BEA Tuxedo

BEA Tuxedo CORBA
要求レベルのインターセプタ

BEA Tuxedo 8.0
8.0 版
2001 年 10 月 31 日

BEA Tuxedo CORBA 要求レベルのインターセプタ

Document Edition	Date	Software Version
8.0	2001 年 10 月 31 日	BEA Tuxedo 8.0

目次

このマニュアルについて

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

1. CORBA 要求レベルのインターセプタの概要

インターセプタのアーキテクチャ 1-2
機能と制限事項 1-4
実行フロー 1-5
クライアント側での実行 1-6
クライアント側での例外処理 1-8
ターゲット側での実行 1-10
ターゲット側での例外処理 1-12
exception_occurred メソッド 1-14
短絡動作について 1-15
複数の要求レベルのインターセプタの使用方法 1-15
複数のクライアント側インターセプタ 1-17
複数のターゲット側インターセプタ 1-17
インターセプタおよびメタ・オペレーション 1-18

2. CORBA 要求レベルのインターセプタの開発

ステップ 1: CORBA アプリケーションのインターフェイスの識別 2-1
ステップ 2: インターセプタ・インプリメンテーション・コードの記述 2-3
インプリメンテーション・ファイルの起動 2-3
実行時のインターセプタの初期化 2-4
要求からのインターセプタ名の取得 2-5
要求内のオペレーションの識別 2-6
インターセプタの応答オペレーションのインプリメント 2-7

データ入力ストリームからのパラメータの読み出し 2-8
例外 2-8
ステップ 3: インターセプタ・ヘッダ・ファイルの作成 2-9
ステップ 4: インターセプタのビルド 2-11
ステップ 5: インターセプタのテスト 2-11

3. CORBA 要求レベルのインターセプタのデプロイ

インターセプタの登録 3-1
インターセプタの登録解除 3-2
インターセプタの呼び出し順の変更 3-2

4. PersonQuery サンプル・アプリケーション

PersonQuery サンプル・アプリケーションのしくみ 4-1
PersonQuery データベース 4-2
クライアント・アプリケーションのコマンド行インターフェイス 4-3
PersonQuery サンプル・アプリケーションの OMG IDL 4-5
PersonQuery サンプル・アプリケーションのビルドと実行 4-8
PersonQuery サンプル・アプリケーション用ファイルのコピー 4-9
PersonQuery アプリケーション・ファイルに対する保護の変更 4-13
環境変数の設定 4-13
CORBA クライアントおよびサーバ・アプリケーションのビルド 4-14
PersonQuery クライアントおよびサーバ・アプリケーションの起動 4-14
PersonQuery サンプル・アプリケーションの実行 4-14
PersonQuery サンプル・アプリケーションの停止 4-15

5. InterceptorSimp サンプル・インターセプタ

PersonQuery サンプル・インターセプタのしくみ 5-1
PersonQuery インターセプタの登録および実行 5-2
インターセプタの出力の検証 5-3
インターセプタの登録解除 5-4
インターセプタの登録解除 5-4

6. InterceptorSec サンプル・インターセプタ

PersonQuery サンプル・インターセプタのしくみ 6-1
InterceptorSec ターゲット側インターセプタのしくみ 6-2
SecurityCurrent オブジェクトの使用 6-3

SecurityCurrent オブジェクトの取得 6-3
ユーザ属性リストの作成 6-4
PersonQuery インターセプタの登録および実行 6-7
インターセプタ出力の検証 6-8
インターセプタの登録解除 6-8

7. InterceptorData サンプル・インターセプタ

InterceptorDataClient インターセプタ 7-1
InterceptorDataTarget インターセプタ 7-2
InterceptorData インターセプタのインプリメント 7-3
InterceptorData インターセプタの登録および実行 7-4
インターセプタ出力の検証 7-5
インターセプタの登録解除 7-7

8. 要求レベルのインターセプタの API

インターセプタの階層構造 8-2
未使用インターフェイスについての注意事項 8-2
Interceptors::Interceptor インターフェイス 8-3
RequestLevelInterceptor::
RequestInterceptor インターフェイス 8-9
RequestLevelInterceptor::
ClientRequestInterceptor インターフェイス 8-19
RequestLevelInterceptor::
TargetRequestInterceptor インターフェイス 8-27
CORBA::DataInputStream インターフェイス 8-36

A. 要求レベルのインターセプタのスタータ・ファイル

スタータ・インプリメンテーション・コード A-1
スタータ・ヘッダ・ファイル・コード A-11

索引

このマニュアルについて

このマニュアルでは、プログラマが、BEA Tuxedo 製品の CORBA 環境に要求レベルのインターセプタをインプリメントする方法について説明します。CORBA 要求レベルのインターセプタは、BEA Tuxedo システムの高度なプログラミング機能です。

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 要求レベルのインターセプタの概要」では、要求レベルのインターセプタの概要、および BEA Tuxedo 製品の CORBA 環境における実行方法について説明します。
- 「第 2 章 CORBA 要求レベルのインターセプタの開発」では、C++ 要求レベルのインターセプタをインプリメントする手順について説明します。
- 「第 3 章 CORBA 要求レベルのインターセプタのデプロイ」では、インターセプタの登録および登録解除を行う管理コマンドについて説明します。
- 「第 4 章 PersonQuery サンプル・アプリケーション」では、PersonQuery サンプル・アプリケーションについて説明します。このサンプル・アプリケーションは、サンプル・インターセプタと共に使用される基本のアプリケーションで、BEA Tuxedo ソフトウェアに同梱されています。
- 「第 5 章 InterceptorSimp サンプル・インターセプタ」では、InterceptorSimp サンプル・インターセプタについて説明します。このサンプル・インターセプタは、PersonQuery クライアントとサーバ・アプリケーションとの間でやり取りされる要求に関する単純なデータを収集します。
- 「第 6 章 InterceptorSec サンプル・インターセプタ」では、基本のセキュリティ・インターセプタである InterceptorSec サンプル・インターセプタについて説明します。
- 「第 7 章 InterceptorData サンプル・インターセプタ」では、PersonQuery サンプル・アプリケーションに固有の 2 つのサンプル・インターセプタについて説明します。
- 「第 8 章 要求レベルのインターセプタの API」では、C++ 用の要求レベルのインターセプタの API について説明します。

- 「付録 A 要求レベルのインターセプタのスタータ・ファイル」では、C++ 要求レベルのインターセプタのインプリメントを開始する際に使用できるコードを示します。

対象読者

このマニュアルは、安全でスケーラブルなトランザクション・ベースのサーバ・アプリケーションを作成するプログラマを対象としています。CORBA および C++ プログラミング言語に読者が精通していることを前提として書かれています。

e-docs Web サイト

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

マニュアルの印刷方法

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

このマニュアルの PDF 版は、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。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 社の Web サイト (http://www.adobe.co.jp/) から無償でダウンロードできます。

関連情報

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

サポート情報

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

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

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

カスタマ・サポートへお問い合わせの際には、以下の情報をご用意ください。

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

表記上の規則

このマニュアルでは、以下の表記規則が使用されています。

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

等幅テキスト	コード・サンプル、コマンドとオプション、データ構造とメンバ、データ型、ディレクトリ、およびファイル名と拡張子を示します。また、キーボードから入力する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。

1 CORBA 要求レベルのインターセプタの概要

要求レベルのインターセプタとは、CORBA アプリケーションのクライアント・コンポーネントとサーバ・コンポーネントの間の呼び出しパスに、セキュリティ・コンポーネントや監視コンポーネントなどの機能を挿入する手段となる、ユーザ記述の CORBA オブジェクトです。特定のマシン上でオブジェクト・リクエスト・ブローカ (ORB) にインストールおよび登録されたインターセプタは、そのマシン上のすべての CORBA アプリケーションに関与します。インターセプタを使用すると、任意の追加機能をクライアント側、サーバ側、または両方でオブジェクト呼び出しの呼び出しパスに挿入できます。

要求レベルのインターセプタは通常、代表的な CORBA 環境の一部ではありません。これらのインプリメントは、高度なプログラミング・タスクと見なされています。

BEA Tuxedo システムの CORBA 環境でサポートされるインターセプタは、2 種類のカテゴリに分類されます。

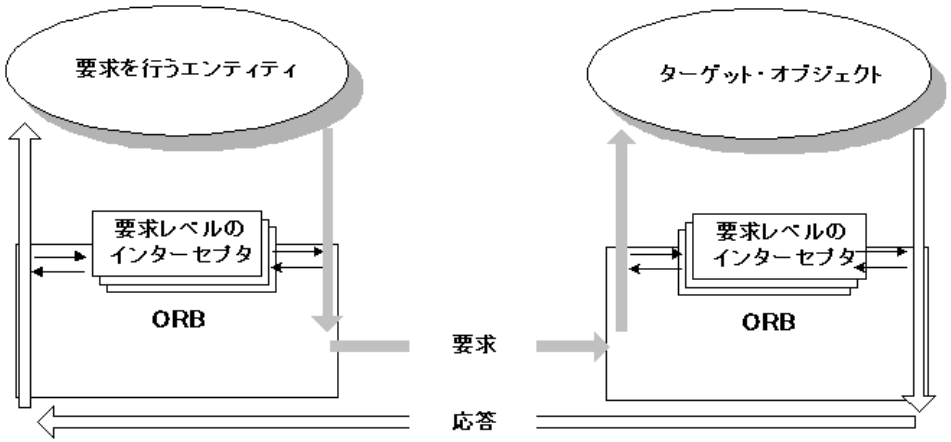
- 1 つはクライアント側インターセプタです。これは呼び出しのクライアント側で ORB によって呼び出され、要求を行うエンティティのプロセスで実行されます。クライアント側インターセプタは、`ClientRequestInterceptor` クラスから継承されます。
- もう 1 つは、ターゲット側インターセプタです。これは呼び出しのターゲット側で ORB によって呼び出され、ターゲット・アプリケーション・プロセスで実行されます。呼び出しのターゲットは、CORBA サーバ・アプリケーションでも CORBA 共同クライアント / サーバ・アプリケーションでもかまいません。ターゲット側インターセプタは、`TargetRequestInterceptor` クラスから継承されます。

CORBA 環境の BEA Tuxedo システムは、クライアント・オブジェクトおよびターゲット・オブジェクトの相対的な場所を基準として、インターセプタをインストールして使用できる場所に関する自由度は非常に高くなっています。クライアント・アプリケーションからは、要求のターゲットが同じプロセス内にあるのか別のプロセス内にあるのかは、見えません。

クライアント側インターセプタとターゲット側インターセプタは別個のインターフェイスから継承されていますが、両者を単一のソース・ファイル内にインプリメントすると都合のよい場合が多くあります。

インターセプタのアーキテクチャ

次の図は、要求レベルのインターセプタと BEA Tuxedo システムの関係を示します。



CORBA インターセプタの BEA Tuxedo インプリメンテーションについては、次の事項に留意してください。

- インターセプタは管理者によって登録され、アプリケーションの実行中の適切な時点で ORB によって呼び出されます。
- クライアント側インターセプタが ORB にインストールおよび登録されると、そのインターセプタはマシン上のどの CORBA クライアント・アプリケーションから要求を受信しても、毎回呼び出されます。

  呼び出しの正常な要求応答サイクル 1 回の間に、クライアント側インターセプタは ORB によって 2 回呼び出されます。

  a. 要求が最初にクライアント・アプリケーションから発行され、ORB に到達した時点 (`client_invoke` オペレーション)

b. ターゲット応答が返されてクライアント・アプリケーションのプロセスに到達した時点 (client_response オペレーション)

- ターゲット側インターセプタが ORB にインストールおよび登録されると、そのインターセプタはマシン上のどのターゲット・オブジェクトに要求が到達しても、毎回呼び出されます。

  呼び出しの要求応答サイクル 1 回の間に、ターゲット側インターセプタは ORB によって 2 回呼び出されます。

  a. クライアント要求が最初に ORB に到達した時点 (target_invoke オペレーション)

  b. ターゲット・オブジェクト応答が ORB に到達した時点 (target_response オペレーション)

- 複数のクライアント側インターセプタまたはターゲット側インターセプタを ORB にインストールおよび登録できます。
- インターセプタは、各々独立しており、ほかのインターセプタが存在するかどうかを認識している必要はありません。
- インターセプタは、まったくターゲット・オブジェクトと関係なく、直接クライアントに応答を返すことによって、呼び出しを短絡できます。
- インターセプタを使用すると、各要求が実行されるたびに手順が追加されるため、アプリケーションの全体的なパフォーマンスに影響が生じます。

