



BEA WebLogic Server™

WebLogic Tuxedo Connector
ATMI プログラミング ガイド

BEA WebLogic Server バージョン 6.1
マニュアルの日付：2003 年 4 月 24 日

著作権

Copyright © 2002 BEA Systems, Inc. All Rights Reserved.

限定的権利条項

本ソフトウェアおよびマニュアルは、BEA Systems, Inc. 又は日本ビー・イー・エー・システムズ株式会社（以下、「BEA」といいます）の使用許諾契約に基づいて提供され、その内容に同意する場合にのみ使用することができ、同契約の条項通りにのみ使用またはコピーすることができます。同契約で明示的に許可されている以外の方法で同ソフトウェアをコピーすることは法律に違反します。このマニュアルの一部または全部を、BEA からの書面による事前の同意なしに、複写、複製、翻訳、あるいはいかなる電子媒体または機械可読形式への変換も行うことはできません。

米国政府による使用、複製もしくは開示は、BEA の使用許諾契約、および FAR 52.227-19 の「Commercial Computer Software-Restricted Rights」条項のサブパラグラフ (c)(1)、DFARS 252.227-7013 の「Rights in Technical Data and Computer Software」条項のサブパラグラフ (c)(1)(ii)、NASA FAR 補遺 16-52.227-86 の「Commercial Computer Software--Licensing」条項のサブパラグラフ (d)、もしくはそれらと同等の条項で定める制限の対象となります。

このマニュアルに記載されている内容は予告なく変更されることがあり、また BEA による責務を意味するものではありません。本ソフトウェアおよびマニュアルは「現状のまま」提供され、商品性や特定用途への適合性を始めとする（ただし、これらには限定されない）いかなる種類の保証も与えません。さらに、BEA は、正当性、正確さ、信頼性などについて、本ソフトウェアまたはマニュアルの使用もしくは使用結果に関していかなる確約、保証、あるいは表明も行いません。

商標または登録商標

BEA、Jolt、Tuxedo、および WebLogic は BEA Systems, Inc. の登録商標です。BEA Builder、BEA Campaign Manager for WebLogic、BEA eLink、BEA Manager、BEA WebLogic Collaborate、BEA WebLogic Commerce Server、BEA WebLogic E-Business Platform、BEA WebLogic Enterprise、BEA WebLogic Integration、BEA WebLogic Personalization Server、BEA WebLogic Process Integrator、BEA WebLogic Server、E-Business Control Center、How Business Becomes E-Business、Liquid Data、Operating System for the Internet、および Portal FrameWork は、BEA Systems, Inc. の商標です。

その他の商標はすべて、関係各社がその権利を有します。

BEA WebLogic Tuxedo Connector ATMI プログラマーズ ガイド

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

目次

このマニュアルの内容

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

1. WebLogic Tuxedo コネクタ プログラミングの概要

WebLogic Tuxedo コネクタ アプリケーションの開発	1-2
WebLogic Tuxedo コネクタ クライアントの開発	1-2
WebLogic Tuxedo コネクタ サーバの開発	1-2
Tuxedo CORBA オブジェクトを相互運用する WebLogic Tuxedo コネク タの使用	1-3
WebLogic Tuxedo コネクタ JATMI プリミティブ	1-3
WebLogic Tuxedo コネクタ TypedBuffers	1-4

2. WebLogic Tuxedo コネクタ クライアント EJB の開発

アプリケーションとの接続および切断	2-1
アプリケーションへの接続	2-2
アプリケーションからの切断	2-3
クライアントの基本操作	2-3
Tuxedo オブジェクトの取得	2-3
メッセージ バッファリングの実行	2-4
メッセージの送信および受信	2-4
要求 / 応答通信	2-5
対話通信	2-5
Tuxedo オブジェクトへの接続の終了	2-6
クライアント EJB の例	2-6

3. WebLogic Tuxedo Connector サービス EJB の開発

サービス EJB の基本操作	3-1
サービス情報へのアクセス	3-1
バッファ メッセージ	3-2
リクエストされたサービスの実行	3-3
要求 / 応答通信でクライアント メッセージを返す	3-3
対話通信での tpsend および tprecv の使用	3-3
サービス EJB の例	3-4

