprecv` を使用してデータを受信します。
- 対話クライアントは、対話サーバにサービス要求を送信するだけです。
- 対話サーバは、`tpforward` を呼び出すことが禁止されています。

WebLogic Tuxedo Connector JATMI 会話プリミティブ

以下の WebLogic Tuxedo Connector プリミティブは、WebLogic Server と Tuxedo の間で通信する対話クライアントおよび対話サーバを作成する際に使用します。

表 6-1 WebLogic Tuxedo Connector の対話クライアントのプリミティブ

名前	操作
tpconnect	Tuxedo 対話サービスとの接続を確立するために使用する。
tpdiscon	接続を中止し、TPEV_DISCONIMM イベントを生成するために使用する。
tprecv	Tuxedo アプリケーションからオープンな接続を介してデータを受信するために使用する。
tpsend	Tuxedo アプリケーションにオープンな接続を介してデータを送信するために使用する。

WebLogic Tuxedo Connector の対話クライアントおよび対話サーバの作成

以降の節では、対話クライアントおよび対話サーバの作成方法を説明します。

対話クライアントの作成

WebLogic Tuxedo Connector の対話クライアントを作成するには、2-1 ページの「WebLogic Tuxedo Connector クライアント EJB の開発」で説明されている手順を行います。次の節では、tpconnect を使用して接続を開き、会話を開始する方法を説明します。

Tuxedo 対話サービスとの接続の確立

WebLogic Tuxedo Connector の対話クライアントは、Tuxedo 対話サービスとの接続を確立する必要があります。JATMI プリミティブ tpconnect を使用すると、接続を開いて対話を開始できます。呼び出しが成功すると、会話でデータを送受信するために使用できるオブジェクトが返されます。

次の表は、tpconnect のパラメータを示しています。

表 6-2 WebLogic Tuxedo Connector JATMI tpconnect のパラメータ

パラメータ	説明
svc	対話サービス名の文字ポインタ。*svc* を指定しないと、呼び出しは失敗し、TPException が TPEV_DISCONIMM に設定される。
data	データ バッファのポインタ。接続を確立するときに、*data* パラメータをバッファを指すように設定することでデータを同時に送信できる。バッファの type および subtype は、呼び出されるサービスによって認識されなければならない。*data* の値を NULL に設定すると、データが送信されないことを指定できる。

表 6-2 WebLogic Tuxedo Connector JATMI tpconnect のパラメータ

パラメータ	説明
flags	アプリケーションの必要に応じて、フラグを単独で、または組み合わせて使用する。有効なフラグは、以下のとおりである。 TPSENDONLY: 開始側が制御権を保持することを指定する。呼び出されるサービスは従属プロセスになり、データの受信だけが可能である。TPRECVONLY と併用することはできない。 TPRECVONLY: 呼び出されたサービスに制御権を渡すことを指定する。開始側は従属プロセスになり、データの受信だけが可能である。TPSENDONLY と併用することはできない。 TPNOTRAN: *svc* が呼び出されて開始側がトランザクション モードのときは、*svc* は開始側のトランザクションの一部ではないものとすることを指定する。呼び出しは、トランザクション タイムアウトの影響を受け続ける。*svc* が失敗しても、開始側のトランザクションは影響を受けない。 TPNOBLOCK: ブロッキング条件が存在する場合は要求を送信しないことを指定する。TPNOBLOCK を指定しないと、条件が解消されるまで、トランザクション タイムアウトが発生するまで、またはブロッキング タイムアウトが発生するまで、開始側はブロックされる。 TPNOTIME: 開始側は無期限にブロックされ、ブロッキング タイムアウトの影響を受けないことを指定する。開始側がトランザクション モードの場合、呼び出しはトランザクション タイムアウトの影響を受ける。

