BEA Tuxedo®

BEA Tuxedo CORBA サーバ間通信

BEA Tuxedo 8.1
2003 年 1 月

Copyright

Restricted Rights Legend

Trademarks or Service Marks

目次

このマニュアルについて

このマニュアルでは、BEA Tuxedo 製品の CORBA サーバ間通信機能の使い方について説明します。このマニュアルでは、サーバ間通信に関連する概念を定義し、Java および C++ 共同クライアント / サーバ・アプリケーションの開発プロセスについて説明します。また、Chat Room および Callback サンプル・アプリケーションのビルドおよび実行方法についても、このマニュアルで説明します。

このマニュアルでは、以下の内容について説明します。

- 「第 1 章 CORBA サーバ間通信について」では、サーバ間通信を使用して共同クライアント / サーバ・アプリケーションをビルドするために理解しておく必要のある概念について説明します。
- 「第 2 章 C++ 共同クライアント / サーバ・アプリケーションの開発」では、C++ 共同クライアント / サーバ・アプリケーションのビルドについて、また Chat Room サンプル・アプリケーションのビルドおよび実行方法について説明します。
- 「第 3 章 Java 共同クライアント / サーバ・アプリケーション」では、Java 共同クライアント / サーバ・アプリケーションをビルドする手順について概説します。

対象読者

このマニュアルは、BEA Tuxedo アプリケーションに CORBA サーバ間通信をインプリメントするプログラマを対象としています。

e-docs Web サイト

BEA Tuxedo 製品のマニュアルは BEA 社の Web サイトで参照することができます。BEA ホーム・ページの [製品のドキュメント] をクリックするか、または http://edocs.beasys.co.jp/e-docs/index.html に直接アクセスしてください。

マニュアルの印刷方法

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

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

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

関連情報

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

サポート情報

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

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

BEA Tuxedo 製品に関するご質問、または BEA Tuxedo のインストールや使用に際して問題が発生した場合は、www.bea.com の BEA WebSUPPORT を通して BEA カスタマ・サポートにお問い合わせください。カスタマ・サポートへの問い合わせ方法は、製品パッケージに同梱されているカスタマ・サポート・カードにも記載されています。

カスタマ・サポートへお問い合わせの際には、以下の情報をご用意ください。

- お客様のお名前、電子メール・アドレス、電話番号、Fax 番号
- お客様の会社名と会社の住所
- ご使用のマシンの機種と認証コード
- ご使用の製品名とバージョン
- 問題の説明と関連するエラー・メッセージの内容

表記上の規則

このマニュアルでは、以下の表記規則が使用されています。

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

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

太字の等幅テキスト	コード内の重要な語を示します。 例 : `void **commit** ( )`
斜体の等幅テキスト	コード内の変数を示します。 例 : `String *expr*`

規則	項目
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します。 例 : LPT1 SIGNON OR
{ }	構文の行で、選択肢の組み合わせを示します。かっこは入力しません。
[ ]	構文の行で、オプション項目を示します。かっこは入力しません。 例 : `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
|	構文の行で、相互に排他的な選択肢の区切りとして使います。記号は入力しません。
...	コマンド・ラインで、以下のいずれかの場合を示します。 ■ コマンド・ラインで、同じ引数を繰り返し使用できることを示します。 ■ 文で、追加のオプション引数が省略されていることを示します。 ■ 追加のパラメータ、値、またはその他の情報を入力できることを示します。 記号は入力しません。 例 : `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
. . .	コード例または構文の行で、項目が省略されていることを示します。記号は入力しません。

章

1 CORBA サーバ間通信について

ここでは、以下の内容について説明します。

- CORBA サーバ間通信の概要
- 共同クライアント / サーバ・アプリケーション
- コールバック・オブジェクトのオブジェクト方針

CORBA サーバ間通信の概要

CORBA サーバ間通信を使用すると、BEA Tuxedo アプリケーションで CORBA オブジェクトを呼び出し、その CORBA オブジェクトからの呼び出しを処理できます (コールバック・オブジェクトと呼ばれる)。CORBA オブジェクトは BEA Tuxedo ドメインの内部にも外部にも配置できます。

BEA Tuxedo 製品は、インターネット ORB 間プロトコル（IIOP）バージョン 1.2 のインプリメンテーションを提供します。これは CORBA オブジェクトとの間でインバウンドおよびアウトバウンドの通信を可能にするものです。サーバ間通信により、ネットワーク・リソースをより効率的に利用できるようになり、サード・パーティのオブジェクト・リクエスト・ブローカ

（ORB）との統合が提供されます。加えて、サーバ間通信は IIOP のバージョン 1.0 および 1.1 を使用してインプリメントされる CORBA オブジェクトによりサポートされています。

共同クライアント / サーバ・アプリケーション

サーバ間通信により、クライアント・アプリケーションがほかのクライアント・アプリケーションからの要求に対処するサーバ・アプリケーションとして動作できるようになります。さらに、サーバ間通信を使用すると BEA Tuxedo サーバ・アプリケーションでほかの ORB に対してオブジェクトを呼び出せます。

注記　BEA Tuxedo および WebLogic Enterprise 製品の旧バージョンでは、クライアント・アプリケーションは、Object Management Group（OMG）インターフェイス定義言語（IDL）で定義された CORBA オブジェクトのオペレーションを呼び出していました。サーバ・アプリケーションは、CORBA オブジェクトのオペレーションをインプリメントしていました。サーバ・アプリケーション内の CORBA オブジェクトは、BEA Tuxedo TP フレームワークおよび環境オブジェクトを使用して、状態管理、セキュリティ、およびトランザクションのインプリメントを行っていました。これらの CORBA オブジェクトは、BEA Tuxedo オブジェクトと呼ばれます。サーバ・アプリケーションはほかのサーバ・アプリケーションに対してクライアント・アプリケーションとして動作できましたが、クライアント・アプリケーションがほかのクライアント・アプリケーションに対し、サーバ・アプリケーションとして動作することはできませんでした。

サーバ間通信機能は、コールバック・オブジェクトを介して利用できます。コールバック・オブジェクトには、次の 2 つの目的があります。

- BEA Tuxedo または CORBA オブジェクトのオペレーションの呼び出し
- CORBA オブジェクトのオペレーションのインプリメント

コールバック・オブジェクトは TP フレームワークを使用しません。また、BEA Tuxedo の管理下には置かれません。コールバック・オブジェクトは、トランザクションの振る舞い、セキュリティ、信頼性、およびスケーラビリティが重要ではない場合にのみ使用してください。

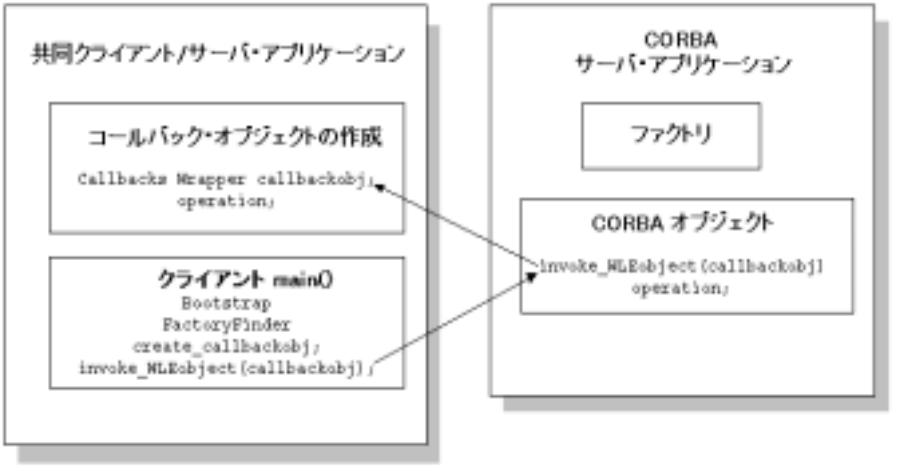
コールバック・オブジェクトは、共同クライアント / サーバ・アプリケーションにインプリメントされます。共同クライアント / サーバ・アプリケーションは、以下のもので構成されています。

- ORB の初期化、BEA Tuxedo 環境オブジェクトを使用しての接続の確立、FactoryFinder オブジェクトへの初期リファレンスの解決、ファクトリを使用した BEA Tuxedo オブジェクトの作成など、BEA Tuxedo クライアント・アプリケーションの機能を実行するプログラム・ロジック。
- コールバック・オブジェクトのサーバントを作成し、オブジェクト ID を使用してそのコールバック・オブジェクトを活性化するプログラム・ロジック。

注記　BEA Tuxedo CORBA 製品のうち、CORBA 環境のリリース 8.0 には、BEA WebLogic Enterprise の旧リリースで提供されていた BEA クライアント環境オブジェクトが引き続き含まれており、BEA Tuxedo 8.0 CORBA クライアントでも使用できます。BEA Tuxedo 8.0 クライアントは、これらの環境オブジェクトを使用して、ブートストラップ処理、セキュリティ、およびトランザクション・オブジェクトへの初期リファレンスを解決する必要があります。このリリースでは、OMG インターオペラブル・ネーミング・サービス（INS）を使用してブートストラップ処理、セキュリティ、およびトランザクション・オブジェクトへの初期リファレンスを解決するためのサポートが追加されています。INS の詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』の「第 4 章 CORBA ブートストラップ処理のプログラミング・リファレンス」を参照してください。