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

Tuxedo CORBA オブジェクト用の WebLogic Tuxedo Connector クライアント Bean を開発する方法	4-2
WTC ORB の使用	4-2
オブジェクト参照の取得	4-3
オブジェクトの呼び出し	4-3
ToupperCorbaBean.java コードの例	4-4
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	4-5
WebLogic Tuxedo Connector を使うための発信 RMI/IIOP アプリケーションの変更方法	4-6
EJB に FederationURL を渡すための ejb-jar.xml ファイルの変更方法	4-6
env-entry-name の割り当て	4-8
env-entry-type の割り当て	4-8
env-entry-value の割り当て	4-8
FederationURL を使ってオブジェクトにアクセスするための EJB の変更方法	4-9
FederationURL 形式の使い方	4-10
corbaloc URL フォーマットの使用	4-11
corbaloc:tgop の例	4-12
-ORBInitRef の使用例	4-12
-ORBDefaultInitRef の使用例	4-13
corbaname URL フォーマットの使用	4-13
-ORBInitRef の使用例	4-13
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	4-14

5. WebLogic Tuxedo コネクタ ATMI トランザクション

グローバル トランザクション	5-1
JTA Transaction API.....	5-2
JTA インタフェースのタイプ.....	5-2
Transaction.....	5-2
TransactionManager	5-2
UserTransaction	5-3
JTA トランザクション プリミティブ	5-3
トランザクションの定義	5-3
トランザクションの開始.....	5-4
TPNOTRAN の使用.....	5-4
トランザクションの終了.....	5-5
WebLogic Tuxedo コネクタ トランザクションのルール.....	5-6
トランザクション コードの例	5-7

6. WebLogic Tuxedo コネクタ JATMI 対話

WebLogic Tuxedo コネクタの対話通信の概要	6-1
WebLogic Tuxedo コネクタの対話の特徴.....	6-2
WebLogic Tuxedo コネクタ JATMI 対話プリミティブ	6-3
WebLogic Tuxedo コネクタの対話クライアントおよび対話サーバの作成	6-4
対話クライアントの作成.....	6-4
Tuxedo 対話サービスとの接続の確立.....	6-4
TuxedoConversationBean.java のコード例	6-5
WebLogic Tuxedo コネクタの対話サーバの作成	6-6
メッセージの送受信.....	6-6
メッセージの送信	6-7
メッセージの受信	6-7
対話の終了.....	6-8
Tuxedo アプリケーションが開始した対話	6-8
WebLogic Tuxedo コネクタ アプリケーションが開始した対話	6-9
階層的な対話の終了	6-10
無秩序な切断の実行.....	6-10
対話通信のイベントについて	6-11
WebLogic Tuxedo コネクタの対話ガイドライン	6-13

7. アプリケーション エラーの管理

アプリケーション エラーのテスト	7-1
例外クラス	7-1
致命的なトランザクション エラー	7-2
WebLogic Tuxedo コネクタのタイムアウト条件	7-2
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	7-2
commit() の影響	7-3
TPNOTRAN の影響	7-3
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	7-4

このマニュアルの内容

このマニュアルでは、BEA WebLogic Server WebLogic Tuxedo コネクタ 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 コネクタ 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 コネクタ 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 コネクタの 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 コネクタ ATMI 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 コネクタ JATMI 対話」](#)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「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、XML、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のメイン トピックを一度に 1 つずつ印刷できます。

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

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

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@bea.com までお送りください。寄せられた意見については、ドキュメントを作成および改訂する BEA の専門の担当者が直に目を通します。

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

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

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

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
[Ctrl] + [Tab]	同時に押すキーを示す。
斜体	強調または本のタイトルを示す。
等幅テキスト	コード サンプル、コマンドとそのオプション、Java クラス、データ型、ディレクトリ、およびファイル名とその拡張子を示す。等幅テキストはキーボードから入力するテキストも示す。 例： <pre>import java.util.Enumeration; chmod u+w * config/examples/applications .java config.xml float</pre>
斜体の等幅テキスト	コード内の変数を示す。 例： <pre>String <i>CustomerName</i>;</pre>

表記法	適用
すべて大文字のテキスト	デバイス名、環境変数、および論理演算子を示す。 例： LPT1 BEA_HOME OR
{ }	構文内の複数の選択肢を示す。
[]	構文内の任意指定の項目を示す。 例： <pre>java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]</pre>
	構文の中で相互に排他的な選択肢を区切る。 例： <pre>java weblogic.deploy [list deploy undeploy update] password {application} {source}</pre>
...	コマンドラインで以下のいずれかを示す。 <ul style="list-style-type: none"> ■ 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 コネクタ プログラミングの概要