TuxedoConversationBean.java のコード例

次に、tpconnect を使用して会話を開始するコードの例を示します。

コード リスト 6-1 会話コードの例

```
.
.
.
Context ctx;
```

```
Conversation myConv;
TuxedoConnection myTux;
TuxedoConnectionFactory tcf;
.
.
.
ctx = new InitialContext();
tcf = (TuxedoConnectionFactory) ctx.lookup ("tuxedo.services.TuxedoConnection");
myTux = tcf.getTuxedoConnection();
flags =ApplicationToMonitorInterface.TPSENDONLY;
myConv = myTux.tpconnect("CONNECT_SVC",null,flags);
.
.
.
```

WebLogic Tuxedo Connector の対話サーバの作成

WebLogic Tuxedo Connector の対話サーバを作成するには、3-1 ページの「WebLogic Tuxedo Connector サービス EJB の開発」で説明されている手順を行います。

メッセージの送受信

対話接続が WebLogic Server アプリケーションと Tuxedo アプリケーションの間で確立されたら、開始プロセス（メッセージを送信）と従属プロセス（メッセージを受信）の間の通信が送信および受信の呼び出しを使用して遂行されます。以下の節では、WebLogic Tuxedo Connector アプリケーションが JATMI プリミティブ `tpsend` および `tprecv` をどのように使用するのかを説明します。

- メッセージの送信
- メッセージの受信

メッセージの送信

JATMI プリミティブ tpsend を使用すると、Tuxedo アプリケーションにメッセージを送信できます。

次の表は、tpsend のパラメータを示しています。

表 6-3 WebLogic Tuxedo Connector JATMI tpsend のパラメータ

パラメータ	説明
data	この会話で送信されるデータが格納されているバッファへのポインタ。
flags	フラグは以下のいずれかを指定できる。 TPRECVONLY: 開始プロセスのデータが送信された後に開始プロセスが接続の制御権を放棄することを指定する。開始プロセスは従属プロセスになり、データを受信することだけができる。 TPNOBLOCK: ブロッキング条件が存在する場合に要求が送信されないことを指定する。TPNOBLOCK を指定しないと、条件が解消されるまで、トランザクション タイムアウトが発生するまで、またはブロッキング タイムアウトが発生するまで、開始側はブロックされる。 TPNOTIME: 開始プロセスが無限にブロックされることを拒否せず、ブロッキング タイムアウトの影響を受けないことを指定する。呼び出しは、トランザクション タイムアウトの影響を受ける。

メッセージの受信

JATMI プリミティブ tprecv を使用すると、Tuxedo アプリケーションからメッセージを受信できます。

次の表は、tprecv のパラメータを示しています。

表 6-4 WebLogic Tuxedo Connector JATMI tprecv のパラメータ

パラメータ	説明
flags	フラグは以下のいずれかを指定できる。 TPNOBLOCK: tprecv が応答の到着を待たないことを指定する。応答があった場合は、tprecv は応答を取得して復帰する。このフラグを指定しない場合は、応答がないと、tprecv は、応答、トランザクション タイムアウト、またはブロッキング タイムアウトのいずれかが発生するのを待つ。 TPNOTIME: tprecv が応答を無限に待つことを指定する。tprecv は、ブロッキング タイムアウトの影響は受けないが、トランザクション タイムアウトの影響は受ける。 フラグの値を 0 にすると、開始プロセスは、条件が解消されるまで、またはタイムアウトが発生するまで、ブロックされる。

会話の終了

WebLogic Server と Tuxedo の会話は、サーバ プロセスがそのタスクを正常に完了したときに終了します。以下の節では、会話の終了方法について説明します。

- Tuxedo アプリケーションが開始した会話
- WebLogic Tuxedo Connector アプリケーションが開始した会話
- 階層的な会話の終了

Tuxedo アプリケーションが開始した会話

WebLogic Server 対話サーバによる return の正常な呼び出しで会話が終了します。TPEV_SVCSUCC イベントが、サービスの正常な終了を示すために接続を開始した Tuxedo クライアントに送信されます。接続はその後に適切な方法で切断されます。