ORB は、登録されたインターセプタのリストを維持します。インターセプタの登録は、管理タスクとして行う操作です。複数のインターセプタをインストールおよび作成できるので、アプリケーション実行時に、ORB はこのリストを使用して、いつ、どの順番でインターセプタを呼び出すべきかを決定します。複数のインターセプタを登録した場合、ORB は各インターセプタを連続的に実行します。複数のインターセプタの呼び出しの順序を設定するのも、管理タスクの 1 つです。

機能と制限事項

要求レベルのインターセプタは、以下のような数種類のサービス・アプリケーションをインプリメントする際に、特に有用です。

- 統計情報を収集するためのインスツルメンテーション・ポイント
- 監視機能またはトレース機能を含むプローブ・ポイント
- ある特定タイプの呼び出しを許可するかどうか、または特定の情報ビットをクライアント・アプリケーションに返せるかどうかを決定する、セキュリティ・チェック。インターセプタとセキュリティの詳細については、「第 6 章 InterceptorSec サンプル・インターセプタ」を参照してください。

CORBA インターセプタに対する現在の制限事項は、以下のとおりです。

- インターセプタは ORB によってのみ呼び出されます。CORBA クライアントもサーバ・アプリケーションもインターセプタを直接呼び出すことはできません。
- 特定のプログラミング言語でインプリメントされたインターセプタは、同じ言語でインプリメントされたエンティティからのみ、呼び出しをインターセプトできます。
- インターセプタは `DataInputStream` オブジェクトへの書き込みはできません。
- インターセプタは、サービス・コンテキスト・オブジェクトを渡したり操作したりできません。
- インターセプタは、トランザクション・カレント・オブジェクトを渡したり操作したりできません。
- インターセプタは、`Tobj_Bootstrap` オブジェクトのメソッドを呼び出すことはできません。
- `REPLY_NO_EXCEPTION` の戻りステータス値はサポートされていません。ただしインターセプタ・クラスのメソッド・シグニチャ・オペレーションでは使用されます。
- インターセプタはほかのオブジェクトに呼び出しを行うことができますが、これらの呼び出しもインターセプトの対象となります。インターセプタがオブジェクトを呼び出す際には、そのインターセプタが自身の呼び出しをインターセプトして無限ループを生じることがないように確認してください。こ

の現象は、呼び出されるオブジェクトがインターセプタと同じサーバ・プロセス内にある場合に起こります。このような状況下では、システムがハングする可能性があります。

- `RequestLevelInterceptor` インターフェイスから派生したクラスのオペレーションのためのメソッド・シグニチャには、次のインターフェイスのパラメータが含まれます。
  - `RequestLevelInterceptor::DataOutputStream`
  - `RequestLevelInterceptor::ServiceContextList`

  これらのインターフェイスは BEA Tuxedo 製品では使用しません。これらのインターフェイスが BEA Tuxedo ソフトウェアで定義されているのは、BEA Tuxedo 製品の将来のリリースでこれらのインターフェイスのインプリメンテーションが提供された場合に、CORBA アプリケーションを再コンパイルする必要性をなくすためです。ORB は常に実際の引数に対する nil オブジェクトを渡します。これらの引数を使用しないでください。プロセスが深刻なエラーにより停止することが予想されます。

実行フロー

以下の節では、インターセプタを使用する CORBA アプリケーションの実行中に行われる処理を説明します。一般に、要求レベルのインターセプタのインスタンス化と初期化は、ORB の初期化時にのみ行われます。それ以外では、要求レベルのインターセプタをインスタンス化できません。

インターセプタの戻りステータスにより、ORB 実行時およびインストールされている可能性があるほかの要求レベルのインターセプタの実行フローが制御されます。

呼び出された後のインターセプタの戻りステータスに応じて、次のイベントのうち 1 つが発生することがあります。

- 呼び出しは、ターゲット・オブジェクトへの通常のパス、クライアント・アプリケーションに戻るパス、または別のインターセプタへのパスを再開します。
- クライアント側またはサーバ側のインターセプタはクライアント要求を処理し、例外をクライアントに返します (この場合、要求はターゲット・オブジェクトへはまったく送信されないか、またはインターセプタにより例外と

置換される応答をターゲット・オブジェクトが提供します。これは、クライアント・アプリケーションに対して透過的に行われます)。

複数の要求レベルのインターセプタを、1 回の呼び出しに関連付けることができます。あるインターセプタが別のインターセプタを認識する必要は、まったくありません。

呼び出しの要求応答サイクル内で起こるイベントは、次の 2 つのカテゴリに分けて提示されます。

- クライアント側での実行
- ターゲット側での実行

クライアント側での実行

呼び出しの要求応答サイクルにおいて、各インターセプタは 2 回呼び出されます。1 回目は要求がクライアントからターゲットに向かうとき、2 回目は応答がクライアントに返るときです。クライアントのインターセプタ・クラス `ClientRequestInterceptor` には、これら 2 つの呼び出しに特に対応するオペレーションが 2 つあります。

- `client_invoke()` ー オブジェクト・リファレンスに対して行われた要求がクライアント側 ORB に到達した時点で呼び出されます。
- `client_response()` ー 要求を行っているエンティティに対して応答が返された時点で呼び出されます。

クライアント側インターセプタを使用する CORBA アプリケーションの実行フローを 図 1-1 に示します。この図では、基本的かつ正常な要求応答呼び出しサイクルを示します。つまり、このサイクルでは例外は発生しません。

図 1-1　クライアント側インターセプタ

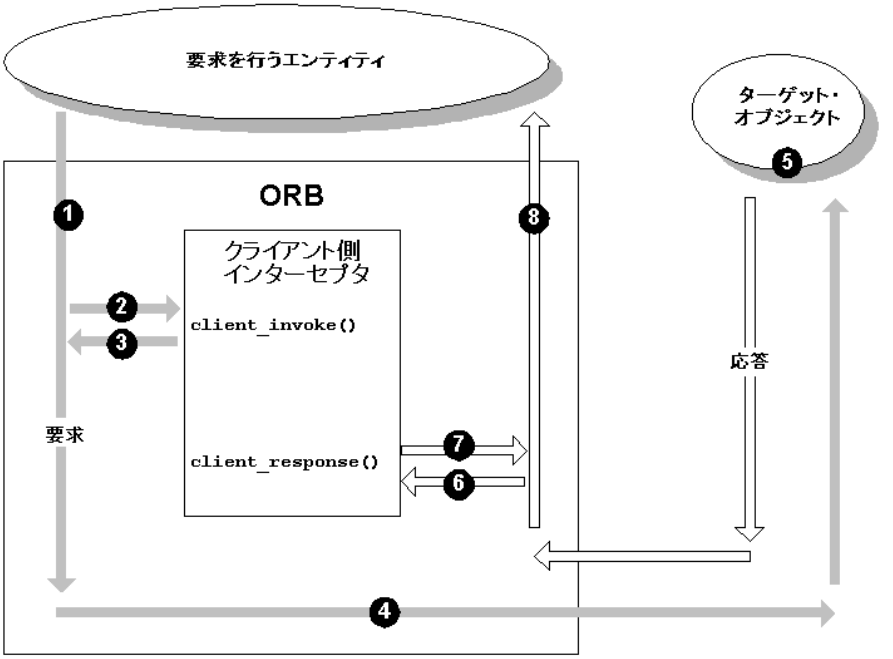


図 1-1 では、以下のイベントを説明しています。

1. 要求がクライアントから出され、ORB に到達します。
2. ORB がクライアント側インターセプタの `client_invoke` オペレーションを呼び出します (「複数の要求レベルのインターセプタの使用方法」では、複数のクライアント側インターセプタがインストールされている場合に生じる現象について説明します)。
3. クライアント側インターセプタは要求を処理し、ORB にステータス・コードを返します。
4. `client_invoke` オペレーションの結果として返される例外がない場合、要求はターゲット・オブジェクトへのパスを再開します。
5. ターゲット・オブジェクトは要求を処理して、応答を発行します。
6. 応答は ORB に戻され、ORB はインターセプタの `client_response` オペレーションを呼び出します。

7. インターセプタは応答を処理し、ORB にステータス・コードを返します。
8. 応答はクライアント・アプリケーションに送信されます。

クライアント側での例外処理

client_invoke および client_response オペレーションは各々、クライアント・インターセプタの処理を続行するかどうかを示すステータス値を返します。インターセプタは、例外処理を引き起こす例外ステータス値を返すことがあります。表 1-1 は、これらのオペレーションから返されるステータス値に応じて生じる現象、およびインターセプタが ORB と共に例外を処理する方法を示します。

表 1-1 クライアント・インターセプタの戻りステータス値

オペレーション	戻りステータス値	生じる現象
client_invoke()	INVOKE_NO_EXCEPTION	ORB はターゲットに送られた要求の通常処理を続行し、ほかにインターセプタが存在する場合は、それを呼び出します。
	REPLY_NO_EXCEPTION (BEA Tuxedo 製品のバージョン 8.0 では、ORB はこの戻り値を処理できないため、これをインターセプタの戻り値としてインプリメントすることは避けてください。)	インターセプタは要求を処理しました。ターゲットに対してこれ以上の処理は必要ありません。要求は、ターゲットによって処理された場合のように、処理済みと見なされます。したがって、ORB は呼び出しを短絡し、クライアント方向のインターセプタの呼び出しを開始します。同じインターセプタの client_response オペレーションは呼び出されませんが、それ以前に呼び出されたインターセプタの client_response オペレーションは呼び出されます。
	REPLY_EXCEPTION	インターセプタは ORB へ例外を返します。次に、ORB はこれより前のクライアント側インターセプタの各 exception_occurred オペレーションを呼び出します。exception_occurred メソッドは、ORB がクライアント・アプリケーションに例外を返す前に状態をクリーンアップする機会を、これより前のインターセプタに提供します。これにより、ORB は呼び出しを短絡し、呼び出しは完了します。exception_occurred メソッドの詳細については、第 1 章の 14 ページ「exception_occurred メソッド」を参照してください。

表 1-1 クライアント・インターセプタの戻りステータス値（続き）

オペレーション	戻りステータス値	生じる現象
client_response()	RESPONSE_NO_EXCEPTION	ORB はクライアントに送られた要求の通常処理を続行し、ほかにインターセプタが存在する場合は、それを呼び出します。
	RESPONSE_EXCEPTION	インターセプタは ORB に例外を返し、それ以前の要求の結果はすべて上書きします。ORB は、クライアント方向に、それ以前の各インターセプタの exception_occurred メソッドを呼び出し、次にクライアント・アプリケーションに例外を返します。

ターゲット側での実行

クライアント側と同じく、ターゲット側インターセプタは要求応答サイクル中に 2 回呼び出されます。ターゲット側インターセプタは TargetRequestInterceptor クラスから継承されます。このクラスには、次のオペレーションが含まれます。

- target_invoke() ー 要求がターゲット・オブジェクトのプロセスの一部である ORB に到達した時点で呼び出されます。
- target_response() ー 応答がクライアントに返るように送信された時点で呼び出されます。

ターゲット側インターセプタを使用する CORBA アプリケーションの実行フローを 図 1-2 に示します。この図では、基本的かつ正常な要求応答呼び出しサイクルを示します。つまり、このサイクルでは例外は発生しません。