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 コネクタ アプリケーションの開発](#)
- [WebLogic Tuxedo コネクタ JATMI プリミティブ](#)
- [WebLogic Tuxedo コネクタ TypedBuffers](#)

WebLogic Tuxedo コネクタ アプリケーションの開発

注意: WebLogic Tuxedo コネクタ JATMI の詳細については、[WebLogic Server クラスの Javadoc](#) を参照してください。WebLogic Tuxedo コネクタ 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 コネクタ 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 コネクタ 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 コネクタ 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 Server の *Startup Class* にある WebLogic Tuxedo コネクタ XML コンフィグレーション ファイルを介して提供されるいかなるサービスにも使用できます。

WebLogic Tuxedo コネクタ 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 コネクタは、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 コネクタの使用

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 コネクタは、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 コネクタ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、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 コネクタ 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
<code>tpacall</code>	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。

表 1-1 JATMI プリミティブ (続き)

名前	操作
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	対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。
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 コネクタ TypedBuffers

注意: WebLogic Tuxedo コネクタは、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 コネクタは、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 コネクタは、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。
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)

2 WebLogic Tuxedo コネクタ クライアント EJB の開発

注意： WebLogic Tuxedo コネクタ JATMI の詳細については、[WebLogic Server クラスの Javadoc](#) を参照してください。WebLogic Tuxedo コネクタ 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 コネクタ 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 コネクタは、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 コネクタ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 コネクタは、BDMCONFIG XML コンフィグレーションファイルを使用して、Tuxedo サービスへのパスを作成するために必要な情報を提供します。BDMCONFIG XML コンフィグレーションファイルの `T_DM_REMOTE_TDOMAIN` および `T_DM_IMPORT` セクションをコンフィグレーションすることによって、セキュリティおよびクライアント認証を提供します。この経路は WebLogic Server が起動され、WebLogic Tuxedo コネクタ XML コンフィグレーションファイルがロードされたときに作成されます。
- WebLogic Tuxedo コネクタは `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 コネクタ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 コネクタ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 コネクタは、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 コネクタは、WebLogic Server 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ときは、次のバッファ タイプを使用します。

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)。

メッセージの送信および受信

WebLogic Tuxedo コネクタ 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 コネクタ 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。
topenqueue	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 コネクタ JATMI 対話](#)」を参照してください。

以下の JATMI 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 コネクタ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	対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。

表 2-3 WebLogic Tuxedo コネクタの対話クライアントのプリミティブ (続き)

名前	操作
<code>tpsend</code>	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

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

TPServiceInformation クラスを使用すると、service.do 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ときは、次のバッファ タイプを使用します。

表 3-2 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。
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。

表 3-2 TypedBuffers (続き)

バッファ タイプ	説明
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)

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

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

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 コネクタ JATMI 対話」](#) 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
<code>tpconnect</code>	Tuxedo 対話サービスとの接続を確立するために使用する。

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

名前	操作
<code>tpdiscon</code>	対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、 <code>TPEV_DISCONIMM</code> イベントを生成するために使用する。
<code>tprecv</code>	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。
<code>tpsend</code>	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、『[WebLogic Tuxedo Connector 管理ガイド](#)』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、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。

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。

- Tuxedo CORBA オブジェクト用の WebLogic Tuxedo Connector クライアント Bean 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
- FederationURL 形式の使い方
- 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

Tuxedo CORBA オブジェクト用の WebLogic Tuxedo Connector クライアン ト Bean を開発する方法

注意： 発信 Tuxedo CORBA オブジェクトのクライアント Bean の開発方法の例については、WebLogic Server 配布キットの `examples\wtc\corba\simpappcns` パッケージを参照してください。

Tuxedo にデプロイされた CORBA オブジェクトを呼び出す EJB を開発するときは、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

WebLogic Tuxedo Connector で CORBA 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、[4-10 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 の CosNaming サーバでオブジェクト参照がバインドされた名前。

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

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:tgio:p: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 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

注意： 着信 RMI/IIOP C++ クライアントの開発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。

WebLogic Tuxedo Connector プログラミングの唯一の要件は、クライアントが COSNaming サービスに、WebLogic Server の名前 サービスがバインドされている正しい名前を渡すことです。コードリスト 4-2 に、`examples\iiop\ejb\stateless\server\tux` サンプルからネーミング コンテキストを取得するサンプル コードが提供されています。「WLS」が、『[WebLogic Tuxedo Connector 管理ガイド](#)』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 を使うための 発信 RMI/IIOP アプリケーションの変更方法