WebLogic Tuxedo Connector アプリケーションが開始した会話

Tuxedo 対話サーバによる tpreturn の正常な呼び出しで会話が終了します。TPEV_SVCSUCC イベントが、サービスの正常な終了を示すために接続を開始した WebLogic Tuxedo Connector クライアントに送信されます。接続はその後に適切な方法で切断されます。

階層的な会話の終了

階層的な会話を適切に終了するためには、会話の終了する順序が重要になります。

A-B および B-C という 2 つのアクティブな接続があると仮定します。B が両方の接続を管理する WebLogic Tuxedo Connector アプリケーションである場合、return を呼び出すと、その呼び出しは失敗し、TPEV_SVCERR イベントがすべての開いている接続にポストされ、それらの接続は無秩序に閉じます。

両方の接続を適切に終了するには、アプリケーションでは次のシーケンスを実行する必要があります。

1. B は TPRECVONLY で tpsend を呼び出し、B-C 接続の制御権を Tuxedo アプリケーション C に引き渡します。
2. C は rval を TPSUCCESS、TPFAIL、または TPEXIT に設定して departure を呼び出します。
3. B は return を呼び出し、A についてイベント（TPEV_SVCSUCC または TPEV_SVCFAIL）をポストします。

対話サービスは、要求または応答の呼び出しを行うことができます。したがって、直前の例で、B から C への呼び出しは tpconnect ではなく tpacall() または tpcall() を使用して実行できます。対話サービスでは、tpforward を呼び出すことは許可されていません。

無秩序な切断の実行

WebLogic Server 対話クライアントまたは対話サーバは、tpdiscon を呼び出して無秩序な切断を実行します。これは、接続を「打ち切る」ような処理です。

次に、tpdiscon の呼び出しについて説明します。

- tpdiscon を呼び出すと、接続が直ちに切断され、接続の反対側で TPEV_DISCONIMM が生成されます。送信先に到着していないデータはすべて失われます。会話がトランザクションの一部である場合は、そのトランザクションをロールバックする必要があります。
- tpdiscon は、会話の開始プロセスでのみ呼び出すことができます。

対話通信のイベントについて

WebLogic Tuxedo Connector JATMI では、5 つのイベントを使用して対話通信を管理します。次の表は、それらのイベント、それらのイベントが返される機能、およびそれぞれの詳しい説明を示しています。

表 6-5 WebLogic Tuxedo Connector の対話通信のイベント

イベント	送信対象	説明
TPEV_SENDONLY	Tuxedo tprecv	接続の制御権が渡された。この Tuxedo プロセスは tpsend を呼び出すことができる。
	JATMI tprecv	接続の制御権が渡された。この JATMI プロセスは tpsend を呼び出すことができる。

表 6-5 WebLogic Tuxedo Connector の対話通信のイベント

イベント	送信対象	説明
TPEV_DISCONIMM	Tuxedo tprecv、tpsend、tpreturn	接続が切断されており、これ以上の通信はできない。JATMI tpdiscon はこのイベントを接続の開始側にポストする。開始側は、tpreturn が呼び出されたときにそのイベントを開いているすべての接続に送信する。接続は無秩序に閉じられ、トランザクションが存在する場合、そのトランザクションは中止される。
	JATMI tprecv、tpsend、return	接続が切断されており、これ以上の通信はできない。Tuxedo tpdiscon はこのイベントを接続の開始側にポストする。開始側は、return が呼び出されたときにそのイベントを開いているすべての接続に送信する。接続は無秩序に閉じられ、トランザクションが存在する場合、そのトランザクションは中止される。
TPEV_SVCERR	Tuxedo tpsend または JATMI tpsend	接続の開始側によって受信され、従属プログラムが tpreturn（Tuxedo）または return（JATMI）を発行し、接続の制御権なしに終了したことを示す。
	Tuxedo tprecv または JATMI tprecv	接続の開始側によって受信され、従属プログラムが接続の制御権なく tpreturn（Tuxedo）または return（JATMI）を正常に発行したが、呼び出しの完了前にエラーが発生したことを示す。
TPEV_SVCSUCC	Tuxedo tprecv	接続の開始側によって受信され、従属サービスが正常に終了した（つまり return が正常に呼び出された）ことを示す。
	JATMI tprecv	接続の開始側によって受信され、従属サービスが正常に終了した（つまり tpreturn が TPSUCCESS で呼び出された）ことを示す。