図 1-2　ターゲット側インターセプタ

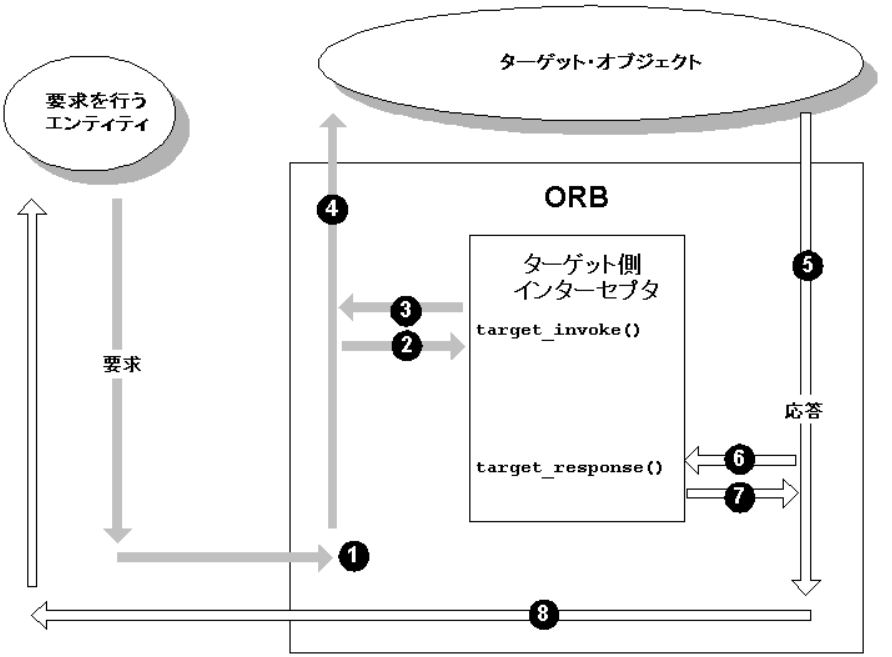


図 1-2 では、以下のイベントを説明しています。

1. クライアント要求が ORB に到達します。
2. ORB がターゲット側インターセプタの `target_invoke` オペレーションを呼び出します (複数のターゲット側インターセプタがインストールされている場合に生じる現象については「複数の要求レベルのインターセプタの使用方法」の節で説明します)。
3. ターゲット側インターセプタが要求を処理し、ORB にステータス・コードを返します。
4. `target_invoke` オペレーションの実行中に発生する例外がない場合、要求がターゲット・オブジェクトへのパスを再開します。
5. ターゲット・オブジェクトが要求を処理して、応答を発行します。
6. ターゲット側 ORB がインターセプタの `target_response` オペレーションを呼び出します。

7. インターセプタが応答を処理し、ORB にステータス・コードを返します。
8. 応答がクライアント・アプリケーションに送信されます。

ターゲット側での例外処理

表 1-2 は、target_invoke および target_response オペレーションによって返されるステータス値に応じて、ターゲット側の呼び出しに生じる現象を示し、例外がスローされた場合の動作を説明します。

表 1-2 ターゲット・インターセプタの戻りステータス値

オペレーション	戻りステータス値	生じる現象
target_invoke()	INVOKE_NO_EXCEPTION	ORB はターゲット (オブジェクト・インプリメンテーション) への要求の通常処理を続行し、ほかにインターセプタが存在する場合は、それを呼び出します。
	REPLY_NO_EXCEPTION (BEA Tuxedo 製品のバージョン 8.0 では、ORB はこの戻り値を処理できないため、これをインターセプタの戻り値としてインプリメントすることは避けてください。)	インターセプタは要求を処理しました。ターゲットに対してこれ以上の処理は必要ありません。要求は、ターゲットによって処理された場合のように、処理済みと見なされます。したがって、ORB は呼び出しを短絡し、クライアント方向のインターセプタの呼び出しを開始します。同じインターセプタの target_response オペレーションは呼び出されませんが、それ以前に呼び出されたインターセプタの target_response オペレーションは呼び出されます。
	REPLY_EXCEPTION	インターセプタは ORB へ例外を返します。次に、ORB はこれより前のターゲット側インターセプタの各 exception_occurred オペレーションを呼び出します。exception_occurred メソッドは、ORB がクライアント ORB に例外を返す前に状態をクリーンアップする機会を、これより前のインターセプタに提供します。これにより、ターゲット ORB は呼び出しを短絡し、呼び出しは完了します。exception_occurred メソッドの詳細については、第 1 章の 14 ページ「exception_occurred メソッド」を参照してください。

表 1-2 ターゲット・インターセプタの戻りステータス値（続き）

オペレーション	戻りステータス値	生じる現象
target_response()	RESPONSE_NO_EXCEPTION	ORB はクライアントに送られた要求の通常処理を続行し、ほかにインターセプタが存在する場合は、それを呼び出します。
	RESPONSE_EXCEPTION	インターセプタは ORB に新しい例外を返し、それ以前の要求の結果はすべて上書きします。クライアントに戻る過程でインターセプタの target_response オペレーションを呼び出す代わりに、ORB はこれらのインターセプタの exception_occurred オペレーションを呼び出します。

exception_occurred メソッド

すべてのインターセプタに exception_occurred メソッドが存在します。ORB はこのメソッドを、次の状況下で呼び出します。

- ORB 内部に問題が見つかった場合。たとえば、オペレーティング・システムのリソース・エラーや通信障害など。
- 例外が ORB またはメソッドで生成されたのではなく、別のインターセプタによって設定された場合。これはたとえば、ORB がインターセプタ A およびインターセプタ B をそれぞれ呼び出している場合です。インターセプタ A で例外が設定されているため、ORB がインターセプタ B に呼び出すのは client_response または target_response メソッドではなく、exception_occurred メソッドとなります。インターセプタはこの振る舞いを利用して、例外を含む応答が処理されるコンテキストと、例外の実際の値を、DataInputStream 構造体からの例外を読み取ることなく検査できます。
- クライアント・アプリケーションは Request オブジェクトに対する遅延同期 DII 呼び出しを使用し、その後 Request オブジェクトを解放します。この場合、クライアントに送信される応答はありません。

前述の状況の 1 つが生じた場合、exception_occurred メソッドが client_response または target_response メソッドの代わりに呼び出されますが、クライアントの呼び出しを完了させる効果は本質的に同じです。

要求のトラッキングの詳細については、第 2 章の 7 ページ「インターセプタの応答オペレーションのインプリメント」を参照してください。

短絡動作について

前述のように、インターセプタは自身で要求を処理したり、例外を返したりすることによって、クライアント要求を短絡できます。どちらの場合でも、クライアント要求が実際にターゲット・オブジェクトによって処理されることはありません。

短絡動作は、`client_invoke` または `target_invoke` メソッドでのみ機能します。`client_response` または `target_response` メソッドには適用されません。

複数の要求レベルのインターセプタの使用方法

複数の要求レベルのインターセプタは、ORB が 1 つずつ逐次的に実行できるように、キューにインストールされます。ORB は、キューに残っている要求レベルのインターセプタがなくなるまで、各要求レベルのインターセプタに連続的に要求を供給します。すべてのインターセプタで正常な状態が示されると要求が処理されます。ORB は得られた応答をクライアントの場合はトランスポートに、ターゲットの場合はオブジェクト・インプリメンテーションに送信します。ORB は要求の処理とは逆の順番で応答を処理するインターセプタを実行します。

インターセプタで正常な状態が示されない場合は、短絡応答が生じます。短絡は `client_invoke` または `target_invoke` オペレーションによって実行されます。インターセプタから返されたステータスにより、例外を伴う要求をターゲット・オブジェクトに処理させるのではなく、インターセプタ自体で応答することが、ORB に通知されます (インターセプタの `client_response` または `target_response` オペレーションでは、短絡動作を行うことはできませんが、ターゲット応答の代わりとなることはできます)。

各インターセプタは通常、明示的に情報を共有していない限り、ほかのインターセプタを認識していません。この独立したプログラミング・モデルは、短絡に関する実行セマンティクスによって維持されます。応答を短絡して、送信先 (クライアント側のトランスポート、およびターゲット側のオブジェクト・インプリメ

ンテーション) に届かないようにする必要があるとインターセプタが示した場合、応答は正常に通過したインターセプタを経由して元に戻ります。たとえば、インターセプタ A が client_invoke オペレーションの処理後に、要求が送信されるようにステータス値 INVOKE_NO_EXCEPTION を返し、次のインターセプタ B が要求を拒否して例外を生成すると、その例外は応答に加えられ、インターセプタ A の exception_occurred オペレーションに送信されます。ターゲット側の類似の実行モデルも有効です。

図 1-3 は複数のクライアント側インターセプタが ORB にインストールされている場合の実行シーケンスを示します (同様の一連のオペレーションは、複数のターゲット側インターセプタでも発生します)。

図 1-3　ORB 上の複数のインターセプタ

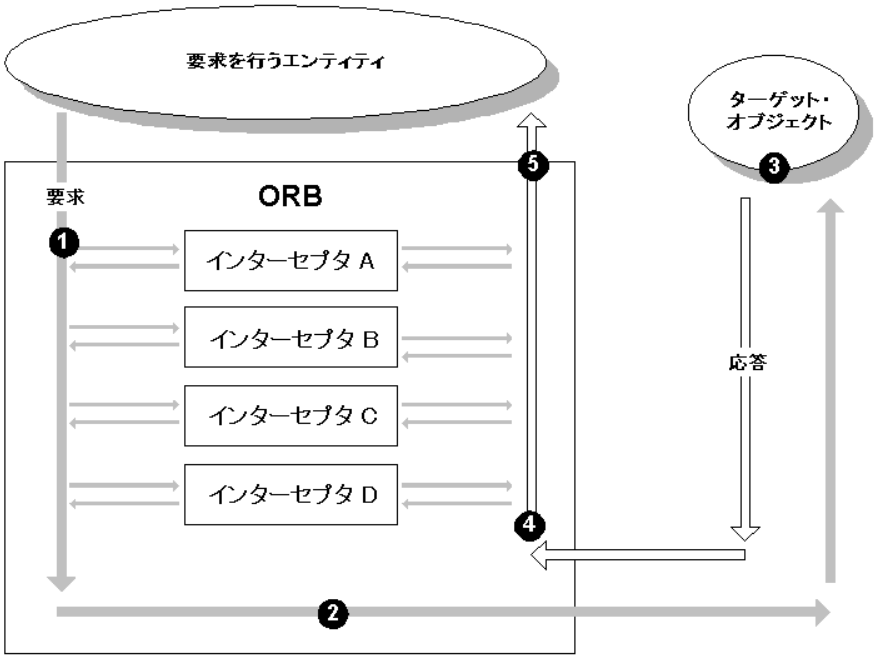


図 1-3 では、以下のイベントを説明しています。

1. クライアント要求が ORB に到達し、ORB がインターセプタ A から D を順に呼び出します。
2. 要求がターゲット・オブジェクトに送られます。

3. ターゲット・オブジェクトが要求を処理して、応答を返します。
4. 応答がクライアント側インターセプタを含む ORB に戻ります。次に、ORB は登録された各インターセプタを、要求が送出されたときとは逆順に呼び出します。
5. 応答が返されてクライアント・アプリケーションに到達します。

複数のクライアント側インターセプタ

ORB は要求を受け取ると、各クライアント側インターセプタの client_invoke オペレーションを呼び出します。戻り値 INVOKE_NO_EXCEPTION が各 client_invoke オペレーションから返された場合（通常の場合）、結果として生じた要求は ORB によってメッセージにマーシャリングされ、ターゲット・オブジェクトに送られます。

次の状況では、ORB は、クライアント方向で残りのインターセプタの client_response オペレーションを呼び出すのではなく、これらのインターセプタの exception_occurred を呼び出し、次に例外をクライアント・アプリケーションに返します。

- どの client_invoke オペレーションからの戻り値も、REPLY_EXCEPTION となります。

  この場合、ORB は要求を残りのインターセプタまたはトランスポートに伝達することを中止します。したがって、要求は ORB によって短絡されます。

- どの client_response オペレーションからの戻り値も、RESPONSE_EXCEPTION となります。

  この場合、インターセプタは ORB に例外を返し、それ以前の要求の結果はすべて上書きします。

複数のターゲット側インターセプタ

クライアント側インターセプタの処理の場合と同じく、ORB は各ターゲット側インターセプタの target_invoke オペレーションを連続的に呼び出します。各 target_invoke オペレーションから戻り値 INVOKE_NO_EXCEPTION が返されると、要求はターゲット・オブジェクトに渡されます。

次の状況では、ORB は、クライアント方向で残りのインターセプタの target_response オペレーションを呼び出すのではなく、これらのインターセプタの exception_occurred を呼び出し、次に例外をクライアント・アプリケーションに返します。

- どの target_invoke オペレーションからの戻り値も、REPLY_EXCEPTION となります。

  この場合、ORB は要求を残りのインターセプタおよびトランスポートに伝達することを中止します。この時点で、ORB はクライアント ORB に応答を返し、ターゲット ORB は要求を短絡します。