注意： 発信 RMI/IIOP クライアントを開発する方法のサンプルについては、WebLogic Server 配布キットの `examples/iiop/ejb/stateless/server/wls` パッケージを参照してください。

WebLogic Tuxedo Connector に関する唯一のプログラミング要件は、EJB が WebLogic Tuxedo Connector 固有の FederationURL を使って、リモート Tuxedo CORBA オブジェクトにアクセスするために使用する初期コンテキストを取得することです。以下の節で説明する手順に従って、WebLogic Tuxedo Connector を使用するよう発信 RMI/IIOP アプリケーションを変更します。

- EJB に FederationURL を渡すための `ejb-jar.xml` ファイルの変更方法
- FederationURL を使ってオブジェクトにアクセスするための EJB の変更方法

EJB に FederationURL を渡すための `ejb-jar.xml` ファイルの変更方法

次に示す `iiop.ejb.stateless.server.wls.ejb-jar.xml` ファイルのコードは、実行時に EJB に FederationURL 形式を渡すよう `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 EJB に FederationURL を渡す 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:tgio:p: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-name>
      </method>
    <trans-attribute>NotSupported</trans-attribute>
  </assembly-descriptor>
</ejb-jar>
```

```
</container-transaction>
</assembly-descriptor>
</ejb-jar>
```

実行時に EJB に FederationURL を渡すには、アプリケーションの `ejb-jar.xml` ファイルに、EJB に対する `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

EJB に `env-entry-value` 要素の値を渡すために使用する変数の名前を指定するには、`env-entry-name` 要素を使用します。図 4-3 のサンプルコードでは、`env-entry-name` に `foreignOrb` を指定しています。

env-entry-type の割り当て

EJB に渡される `env-entry-value` 要素のデータ型 (String、Integer、Double など) を指定するには、`env-entry-type` 要素を使用します。図 4-3 のサンプルコードでは、`foreignOrb` 変数が String データを EJB に渡すように指定しています。

env-entry-value の割り当て

EJB に渡すデータを指定するには、`env-entry-value` 要素を使用します。図 4-3 のサンプルコードでは、`foreignOrb` 変数が次の FederationURL 形式を EJB に渡すよう指定しています。

```
corbaloc:tgiop:simpapp
```

FederationURL を使ってオブジェクトにアクセスするための EJB の変更方法

ここでは、リモート Tuxedo CORBA オブジェクトにアクセスするために使用する InitialContext を FederationURL を使って取得する方法について説明します。

次に示す `iiop.ejb.stateless.server.wls.TraderBean.java` ファイルのコードは、FederationURL を使って InitialContext を取得する方法の例です。

コードリスト 4-4 InitialContext を取得するための TraderBean.java のサンプルコード

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

    try {
        InitialContext ic = new InitialContext();

        // リモート CORBA ドメインで EJB 類似の CORBA サーバ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 environment value "+ne);
    }

    catch (RemoteException re) {
        throw new CreateException("Error creating remote ejb "+re);
    }
}
.
.
.
```

FederationURL を使って、リモート Tuxedo CORBA オブジェクトに対する InitialContext を取得するには、以下の手順を使用します。

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

例：

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

図 4-3 のサンプル コードでは、`foreignOrb` 変数が次の FederationURL 形式を EJB に渡すよう指定しています。

```
corbaloc:tgio:p:simpapp
```

2. FederationURL 形式と「`/NameService`」を連結して、FederationURL を形成します。

例：

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

次のような FederationURL が生成されます。

```
corbaloc:tgio:p: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);
```

結果として、Tuxedo CORBA オブジェクトの InitialContext が取得されます。

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:tgio:p` フォームは、BEA `tgio:p` 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:tgioip":[<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 コネクタ ATMI 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 コネクタ 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 コネクタは、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オペレーションを許可します。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が、`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 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()` へ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オペレーションは、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()` への呼び出しによって終了されます。`commit()` が正常に返された場合、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 コネクタ 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()` へ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()` への呼び出しは、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では `TPEBADDESC` が返されます。

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");  
    }  
}
```

5 WebLogic Tuxedo コネクタ ATMI トランザクション