表 6-5 WebLogic Tuxedo Connector の対話通信のイベント

イベント	送信対象	説明
TPEV_SVCFAIL	Tuxedo tpsend または JATMI tpsend	接続の開始側によって受信され、従属プログラムが tpreturn（Tuxedo）または return（JATMI）を発行し、接続の制御権なしに終了したことを示す。サービスはステータス TPFAIL または TPEXIT で完了しており、データは Null に設定される。
	Tuxedo tprecv または JATMI tprecv	接続の開始側によって受信され、従属プログラムが正常に終了しなかったことを示す。サービスはステータス TPFAIL または TPEXIT で完了した。

WebLogic Tuxedo Connector の会話ガイドライン

会話が正常に完了するように、会話モードでは以下のガイドラインに従ってください。

- JATMI 対話プリミティブは、WebLogic Tuxedo Connector Conversation インタフェースおよび ApplicationToMonitorInterface インタフェースの定義に従って使用します。
  - 常にフラグを使用します。
  - WebLogic Tuxedo Connector JATMI で定義されているフラグのみを使用します。
- WebLogic Tuxedo Connector には、同時会話の数を制限して WebLogic Server ネットワークの過負荷を防止するために使用できるパラメータがありません。
- Tuxedo が会話の最大許容数（MAXCONV パラメータで定義）を超えた場合、WebLogic Tuxedo Connector の例外値としては TPEV_DISCONIMM が予期されます。

- 無認可の Tuxedo サービスに対する `tprecv` では、`TPEV_DISCONIMM` 例外値が生じます。
- WebLogic Tuxedo Connector クライアントが、別の対話サービスへの `tpforward` を実行する Tuxedo 対話サービスに接続された場合、WebLogic Tuxedo Connector の例外値としては `TPEV_DISCONIMM` が予期されます。
- 会話はトランザクションの中で開始できます。その際は、会話をトランザクション モードでプログラム文の一部として開始します。トランザクションの詳細については、5-1 ページの「WebLogic Tuxedo Connector JATMI トランザクション」を参照してください。
- WebLogic Tuxedo Connector リモート ドメインで `TPENOENT` が生じた場合、そのリモート ドメインは切断イベント メッセージを送り返し、そのメッセージは WebLogic Tuxedo Connector アプリケーション `tprecv` で `TPEV_DISCONIMM` 例外として捕捉されます。

7 WebLogic Tuxedo Connector JATMI VIEW

次の節では、WebLogic Tuxedo Connector VIEW バッファの使い方について説明します。

- WebLogic Tuxedo Connector VIEW バッファの概要
- VIEW 記述ファイルの作成方法
- viewj コンパイラの使用法
- JATMI アプリケーションでの VIEW バッファの使用法

WebLogic Tuxedo Connector VIEW バッファの概要

注意： Tuxedo VIEW バッファに関する詳細については、「VIEW 型バッファ」を参照してください。

WebLogic Tuxedo Connector を使用すると、非依存型 C 構造体から派生した Tuxedo VIEW バッファ タイプと同様の Java VIEW バッファを作成することができます。これにより、WebLogic Server アプリケーションおよび Tuxedo アプリケーションが共通の構造体を用いて情報を受け渡しできます。WebLogic Tuxedo Connector VIEW バッファは、FML VIEW も FML VIEW/Java 対話もサポートしていません。