- どの target_response オペレーションからの戻り値も、RESPONSE_EXCEPTION となります。

  この場合、インターセプタは ORB に例外を返し、それ以前の要求の結果はすべて上書きします。

インターセプタおよびメタ・オペレーション

メタ・オペレーションとは、CORBA Object インターフェイスをサポートする is_a、get_interface、および non_existent などのオペレーションです。一部のメタ・オペレーションは、呼び出しを発行することなく ORB によって実行されますが、それ以外のオペレーション (is_a、get_interface、および non_existent メソッド) ではオブジェクトの呼び出しを必要とする場合あります。したがって、これらのオペレーションはインターセプタを開始できます。

オペレーションを CORBA 固有の言語でバインディングすると、オペレーション名を IDL で定義した名前から、次の名前に変換できます。

- _is_a
- _interface
- _non_existent (または _not_existent)

セキュリティ・ベースのインターセプタをインプリメントしている場合は、ORB がこれらのオペレーションをクライアント要求の一部として呼び出す可能性があるため、この振る舞いには注意してください。通常、インターセプタが特定のクライアント要求にだけターゲット・オブジェクトへの送信を許可し、これらのメタ・オペレーションを考慮できないような状況は避けてください。

2 CORBA 要求レベルのインターセプタの開発

CORBA 要求レベルのインターセプタの開発を行うには通常、次の手順に従います。

- ステップ 1: CORBA アプリケーションのインターフェイスの識別

  インターセプタをデプロイするマシンの識別も行います。
- ステップ 2: インターセプタ・インプリメンテーション・コードの記述
- ステップ 3: インターセプタ・ヘッダ・ファイルの作成
- ステップ 4: インターセプタのビルド
- ステップ 5: インターセプタのテスト

上記の手順は通常、繰り返し行います。たとえば、最初にインターセプタをビルドしてテストするときは、そのインターセプタが実行中であることを確認するだけの最も基本的なコードのみとします。その後のビルドとテストで、インターセプタの全機能を徐々にインプリメントしていきます。

以降の節では、BEA Tuxedo ソフトウェアに同梱されたサンプルのインターセプタを例として、各手順を詳しく説明します。

ステップ 1: CORBA アプリケーションのインターフェイスの識別

所定のマシンにインターセプタをデプロイすると、そのインターセプタはマシン上のアプリケーションが要求を発行 (クライアント側インターセプタの場合) または受信 (ターゲット側インターセプタの場合) するたびに呼び出されるため、著しいオーバヘッドが生じます。したがって、作成したインターセプタはすべて、これらのアプリケーションによく適合している必要があります。

たとえば、セキュリティ・インターセプタは通常、問題となっている要求の種類、および要求で処理されているデータの種類を認識する必要があります。

特定の要求を扱うインターセプタはすべて、要求からインターフェイス・リポジトリ ID を抽出できる必要があります。インターフェイスについてこのような情報があれば、インターセプタは要求に含まれるデータの種類を知り、そのデータを要求固有のやり方で処理できます。

さらに、対象外の要求が送信された場合、インターセプタはその要求を迅速かつ効率的に渡すことができなければなりません。

「第 4 章 PersonQuery サンプル・アプリケーション」で説明されている PersonQuery の例では、PersonQuery クライアント・アプリケーションのユーザがアドレスを受信できるかどうかを決定するインターセプタを使用します。ユーザの識別情報が特定の基準に一致する場合には、インターセプタは完全なアドレス番号をクライアントに返すことを許可します。一致しなかった場合、インターセプタはアドレスの代わりに x 文字からなる文字列のみをログ・ファイルに返します。

ステップ 2: インターセプタ・インプリメンテーション・コードの記述

インターセプタをインプリメントする場合、次の事項に留意します。

- インプリメントする最初の段階では、インターセプタは単純なものにしておきます。たとえば、各関数メンバで、メッセージをログ・ファイルに出力する文をインプリメントするよう決定するとします。これにより確認されるのは、単にインターセプタが適正にビルドおよび登録され、実行中であるかどうかということのみです。いったんインターセプタが適正に機能していると分かれば、必要な機能がすべて揃うまでコードの追加を繰り返すことができます。
- ある特定の機能をインプリメントするためにクライアント側およびターゲット側のインターセプタをデプロイする場合は、両方のインターセプタを単一のソース・ファイルにインプリメントできます。その後、インターセプタをビルドし、デプロイする際に、必要に応じてそれらをクライアント側マシン上とターゲット側マシン上で個別にコンフィギュレーションできます。BEA Tuxedo ソフトウェアに付属しているサンプルのインターセプタでは、この方式を採用しています。

以下のトピックでは、多くのインターセプタについて一般的なインプリメンテーション上の検討事項について説明します。「第 7 章 InterceptorData サンプル・インターセプタ」で説明する InterceptorData インターセプタから例を取ります。

インプリメンテーション・ファイルの起動

付録 A に示すコードの抜粋を、インターセプタのインプリメントを開始する際に使用できます。付録 A のコードを使用しても、BEA の Web サイト上の WebLogic Enterprise ディベロッパ・センタで入手できるスタータ・ファイルをコピーしてもかまいません。

ファイル名	説明
intercep.h	インターセプタ・ヘッダのスタータ・ファイルです。このファイルの内容と、使用方法の説明は、第 2 章の 9 ページ「ステップ 3: インターセプタ・ヘッダ・ファイルの作成」に示しています。
intercep.cpp	インターセプタ・インプリメンテーションのスタータ・ファイルです。

WebLogic Enterprise ディベロッパ・センタからこれらのスタータ・ファイルを取得する方法については、『BEA Tuxedo 8.0 リリース・ノート』を参照してください。

付録 A に示すサンプル・インターセプタのコードを使用して、インターセプタのインプリメンテーションを開始できます。コード中で、*YourInterceptor* はインプリメントしているインターセプタの名前を表します。ORB は常に ServiceContextList および CORBA::DataOutputStream の各パラメータについて、ニル・リファレンスを渡します。これらのパラメータは、使用も参照もできません。またこの制限事項は将来のバージョンで変更される可能性があるため、nil についてのこれらのパラメータのテストはしないでください。

実行時のインターセプタの初期化

ORB の初期化時に、すべてのインターセプタがインスタンス化されます。それ以外では、要求レベルのインターセプタはインスタンス化されません。初期化の一環として、インターセプタの初期化ルーチンにより、インターセプタのサポート対象に応じて、クライアント・インターセプタとターゲット・インターセプタの一方または両方のインプリメンテーションのインスタンスをインスタンス化する必要があります。前述のように、単一の共有可能なイメージで、クライアント側インターセプタとターゲット側インターセプタの双方をサポートできます。その後、任意のインスタンス化されたインターセプタのインスタンスが、初期化ルーチンから返され、ORB 実行時に登録されます。

次のコードの抜粋部分は、InterceptorData インターセプタからのものです。クライアント側 ORB の初期化時に、ORB によって呼び出された初期化オペレーションの宣言を示します。

```
void InterceptorDataClientInit(
    CORBA::ORB_ptr                                          TheORB,
    RequestLevelInterceptor::ClientRequestInterceptor ** ClientPtr,
    RequestLevelInterceptor::TargetRequestInterceptor ** TargetPtr,
    CORBA::Boolean *                                        RetStatus)
```

次のコードの抜粋部分は、InterceptorData クライアント・インターセプタ・クラスをインスタンス化するための文を示します。この抜粋部分は、受信される各クライアント要求のトラッキングを行い、ターゲット・オブジェクトから返される応答に一致させるための、tracker というクラスを使用します。tracker クラスは、第 2 章の 6 ページ「要求内のオペレーションの識別」で説明します。

```
ClientInterceptorData * client = new ClientInterceptorData(TheORB, tracker);
    if  (!client)
    {
        tmpfile << "InterceptorDataClientInit: Client alloc failed"
                << endl << endl;
        *RetStatus = CORBA_FALSE;
        delete tracker;
        return;
    }
```

次のコード抜粋部分は、インターセプタ・クラスを ORB に返すための文を示します。

```
*ClientPtr = client;
*TargetPtr = 0;
*RetStatus = CORBA_TRUE;
return;
```

要求からのインターセプタ名の取得

特定のインターフェイスや要求で機能するインターセプタの場合は、要求に関連付けられたインターフェイス ID を抽出する方法が必要です。それにより、インターセプタは要求を識別し、その中のデータをどのように処理すればよいかを理解できます。たとえば、InterceptorData インターセプタは PersonQuery アプリケーションからの要求で送信された要求パラメータを操作します。要求パラメータを操作するには、インターセプタはどの要求が送信されているのかを認識する必要があります。

次の InterceptorData サンプルからのコード抜粋部分は、RequestContext 構造体から抽出されたインターフェイス ID を示します。

```
            if (strcmp(request_context.interface_id.in(),
                       PersonQuery::_get_interface_name()) != 0)
                return ret_status;
```

要求内のオペレーションの識別

抽出されたインターフェイス ID を使用して、InterceptorData サンプルは単純な switch 文でクライアント要求内のオペレーションを識別します。これにより、インターセプタは要求に含まれる要求パラメータの扱い方を認識します。

次のコード抜粋部分は、Exit オペレーション、またはデータベースに対しある人物を名前で指定した問合せを行うためのオペレーションがあるかどうか確認する、switch 文を示します。parser オブジェクトが使用されていることに注意してください。このオブジェクトは tracker オブジェクトより受信した要求からオペレーションを抽出します。

```
m_outfile << "   Operation:      " << request_context.operation << endl;
    PQ parser;
    PQ::op_key key = parser.MapOperation(request_context.operation.in());
    switch (key)
    {
        default:
          m_outfile << "    ERROR: operation is not member of "
                  << request_context.interface_id.in() << endl;
                     excep_val = new CORBA::BAD_OPERATION();
          return Interceptors::REPLY_EXCEPTION;

        case PQ::Exit:
          m_outfile << endl;
          return  ret_status;

        case PQ::ByPerson:
          {
             PersonQuery::Person per;
             parser.GetByPerson(request_arg_stream, &per);
             m_outfile << "   Parameters:" << endl;
             m_outfile << per << endl;
          }
          break;
```

インターセプタの応答オペレーションのインプリメント

クライアント要求からインターフェイス ID を抽出するのは、かなり単純な作業です。しかし、ターゲット応答で同じ処理を行う場合はそれほど単純ではありません。ORB から受信する応答に関連付けられたインターフェイスおよびオペレーションを知る必要がある場合、インターセプタは要求をトラッキングするための特別なロジックを必要とします。クライアントから受信した要求のトラッキングは、インターセプタの役割です。

InterceptorData サンプルは、`Tracker` と呼ばれる言語オブジェクトをインプリメントします。これはターゲットに送られる要求の記録を取り、インターセプタにターゲット応答が返ると、それらの要求に応答を適合させるオブジェクトです。

InterceptorData サンプルの `client_response` および `target_response` オペレーションは、ターゲットから応答が返ると、`Tracker` オブジェクトからインターフェイスおよびオペレーションの情報を抽出します。

次の InterceptorData コードの抜粋部分は、応答に関連付けられた要求を抽出するものです。

```
RequestInfo * req_info = m_tracker->CompleteRequest(reply_context);
    if (!req_info)
    {
       m_outfile << "   unable to find request for this reply (must not be one
       we care about)" << endl << endl;
       return Interceptors::RESPONSE_NO_EXCEPTION;
    }

    //
    // 必要なインターフェイス。ここで、要求パラメータを
    // 解析できるように、呼び出されているオペレーションを識別する
    //

    m_outfile << "   ReplyStatus:    ";
    OutputReplyStatus(m_outfile, reply_context.reply_status);
    m_outfile << endl;
    m_outfile << "   Interface:      " << req_info->intf() << endl;
    m_outfile << "   Operation:      " << req_info->op() << endl;
    PQ parser;
    PQ::op_key key = parser.MapOperation(req_info->op());
```

これでインターセプタは応答に関連付けられた要求を取得したので、応答内のデータを適切に処理できます。

データ入力ストリームからのパラメータの読み出し

次のコードの抜粋部分は、InterceptorData のサンプルがどのようにして、データ・ストリームからの要求パラメータを構造体に入れるのかの例を示します。次のコード抜粋部分のパラメータ S は、インターセプタのインプリメンテーションにより PersonQuery オペレーションの応答パラメータ値を取得するのに使用可能な DataInputStream 構造体へのポインタを表します。このコード抜粋部分において、中かっこで囲まれたコードは、DataInputStream 構造体からの応答のパラメータを抽出するものです。DataInputStream 構造体の詳細については、「第 8 章 要求レベルのインターセプタの API」を参照してください。

```
void PQ::get_addr(CORBA::DataInputStream_ptr S,
                  PersonQuery::Address *addr)
{
    addr->number = S->read_short();
    addr->street = S->read_string();
    addr->town = S->read_string();
    addr->state = S->read_string();
    addr->country = S->read_string();
}
```