図 1-1 では、共同クライアント / サーバ・アプリケーションの構造を示します。

図 1-1 共同クライアント / サーバ・アプリケーションの構造



C++ および Java による共同クライアント / サーバ・アプリケーションがサポートされています。

Java アプレットでのコールバック・オブジェクトの使用は、Java アプレットのセキュリティ・メカニズムにより制限されています。独自の環境またはプロトコルを使用した Java アプレットによるソケットの作成およびリッスンが可能な、任意の Java アプレット実行時環境を、共同クライアント / サーバ・アプリケーションとして動作させることができます。しかし、Java アプレット実行時環境がソケット通信を制限している場合、その Java アプレットは共同クライアント / サーバ・アプリケーションとして動作できません。

共同クライアント / サーバ・アプリケーションでは、IIOP を使用して BEA Tuxedo サーバ・アプリケーションと通信します。IIOP は、使用している IIOP プロトコルのバージョンに応じて、以下の方法で動作できます。

- 双方向

  共同クライアント / サーバ・アプリケーションは、常に BEA Tuxedo のドメイン内の同じ IIOP サーバ・ハンドラ（ISH）に接続されます。その ISH は、共同クライアント / サーバ・アプリケーションとの間で要求を送受信するのに、同じ接続を再利用します。

- デュアル・ペア接続

  共同クライアント / サーバ・アプリケーションでは、Bootstrap オブジェクトの `register_callback_port` メソッドを使用して、共同クライアント / サーバ・アプリケーションの受信ポートを ISH に登録します。共同クライアント / サーバ・アプリケーション内のコールバック・オブジェクトに対するサーバ・アプリケーションからの呼び出しは、共同クライアント / サーバ・アプリケーションに接続された ISH 経由でルーティングされます。この ISH では、接続された共同クライアント / サーバ・アプリケーションとの間で要求の送信と応答の受信を行うために、第 2 のアウトバウンド接続を使用します。アウトバウンド接続は、受信時接続とペアになります。これは、接続が 1 つしか使用されない双方向 IIOP とは異なる点です。

- 非対称

  共同クライアント / サーバ・アプリケーションにより、任意のコールバック・オブジェクトを呼び出せます。呼び出す対象は、ISH に接続された共同クライアント / サーバ・アプリケーションでインプリメントされたコールバック・オブジェクトに限りません。非対称 IIOP を使用すると、呼び出しを処理するために使用可能な ISH を探すよう ORB インフラストラクチャに強制できます。

注記　リモート・オブジェクトのタイプと希望するアウトバウンド IIOP コンフィギュレーションによっては、さらに別のプログラミング・タスクを実行しなければならない場合があります。

表 1-1 では、オブジェクトの各タイプとアウトバウンド IIOP についての要件を示します。

表 1-1 アウトバウンド IIOP を使用するためのプログラミング要件

オブジェクトのタイプ	非対称要件	ペア接続要件	双方向要件
リモート共同クライアント / サーバ	`ISL CLOPT -O` オプションを設定します。	`Tobj_Bootstrap::register_callback_port` メソッドを使用してコールバック・ポートを登録します。	`CORBA::ORB::create_policy` メソッドを使用して POA 上で `BiDirPolicy` を設定します。
外部 (非 BEA-Tuxedo) ORB	`ISL CLOPT -O` オプションを設定します。	適用されません。	ORB が POA および `BiDirPolicy` をサポートしている場合は、`CORBA::ORB::create_policy` メソッドを使用して POA 上で `BiDirPolicy` を設定します。
リモート・クライアント	リモート・クライアントはサーバではないため、アウトバウンド IIOP は使用できません。		
ネイティブ共同クライアント / サーバ	アウトバウンド IIOP は使用されません。		
ネイティブ・クライアント	アウトバウンド IIOP は使用されません。		

双方向、デュアル・ペア接続、および非対称 IIOP の詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』を参照してください。

コールバック・オブジェクトのオブジェクト方針

コールバック・オブジェクトには、オブジェクト・リファレンスの有効期間やオブジェクト ID がそのオブジェクトに割り当てられる方法を制御する方針が割り当てられます。オブジェクト方針は、コールバック・オブジェクトへのリファレンス作成時に定義されます。また、これらの方針はコールバック・ラッパー・オブジェクト内で定義されるので、共同クライアント / サーバ・アプリケーションの開発が簡略化されます。

コールバック・オブジェクトでは、次のオブジェクト方針がサポートされています。

- 一時 / システム ID ― このタイプのコールバック・オブジェクトのオブジェクト・リファレンスは、共同クライアント / サーバ・アプリケーションの持続期間中のみ有効です。オブジェクト ID は BEA Tuxedo システムによって割り当てられます。このタイプのオブジェクトは、共同クライアント / サーバ・アプリケーションが終了するまでの間にのみ受信する呼び出しに使用されます。
- 永続 / システム ID ― このタイプのコールバック・オブジェクトのオブジェクト・リファレンスは、共同クライアント / サーバ・アプリケーションでの複数の呼び出しにわたって有効です。オブジェクト ID は BEA Tuxedo システムによって割り当てられます。このタイプのオブジェクトは、ある一定の期間にわたって終了および再起動する共同クライアント / サーバ・アプリケーションで有用です。共同クライアント / サーバ・アプリケーションは、実行中に、ある特定のコールバック・オブジェクトに対する要求を、そのオブジェクト・リファレンスと共に受け取ることができます。一般的には、共同クライアント / サーバ・アプリケーションが、オブジェクト・リファレンスを一度作成し、それを自身の恒久的な記憶域に保存して、起動のたびにそのオブジェクトのサーバントを再活性化します。
- 永続 / ユーザ ID ― このオブジェクト方針は永続 / システム ID と同じですが、オブジェクト ID が共同クライアント / サーバ・アプリケーションによって割り当てられる点が異なります。

一時オブジェクト方針を備えたコールバック・オブジェクトを作成する場合、オブジェクト・リファレンスは共同クライアント / サーバ・アプリケーションが終了するまで、または `stop_all_objects` メソッドが呼び出されるまでの間のみ有効です。

永続オブジェクト方針を備えたコールバック・オブジェクトを作成する場合、オブジェクト・リファレンスは共同クライアント / サーバ・アプリケーションの終了後でも有効です。共同クライアント / サーバ・アプリケーションが、同一オブジェクト ID についてサーバントを終了、再起動、および活性化すると、サーバントはそのオブジェクト・リファレンスに対して行われた要求を受け取ります。

注記　ネイティブ共同クライアント / サーバ・アプリケーション、つまり呼び出し側のサーバ・アプリケーションと同じ BEA Tuxedo ドメインに配置された共同クライアント / サーバ・アプリケーションを作成する場合は、永続 / システム ID または永続 / ユーザ ID のオブジェクト方針を使用することはできません。

章

2 C++ 共同クライアント / サーバ・アプリケーションの開発

ここでは、以下の内容について説明します。

- 開発プロセス
- Chat Room サンプル・アプリケーション
- ステップ 1: OMG IDL の記述
- ステップ 2: スケルトンおよびクライアント・スタブの生成
- ステップ 3: 各オブジェクトのオペレーションをインプリメントするメソッドの記述
- ステップ 4: 共同クライアント / サーバ・アプリケーションのクライアント部分の記述
- ステップ 5: コールバック・ラッパー・オブジェクトを使用したコールバック・オブジェクトの作成
- ステップ 6: コールバック・オブジェクトにリファレンスを渡すことによるオブジェクトのオペレーションの呼び出し
- ステップ 7: コンフィギュレーション情報の指定
- ステップ 8: 共同クライアント / サーバ・アプリケーションのコンパイル

- POA の使用によるコールバック・オブジェクトの作成
- C++ 共同クライアント / サーバ・アプリケーションのスレッドに関する留意事項
- Chat Room サンプル・アプリケーションのビルドと実行

開発プロセス

表 2-1 では、C++ 共同クライアント / サーバ・アプリケーションの開発プロセスの概略を示します。

表 2-1 C++ 共同クライアント / サーバ・アプリケーションの開発プロセス

手順	**説明**
1	BEA Tuxedo アプリケーションで使用するコールバック・インターフェイスおよび CORBA インターフェイスの OMG IDL を記述します。
2	スケルトンおよびクライアント・スタブを生成します。
3	各オブジェクトのオペレーションをインプリメントするメソッドを記述します。
4	共同クライアント / サーバ・アプリケーションのクライアント部分を記述します。
5	コールバック・ラッパー・オブジェクトを使用してコールバック・オブジェクトを作成します。
6	コールバック・オブジェクトのオブジェクト・リファレンスを渡すことにより、BEA Tuxedo オブジェクトのオペレーションを呼び出します。
7	コンフィギュレーション情報を指定します。
8	共同クライアント / サーバ・アプリケーションをコンパイルします。