VIEW 記述ファイルの作成方法

注意： fbname フィールドと null フィールドは、非依存型 Java 構造体や C 構造体とは関係がなく、Java や C の VIEW コンパイラには無視されます。これらのフィールドにはプレースホルダとして値 (たとえば、ダッシュ) を入れる必要があります。

WebLogic Server アプリケーションと Tuxedo アプリケーションは、VIEW 記述で定義された同一の情報構造を共有する必要があります。VIEW 記述ファイルの各構造体には下記の形式を使用します。

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

各値に関する説明は以下のとおりです。

- ファイル名は VIEW 名と同一です。
- ファイルごとに 1 つの VIEW しか持てません。
- WebLogic Tuxedo Connector viewj コンパイラと Tuxedo viewc コンパイラの両方で同一の VIEW 記述ファイルが使用されます。
- *viewname* は情報構造の名前です。
- コメント行は、「#」または「$」文字で始めます。
- 各構造体用の VIEW 記述ファイルに指定しなければならないフィールドを下表で説明します。

表 7-1 VIEW 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。設定できる型は、short、long、float、double、char、string、または carray。
cname	情報構造内でのフィールド名。
fbname	無視される。

フィールド	説明
count	フィールドの反復回数。
flag	次のいずれかのオプション フラグを指定する。 ■ N - ゼロ方向マッピング ■ C - 連想カウント メンバ（ACM）に追加フィールドを生成 ■ L - STRING および CARRAY に転送されるバイト数を保持
size	STRING および CARRAY バッファ タイプの場合、バッファの値の最大長を指定する。その他のバッファ タイプの場合には、このフィールドは無視される。
null	ユーザ指定の NULL 値、または、マイナス記号（-）の場合にはフィールドのデフォルト値を示す。VIEW 型バッファで使用される NULL 値は空の C 構造体メンバを示す。 デフォルトの NULL 値は、数値型の場合はすべて 0（dec_t では 0.0）。文字型の場合は `\0'。STRING および CARRAY 型では " "。 規約により、エスケープ文字として使用される定数にも、NULL 値を指定できる。VIEW コンパイラが認識するエスケープ定数は右のとおり。\ddd（d は 8 進数）、\0、\n、\t、\v、\r、\f、\\、\'、および \"。 STRING、CARRAY、および char 型の NULL 値を、二重引用符または単一引用符で囲んでもかまわない。ユーザ指定の NULL 値内のエスケープされていない引用符は、VIEW コンパイラでは受け付けられない。 VIEW メンバ記述の NULL フィールドでキーワード「NONE」を指定することもできる。この指定は、そのメンバには NULL 値がないことを意味する。文字列および文字配列メンバの最大サイズのデフォルト値は 2660 文字である。

サンプル VIEW 記述ファイル

次に、Tuxedo アプリケーションと情報を送受信するために VIEW バッファを使用するサンプル VIEW 記述を示します。この VIEW のファイル名は `infoenc` です。

コード リスト 7-1 サンプル VIEW 記述

```
VIEW infoenc
#type    cname    fbname  count flag size null
float    amount   AMOUNT  2     -    -    0.0
short    status   STATUS  2     -    -    0
init     term     TERM    2     -    -    0
char     mychar   MYCHAR  2     -    -    -
string   name     NAME    1     -    16   -
carray   carray1  CARRAY1 1     -    10   -
END
```

viewj コンパイラの使用法

`VIEW` 型バッファをコンパイルするには、引数としてパッケージ名と VIEW 記述ファイルの名前を指定して、`viewj` コマンドを実行します。出力ファイルはカレント ディレクトリに作成されます。

`viewj` コンパイラを使用するには、次のコマンドを入力します。

```
java weblogic.wtc.jatmi.viewj [package] viewfile
```