例外

excep_val パラメータ経由で返されたインターセプタからの例外は、CORBA::SystemException 基本クラスから派生したタイプのみです (これ以外で、インターセプタのインプリメンテーションが ORB に返す例外のタイプはすべて、ORB によって CORBA::UNKNOWN 例外に変換され、excep_val パラメータを介して渡されます)。例外は CORBA::SystemException クラスまたはその派生クラスの 1 つにマッピングする必要があります。

ステップ 3: インターセプタ・ヘッダ・ファイルの作成

インターセプタのインプリメンテーション・ファイル内に何らかのインプリメンテーション・コードを作成した後は、必要に応じてインターセプタ・ヘッダ・ファイルにデータまたはオペレーションを加える必要があります。

次のコード例は、クライアント側インターセプタとターゲット側インターセプタの双方をインプリメントする、インターセプタ・インプリメンテーション・ファイル用のヘッダ・ファイルで必要な基本情報を示します。

また、この例では次のものも示されます。

- セキュリティに必要な include ファイル
- セキュリティのためのターゲット・データ・メンバ

このコード例では、YourInterceptor は作成しているインターセプタの名前を表します。

```
#include <CORBA.h>
#include <RequestLevelInterceptor.h>
#include <security_c.h>          // セキュリティ用

class YourInterceptorClient : public virtual
RequestLevelInterceptor::ClientRequestInterceptor
{
private:
    YourInterceptorClient() {}
    CORBA::ORB_ptr  m_orb;
public:
    YourInterceptorClient(CORBA::ORB_ptr TheOrb);
    ~YourInterceptorClient() {}
    Interceptors::ShutdownReturnStatus shutdown(
        Interceptors::ShutdownReason reason,
        CORBA::Exception_ptr & excep_val);
    CORBA::String id();
    void exception_occurred (
        const RequestLevelInterceptor::ReplyContext & reply_context,
        CORBA::Exception_ptr excep_val);
    Interceptors::InvokeReturnStatus client_invoke (
        const RequestLevelInterceptor::RequestContext & request_context,
```

```
        RequestLevelInterceptor::ServiceContextList_ptr service_context,
        CORBA::DataInputStream_ptr request_arg_stream,
        CORBA::DataOutputStream_ptr reply_arg_stream,
        CORBA::Exception_ptr & excep_val);
    Interceptors::ResponseReturnStatus client_response (
        const RequestLevelInterceptor::ReplyContext & reply_context,
        RequestLevelInterceptor::ServiceContextList_ptr service_context,
        CORBA::DataInputStream_ptr arg_stream,
        CORBA::Exception_ptr & excep_val);

};

class YourInterceptorTarget : public virtual
RequestLevelInterceptor::TargetRequestInterceptor
{
private:
    YourInterceptorTarget() {}
    CORBA::ORB_ptr  m_orb;
    SecurityLevel1::Current_ptr m_security_current;          // セキュリティ用
    Security::AttributeTypeList * m_attributes_to_get;       // セキュリティ用
public:
    YourInterceptorTarget(CORBA::ORB_ptr TheOrb);
    ~YourInterceptorTarget();
    Interceptors::ShutdownReturnStatus shutdown(
        Interceptors::ShutdownReason reason,
        CORBA::Exception_ptr & excep_val);
    CORBA::String id();
    void exception_occurred (
        const RequestLevelInterceptor::ReplyContext & reply_context,
        CORBA::Exception_ptr excep_val);
    Interceptors::InvokeReturnStatus target_invoke (
        const RequestLevelInterceptor::RequestContext & request_context,
        RequestLevelInterceptor::ServiceContextList_ptr service_context,
        CORBA::DataInputStream_ptr request_arg_stream,
        CORBA::DataOutputStream_ptr reply_arg_stream,
        CORBA::Exception_ptr & excep_val);

    Interceptors::ResponseReturnStatus target_response (
        const RequestLevelInterceptor::ReplyContext & reply_context,
        RequestLevelInterceptor::ServiceContextList_ptr service_context,
        CORBA::DataInputStream_ptr arg_stream,
        CORBA::Exception_ptr & excep_val);

};
```

ステップ 4: インターセプタのビルド

インターセプタは、共有可能ライブラリに組み込まれます。したがって、インターセプタをビルドする手順は、プラットフォーム固有です。任意の特定プラットフォーム上でインターセプタをビルドするのに使用される特定のコマンドおよびオプションの詳細を知るには、BEA Tuxedo ソフトウェア付属のインターセプタのサンプル・アプリケーションをビルドする makefile を実行し、ビルドによって生じるログ・ファイルで、ビルド結果を参照します。

サンプルのインターセプタをビルドするコマンドは、次のとおりです。

Windows 2000

```
> nmake -f makefile.nt
```

UNIX

```
> make -f makefile.mk
```

BEA Tuxedo ソフトウェアに付属するサンプルのインターセプタのビルドおよび実行の詳細については、「第 4 章 PersonQuery サンプル・アプリケーション」を参照してください。

ステップ 5: インターセプタのテスト

インターセプタをテストするには、次のタスクを実行する必要があります。

- インターセプタを登録します。
- `tmboot` コマンドを使用して、CORBA サーバ・アプリケーションをブートします。
- CORBA クライアント・アプリケーションを実行します。
- インターセプタのログ・ファイルを確認し、インターセプタの振る舞いを検証します。

インターセプタの登録については、「第 3 章 CORBA 要求レベルのインターセプタのデプロイ」を参照してください。

3 CORBA 要求レベルのインターセプタのデプロイ

CORBA 要求レベルのインターセプタの登録管理と関連する管理タスクは 3 つあります。

- インターセプタの登録
- インターセプタの登録解除
- インターセプタの呼び出し順の変更

ここでは、これら 3 つのタスクについて説明します。

インターセプタの登録

ORB にインターセプタを登録するには、`epifreg` コマンドを使用します。インターセプタを登録する場合は、すでに ORB に登録されたインターセプタのリストの末尾に、そのインターセプタを追加します。これは複数のインターセプタを ORB に登録する場合には重要です。

インターセプタを登録するための `epifreg` コマンドの構文は、次のとおりです。

```
epifreg -t bea/wle -i AppRequestInterceptor \
        -p <InterceptorName> -f <FileName> -e <EntryPoint> \
        -u "DisplayName=<Administrative Name>" -v 1.0
```

前述のコマンド・ラインの詳細は次のとおりです。

- *`InterceptorName`* は、ORB に登録されたインターセプタの名前を表します。選択する名前はそれ以前に登録した名前と重複しないものである必要があります。この名前は、複数のインターセプタの順序を指定したり、インターセプタの登録を解除したりするのに使用します。その後に続く引数 *`FileName`*、*`EntryPoint`* および `DisplayName` は、この名前に関連付けられます。

- *FileName* は、インターセプタのインプリメンテーションを含むファイルの場所を表します。この名前は、オペレーティング・システムおよび言語に依存します。このファイルは、共有可能なイメージ・ファイルです。
- *EntryPoint* は、インターセプタのエントリ・ポイント名である文字列値を表します。この名前はプログラミング言語固有です。この値は、インターセプタのインスタンスを作成する共有可能イメージの初期化関数名です。
- DisplayName は、管理機能およびその他の報告目的に使用される文字列値を指定します。この名前は、あくまでも管理用の名前です。

注記 BEA Tuxedo CORBA サーバ・プロセスがすでに実行されているマシン上にインターセプタを登録する場合、これらのプロセスはインターセプトの対象にはなりません。インターセプトの対象となるのは、インターセプタの登録後に開始されたプロセスのみです。すべての CORBA サーバ・プロセスを確実にインターセプトの対象とするには、必ず CORBA サーバ・プロセスのブート前にインターセプタを登録するようにします。

インターセプタの登録解除

ORB からインターセプタの登録を解除するには、epifunreg コマンドを使用します。このコマンドの構文は次のとおりです。

```
epifunreg -t bea/wle -p <InterceptorName>
```

引数 *<InterceptorName>* は epifreg コマンドで指定した、大文字と小文字を区別しない名前と同じものです。インターセプタは、登録解除するとインターセプタの順序から除外されます。

インターセプタの呼び出し順の変更

次のコマンドを使用すると、インターセプタの登録順序と、それに従った呼び出し順序を確認できます。

```
epifregedt -t bea/wle -g -k SYSTEM/interfaces/AppRequestInterceptor
```

epifregedit により、ORB が要求を受信したときのインターセプタの実行順序が表示されます。

インターセプタの実行順序は、次のコマンドで変更できます。

```
epifregedt -t bea/wle -s -k SYSTEM/interfaces/AppRequestInterceptor \
-a Selector=Order -a Order=<InterceptorName1>,<InterceptorName2>,...
```

各 *<InterceptorName>* は、事前に登録しておく必要のある、大文字と小文字を区別しないインターセプタ名です。このコマンドにより、現在レジストリにある順序が置換されます。`epifregedt` コマンドでは、ORB によってロードおよび実行するインターセプタをすべて指定する必要があります。インターセプタが登録されたままでも、名前を `epifregedt` コマンドで指定していないと、そのインターセプタはロードされません。

4 PersonQuery サンプル・アプリケーション

BEA Tuxedo ソフトウェアに同梱されたインターセプタ例を理解して使用するには、PersonQuery サンプル・アプリケーションをビルドして実行する必要があります。PersonQuery サンプル・アプリケーション自体には、インターセプタは含まれていません。しかし、このアプリケーションは、以降の 3 つの章で説明するインターセプタのサンプル・アプリケーションの基本として使用されます。

ここでは、次の内容について説明します。

- PersonQuery サンプル・アプリケーションのしくみ
- PersonQuery サンプル・アプリケーションの OMG IDL
- PersonQuery サンプル・アプリケーションのビルドと実行

PersonQuery サンプル・アプリケーションのしくみ

PersonQuery サンプル・アプリケーションは、単純なデータベース問い合せインターフェイスをインプリメントします。PersonQuery アプリケーションを使用すると、データベースから次のような特定の検索基準に一致する人物の情報を取得できます。

- 年齢、体重、髪の色、目の色、肌の色などの身体的特性
- 名前、住所、その他の詳細情報

PersonQuery アプリケーションには、次のコンポーネントが含まれます。

- さまざまなデータ型をパラメータとして含む要求を発行する、クライアント・アプリケーション。クライアント・アプリケーションは特定の形式で

ユーザからのコマンド行の入力を受け付け、サンプル・インターフェイスに従って入力をパッケージ化し、適切な要求を送信します。

クライアント・アプリケーションは、サーバからの問い合せ結果を受け取ると、見つかった項目の数を報告します。ユーザは次に、最新の問い合せ結果を表示するコマンドを入力するか、新しい問い合せを指定します。

- 単純な組み込みデータベースを含むサーバ・アプリケーション。サーバ・アプリケーションは、データベースにアクセスしてクライアント要求を処理します。

PersonQuery データベース

サーバ・アプリケーションの PersonQuery データベースには、データベース内の各人物に関する次の情報が含まれます。

- 名前
- 住所
- 米国社会保障番号
- 性別
- 年齢
- 配偶者の有無
- 趣味
- 生年月日
- 身長
- 体重
- 髪の色
- 肌の色
- 目の色
- その他の身体的特性

クライアント・アプリケーションのコマンド行インターフェイス

PersonQuery サンプル・アプリケーションは、ユーザがデータベース問い合せコマンド、およびアプリケーション終了コマンドを入力できるクライアント・コンポーネントの単純なコマンド行インターフェイスをインプリメントします。

データベース問い合せコマンドの構文は次のとおりです。

```
Option? command [keyword] [command [keyword]]...
```

このコマンド構文の詳細は以下のとおりです。

- `Option?` は PersonQuery コマンド・プロンプトです。
- *`command`* は、表 4-1 に示す PersonQuery コマンドの 1 つです。
- *`keyword`* は、表 4-1 に示すキーワードの 1 つです。キーワードの指定にあたっては、次のルールに留意してください。
  - 通常 name コマンドや address コマンドについて指定される複合キーワードは、次のコマンドのように、スペースで区切って二重引用符 ("") で囲む必要があります。