これらの手順については、以降の節で説明します。

共同クライアント / サーバ・アプリケーション内のコールバック・オブジェクトはトランザクションに関与せず、オブジェクト管理機能を持たないため、このオブジェクト用のインプリメンテーション・コンフィギュレーション・ファイル (*filename*.icf) を作成する必要はありません。ただし、BEA Tuxedo アプリケーション内の BEA Tuxedo オブジェクトの ICF ファイルは作成する必要があります。ICF ファイルの記述については、『BEA Tuxedo CORBA サーバ・アプリケーションの開発方法』を参照してください。

Chat Room サンプル・アプリケーション

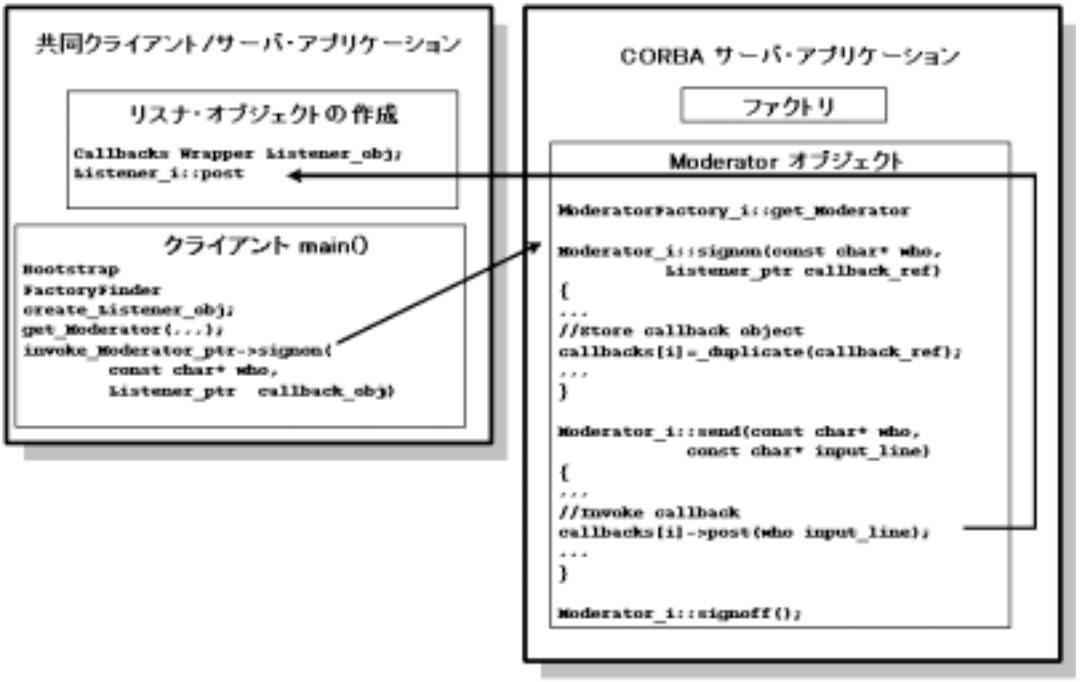
ここでは、Chat Room サンプル・アプリケーションを使用して開発の手順を説明します。チャット・ルームとは、さまざまな場所にいる人々が相互に通信できるアプリケーションです。ログインしたクライアント・アプリケーションをトラッキングし、それらにメッセージを配信する役目を持つモデレータが、チャット・ルームであると考えてください。

クライアント・アプリケーションは、ユーザ名を指定してモデレータにログインします。キーボードにメッセージが入力されると、クライアント・アプリケーションはモデレータを呼び出してそのメッセージを渡します。次に、モデレータはコールバック・オブジェクトを呼び出して、ほかのすべてのクライアント・アプリケーションにメッセージを配信します。

Chat Room サンプル・アプリケーションは、C++ 共同クライアント / サーバ・アプリケーションおよび BEA Tuxedo サーバ・アプリケーションで構成されています。共同クライアント / サーバ・アプリケーションはキーボード入力を受け取り、モデレータに対して呼び出しを行います。また共同クライアント / サーバ・アプリケーションは、モデレータからのメッセージをリッスンする、つまりモデレータからの呼び出しを受信するように、コールバック・オブジェクトを設定します。Chat Room サンプル・アプリケーションの BEA Tuxedo サーバ・アプリケーションでは、モデレータがインプリメントされます。

図 2-1 では、Chat Room サンプル・アプリケーションがどのように動作するのかを示します。

図 2-1Chat Room サンプル・アプリケーションの動作方法



Chat Room サンプル・アプリケーションは、次のように動作します。

1. 共同クライアント / サーバ・アプリケーションは、コールバック・オブジェクト (Listener オブジェクト) のロジックをインプリメントし、Listener オブジェクトのサーバントを作成し、Listener オブジェクトを活性化します。
2. 共同クライアント / サーバ・アプリケーションは、Listener オブジェクトのオブジェクト・リファレンスを作成し、`signon` オペレーションの一部として Moderator オブジェクトに渡します。
3. Chat Room サンプル・アプリケーションのサーバ・アプリケーションでは、キーボード入力によるメッセージを確認します。

4. キーボード入力によりメッセージが生成されると、Chat Room サンプル・アプリケーションは send オペレーションを介してそのメッセージを Moderator オブジェクトに送信します。
5. Chat Room サンプル・アプリケーションは、一時的に制御を ORB に渡して、共同クライアント / サーバ・アプリケーションの Listener オブジェクトが Moderator オブジェクトから post 呼び出しを受け取れるようにします。
6. 共同クライアント / サーバ・アプリケーション内の Listener オブジェクトは、クライアント・アプリケーションからの要求があるまで、ポストされたメッセージを保存します。

Chat Room サンプル・アプリケーションのソース・ファイルは、BEA Tuxedo ソフトウェア・ディレクトリの *TUXDIR*\samples\corba\chatroom ディレクトリに格納されています。詳細については、第 2 章の 30 ページ「Chat Room サンプル・アプリケーションのビルドと実行」を参照してください。

ステップ 1: OMG IDL の記述

使用できる CORBA インターフェイスをクライアント・アプリケーションに記述するには、Object Management Group (OMG) インターフェイス定義言語 (IDL) を使用します。OMG IDL で記述したインターフェイス定義を使用すると、完全に CORBA インターフェイスを定義し、各オペレーションの引数を指定できます。OMG IDL は、純粋な宣言型言語です。つまり、インプリメンテーションの詳細は含まれていません。OMG IDL の詳細については、『BEA Tuxedo CORBA クライアント・アプリケーションの開発方法』を参照してください。

Chat Room サンプル・アプリケーションにでは、表 2-2 に示した CORBA インターフェイスがインプリメントされます。

表 2-2 Chat Room サンプル・アプリケーションの CORBA インターフェイス

インターフェイス	説明	オペレーション
`Listener`	コールバック・オブジェクト。	`post()`

`Moderator`	クライアント・アプリケーションからの入力を受信し、コールバック・オブジェクトによりメッセージを転送して共同クライアント / サーバ・アプリケーションに戻します。	`signon()`
		`send()`
		`signoff()`

`ModeratorFactory`	Moderator オブジェクトへのオブジェクト・リファレンスを作成します。	`get_moderator()`

リスト 2-1 は、Listener インターフェイスを定義する chatclient.idl を示します。

コード リスト 2-1 Listner **インターフェイスの** OMG IDL

```
module ChatClient{
       interface Listener  {
              oneway void post (in string from,
                                in string output_line);
       };
};
```

リスト 2-2 は、Chat Room サンプル・アプリケーションの Moderator および ModeratorFactory インターフェイスを定義する chatroom.idl を示します。別の OMG IDL ファイルのインターフェイスに対するリファレンスを解決するには、#include を使用します。Chat Room サンプル・アプリケーションでは、signon メソッドにはパラメータとして Listener オブジェクトが必要であり、その IDL ファイルは Listener インターフェイスを定義する OMG IDL ファイルを #include している必要があります。

コード リスト 2-2 Moderator **および** ModeratorFactory **インターフェイスの** OMG IDL

```
#include "ChatClient.idl"

module ChatRoom {

       interface Moderator {
           exception IdAlreadyUsed{};
           exception NoRoomLeft{};
           exception IdNotKnown{};

           void signon( in string who,
                        in ChatClient::Listener callback_ref )
                            raises( IdAlreadyUsed, NoRoomLeft );

           void send  (in string who,
                       in string input_line )
                            raises( IdNotKnown );
```

```
        void signoff(in string who )
                        raises( IdNotKnown );
};
```

```
        interface ModeratorFactory {
           Moderator get_moderator( in string chatroom_name );
        };
};
```

ステップ 2: スケルトンおよびクライアント・スタブの生成

OMG IDL で定義したインターフェイス仕様は、IDL コンパイラによってスケルトンおよびクライアント・スタブの生成に使用されます。共同クライアント / サーバ・アプリケーションは、BEA Tuxedo オブジェクトにクライアント・スタブを、コールバック・オブジェクトにスケルトンおよびクライアント・スタブを使用することに注意してください。