`viewj32` コンパイラを使用するには、次のコマンドを入力します。

```
java weblogic.wtc.jatmi.viewj32 [package] viewfile
```

このコマンドの引数は以下のように定義されます。

引数	説明
package	.java ソース ファイルに含まれるパッケージ名。 例 : examples.wtc.atmi.simpview
viewfile	VIEW 記述ファイルの名前。 例 : Infoenc

たとえば、

- VIEW バッファは次のようにしてコンパイルします。

```
java weblogic.wtc.jatmi.viewj examples.wtc.atmi.simpview infoenc
```

- VIEW32 バッファは次のようにしてコンパイルします。

```
java weblogic.wtc.jatmi.viewj32 examples.wtc.atmi.simpview infoenc
```

viewj コマンドの出力は .java ソース ファイルです。このソース ファイルには、VIEW 記述ファイル内の各フィールドに対する set および get アクセサ メソッドが含まれます。

JATMI アプリケーションでの VIEW バッファの使用法

注意 : VIEW バッファを使用する JATMI アプリケーションを開発する方法の例については、WebLogic Server 配布キットの examples/wtc/atmi/simpview パッケージを参照してください。

JATMI アプリケーションに VIEW バッファを取り込む際には、下記の手順に従います。

1. 「VIEW 記述ファイルの作成方法」で説明されているとおりに、アプリケーション用の VIEW 記述ファイルを作成します。

2. 「viewj コンパイラの使用法」で説明されているとおりに、VIEW 記述ファイルをコンパイルします。
3. set および get アクセサ メソッドを使用して、VIEW バッファとの間で情報を送受信します。

   クライアントと VIEW バッファとの間でアクセサを使用して情報を送受信する方法の例については、WebLogic Server 配布キットの `examples/wtc/atmi/simpview/ViewClient.java` ファイルを参照してください。
4. VIEW コンパイラの出力ファイルをソース コードにインポートします。
5. 必要な場合は、「VIEW 型バッファ」で説明されているとおりに、Tuxedo アプリケーション用の VIEW 記述ファイルをコンパイルし、その出力を C ソース ファイルにインクルードします。
6. コンパイル済み Java VIEW 記述ファイルの完全修飾クラス名と VIEW バッファ タイプ（VIEW または VIEW32）を指定した Resources MBean で WTCServer MBean をコンフィグレーションします。
7. Tuxedo アプリケーションをビルドし、起動します。
8. WebLogic Server アプリケーションをビルドし、起動します。

8　アプリケーション エラーの管理

次の節では、アプリケーションでエラー条件を管理し解釈するメカニズムについて説明します。

- アプリケーション エラーのテスト
- WebLogic Tuxedo Connector のタイムアウト条件
- アプリケーション イベントのトラッキングのガイドライン

アプリケーション エラーのテスト

注意： エラー条件のテスト方法を示すサンプルを表示するには、5-7 ページの「トランザクション コードの例」を参照してください。

アプリケーション ロジックは、戻り値を持つ呼び出しの後にエラー条件をテストし、それらの条件に基づいて適切な手順を実行する必要があります。関数が値を返すイベントでは、特定の値をテストする関数を呼び出し、各条件に適切なアプリケーション ロジックを実行できます。

例外クラス

WebLogic Tuxedo Connector では、次の例外クラスが送出されます。

- `Ferror`: FML 操作中に発生したエラーで送出される例外。
- `TPException`: TPException 障害を表す例外。
- `TPReplyException`: ユーザ データが送出される例外と関連付けられている場合に TPException 障害を表す例外。

致命的なトランザクション エラー

トランザクションの管理では、どのエラーがトランザクションにとって致命的であるかを理解することが重要です。致命的なエラーが発生した場合、トランザクションのイニシエータによって `commit()` を呼び出し、そのトランザクションをアプリケーション レベルで明示的にアボートする必要があります。トランザクションは、次のような理由によりエラーとなります。