    ```
    Option? name "Thomas Mann"
    ```

  - アドレスを指定する場合は、次のコマンドのように、通りの名前、都市名、州または地区名、国名など住所の構成要素を、常にカンマで区切ります。

    ```
    Option? address "116 Einbahnstrasse, Frankfurt am Main, BRD"
    ```

- 次の例のように 1 行に複数のコマンドを指定できます。

  ```
  Option? hair brown eyes blue
  ```

表 4-1 PersonQuery アプリケーションのコマンドおよびキーワード

コマンド	キーワード	説明
name	*"firstname lastname"*	名前による問い合せ。スペースの入った文字列は引用符で囲む必要があります。

コマンド	キーワード	説明
address	*"number street, city..."*	住所による問い合せ。スペースの入った文字列は引用符で囲む必要があります。住所の構成要素は番地、通りの名前、都市名、州名、および国名です。通り、都市、州、および国は、カンマで区切る必要があります。
ss	*xxx-xx-xxxx*	米国社会保障番号による問い合せ。キーワードは、*xxx-xx-xxxx* の形式にする必要があります。
sex	*sex*	性別による問い合せ。選択肢は、male、female、および cant_tell です。
age	*age*	年齢による問い合せ。
marriage	*status*	配偶者の有無による問い合せ。選択肢は、married、single、divorced、および not_known です。
hobby	*hobby*	趣味による問い合せ。選択肢は、who_cares、rocks、swim、tv、stamps、photo、および weaving です。
dob	*mm/dd/yyyy*	日付による問い合せ。キーワードは、*mm/dd/yyyy* の形式にする必要があります。
height	*inches*	身長による問い合せ。単位はインチです。
weight	*pounds*	体重による問い合せ。単位はポンドです。
hair	*color*	髪の色による問い合せ。選択肢は white、black、red、brown、green、yellow、blue、gray、および unknown です。

コマンド	キーワード	説明
skin	*color*	肌の色による問い合せ。選択肢は white、black、brown、yellow、green、および red です。
eyes	*color*	目の色による問い合せ。選択肢は blue、brown、gray、green、purple、black、および hazel です。
other	*feature*	その他の身体的特性による問い合せ。選択肢は tattoo、limb (手足の欠如)、scar、および none です。
result		最新の問い合せの出力結果を表示します。
exit		提供されたサービスに対する課金を表示してアプリケーションを終了します。

PersonQuery サンプル・アプリケーションの OMG IDL

リスト 4-1 は、PersonQuery サンプル・アプリケーションでインプリメントされている OMG IDL コードを示します。

リスト 4-1 PersonQuery アプリケーションのインターフェイス用 OMG IDL

```
#pragma prefix "beasys.com"

interface PersonQuery
{
       enum MONTHS {Empty,Jan, Feb, Mar, Apr, May, Jun, Jul, Aug,
                    Sep, Oct, Nov, Dec};
       struct date_ {
                     MONTHS Month;
```

```
                    short Day;
                    short Year;
                };
        typedef date_ Date;
        struct addr_ {
               short number;
               string street;
               string town;
               string state;
               string country;
        };
        typedef addr_ Address;
        enum MARRIAGE {not_known, single, married, divorced};
        enum HOBBIES {who_cares, rocks, swim, tv, stamps, photo,
                      weaving};
        enum SEX {cant_tell, male, female};
        enum COLOR {white, black, brown, yellow, red, green, blue,
                    gray, violet, hazel, unknown, dontcare};
        enum MARKINGS {dont_care, tattoo, scar, missing_limb,
                       none};
        struct person_ {
               string        name;
               Address       addr;
               string        ss;
               SEX           sex;
               short         age;
               MARRIAGE      mar;
               HOBBIES       rec;
               Date          dob;
               short         ht;
               long          wt;
               COLOR         hair;
               COLOR         eye;
               COLOR         skin;
               MARKINGS      other;
        };
typedef person_ Person;
        typedef sequence <Person> Possibles;
        union reason_ switch (short)
                {
                case 0:      string      name;
                case 1:      Address     addr;
                case 2:      string      ss;
                case 3:      SEX         sex;
                case 4:      short       age;
                case 5:      MARRIAGE    mar;
                case 6:      HOBBIES     rec;
                case 7:      Date        dob;
                case 8:      short       ht;
                case 9:      long        wt;
```

```
            case 10:     COLOR       hair;
            case 11:     COLOR       eyes;
            case 12:     COLOR       skin;
            case 13:     MARKINGS    other;
 };
 typedef reason_ Reason;

 exception DataOutOfRange
 {

        Reason why;
 };
boolean   findPerson (
          in Person who, out Possibles hits)
          raises (DataOutOfRange);
boolean   findPersonByName (
          in string name, out Possibles hits)
          raises (DataOutOfRange);
boolean   findPersonByAddress (
          in Address addr, out Possibles hits)
          raises (DataOutOfRange);
boolean   findPersonBySS (
          in string ss, out Possibles hits)
          raises (DataOutOfRange);
boolean   findPersonByAge (
          in short age, out Possibles hits)
          raises (DataOutOfRange);
boolean   findPersonByMarriage (
          in MARRIAGE mar, out Possibles hits)
          raises (DataOutOfRange);
boolean   findPersonByHobbies (
          in HOBBIES rec, out Possibles hits)
          raises (DataOutOfRange);
boolean   findPersonBydob (
          in Date dob, out Possibles hits)
          raises (DataOutOfRange);
boolean   findPersonByHeight (
          in short ht, out Possibles hits)
          raises (DataOutOfRange);
boolean   findPersonByWeight (
          in long wt, out Possibles hits)
          raises (DataOutOfRange);
boolean   findPersonByHairColor (
          in COLOR col, out Possibles hits)
          raises (DataOutOfRange);
boolean   findPersonBySkinColor (
          in COLOR col, out Possibles hits)
```

```
                raises (DataOutOfRange);
        boolean   findPersonByEyeColor (
                in COLOR col, out Possibles hits)
                raises (DataOutOfRange);
        boolean   findPersonByOther (
                in MARKINGS other, out Possibles hits)
                raises (DataOutOfRange);
        void      exit();
};
interface QueryFactory
{

        PersonQuery createQuery (in string name);
};
```

PersonQuery サンプル・アプリケーションのビルドと実行

PersonQuery 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. PersonQuery 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. PersonQuery サンプル・アプリケーションのファイルの保護を変更します。
3. 環境変数を設定します。
4. CORBA クライアントおよびサーバのサンプル・アプリケーションをビルドします。
5. PersonQuery クライアントおよびサーバ・アプリケーションを起動します。
6. クライアント・アプリケーションを使用して、サーバ上のデータベースを検索するためのいくつかのコマンドを入力します。
7. PersonQuery サンプル・アプリケーションを停止します。

PersonQuery サンプル・アプリケーション用ファイルのコピー

要求レベルのインターセプタのサンプル・アプリケーション用ファイルは、次のディレクトリに格納されています。

```
$TUXDIR\samples\corba\interceptors_cxx
```

これらのファイルをビルドおよび実行できるようにコピーするには、次の手順に従います。

1. サンプル・ファイルのコピー先となる作業ディレクトリを作成します。
2. 1 つ前の手順で作成した作業ディレクトリに interceptors_cxx サンプルをコピーします。

Windows 2000

```
> xcopy /s/i %TUXDIR%\samples\corba\interceptors_cxx <workdirectory>\cxx
```

UNIX

```
> cp -R $TUXDIR/samples/corba/interceptors_cxx <workdirectory>/cxx
```

3. サンプル・ファイルが格納された作業ディレクトリに移動します。

Windows 2000

```
> cd <workdirectory>\cxx
```

UNIX

```
> cd <workdirectory>/cxx
```

PersonQuery サンプル・アプリケーションには 表 4-2 でリストおよび説明されているファイルを使用します。

表 4-2 インターセプタのサンプル・アプリケーションに含まれるファイル

ディレクトリ	ファイル	説明
app_cxx (interceptors_cxx のサブディレクトリ)	Readme.txt	インターセプタのサンプル・アプリケーション一式のビルドと実行に関する最新情報が記載されたファイル。
	makefile.mk	UNIX システムでインターセプタのサンプル・アプリケーション一式すべて (PersonQuery アプリケーションおよび全サンプル・インターセプタ) をビルドするための makefile。
	makefile.nt	Windows 2000 システムでインターセプタのサンプル・アプリケーション一式すべて (PersonQuery アプリケーションおよび全サンプル・インターセプタ) をビルドするための makefile。
	makefile.inc	適切な *platform*.inc ファイルで定義されるマクロを使用する、一般的な makefile。
	personquery_i.h and personquery_i.cpp	PersonQuery インターフェイスのインプリメンテーション。
	personqueryc.cpp	PersonQuery アプリケーションのソース・ファイル。
	personquerys.cpp	PersonQuery データベース・サーバのソース・ファイル。
	setenv.ksh	UNIX システムでインターセプタのサンプル・アプリケーション一式すべてをビルドするために必要な全環境変数を設定するシェル・ファイル。
	setenv.cmd	Windows 2000 システムでインターセプタのサンプル・アプリケーション一式すべてをビルドするために必要な全環境変数を設定するコマンド・ファイル。

表 4-2 インターセプタのサンプル・アプリケーションに含まれるファイル（続き）

ディレクトリ	ファイル	説明
data_cxx (interceptors_cxx のサブディレクトリ)	InterceptorData.cpp	InterceptorData C++ ソース・ファイル。
	InterceptorData.h	InterceptorData クラスの定義ファイル。
	makefile.inc	InterceptorData インターセプタのビルドに、適切な *platform*.inc ファイルで定義されたマクロを使用する、一般的な makefile。
	makefile.mk	UNIX システムで InterceptorData インターセプタをビルドする makefile。
	makefile.nt	Windows 2000 システムで InterceptorData インターセプタをビルドする makefile。
simple_cxx(interceptors_cxx のサブディレクトリ)	InterceptorSimp.cpp	InterceptorData C++ ソース・ファイル。
	InterceptorSimp.h	InterceptorSimp クラスの定義ファイル。
	makefile.inc	InterceptorSimp インターセプタのビルドに、適切な *platform*.inc ファイルで定義されたマクロを使用する、一般的な makefile。
	makefile.mk	UNIX システムで InterceptorSimp インターセプタをビルドする makefile。
	makefile.nt	Windows 2000 システムで InterceptorSimp インターセプタをビルドする makefile。

表 4-2 インターセプタのサンプル・アプリケーションに含まれるファイル（続き）

ディレクトリ	ファイル	説明
security_cxx(interceptors_cxx のサブディレクトリ)	InterceptorSec.cpp	InterceptorSec C++ ソース・ファイル。
	InterceptorSec.h	InterceptorSec クラスの定義ファイル。
	makefile.inc	InterceptorSec インターセプタのビルドに、適切な *platform*.inc ファイルで定義されたマクロを使用する、一般的な makefile。
	makefile.mk	UNIX システムで InterceptorSec インターセプタをビルドする makefile。
	makefile.nt	Windows 2000 システムで InterceptorSec インターセプタをビルドする makefile。
common (interceptors_cxx のサブディレクトリ)	app.inc	アプリケーションのコンフィギュレーションのための makefile 定義を含むファイル。
	platform.inc	インターセプタのサンプル・アプリケーション一式をビルドするための、プラットフォーム固有の make 定義が含まれるファイル。*platform* は、使用しているマシンのシステム・プラットフォームを表します。
	common.mk	UNIX システムの makefile 定義が含まれるファイル。
	makefile.inc	適切な *platform*.inc ファイルで定義されるマクロを使用する、一般的な makefile。
	makefile.mk	UNIX システムでサンプル・アプリケーション用ファイル一式すべてをビルドする makefile。
	makefile.nt	Windows システムでサンプル・アプリケーション用ファイル一式すべてをビルドする makefile。
	personquery.idl	PersonQuery サンプル・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。

PersonQuery アプリケーション・ファイルに対する保護の変更

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。PersonQuery 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できるようにするには、次のように作業ディレクトリにコピーしたファイルの保護属性を変更する必要があります。まず、サンプル・アプリケーションのファイルをコピーしたディレクトリにアクセスしていることを確認します。

Windows 2000

```
prompt>attrib -r /s *.*
```

UNIX

```
prompt>/bin/ksh
ksh prompt>chmod -R u+w *.*
```

環境変数の設定

PersonQuery サンプル・アプリケーションをビルドして実行する前に、アプリケーションが実行される環境を設定する必要があります。PersonQuery サンプル・アプリケーションのビルドと実行に必要な環境変数およびプロパティの設定を行うには、次のコマンドを入力します。