たとえば Chat Room サンプル・アプリケーションでは、共同クライアント / サーバ・アプリケーションは Listener オブジェクト (つまりコールバック・オブジェクト) のためのスケルトンおよびクライアント・スタブを使用して、オブジェクトをインプリメントします。また、共同クライアント / サーバ・アプリケーションは `Moderator` および `ModeratorFactory` インターフェイスのためのクライアント・スタブを使用して、オブジェクトのオペレーションを呼び出します。BEA Tuxedo サーバ・アプリケーションは Moderator および ModeratorFactory オブジェクトのスケルトンを使用してオブジェクトをインプリメントし、Listener オブジェクトのためのクライアント・スタブを使用してオブジェクトのオペレーションを呼び出します。

開発プロセスでは、`idl` コマンドを `-P` および `-i` の各オプションを指定して使用し、コールバック・オブジェクトを定義する OMG IDL ファイルをコンパイルします。たとえば Chat Room サンプル・アプリケーションでは、`chatclient.idl` ファイルが OMG IDL ファイルです。各オプションは、次のように動作します。

- `-P` オプションを指定すると、`PortableServer::ServantBase` クラスから直接継承したスケルトン・クラスが作成されます。`PortableServer::ServantBase` からの継承は、共同クライアント / サー

バ・アプリケーションが明示的にコールバック・オブジェクトのサーバントを作成し、そのサーバントの状態を初期化する必要があることを意味しています。activate_object および deactivate_object メソッドは PortableServer::ServantBase クラスのメンバであるため、コールバック・オブジェクトのサーバントでは使用できません。

- -i オプションを指定すると、インプリメンテーション・テンプレート・ファイルが生成されます。このファイルは、Listener オブジェクトの OMG IDL で定義されたインターフェイスをインプリメントするコードのテンプレートです。

次に、BEA Tuxedo サーバ・アプリケーションのインターフェイスを定義する OMG IDL ファイルをコンパイルする必要があります。たとえば Chat Room サンプル・アプリケーションでは、chatroom.idl ファイルが OMG IDL ファイルです。この OMG IDL ファイルをコンパイルするには、idl コマンドを -i オプションのみ指定して使用します。

表 注記 に、idl コマンドによって作成されるファイルの一覧を示します。

注記 Chat Room サンプル・アプリケーションでは、ChatClient.idl および ChatRoom.idl の各ファイル用に生成されたテンプレート・ファイルは、それらによってインプリメントされるオブジェクト (Listener および Moderator) を反映するよう名前を変更されています。また、Moderator オブジェクト用のテンプレート・ファイルには、ModeratorFactory オブジェクトのインプリメンテーションが含まれています。

表 2-3 idl ファイルにより生成されるファイル

ファイル	idl コマンドで作成される Chat Room サンプル・アプリケーションのファイル	説明
クライアント・スタブ・ファイル	Listener_c.cpp Listener_c.h Moderator_c.cpp Moderator_c.h	OMG IDL ファイルで指定した各インターフェイス用のクライアント・スタブが含まれます。クライアント・スタブは、オブジェクトに対する要求の送信に使用されます。

表 2-3 idl ファイルにより生成されるファイル

ファイル	idl コマンドで作成される Chat Room サンプル・アプリケーションのファイル	説明
インプリメンテーション・ファイル	`Listener_i.cpp` `Moderator_i.cpp`	OMG IDL ファイルで指定した `Listener`、`Moderator`、および `ModeratorFactory` インターフェイスのオペレーションをインプリメントするメソッドのシグニチャが含まれます。`Listener_i.h` ファイルには、`POA_ChatClient::Listener` クラスから継承されたインプリメンテーション・ファイルが含まれます。

スケルトン・ファイル	`Listener_s.cpp`
	`Listener_s.h`
	`Moderator_s.cpp`
	`Moderator_s.h`
		OMG IDL ファイルで指定した各インターフェイス用のスケルトンが含まれます。実行時に、スケルトンはクライアント要求をサーバ・アプリケーション内の適切なオペレーションにマッピングします。`Listener_s.h` ファイルには、`POA_skeleton` クラス定義 (`POA_ChatClient::Listener` など) が含まれます。

ステップ 3: 各オブジェクトのオペレーションをインプリメントするメソッドの記述

各 OMG IDL ファイルをコンパイルしたら、各オブジェクトのオペレーションをインプリメントするメソッドを記述する必要があります。共同クライアント / サーバ・アプリケーションでは、コールバック・オブジェクト (つまり Listener オブジェクト) のインプリメンテーション・ファイルを記述します。コールバック・オブジェクトのインプリメンテーションは、ほかの

CORBA オブジェクトのインプリメンテーションと同じように記述しますが、TP フレームワークの代わりに POA を使用する点が異なります。また、BEA Tuxedo サーバ・アプリケーションの BEA Tuxedo オブジェクト (Moderator および ModeratorFactory オブジェクト) のインプリメンテーション・ファイルも記述します。

インプリメンテーション・ファイルには、以下のものが含まれます。

- OMG IDL ファイルで指定された各オペレーションのメソッド宣言
- アプリケーションのビジネス・ロジック
- 各インターフェイスのインプリメンテーションのコンストラクタ (インプリメントは任意)
- オプションとして、BEA Tuxedo オブジェクト用の `com.beasys.Tobj_Servant.activate_object` メソッドおよび `com.beasys.Tobj_Servant.deactivate_object` メソッド

  `activate_object` メソッドおよび `deactivate_object` メソッド内では、オブジェクトの活性化または不活性化に関連する特定の手順を実行するコードを記述します。

リスト 2-3 は Listener オブジェクトのインプリメンテーション・ファイルを、リスト 2-4 は Moderator および ModeratorFactory オブジェクトのインプリメンテーション・ファイルを示します。

注記 Moderator および ModeratorFactory オブジェクトのインプリメンテーション・ファイルには、メソッドとデータがさらに追加されています。インプリメンテーション・ファイルのテンプレートは、`idl -i` コマンドによって作成されています。

コード リスト 2-3 Listener **オブジェクトのインプリメンテーション・ファイル**

```
// このモジュールにはインプリメンテーション・クラス Listener_i の定義が
// 含まれる

#ifndef _Listener_i_h
#define _Listener_i_h
```

```
#include "ChatClient_s.h"

class Listener_i : public POA_ChatClient::Listener {
       public:

              Listener_i ();
              virtual ~Listener_i();

              void post (
                     const char * from,
                     const char * output_line);
...
};

#endif
```

コード リスト 2-4　Moderator **および** ModeratorFactory **オブジェクトのインプリメンテーション・ファイル**

```
// このモジュールにはインプリメンテーション・クラス Moderator および
ModeratorFactory の定義が
// 含まれる

#ifndef _Moderator_i_h
#define _Moderator_i_h

#include "ChatRoom_s.h"

const int CHATTER_LIMIT = 5;
// チャット参加者の最多許容人数

class Moderator_i : public POA_ChatRoom::Moderator {
       public:

       // オペレーションを定義
       void signon ( const char*      who,
                     ChatClient::Listener_ptr   callback_ref);
       void send ( const char *       who,
                   const char *       input_line);

       void signoff ( const char * who);

       // フレームワーク関数を定義
       virtual void activate_object  ( const char* stroid );
```

```
        virtual void deactivate_object( const char* stroid,
                                        TobjS::DeactivateReasonValue
                                        reason);
        private:

        // リスト内の名前を見つける関数を定義
        int find( const char * handle );

        // モデレータによって監督されるチャット・ルームの名前を定義
        char* m_chatroom_name;

        // リスト Chatter[n] id を維持するための
        // データ
        CORBA::String            chatters[CHATTER_LIMIT];

        // Chatter[n] のコールバック・リファレンス
        ChatClient::Listener_var callbacks[CHATTER_LIMIT];
};

class ModeratorFactory_i : public POA_ChatRoom::ModeratorFactory {
        public:
         ChatRoom::Moderator_ptr get_moderator ( const char*
                                                 chatroom_name );
};
#endif
```

ステップ 4: 共同クライアント / サーバ・アプリケーションのクライアント部分の記述

共同クライアント / サーバ・アプリケーションの開発では、そのクライアント部分を BEA Tuxedo クライアント・アプリケーションと同じように記述します。クライアント・アプリケーションには、次の処理を行うコードを含める必要があります。

1. ORB を初期化します。BEA Tuxedo システムは、適正なプロトコルを使用して ORB を活性化します。この場合、該当するプロトコルは IIOP です。
2. Bootstrap オブジェクトを使用して、BEA Tuxedo のドメインとの通信を確立します。
3. FactoryFinder オブジェクトへの初期リファレンスを解決します。
4. ファクトリを使用して、目的の BEA Tuxedo オブジェクト (つまり Moderator オブジェクト) のオブジェクト・リファレンスを取得します。