- トランザクションのイニシエータまたは参加コンポーネントが、トランザクションをロールバックとしてマーク付けした。
- トランザクションがタイムアウトした。
- `commit()` が、トランザクションのオリジネータではなく、参加コンポーネントによって呼び出された。

WebLogic Tuxedo Connector のタイムアウト条件

WebLogic Tuxedo Connector を使用するときに発生するタイムアウトには、次の 2 つのタイプがあります。

- ブロッキング タイムアウト
- トランザクション タイムアウト

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

ブロッキング タイムアウトは、呼び出しがブロッキング条件を消去するために待機できる時間を超えると発生します。トランザクション タイムアウトは、トランザクションが `setTransactionTimeout()` 内で定義された時間以上かかっている場合に発生します。デフォルトでは、プロセスがトランザクション モード

でない場合、ブロッキング タイムアウトが実行されます。通信呼び出しの *flags* パラメータが TPNOTIME に設定されている場合は、ブロッキング タイムアウトのみが適用されます。プロセスがトランザクション モードである場合、ブロッキング タイムアウトおよび TPNOTIME フラグは適切ではありません。トランザクションが開始されたときにタイムアウトが定義されていた場合、プロセスはトランザクション タイムアウトのみに対応します。2 つの異なるタイプのタイムアウトの関係は、次のようになります。

- プロセスがトランザクション モードでなく、ブロッキング タイムアウトが非同期呼び出しで発生した場合、ブロックした通信呼び出しは失敗するが、呼び出し記述子はそのまま有効となり、再発行された呼び出しで使用可能である。今後の一般の通信に影響はない。
- トランザクション タイムアウトの場合、非同期トランザクション応答への呼び出し記述子（TPNOTRAN フラグなしで実行）は停滞状態となり、参照されなくなることがある。今後許可される通信は、応答、ブロッキング、およびトランザクションのない前述のケースのみである。

commit() の影響

タイムアウトが commit() への呼び出しの後に発生した場合のトランザクションの状態は不定です。トランザクションがタイムアウトし、そのトランザクションがアボートしたことをシステムが認識した場合、setRollbackOnly() はエラーを返します。

トランザクションの状態が明確ではない場合、トランザクションがコミットされたかアボートされたかを調べるために、リソースをクエリしてそのトランザクションの一部だった変更作業が適用されているかどうかを判断する必要があります。

TPNOTRAN の影響

注意： トランザクションは、そのトランザクションの一部ではないサービスからの応答を待機中にタイムアウトすることがあります。

プロセスがトランザクション内にあり、TPNOTRAN に設定された *flags* を使用して通信呼び出しを行う場合、呼び出されたサービスはそのトランザクションの参加コンポーネントになることはできません。サービスの成功または失敗は、そのトランザクションの結果に影響を与えません。

アプリケーション イベントのトラッキングのガイドライン

System.out.println() を使用してアプリケーションの実行を追跡し、WebLogic Server トレース ログにメッセージを書き込むことができます。String 型の変数を取得する log() メソッドを作成し、呼び出しへの引数として変数名を使用するか、呼び出しへの引数としてリテラルを引用符で囲んだメッセージを使用します。次の例は、tpcall() の進行状況を追跡するために使用する一連のメッセージです。

コード リスト 8-1　イベント ロギングの例

```
.
.
.
log("About to call tpcall");
       try {
              myRtn = myTux.tpcall("TOUPPER", myData, 0);
       }
       catch (TPReplyException tre) {
              log("tpcall threw TPReplyExcption " + tre);
              throw tre;
       }
       catch (TPException te) {
              log("tpcall threw TPException " + te);
              throw te;
       }
       catch (Exception ee) {
              log("tpcall threw exception: " + ee);
              throw new TPException(TPException.TPESYSTEM,
"Exception: " + ee);
       }
       log("tpcall successfull!");
.
```

```
.
.
private static void
log(String s)
{      System.out.println(s);}
.
.
.
```