Windows 2000

```
> setenv.cmd
```

UNIX:

```
> $ . ./setenv.ksh
```

CORBA クライアントおよびサーバ・アプリケーションのビルド

次のコマンドにより、PersonQuery アプリケーションをビルドし、マシン固有の UBBCONFIG ファイルを作成し、UBBCONFIG ファイルをロードできます。

Windows 2000

```
> nmake -f makefile.nt
```

UNIX

```
$ make -f makefile.mk
```

注記 便宜上、この手順で実行される makefile によってインターセプタのサンプル一式すべてをビルドします。これには、InterceptorSimp、InterceptorSec、および InterceptorData インターセプタも含まれます。これらのインターセプタのインプリメントとビルド、および PersonQuery サンプル・アプリケーションによる実行については、以降の章で説明します。

PersonQuery クライアントおよびサーバ・アプリケーションの起動

次のコマンドを入力して、PersonQuery サンプル・アプリケーションを起動します。

```
prompt> tmboot -y
```

PersonQuery サンプル・アプリケーションの実行

PersonQuery サンプル・アプリケーションの代表的な使用方法として、次の手順に従います。

1. 次の例のように、1 つの特性について問い合せコマンドを入力し、返された項目数を確認します。

   ```
   Option? hair brown eyes blue
   ```

2. 1 つ前の手順で問い合せを行った特性に関し、さらに問い合せデータを入力します。
3. すべての問い合せデータが必要なレベルに絞られるまで、問い合せを続行します。
4. `result` コマンドを入力して、最終的な問い合せ結果を表示します。
5. 新しい問い合せサイクルを開始します。
6. `exit` コマンドを入力して、アプリケーションを終了します。

PersonQuery サンプル・アプリケーションの停止

PersonQuery サンプル・アプリケーションを停止するには、次のコマンドを入力します。

```
prompt>tmshutdown -y
```

5 InterceptorSimp サンプル・インターセプタ

ここでは、次の内容について説明します。

- PersonQuery サンプル・インターセプタのしくみ
- PersonQuery インターセプタの登録および実行
- インターセプタの出力の検証
- インターセプタの登録解除

この章で説明する手順を実行してみる前に、「第 4 章 PersonQuery サンプル・アプリケーション」で解説した手順がすべて完了していることを確認してください。

PersonQuery サンプル・インターセプタのしくみ

InterceptorSimp サンプル・インターセプタでは、インターセプタに渡された要求におけるオペレーションに RequestContext オブジェクトを介してアクセスする方法を示します。InterceptorSimp サンプルが要求をインターセプトすると、インターセプタは次の処理を行います。

- データ・ファイルからオペレーション名を書き出します。ただし、要求のパラメータを解釈したり変更したりはしません。
- インターセプタ・メソッドから適切なステータスを返します。

インターセプタに対して正常な呼び出しが行われた場合、クライアント呼び出しはターゲット・オブジェクトに渡されて通常の方法で処理されます。
InterceptorSimp サンプル・インターセプタでは、次のことが示されます。

- 基本的な監視サービスのインプリメンテーション。これは、呼び出されたターゲット・オブジェクトの各オペレーションをトラッキングするだけのサービスです。
- インターセプタが、ORB によってインターセプタ・メソッドに渡されたパラメータにアクセスして、要求に含まれるオペレーションを識別する方法。

InterceptorSimp サンプル・インターセプタでは、定義および登録されるものの単一のソース・ファイル内でインプリメントされる 2 種類のインターセプタも示されます。この例では、クライアント・インターセプタとターゲット・インターセプタは個別に登録されます。先に初期化されるのは、クライアント・インターセプタです。

PersonQuery インターセプタの登録および実行

「第 4 章 PersonQuery サンプル・アプリケーション」の PersonQuery サンプル・アプリケーションをビルドする makefile を実行すると、InterceptorSimp インターセプタを含むサンプル・インターセプター式全部が、同様にビルドされます。この節では、実行時に PersonQuery アプリケーションと共に機能するように InterceptorSimp インターセプタを登録する方法を説明します。

InterceptorSimp クライアント・インターセプタおよびサーバ・インターセプタを登録して実行するには、次の手順に従います。

1. InterceptorSimp サンプルのディレクトリに移動します。ここで *workdirectory* は、「第 4 章 PersonQuery サンプル・アプリケーション」に示す手順でインターセプタのサンプル・アプリケーションをコピーしたディレクトリの名前を表します。

   Windows 2000

   ```
   > cd <workdirectory>\cxx\simple_cxx
   ```

   UNIX

   ```
   $ cd <workdirectory>/cxx/simple_cxx
   ```

2. インターセプタを登録します。

   Windows 2000

   ```
   > nmake -f makefile.nt config
   ```

   UNIX

   ```
   $ make -f makefile.mk config
   ```

3. CORBA サーバをブートしてクライアントを実行します。

   Windows 2000

   ```
   > cd <workdirectory>\cxx\app_cxx
   > tmboot -y
   > PersonQueryClient
   ```

   UNIX

   ```
   > cd <workdirectory>/cxx/app_cxx
   > tmboot -y
   > PersonQueryClient
   ```

4. 「第 4 章 PersonQuery サンプル・アプリケーション」で説明したコマンド構文を使用して、PersonQuery クライアント・アプリケーションで呼び出しを任意の回数行います。

5. PersonQuery アプリケーションを停止します。

   ```
   > tmshutdown -y
   ```

インターセプタの出力の検証

単純なクライアント・インターセプタからの出力は、次の構文で名前を付けられたファイルに入っています。

`InterceptorSimpClient`*xxxx*`.out`

上述の構文の行で、*xxxx* はインターセプタが実行された実行可能ファイルのプロセス ID を表します。たとえば、`InterceptorSimpClient`*xxx*`.out` ファイルは以下のそれぞれについて 1 つずつ、全部で 3 つあります。

- FactoryFinder, TMFFNAME
- PersonQueryServer
- PersonQueryClient

各ファイルの内容は、ORB と実行可能ファイルがどのようにやり取りしたかに応じて変わります。たとえば、ターゲット・インターセプタはサーバ上で実行され、クライアント・インターセプタはクライアント上で実行されるので、InterceptorSimpClient ログ・ファイルに記録されるターゲット・インターセプタからの出力は通常、非常に少ない一方で、クライアント・インターセプタからの出力はより多くなっています。

インターセプタの登録解除

InterceptorSimp サンプル・インターセプタで PersonQuery サンプル・アプリケーションを実行後に、次の手順によってこれらのインターセプタの登録を解除できます。

1. 次のコマンドを入力して、実行中の CORBA アプリケーションをすべてシャットダウンします。

   ```
   > tmshutdown -y
   ```

2. インターセプタの登録を解除します。

インターセプタの登録解除

InterceptorSimp クライアント・インターセプタおよびサーバ・インターセプタの登録を解除するには、次の手順に従います。

1. InterceptorSimp サンプルのディレクトリに移動します。ここで *workdirectory* は、「第 4 章 PersonQuery サンプル・アプリケーション」に示す手順でインターセプタのサンプル・アプリケーションをコピーしたディレクトリの名前を表します。

   Windows 2000

   ```
   > cd <workdirectory>\cxx\simple_cxx
   ```

   UNIX

   ```
   $ cd <workdirectory>/cxx/simple_cxx
   ```

2. インターセプタの登録を解除します。

Windows 2000

```
> nmake -f makefile.nt unconfig
```

UNIX

```
$ make -f makefile.mk unconfig
```

6 InterceptorSec サンプル・インターセプタ

ここでは、次の内容について説明します。

- PersonQuery サンプル・インターセプタのしくみ
- PersonQuery インターセプタの登録および実行
- インターセプタ出力の検証
- インターセプタの登録解除

この章で説明する手順を実行してみる前に、「第 4 章 PersonQuery サンプル・アプリケーション」で解説した手順がすべて完了していることを確認してください。

PersonQuery サンプル・インターセプタのしくみ

InterceptorSec サンプル・インターセプタにより、基本的なセキュリティ・モデルをインプリメントする単純なクライアント / サーバ・インターセプタのペアが示されます。InterceptorSec クライアント側インターセプタは単に、ORB で処理される各クライアント要求をログに記録します。InterceptorSec ターゲット側インターセプタは、クライアント・アプリケーションのユーザが要求のオペレーションの実行を認可されているかどうかを確認する、単純なセキュリティ・メカニズムをインプリメントします。

InterceptorSec サンプル・インターセプタでは、単一の初期化関数で初期化され、単一のライブラリにインプリメントされた、クライアントおよびターゲット・インターセプタのペアが示されます。単一の初期化関数が呼び出されるので、インターセプタ登録コマンドで登録されるのは初期化関数 1 つとインプリメンテーション・ライブラリ 1 つです。

InterceptorSec ターゲット側インターセプタのしくみ

ターゲット側 ORB で要求が受信されると、ORB は InterceptorSec ターゲット側インターセプタを呼び出し、クライアント要求からの RequestContext および DataInputStream オブジェクトを渡します。

ターゲット側インターセプタはその後、次の処理により、クライアント・アプリケーションのユーザに対して要求に含まれるオペレーションの認可を行います。

1. 要求が PersonQuery インターフェイスに対する呼び出しであるかどうかを確認します。違っていた場合、インターセプタは INVOKE_NO_EXCEPTION を返します。
2. 要求に含まれるオペレーションが PersonQuery インターフェイスに対する呼び出しであった場合、インターセプタは次の処理を行います。
   a. SecurityCurrent オブジェクトへのリファレンスを取得します。その後、このオブジェクトはインターセプタによってナロー変換されます。
   b. SecurityContext オブジェクトを呼び出し、クライアント・アプリケーションのユーザの属性リストを要求します。
   c. 属性リストの中から、2 つの属性を取得します。

PrimaryGroupId	クライアント・アプリケーションのユーザのクライアント名を識別します。このインターセプタでは、クライアント名には文字 r と NULL 文字列のどちらかが含まれている必要があります。
AccessId	クライアント・アプリケーションのユーザを識別します。このインターセプタでは、ユーザ名に文字 R、P 、または N が入っている必要があります (大文字でも小文字でも可)。

   d. ユーザを PrimaryGroupId および AccessId と照合します。ユーザがこれら 2 つの属性の判定基準に問題なく一致していた場合、インターセプタは INVOKE_NO_EXCEPTION を返します。

e. 一致していなければ、インターセプタは REPLY_EXCEPTION を返します。これにより、ターゲット・オブジェクトへの要求の送信は回避されます。その代わりに、ORB はクライアント・アプリケーションに例外を返します。

以降の節ではインターセプタのセキュリティについて説明し、InterceptorSec ターゲット側インターセプタの中から、該当するコードの抜粋部分を示します。

SecurityCurrent オブジェクトの使用

インターセプタは、Bootstrap オブジェクトではなく、ORB から SecurityCurrent オブジェクトを取得します。ORB から取得可能な SecurityCurrent オブジェクトには、インターセプタがクライアントの情報を取得するために必要とする API が含まれます。

インターセプタで ORB に対する `resolve_initial_references("SecurityCurrent")` オペレーションを呼び出すと、SecurityCurrent オブジェクトを取得できます。インターセプタは次に、SecurityCurrent リファレンスを `SecurityCurrentLevel1` カレントにナロー変換します。

SecurityCurrent オブジェクトの取得

SecurityCurrent オブジェクトは、ORB からのみ取得可能です。このオブジェクトの主要な機能は、CORBA サーバ・アプリケーションがクライアント呼び出し関連の属性にアクセスできるようにすることです。

ORB の `resolve_initial_references("SecurityCurrent")` メソッドは、インターセプタに SecurityCurrent オブジェクトのリファレンスを提供します。このオブジェクトから、インターセプタにレベル 1 のセキュリティ機能が与えられます。インターセプタは、SecurityCurrent オブジェクトの `get_attributes` メソッドを介してクライアント呼び出しの属性を取得します。このオブジェクトは、属性リストをインターセプタに返します。属性リストには、インターセプトされている呼び出しを実行したクライアント・アプリケーションのユーザに関連する属性が含まれます。上記の例外を除けば、CORBA セキュリティ・サービスからのどのメソッドの振る舞いも同じです。