注記　BEA Tuxedo 製品のうち、CORBA 環境のリリース 8.0 には、BEA WebLogic Enterprise の旧リリースで提供されていた BEA クライアント環境オブジェクトが引き続き含まれており、BEA Tuxedo 8.0 CORBA クライアントでも使用できます。BEA Tuxedo 8.0 クライアントは、これらの環境オブジェクトを使用して、ブートストラップ処理、セキュリティ、およびトランザクション・オブジェクトへの初期リファレンスを解決する必要があります。このリリースでは、OMG インターオペラブル・ネーミング・サービス（INS）を使用してブートストラップ処理、セキュリティ、およびトランザクション・オブジェクトへの初期リファレンスを解決するためのサポートが追加されています。INS の詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』の「第 4 章 CORBA ブートストラップ処理のプログラミング・リファレンス」を参照してください。

クライアントの開発手順を、リスト 2-5 で説明します。これは Chat Room サンプル・アプリケーションのコードの一部です。Chat Room サンプル・アプリケーションでは、共同クライアント / サーバ・アプリケーションのクライアント部分がファクトリを使用して Moderator オブジェクトへのオブジェクト・リファレンスをを取得し、次に Moderator オブジェクトの `signon`、`send`、および `signoff` メソッドを呼び出します。

コード リスト 2-5　Chat Room **共同クライアント / サーバ・アプリケーションのクライアント部分**

```
...
```

```
// ORB の初期化

orb_ptr = CORBA::ORB_init(argc, argv, "BEA_IIOP");

// Bootstrap オブジェクトを作成して、
// ドメインとの通信を確立

bootstrap = new Tobj_Bootstrap(orb_ptr,"");

// FactoryFinder オブジェクトを取得し、それを使用して Moderator ファクトリ
を検索し、
// Moderator を取得

// Bootstrap オブジェクトを使用して FactoryFinder オブジェクトを見つける

CORBA::Object_var var_factory_finder_oref =
           bootstrap->resolve_initial_references("FactoryFinder");

// FactoryFinder オブジェクトをナロー変換

Tobj::FactoryFinder_var var_factory_finder =
       Tobj::FactoryFinder::_narrow(var_factory_finder_oref.in());

// FactoryFinder オブジェクトを使用して Moderator のファクトリを見つける

CORBA::Object_var var_moderator_factory_oref =
       var_factory_finder->find_one_factory_by_id(
       "ModeratorFactory" );

// ModatatorFactory をナロー変換

ChatRoom::ModeratorFactory_var var_moderator_factory =
       ChatRoom::ModeratorFactory::_narrow(
       var_moderator_factory_oref.in() );

// Moderator を取得
// チャット・ルーム名がコマンド行パラメータとして渡される

var_moderator_oref =
            var_moderator_factory->get_moderator
            (var_chat_room_name.in() );
...
```

ステップ 5: コールバック・ラッパー・オブジェクトを使用したコールバック・オブジェクトの作成

コールバック・オブジェクトの基本的な作成手順は常に同じであるため、BEA Tuxedo 製品ではコールバック・オブジェクトの開発を簡略化するコールバック・ラッパー・オブジェクトを提供しています。

コールバック・ラッパー・オブジェクトは、以下のことを行います。

- コールバック・オブジェクトのオブジェクト方針を定義します。以下のオブジェクト方針がサポートされています。
  - 一時 / システム ID (`_transient`)
  - 永続 / システム ID (`_persistent/systemid`)
  - 永続 / ユーザ ID (`_persistent/userid`)

  コールバック・オブジェクトに対するオブジェクト方針の完全な説明については、第 1 章の 7 ページ「コールバック・オブジェクトのオブジェクト方針」を参照してください。
- コールバック・オブジェクトのサーバントを作成します。
- ORB および POA を、コールバック・オブジェクトに対する要求を受け付ける状態に設定します。
- 活性化されたコールバック・オブジェクトにオブジェクト・リファレンスを返します。オブジェクト ID はシステムで生成することも、ユーザが指定することもできます。
- 単一のサーバントまたはすべての活性化されたサーバントに対する要求の受け付けを停止するよう ORB に指示します。

コールバック・ラッパー・オブジェクトおよびそのメソッドの完全な説明については、『BEA Tuxedo CORBA プログラミング・リファレンス』を参照してください。

リスト 2-6 は、Chat Room サンプル・アプリケーションでのコールバック・ラッパー・オブジェクトの使用方法を示します。

コード リスト 2-6 Chat Room サンプル・アプリケーションでのコールバック・ラッパー・オブジェクトの使用方法

```
...

// コールバック・オブジェクトを使用して Listener オブジェクトのサーバントを
// 作成し、Listener オブジェクトを活性化し、Listener オブジェクトに
// 対するオブジェクト・リファレンスを作成する

BEAWrapper::Callbacks* callbacks =
                        new BEAWrapper::Callbacks( orb_ptr );
Listener_i * listener_callback_servant = new Listener_i();
CORBA::Object_var v_listener_oref=callbacks->start_transient(
                                  listener_callback_servant,
                                  ChatClient::_tc_Listener->id());
ChatClient::Listener_var v_listener_callback_oref =
                                  ChatClient::Listener::_narrow(
                                  var_listener_oref.in());
...
```

ステップ 6: コールバック・オブジェクトにリファレンスを渡すことによるオブジェクトのオペレーションの呼び出し

コールバック・オブジェクトへのオブジェクト・リファレンスを取得したら、そのコールバック・オブジェクト・リファレンスを BEA Tuxedo オブジェクトのメソッドにパラメータとして渡すことができます。Chat Room サンプル・アプリケーションでは、Moderator オブジェクトは `signon` メソッドへのパラメータとして、Listener オブジェクトのオブジェクト・リファレンスを使用します。リスト リスト 2-7 では、この手順を示します。

コード リスト 2-7　signon **メソッドの呼び出し**

```
// ユーザ定義のハンドルと Listener オブジェクト ( コールバック・オブジェクト )
// へのリファレンスを使用してチャット・ルームにログインし、ログイン中の
// クライアント・アプリケーションから入力を受信する

var_moderator_reference->signon(handle,
                                var_listener_callback_oref.in() );
```

ステップ 7: コンフィギュレーション情報の指定

IIOP を使用するリモート共同クライアント / サーバ・アプリケーションを実行する場合は、次のようにコールバック・オブジェクトのオブジェクト・リファレンスにホストおよびポート番号が含まれている必要があります。

- 一時コールバック・オブジェクトの場合、任意の有効なポート番号 (TCP/IP で定義) を使用できます。この番号は、ORB によって動的に取得できます。
- 永続コールバック・オブジェクトの場合、コールバック・オブジェクトのオブジェクト・リファレンスが作成されたのと同じポートでコールバック・オブジェクトへの要求を受け付けるように、ORB を設定する必要があります。

ポート番号の指定は、動的範囲ではなくユーザ範囲から行ってください。ユーザ範囲からポート番号を割り当てることで、共同クライアント / サーバ・アプリケーションで使用するポートの不整合を回避できます。使用する共同クライアント / サーバ・アプリケーション用に特定のポートを指定するには、共同クライアント / サーバ・アプリケーションのためのプロセスを開始するコマンド行に、以下を含めます。

```
-ORBport nnn
```

ここで、*nnn* は共同クライアント / サーバ・アプリケーション内のコールバック・オブジェクトに対する呼び出しを生成およびリッスンする際に ORB によって使用されるポートの数です。

このコマンドは、共同クライアント / サーバ・アプリケーション内のコールバック・オブジェクトのオブジェクト・リファレンスを永続的なものにする場合、および共同クライアント / サーバ・アプリケーションを停止して再起動する場合に使用します。このコマンドを使用しない場合、ORB ではランダムなポートが使用されます。共同クライアント / サーバ・アプリケーションを停止してから再起動すると、その共同クライアント / サーバ・アプリケーション内のコールバック・オブジェクトへの呼び出しは失敗します。

ポート番号は、`CORBA::orb_init` メンバ関数の `argv` 引数への入力の一部です。`argv` 引数が渡されると、ORB がその情報を読み取り、そのプロセス中に作成されるすべてのオブジェクト・リファレンス用のポートを確立します。また、同じ目的にブートストラップ・オブジェクトの `register_callback_port` オペレーションを使用することもできます。

共同クライアント / サーバ・アプリケーションが同じドメイン内の BEA Tuxedo オブジェクトと通信するには、サーバ・アプリケーションのコンフィギュレーション・ファイルが必要です。このコンフィギュレーション・ファイルは、サーバ・アプリケーション開発の一環として記述します。コンフィギュレーション・ファイルのバイナリ・バージョンである `TUXCONFIG` ファイルは、共同クライアント / サーバ・アプリケーションを起動する前に存在している必要があります。`TUXCONFIG` ファイルは、`tmloadcf` コマンドを使用して作成されます。`TUXCONFIG` ファイルの作成については、『BEA Tuxedo CORBA アプリケーション入門』および『BEA Tuxedo アプリケーションの設定』を参照してください。