```
// 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 データベース処理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処理のロギングなど）
    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 コネクタ JATMI 対話

注意： BEA Tuxedo の対話通信の詳細については、「[Writing Conversational Clients and Servers](#)」を参照してください。

以下の節では、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。

- [WebLogic Tuxedo コネクタの対話通信の概要](#)
- [WebLogic Tuxedo コネクタの対話の特性](#)
- [WebLogic Tuxedo コネクタ JATMI 対話プリミティブ](#)
- [WebLogic Tuxedo コネクタ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](#)
- [対話の終了](#)
- [無秩序な切断の実行](#)
- [対話通信のイベントについて](#)
- [WebLogic Tuxedo コネクタの対話ガイドライン](#)

WebLogic Tuxedo コネクタの対話通信の概要

WebLogic Tuxedo コネクタは、BEA Tuxedo の対話を 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、各サイドで対話の状態に関する情報が維持されます。接続を開

き、対話を開始するプロセスが、対話の開始側です。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 コネクタの対話の特性

WebLogic Tuxedo コネクタ JATMI 対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 コネクタ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 コネクタ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 コネクタ JATMI 対話プリミティブ

注意： WebLogic Tuxedo コネクタ JATMI の詳細については、[WebLogic クラスの Javadoc](#) の `weblogic.wtc.jatmi` を参照してください。

以下の WebLogic Tuxedo コネクタ 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 コネクタの対話クライアントのプリミティブ

名前	操作
<code>tpconnect</code>	Tuxedo 対話サービスとの接続を確立するために使用する。
<code>tpdiscon</code>	接続を中止し、TPEV_DISCONIMM イベントを生成するために使用する。
<code>tprecv</code>	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。
<code>tpsend</code>	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 コネクタ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 コネクタの対話クライアントを作成するには、[2-1 ページの「WebLogic Tuxedo コネクタ 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、対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

注意： `tpconnect` の詳細については、[WebLogic クラスの Javadoc](#) の `weblogic.wtc.jatmi` を参照してください。

WebLogic Tuxedo コネクタ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、対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 コネクタ JATMI `tpconnect` のパラメータ

パラメータ	説明
<code>svc</code>	対話サービス名の文字ポインタ。 <code>svc</code> を指定しないと、呼び出しは失敗し、 <code>TPException</code> が <code>TPEV_DISCONIMM</code> に設定される。

表 6-2 WebLogic Tuxedo コネクタ JATMI tpconnect のパラメータ (続き)

パラメータ	説明
<i>data</i>	データバッファのポインタ。接続を確立するときに、 <i>data</i> 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 <i>type</i> および <i>subtype</i> 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。 <i>data</i> 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。
<i>flags</i>	<p>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。</p> <p>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。</p> <p>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。</p> <p>TPNOTRAN: <i>svc</i> 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、<i>svc</i> 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。<i>svc</i> 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。</p> <p>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。</p> <p>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。</p>

TuxedoConversationBean.java のコード例

次に、`tpconnect` を使用して対話を開始するコードの例を示します。

```
.  
. .  
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.tpsconnect("CONNECT_SVC",null,flags);  
. .  
. .
```

WebLogic Tuxedo コネクタの対話サーバの作成

WebLogic Tuxedo コネクタ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 コネクタ 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

注意: `tpsend` の詳細については、[WebLogic クラスの Javadoc](#) の `weblogic.wtc.jatmi` を参照してください。

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 コネクタ JATMI `tpconnect` のパラメータ

パラメータ	説明
<code>data</code>	この対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。
<code>flags</code>	<p>フラグは以下のいずれかを指定できる。</p> <p>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。</p> <p>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。</p> <p>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。</p>

メッセージの受信

注意: `tprecv` の詳細については、[WebLogic クラスの Javadoc](#) の `weblogic.wtc.jatmi` を参照してください。

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 コネクタ JATMI `tpconnect` のパラメータ

パラメータ	説明
<code>flags</code>	<p>フラグは以下のいずれかを指定できる。</p> <p>TPNOBLOCK: <code>tprecv</code> 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、<code>tprecv</code> は応答を取得して返る。このフラグを指定しない場合は、応答がないと、<code>tprecv</code> 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。</p> <p>TPNOTIME: <code>tprecv</code> が応答を無限に待つことを指定する。<code>tprecv</code> は、ブロッキング タイムアウトの影響は受けませんが、トランザクション タイムアウトの影響は受ける。</p> <p>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。</p>