使用する共同クライアント / サーバ・アプリケーションで IIOP バージョン 1.0 または 1.1 が使用されている場合、管理者は IIOP サーバ・ハンドラ (ISH) に接続されていないコールバック・オブジェクトをアウトバウンド IIOP が呼び出すことを可能にする起動パラメータを指定して、IIOP サーバ・リスナ (ISL) をブートする必要があります。ISL コマンドの `-O` オプションを指定すると、アウトバウンド IIOP が有効になります。パラメータを追加すると、管理者は BEA Tuxedo アプリケーションに最適なコンフィギュレーションを得ることができます。ISL コマンドの詳細については、『BEA Tuxedo コマンド・リファレンス』を参照してください。

ステップ 8: 共同クライアント / サーバ・アプリケーションのコンパイル

共同クライアント / サーバ・アプリケーション開発の最終手順は、実行可能プログラムの生成です。これを行うには、スケルトンおよびクライアント・スタブに対してコードおよびリンクをコンパイルする必要があります。

`buildobjclient` コマンドを `-P` オプションを指定して使用し、共同クライアント / サーバ・アプリケーションの実行可能プログラムを生成します。実行可能プログラムをビルドするには、コマンドにより BEA Tuxedo オブジェクトのクライアント・スタブ、コールバック・オブジェクトのクライアント・スタブ、コールバック・オブジェクトのスケルトン、およびコールバック・オブジェクトのインプリメンテーションを、適切な POA ライブラリと組み合わせます。

注記 `buildobjclient` コマンドの `-P` オプションを使用するには、コールバック・オブジェクトのスケルトンとクライアント・スタブを作成したときに `idl` コマンドの `-P` オプションを使用しておく必要があります。

POA の使用によるコールバック・オブジェクトの作成

POA を直接使用してコールバック・オブジェクトを作成できます。POA を直接使用するのは、コールバック・ラッパー・オブジェクトからでは利用できない POA 機能およびオブジェクト方針を使用する場合です。たとえば、POA の最適化機能を使う場合は、POA を直接使用する必要があります。以下のトピックでは、POA を使用して、サポートされているオブジェクト方針を備えたコールバック・オブジェクトを作成する方法を説明します。

注記　BEA Tuxedo 製品のこのバージョンでサポートされているのは、POA インターフェイスのサブセットのみです。サポートされているインターフェイスのリストについては、『BEA Tuxedo CORBA プログラミング・リファレンス』を参照してください。

一時オブジェクト方針を備えたコールバック・オブジェクトの作成

POA を使用して一時オブジェクト方針を備えたコールバック・オブジェクトを作成するには、以下のことを行うコードを記述する必要があります。

1. POA との接続を確立します。
2. 子 POA を作成します。

   子 POA を作成する必要があるのは、ルート POA では双方向 IIOP を使用できないためです。子 POA では `LifespanPolicy (TRANSIENT)` および `IDAssignmentPolicy (SYSTEM)` のデフォルト値を使用できます。`BOTH` の `BiDirPolicy` 方針を指定する必要があります。

   IIOP バージョン 1.2 は、入力される要求と出力される要求の双方について TCP/IP 接続の再利用をサポートします。TCP/IP 接続の再利用の許可を選択するのは、ORB です。再利用を許可するには、TCP/IP 接続の再利用を許可する ORB 方針オブジェクトを作成し、方針リスト内の方針オブジェクトを ORB 初期化時に使用します。方針オブジェクトは、`CORBA::ORB::create_policy` オペレーションを使用して作成します。`CORBA::ORB::create_policy` オペレーションの詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』を参照してください。
3. コールバック・オブジェクトのサーバントを作成します。
4. サーバントでコールバック・オブジェクトに対する要求を受け付ける準備ができていることを POA に通知します。

   この手順では、共同クライアント / サーバ・アプリケーションが、オブジェクト ID を使用して POA のコールバック・オブジェクトを活性化します。
5. POA を活性化します。
6. コールバック・オブジェクトのオブジェクト・リファレンスを作成します。

7. コールバック・オブジェクトのオブジェクト・リファレンスを、BEA Tuxedo オブジェクトの 1 つのメソッドへのパラメータとして使用し、BEA Tuxedo オブジェクトに対して呼び出しを行います。

リスト 2-8 では、POA を使用して Listener オブジェクトを作成する Chat Room サンプル・アプリケーションの一部を示します。

コード リスト 2-8 POA **の使用による** Listener **オブジェクトの作成**

```
// POA との通信を確立

orb_ptr = CORBA::ORB_init(argc, argv, "BEA_IIOP");
CORBA::PolicyList policy_list(1);
CORBA::Any val;

CORBA::Object_ptr o_init_poa;
o_init_poa = orb_ptr->resolve_initial_references("RootPOA");

// ナロー変換してルート POA を取得

root_poa_ptr = PortableServer::POA::_narrow(o_init_poa);
CORBA::release(o_init_poa);


// POA について双方向の IIOP 方針を指定

val <<= BiDirPolicy::BOTH;
CORBA::Policy_ptr bidir_pol_ptr = orb_ptr->create_policy(
                    BiDirPolicy::BIDIRECTIONAL_POLICY_TYPE, val);
policy_list.length ( 1 );
policy_list[0] = bidir_pol_ptr;

// 双方向の POA を作成

bidir_poa_ptr = root_poa_ptr->create_POA("BiDirPOA",
                                         root_poa_ptr->
                                         the_POAManager(),
                                         policy_list);
// POA を活性化

root_poa_ptr->the_POAManager()->activate();

// Listener オブジェクトを作成

ChatClient::Listener_var v_listener_callback_ref;
```

```
// Listener オブジェクトのサーバントを作成して活性化

listener_callback_servant = new Listener_i();
  CORBA::Object_var           v_listener_oref;
  PortableServer::ObjectId_var temp_OId =
     bidir_poa_ptr ->activate_object(listener_callback_servant );

// システム生成のオブジェクト ID を備えた Listener オブジェクトに対する

// オブジェクト・リファレンスを作成

v_listener_oref = bidir_poa_ptr->create_reference_with_id
                                  (temp_OId,
                                  ChatClient::_tc_Listener->id() );
v_listener_callback_ref = ChatClient::Listener::_narrow
                                  ( v_listener_oref.in() );
```

永続 / ユーザ ID オブジェクト方針を備えたコールバック・オブジェクトの作成

POA を使用して永続 / ユーザ ID オブジェクト方針を備えたコールバック・オブジェクトを作成するには、以下のことを行うコードを記述する必要があります。

1. ユーザ ID を格納する文字列を使用し、それをオブジェクト ID に変換します。
2. `LifespanPolicy` が `PERSISTENT` に設定され、`IDAssignmentPolicy` が `USERID` に設定された子 POA を作成します。
3. Listener オブジェクトのサーバントを作成します。
4. Listener オブジェクトの文字列化されたオブジェクト ID とリポジトリ ID を使用して、Listener オブジェクトのオブジェクト・リファレンスを作成します。
5. Listener オブジェクトを活性化します。

注記　永続 / ユーザ ID オブジェクト方針は、リモート共同クライアント / サーバ・アプリケーション、つまり BEA Tuxedo ドメイン内には存在しない共同クライアント / サーバ・アプリケーションでのみ使用されます。

リスト 2-9 では、これらの手順を行うコードを示します。

注記　このコード例では、双方向 IIOP を使用しません。

コード リスト 2-9　永続 / ユーザ ID オブジェクト方針を備えた Listener オブジェクトのコード例

```
// 文字列を宣言してオブジェクト ID に変換
const char* oid_string = "783";
PortableServer::ObjectID_var oid=
              PortableServer::string_to_ObjectId(oid_string);

// ルート POA を見つける
CORBA::Object_var oref =
              orb_ptr->resolve_initial_references("RootPOA");
PortableServer::POA_var root_poa =
              PortableServer::POA::_narrow(oref);

// 永続 / ユーザ ID POA を作成して活性化
CORBA::PolicyList policies(2);
policies.length(2);
policies[0] = root_poa->create_lifespan_policy(
              PortableServer::PERSISTENT);
policies[1] = root_poa->create_id_assignment_policy(
              PortableServer::USER_ID );
PortableServer::POA_var poa_ref =
              root_poa->create_POA("poa_ref",
              root_poa->the_POAManager(),policies);
root_poa->the_POAManager()->activate();

// Listener オブジェクトのオブジェクト・リファレンスを作成
oref = poa_ref->create_reference_with_id(oid,
              ChatClient::_tc_Listener->id());
ChatClient::Listener_ptr Listener_oref =
              ChatClient::Listener::_narrow( oref );

// Listener_i サーバントを作成して、Listener オブジェクトを活性化
Listener_i* my_Listener_i = new Listener_i();
poa_ref->activate_object_with_id( oid, my_Listener_i);
```

```
// Listener オブジェクトにリファレンスを渡す呼び出しを行う
v_moderator_ref->signon( handle, Listener_oref);
```

永続 / システム ID オブジェクト方針を備えたコールバック・オブジェクトの作成

POA を使用して永続 / システム ID オブジェクト方針を備えたコールバック・オブジェクトを作成するには、次のことを行うコードを記述する必要があります。

1. `LifespanPolicy` が `PERSISTENT` に設定され、`IDAssignmentPolicy` がデフォルトに設定された子 POA を作成します。
2. Listener オブジェクトのサーバントを作成します。
3. システム生成のオブジェクト ID (Listener オブジェクトのリポジトリ ID) を使用して、Listener オブジェクトのオブジェクト・リファレンスを作成します。
4. Listener オブジェクトを活性化します。

注記　永続 / システム ID オブジェクト方針は、リモート共同クライアント / サーバ・アプリケーション、つまり BEA Tuxedo ドメイン内には存在しない共同クライアント / サーバ・アプリケーションでのみ使用されます。

リスト 2-10 では、これらの手順を行うコードを示します。

コード リスト 2-10　永続 / システム ID オブジェクト方針を備えた Listener オブジェクトのコード例

```
// ルート POA を見つける
CORBA::Object_var oref=
       orb_ptr->resolve_initial_references("RootPOA")
PortableServer::POA_var root_poa =
       PortableServer::POA::_narrow(oref);

// 永続 / システム ID POA を作成して活性化
CORBA::PolicyList policies(1);
policies.length(1);
policies[0] = root_poa->create_lifespan_policy(
       PortableServer::PERSISTENT);
```

```
//IDAssignmentPolicy はデフォルトのため、指定する必要はない
PortableServer::POA_var poa_ref = root_poa->create_POA(
        "poa_ref", root_poa->the_POAManager(), policies);
root_poa->the_POAManager()->activate();

// Listener_i サーバントを作成して、Listener オブジェクトを活性化
Listener_i* my_Listener_i = new Listener_i();
PortableServer::ObjectId_var temp_OId =
       poa_ref->activate_object ( my_Listener_i );

// 戻り値のシステム・オブジェクト ID を備えた Listener オブジェクトの
// オブジェクト・リファレンスを作成
oref = poa_ref->create_reference_with_id(
       temp_OId, ChatClient::_tc_Listener->id() );
ChatClient::Listener_var Listener_oref =
       ChatClient::Listener::_narrow(oref);

// Listener オブジェクトにリファレンスを渡す呼び出しを行う
v_moderator_ref->signon( handle, Listener_oref );
```

C++ 共同クライアント / サーバ・アプリケーションのスレッドに関する留意事項

共同クライアント / サーバ・アプリケーションは、まずクライアント・アプリケーションとして機能し、次にサーバ・アプリケーションとして機能するように切り替わることができます。そのために、共同クライアント / サーバ・アプリケーションでは、以下の呼び出しを行うことにより、スレッドの完全な制御を ORB に提供します。

```
orb -> run();
```

共同クライアント / サーバ・アプリケーションのサーバ部分のメソッドが `ORB::shutdown()` を呼び出すと、サーバのアクティビティはすべて停止し、共同クライアント / サーバ・アプリケーションのサーバ部分で `ORB::run()`

が呼び出されると、制御はステートメントに返されます。制御が共同クライアント / サーバ・アプリケーションのクライアント機能に返されるのは、この条件においてのみです。

クライアント・アプリケーションにはスレッドが 1 つしかないため、共同クライアント / サーバ・アプリケーションのクライアント機能とサーバ機能で、中央演算処理装置 (CPU) を共有する必要があります。この共有は、ORB をときどき確認して、共同クライアント / サーバ・アプリケーションに、実行すべきサーバ・アプリケーション作業があるかどうかを調べることによって行われます。ORB の確認を実行するには、次のコードを使用します。

```
if ( orb->work_pending() ) orb->perform_work();
```

ORB はサーバ・アプリケーションの作業を完了すると、共同クライアント / サーバ・アプリケーションに戻ります。共同クライアント / サーバ・アプリケーションはその後、クライアント・アプリケーション機能を実行します。共同クライアント / サーバ・アプリケーションでは、ORB の不定期な確認を、確実に行う必要があります。行われない場合、共同クライアント / サーバ・アプリケーションは呼び出しをまったく処理しません。

共同クライアント / サーバ・アプリケーションが要求をブロックしている間は、ORB がコールバックを扱うことはできません。共同クライアント / サーバ・アプリケーションが別の BEA Tuxedo サーバ・アプリケーションのオブジェクトを呼び出した場合、ORB は応答待機中にブロックします。ブロック中の ORB はコールバックを扱うことができないため、要求が完了するまでコールバックはキュー入れられます。

Chat Room サンプル・アプリケーションのビルドと実行

Chat Room サンプル・アプリケーションをビルドして実行するには、以下の手順に従います。

1. Chat Room サンプル・アプリケーションのファイルを作業ディレクトリにコピーします。
2. Chat Room サンプル・アプリケーションのファイルの保護属性を変更します。
3. 環境変数の設定を確認します。
4. `ChatSetup` コマンドを実行します。

以降の節では、上記の各手順について説明します。

Chat Room サンプル・アプリケーション用ファイルの作業ディレクトリへのコピー

Chat Room サンプル・アプリケーションのファイルを、ローカル・マシンの作業ディレクトリにコピーする必要があります。Chat Room サンプル・アプリケーションのファイルは、次のディレクトリに格納されています。

Windows

drive:*TUXDIR*\samples\corba\chatroom

UNIX

/usr/local/TUXDIR/samples/corba/chatroom

Chat Room サンプル・アプリケーションをビルドして実行するには、表 2-4 に示すファイルを使用します。

表 2-4 Chat Room サンプル・アプリケーションに含まれるファイル

ファイル	説明
ChatRoom.idl	Moderator および ModeratorFactory インターフェイスを宣言する、OMG IDL コード。
ChatClient.idl	Listener インターフェイスを宣言する OMG IDL コード。
Listener_i.h Listener_i.cpp	共同クライアント / サーバ・アプリケーションの Listener オブジェクトのメソッド・インプリメンテーション用 C++ ソース・コード。
Moderator_i.h Moderator_i.cpp	BEA Tuxedo サーバ・アプリケーションの Moderator および ModeratorFactory オブジェクトのメソッド・インプリメンテーション用 C++ ソース・コード。
ChatClientMain.cpp	共同クライアント / サーバ・アプリケーションの C++ ソース・コード。

表 2-4 Chat Room サンプル・アプリケーションに含まれるファイル (続き)

ファイル	説明
ChatRoomServer.cpp	BEA Tuxedo サーバ・アプリケーションの C++ ソース・コード。
KeyboardManager.h KeyboardManager.cpp	Chat Room サンプル・アプリケーションでキーボードからの入力を処理する C++ ソース・コード。このコードは、ChatClientMain.cpp によって使用されます。
ChatRoom.icf	Chat Room サンプル・アプリケーションにおける BEA Tuxedo サーバ・アプリケーションの Moderator および ModeratorFactory オブジェクトのインプリメンテーション・コンフィギュレーション・ファイル (ICF)。
ChatRoom.ksh	UNIX システムで、環境変数を設定して Chat Room サンプル・アプリケーションをビルドするスクリプト。
ChatRoom.cmd	Windows システムで、環境変数を設定して Chat Room サンプル・アプリケーションをビルドするコマンド・プロシージャ。
ChatRoom.mk	Chat Room サンプル・アプリケーション用の、UNIX オペレーティング・システムの makefile。
ChatRoom.nt	Chat Room サンプル・アプリケーション用の、Windows オペレーティング・システムの makefile。
Readme.txt	Chat Room サンプル・アプリケーションのビルドと実行に関する最新情報を示したファイル。

Chat Room サンプル・アプリケーションのファイルに対する保護属性の変更

BEA Tuxedo ソフトウェアのインストール中、サンプル・アプリケーション・ファイルは読み取り専用としてマークされます。Chat Room サンプル・アプリケーションのファイルを編集またはビルドするには、作業ディレクトリにコピーしたファイルの保護属性を次のように変更しておく必要があります。

Windows

```
prompt> attrib /S -r drive:\workdirectory\*.*
```

UNIX

```
prompt> /bin/ksh

ksh prompt> chmod u+w /workdirectory/*.*
```

UNIX オペレーティング・システムのプラットフォームでは、次のように ChatRoom.ksh のパーミッションも変更して、ファイルの実行許可を付与する必要があります。

```
ksh prompt> chmod +x ChatRoom.ksh
```

TUXDIR 環境変数の設定の確認

Chat Room サンプル・アプリケーションをビルドして実行する前に、システムで TUXDIR 環境変数が設定されていることを確認する必要があります。ほとんどの場合、この環境変数はインストール手順の一環として設定済みです。TUXDIR 環境変数は、BEA Tuxedo ソフトウェアをインストールしたディレクトリ・パスを定義します。たとえば、次のように入力します。

Windows

```
TUXDIR=C:\TUXDIR
```

UNIX

```
TUXDIR=/usr/local/TUXDIR
```

インストール中に定義された環境変数の情報が正しいことを確認するには、以下の手順に従います。

Windows

1. [スタート] メニューの、[設定] をポイントします。
2. [設定] メニューから、[コントロール パネル] をクリックします。

   [コントロール パネル] が表示されます。
3. [システム] アイコンをクリックします。

   [システムのプロパティ] ウィンドウが表示されます。
4. [詳細] タブの [環境変数] をクリックします。

   [環境変数] ページが表示されます。
5. `TUXDIR` の設定を確認します。

UNIX

```
ksh prompt>printenv TUXDIR
```

設定を変更するには、以下の手順に従います。

Windows

1. [システムのプロパティ] ウィンドウの [環境] ページで、`TUXDIR` 環境変数をクリックします。
2. 値フィールドに環境変数の正しい情報を入力します。
3. [OK] をクリックして変更を保存します。