対話の終了

WebLogic Server と Tuxedo の対話は、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、対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対話](#)
- [WebLogic Tuxedo コネクタ アプリケーションが開始した対話](#)
- [階層的な対話の終了](#)

Tuxedo アプリケーションが開始した対話

WebLogic Server 対話サーバによる `return` の正常な呼び出しで対話が終了します。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 コネクタ アプリケーションが開始した対話

Tuxedo 対話サーバによる `treturn` の正常な呼び出しで対話が終了します。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 コネクタ 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。

階層的な対話の終了

階層的な対話を適切に終了するためには、対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 コネクタ 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 `tpcall` 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。対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、対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

注意： WebLogic Tuxedo コネクタ JATMI の詳細については、[WebLogic クラスの Javadoc](#) の `weblogic.wtc.jatmi` を参照してください。

WebLogic Tuxedo コネクタ 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 コネクタの対話通信のイベント

イベント	送信対象	説明
TPEV_SENDOONLY	Tuxedo <code>tprecv</code>	接続の制御権が渡された。この Tuxedo プロセスは <code>tpsend</code> を呼び出すことができる。
	JATMI <code>tprecv</code>	接続の制御権が渡された。この JATMI プロセスは <code>tpsend</code> を呼び出すことができる。
TPEV_DISCONIMM	Tuxedo <code>tprecv</code> 、 <code>tpsend</code> 、 <code>tpreturn</code>	接続が切断されており、これ以上の通信はできない。JATMI <code>tpdiscon</code> はこのイベントを接続の開始側にポストする。開始側は、 <code>tpreturn</code> 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 <code>tprecv</code> 、 <code>tpsend</code> 、 <code>return</code>	接続が切断されており、これ以上の通信はできない。Tuxedo <code>tpdiscon</code> はこのイベントを接続の開始側にポストする。開始側は、 <code>return</code> 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。

表 6-5 WebLogic Tuxedo コネクタの対話通信のイベント（続き）

イベント	送信対象	説明
TPEV_SVCERR	Tuxedo <code>tpsend</code> または JATMI <code>tpsend</code>	接続の開始側によって受信され、従属プログラムが <code>tpreturn</code> (Tuxedo) または <code>return</code> (JATMI) を発行し、接続の制御権なしに終了したことを示す。
	Tuxedo <code>tprecv</code> または JATMI <code>tprecv</code>	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 <code>tpreturn</code> (Tuxedo) または <code>return</code> (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 <code>tprecv</code>	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 <code>return</code> が正常に呼び出された）ことを示す。
	JATMI <code>tprecv</code>	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 <code>tpreturn</code> が <code>TPSUCCESS</code> で呼び出された）ことを示す。
TPEV_SVCFAIL	Tuxedo <code>tpsend</code> または JATMI <code>tpsend</code>	接続の開始側によって受信され、従属プログラムが <code>tpreturn</code> (Tuxedo) または <code>return</code> (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 <code>TPFAIL</code> または <code>TPEXIT</code> で完了しており、データは <code>Null</code> に設定される。
	Tuxedo <code>tprecv</code> または JATMI <code>tprecv</code>	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 <code>TPFAIL</code> または <code>TPEXIT</code> で完了した。

WebLogic Tuxedo コネクタの対話ガイドライン

対話が正常に完了するように、対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 コネクタ対話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 コネクタ JATMI で定義されているフラグのみを使用します。
- WebLogic Tuxedo コネクタには、同時対話の数を制限して 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 が対話の最大許容数 (MAXCONV パラメータで定義) を超えた場合、WebLogic Tuxedo コネクタの例外値としては 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 コネクタ 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 コネクタの例外値としては TPEV_DISCONIMM が予期されます。
- 対話はトランザクションの中で開始できます。その際は、対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 コネクタ ATMI トランザクション」](#)を参照してください。
- WebLogic Tuxedo コネクタ 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 コネクタ アプリケーション `tprecv` で TPEV_DISCONIMM 例外として捕捉されます。

7 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 コネクタ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 コネクタでは、次の例外クラスが送出されます。

- **Error** : 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ポ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 コネクタのタイムアウト条件

WebLogic Tuxedo コネクタ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しますが、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す。今後の一般の通信に影響はありません。
- 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ります。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す。

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明確です。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ポ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ポ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 7-1 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);
}
```

- -
 -
-