UNIX

```
ksh prompt>export TUXDIR=directorypath
```

ChatSetup コマンドの実行

`ChatSetup` コマンドを使用すると、以下の手順を自動化できます。

1. システム環境変数を設定します。
2. コンフィギュレーション・ファイルを作成およびロードします。
3. クライアント・アプリケーションのコードをコンパイルします。
4. サーバ・アプリケーションのコードをコンパイルします。

ChatSetup コマンドを実行する前に、次のことを確認する必要があります。

- アプリケーションのビルドおよび実行を行う適切な管理者権限があること。
- 使用しているマシンのパスに nmake が存在すること。
- UNIX の場合は、PATH 変数に make 実行可能プログラムが含まれていること。

サンプル・アプリケーションをビルドして実行するには、次のように ChatSetup コマンドを入力します。

Windows

```
prompt>cd workdirectory
prompt> ChatSetup.cmd
```

UNIX

```
ksh prompt> cd workdirectory
ksh prompt> ./ChatSetup.ksh
```

サーバ・アプリケーションの起動

次のコマンドを入力して、Chat Room サンプル・アプリケーションにおけるサーバ・アプリケーションとシステム・サーバのプロセスを開始します。

```
prompt> tmboot -y
```

このコマンドを入力すると、次のサーバ・プロセスが開始されます。

- TMSYSEVT

システムの EventBroker です。このサーバ・プロセスを使用するのは、BEA Tuxedo システムのみです。

- TMFFNAME

  次の 3 つの TMFFNAME サーバ・プロセスが開始されます。

  - -N および -M オプションを指定して開始された TMFFNAME サーバ・プロセスは、マスタ NameManager サービスです。NameManager サービスは、アプリケーションによって指定された名前のオブジェクト・リファレンスへのマッピングを維持します。このサーバ・プロセスを使用するのは、BEA Tuxedo システムのみです。
  - -N オプションのみを指定して開始された TMFFNAME サーバ・プロセスは、スレーブ NameManager サービスです。
  - -F オプションを指定して開始された TMFFNAME サーバ・プロセスには、FactoryFinder オブジェクトが含まれます。

- ChatRoom

  Chat Room サンプル・アプリケーションのためのサーバ・アプリケーション・プロセスです。

- ISL

  IIOP リスナ / ハンドラ・プロセス。

クライアント・アプリケーションの起動

次のコマンドを入力して、Chat Room サンプル・アプリケーションのクライアント・アプリケーションを起動します。

```
prompt> ChatClient  chatroom_name -ORBport nnn
```

ここで、chatroom_name は接続するチャット・ルームの名前です。任意の値を入力できます。自分自身を識別するためのハンドルを入力するよう求められます。任意の値を入力できます。選択したハンドルが使用中の場合は、別のハンドルの入力を求められます。

Chat Room サンプル・アプリケーションを使いやすいように最適化するには、同じチャット・ルーム名を使用する第 2 のクライアント・アプリケーションを実行します。

クライアント・アプリケーションを終了するには、「\」と入力します。

Chat Room サンプル・アプリケーションの停止

別のサンプル・アプリケーションを使用する前に、次のコマンドを入力して Chat Room サンプル・アプリケーションを停止し、作業ディレクトリから不要なファイルを削除します。

Windows

```
prompt> tmshutdown -y
prompt> Admin\setenv
prompt> nmake -f ChatRoom.nt superclean
prompt> nmake -f ChatRoom.nt adminclean
```

UNIX

```
ksh prompt> tmshutdown -y
ksh prompt> . ./Admin/setenv.ksh
ksh prompt> make -f ChatRoom.mk superclean
ksh prompt> make -f ChatRoom.nt adminclean
```

章

3 Java 共同クライアント / サーバ・アプリケーション

ここでは、以下の内容について説明します。

- 開発プロセス
- 共同クライアント / サーバ・アプリケーションのサポート

開発プロセス

表 3-1 では、Java 共同クライアント / サーバ・アプリケーションの開発プロセスの概略を示します。

表 3-1 Java 共同クライアント / サーバ・アプリケーションの開発プロセス

手順	説明
1	BEA Tuxedo アプリケーションで使用するコールバック・インターフェイスおよび CORBA インターフェイスの OMG IDL を記述します。

手順	説明
2	スケルトンおよびクライアント・スタブを生成します。
3	各インターフェイスのオペレーションをインプリメントするメソッドを記述します。
4	ORB を初期化します。
5	共同クライアント / サーバ・アプリケーションのクライアント主要部分を記述します。
6	コールバック・ラッパー・オブジェクトを使用してコールバック・オブジェクトを作成します。
7	ISH との通信を確立します。
8	コールバック・オブジェクトのオブジェクト・リファレンスを渡すことにより、BEA Tuxedo オブジェクトのオペレーションを呼び出します。
9	コンフィギュレーション情報を指定します。
10	共同クライアント / サーバ・アプリケーションをコンパイルします。

共同クライアント / サーバ・アプリケーション内のコールバック・オブジェクトはトランザクションに関与せず、オブジェクト管理機能を持たないため、このオブジェクト用のサーバ記述ファイル (*filename*.xml) を作成する必要はありません。ただし、BEA Tuxedo アプリケーション内の BEA Tuxedo オブジェクト用サーバ記述ファイルは、作成する必要があります。

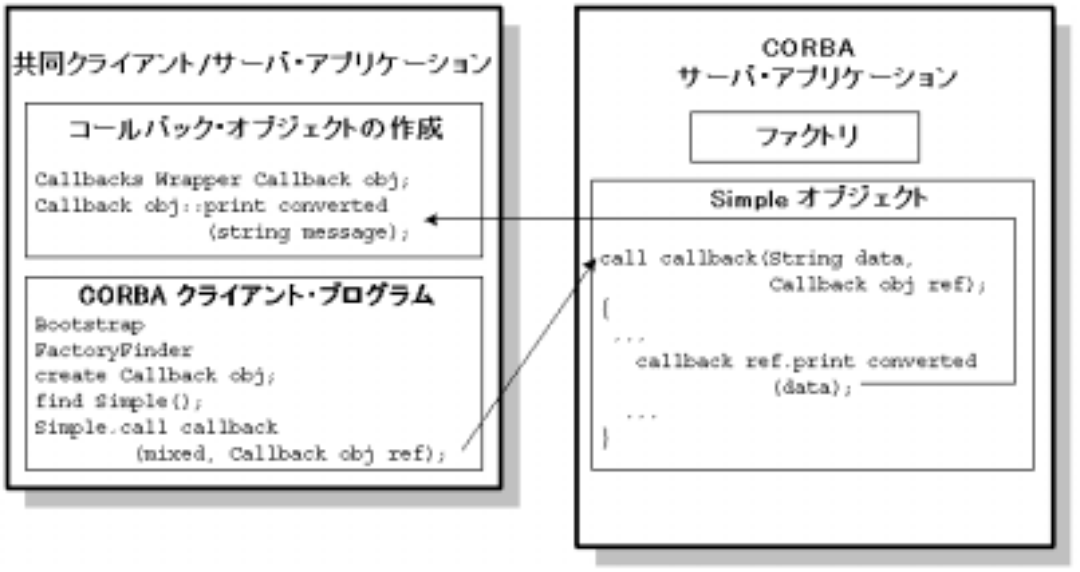
共同クライアント / サーバ・アプリケーションのサポート

BEA Tuxedo CORBA は、Java によるクライアントおよび共同クライアント / サーバをサポートしています。

注記　BEA Tuxedo 製品のうち CORBA 環境のリリース 8.0 は、Java サーバをサポートしていません。BEA WebLogic Enterprise 製品のバージョン 5.0 および 5.1 には、Java サーバのサポートが含まれていました。このサポートは、BEA WebLogic Enterprise が BEA Tuxedo のリリース 8.0 に統合された際に中止されました。

共同クライアント / サーバのインプリメンテーションの 1 つで、コールバック・オブジェクトが使用されています。図 3-1 は、コールバック・オブジェクトを使用する共同クライアント / サーバ・アプリケーションの概念を示します。

図 3-1 共同クライアント / サーバ・アプリケーションの概念



共同クライアント / サーバ・アプリケーションの完全な例については、『BEA Tuxedo CORBA ノーティフィケーション・サービス』の「第 6 章 Advanced サンプル・アプリケーションのビルド」を参照してください。Advanced サンプル・アプリケーションのサブスクライバ・コンポーネントは、共同クライアント / サーバ・アプリケーションをインプリメントします。Java Advanced サンプル・アプリケーションは、BEA Simple Events API を使用してインプリメントされます。

索引

B

C

I

J

ほ

り