次の C++ コードの抜粋では、SecurityCurrent オブジェクトの取得方法を示します。

```
try
        {
            sec_current = m_orb->resolve_initial_references("SecurityCurrent");
        }
        catch (...)
        {
            *m_outfile <<
                    "ERROR: ORB::resolve_initial_references threw exception"
                    << endl << endl << flush;
            excep_val = new CORBA::UNKNOWN();
            return Interceptors::REPLY_EXCEPTION;
        }
        if  (CORBA::is_nil(sec_current.in()))
        {
            *m_outfile << "ERROR: No SecurityCurrent present"
                    << endl << endl << flush;
            excep_val = new CORBA::NO_PERMISSION();
            return Interceptors::REPLY_EXCEPTION;
        }

        m_security_current = SecurityLevel1::Current::_narrow(sec_current.in());
        if  (!m_security_current)
        {
            *m_outfile << "ERROR: Couldn't narrow security
                           current to SecurityLevel1::Current"
                    << endl << endl << flush;
            excep_val = new CORBA::NO_PERMISSION();
            return Interceptors::REPLY_EXCEPTION;
        }
```

ユーザ属性リストの作成

この節のコード抜粋部分は、InterceptorSec ターゲット側インターセプタが、ユーザ属性のリストを作成し、その後このリスト内を調べて、ユーザが認可基準に適合しているかどうかを判定する方法を示します。

InterceptorSec サンプルでは、属性リストの作成と、そのリスト内容の検討は、個別の手順で行われます。長さ 0 のクライアント属性リストが返るように指定を行うと、SecurityCurrent オブジェクトはそのクライアントの全属性を返すことに注意してください。

```
// 認可チェックを行う必要のある情報に対応する
// 属性を取得する
//      PrimaryGroupId（ログインしたクライアントのクライアント名）
//      AccessId（ログインしたクライアントのユーザ名）
Security::AttributeList_var client_attr = 0;
try
{
         client_attr = m_security_current->get_attributes(*m_attributes_to_get);
```

次の抜粋部分では、リストの作成方法を示します。

```
   Security::AttributeTypeList_var attr = new Security::AttributeTypeList(2);
    if (!attr.ptr())
    {
        cout <<
                "ERROR: can't allocation security list: Out of memory"
                << endl << endl << flush;
        return;
    }
    attr.inout().length(2);
    attr[(CORBA::ULong)0].attribute_family.family_definer = 0;
    attr[(CORBA::ULong)0].attribute_family.family = 1;
    attr[(CORBA::ULong)0].attribute_type = Security::PrimaryGroupId;
    attr[(CORBA::ULong)1].attribute_family.family_definer = 0;
    attr[(CORBA::ULong)1].attribute_family.family = 1;
    attr[(CORBA::ULong)1].attribute_type = Security::AccessId;
    m_attributes_to_get = attr._retn();
    return;
```

次の抜粋部分は、属性リストでユーザが認可基準に適合するかどうかを確認する方法を示します。

```
if  (client_attr[i].attribute_type.attribute_type == Security::PrimaryGroupId)
           {
               //
               // この属性はクライアント名。
               // 何らかのクライアント名値と比較する。
               // たとえば、「r」を含むすべての文字列か、
               // NULL 文字列を受け入れるとする。その場合、クライアント名を、
               // 認可済みの値のセットのどれかに比較する
```

```
            //
            if  ((strlen(value_buffer) == 0) ||
                 (strchr(value_buffer, 'r') != 0))
            {
                *m_outfile << "    INFO: Valid client name found: "
                           << value_buffer << endl;
                clientname_ok = 1;
            }
            else
            {
                *m_outfile << "    ERROR: Invalid client name found: "
                       << value_buffer << endl;
            }
        }
else if (client_attr[i].attribute_type.attribute_type == Security::AccessId)
        {
            // この属性はユーザ名。ユーザ ID 中に「r」、「n」
            // または「p」が含まれるユーザであれば、任意に選択し、
            // 認可する。別の認可基準を
            // 選択することもできる
            //
            if  ((strchr(value_buffer, 'r') != 0) ||
                 (strchr(value_buffer, 'R') != 0) ||
                 (strchr(value_buffer, 'P') != 0) ||
                 (strchr(value_buffer, 'p') != 0) ||
                 (strchr(value_buffer, 'N') != 0) ||
                 (strchr(value_buffer, 'n') != 0))
            {
                *m_outfile << "    INFO: Valid username found: "
                           << value_buffer << endl;
                username_ok = 1;
            }
```

PersonQuery インターセプタの登録および実行

「第 4 章 PersonQuery サンプル・アプリケーション」の PersonQuery サンプル・アプリケーションをビルドする makefile を実行すると、InterceptorSec インターセプタを含むサンプル・インターセプター式全部が、同様にビルドされます。この節では、実行時に PersonQuery アプリケーションと共に機能するように InterceptorSec インターセプタを登録する方法を説明します。

InterceptorSec クライアント・インターセプタおよびサーバ・インターセプタを登録して実行するには、以下の手順に従います。

1. InterceptorSec サンプルのディレクトリに移動します。ここで *workdirectory* は、「第 4 章 PersonQuery サンプル・アプリケーション」に示す手順でインターセプタのサンプル・アプリケーションをコピーしたディレクトリの名前を表します。

   Windows 2000

   ```
   > cd <workdirectory>\cxx\security_cxx
   ```

   UNIX

   ```
   $ cd <workdirectory>/cxx/security_cxx
   ```

2. インターセプタを登録します。

   Windows 2000

   ```
   > nmake -f makefile.nt config
   ```

   UNIX

   ```
   $ make -f makefile.mk config
   ```

3. CORBA サーバをブートして CORBA クライアントを実行します。

   Windows 2000

   ```
   > cd <workdirectory>\cxx\app_cxx
   > tmboot -y
   > PersonQueryClient
   ```

UNIX

```
> cd <workdirectory>/cxx/app_cxx
> tmboot -y
> PersonQueryClient
```

4. 「第 4 章 PersonQuery サンプル・アプリケーション」で説明したコマンド構文を使用して、PersonQuery クライアント・アプリケーションで呼び出しを任意の回数行います。

5. PersonQuery アプリケーションを停止します。

```
> tmshutdown -y
```

インターセプタ出力の検証

InterceptorSec クライアント・インターセプタおよびターゲット・インターセプタは、指定されたファイルに出力のログを記録します。ファイル名はそれぞれ、`InterceptorSecClientxxx.out` および `InterceptorSecTargetxxx.out` です。これらのファイルには、インターセプタからのデバッグ出力が含まれます。これは PersonQuery アプリケーション用に、ORB によって自動的にロードおよび実行されます。

インターセプタの登録解除

InterceptorSec サンプル・インターセプタで PersonQuery サンプル・アプリケーションを実行後に、次の手順でこれらのインターセプタの登録を解除できます。

1. 次のコマンドを入力して、実行中の CORBA アプリケーションをすべてシャットダウンします。

```
> tmshutdown -y
```

2. InterceptorSec サンプルのディレクトリに移動します。ここで *workdirectory* は、「第 4 章 PersonQuery サンプル・アプリケーション」に示す手順でインターセプタのサンプル・アプリケーションをコピーしたディレクトリの名前を表します。

Windows 2000

```
> cd <workdirectory>\cxx\security_cxx
```

UNIX

```
$ cd <workdirectory>/cxx/security_cxx
```

3. インターセプタの登録を解除します。

Windows 2000

```
> nmake -f makefile.nt unconfig
```

UNIX

```
$ make -f makefile.mk unconfig
```

7 InterceptorData サンプル・インターセプタ

この章では、PersonQuery サンプル・アプリケーションで使用するために設計されている、次の 2 つのサンプル・インターセプタについて説明します。

- PersonQuery クライアント・コンポーネントのホスト・マシン上にインストールされる InterceptorDataClient インターセプタ
- PersonQuery サーバ・コンポーネントのホスト・マシン上にインストールされる InterceptorDataTarget インターセプタ

ここでは、各インターセプタのしくみについて説明し、次にそれらを PersonQuery サンプル・アプリケーションでビルドして実行する方法を示します。

InterceptorDataClient インターセプタ

InterceptorDataClient インターセプタは、各クライアント・アプリケーションの要求パラメータと応答パラメータをインターセプトしてログに記録します。このインターセプタはまた、PersonQuery サーバ・アプリケーションに対するある特定のオペレーションを、クライアント・アプリケーションの特定の基準に一致するユーザが呼び出せるようにします。InterceptorDataClient インターセプタは、`ClientRequestInterceptor` クラスから継承される `InterceptorDataClient` インターフェイスをインプリメントします。

InterceptorDataClient クラスは、次のメソッドをインプリメントします。

- `id()`

  このメソッドは、文字列 `InterceptorDataClient` を返します。

- `shutdown()`

  このメソッドは `tracker` オブジェクトからの要求を削除します。

- exception_occurred()

  このメソッドは、ORB によって呼び出されると、tracker オブジェクトから要求を削除します。

- client_invoke()

  このメソッドは、インターフェイスおよびオペレーションが「関心の対象」であるかどうかを判断します。クライアント要求が「関心の対象」である場合、このメソッドは要求パラメータを解析して、パラメータをログ・ファイルに出力します。クライアント要求が「関心の対象」ではない場合、メソッドは単に復帰します。

- client_response()

  このメソッドは、要求内のインターフェイスおよびオペレーションが「関心の対象」であるかどうかを判断します。インターフェイスおよびオペレーションが「関心の対象」である場合、メソッドは CORBA DataInputStream パラメータの中から応答パラメータを取得して、それをログ・ファイルに書き込みます。要求内のインターフェイスおよびオペレーションが「関心の対象」ではない場合、メソッドは単に復帰します。

さらに、データ・インターセプタが InterceptorDataClientInit メソッドを提供して、クライアント・インターセプタ・クラスを初期化します。

InterceptorDataTarget インターセプタ

InterceptorDataTarget インターセプタは要求および応答データのパラメータをインターセプトしてログに記録します。このインターセプタはまた、x の文字でデータをマスキングして特定の応答パラメータから、機密データを削除します。InterceptorDataTarget インターセプタは、InterceptorDataTarget クラスから継承される InterceptorDataTarget インターフェイスをインプリメントします。

InterceptorDataTarget クラスは、次のメソッドをインプリメントします。

- id()

  このメソッドは、文字列 InterceptorDataTarget を返します。

- shutdown()

  このメソッドは、単に復帰します。

- exception_occurred()

  このメソッドは tracker オブジェクトからの要求を削除します。

- target_invoke()

  このメソッドは、インターフェイスおよびオペレーションが「関心の対象」であるかどうかを判断します。「関心の対象」である場合、このメソッドは要求パラメータを解析して、そのデータをログ・ファイルに出力します。要求内のインターフェイスおよびオペレーションが「関心の対象」ではない場合、メソッドは単に復帰します。要求内のオペレーションが exit である場合、このメソッドはステータス値 INVOKE_NO_EXCEPTION を返します。

- target_response()

  このメソッドは、インターフェイスおよびオペレーションが「関心の対象」であるかどうかを判断します。「関心の対象」である場合、このメソッドは DataInputStream パラメータの中から、応答パラメータを取得してログ・ファイルに出力します。機密データは、ログ内では置換されます。たとえば、個人の社会保障番号は、ログには出力されません。要求内のインターフェイスおよびオペレーションが「関心の対象」ではない場合、メソッドは単に復帰します。

さらに、データ・インターセプタが InterceptorDataTargetInit メソッドを提供して、ターゲット・インターセプタ・クラスを初期化します。

InterceptorData インターセプタのインプリメント

InterceptorData インターセプタをインプリメントするために使用されるコードについては、「第 2 章 CORBA 要求レベルのインターセプタの開発」で示しています。次の操作を行う方法については、そちらを参照してください。

1. 第 2 章の 3 ページ「インプリメンテーション・ファイルの起動」
2. 第 2 章の 4 ページ「実行時のインターセプタの初期化」
3. 第 2 章の 5 ページ「要求からのインターセプタ名の取得」
4. 第 2 章の 6 ページ「要求内のオペレーションの識別」
5. 第 2 章の 7 ページ「インターセプタの応答オペレーションのインプリメント」
6. 第 2 章の 8 ページ「データ入力ストリームからのパラメータの読み出し」

InterceptorData インターセプタの登録および実行

「第 4 章 PersonQuery サンプル・アプリケーション」で説明した PersonQuery サンプル・アプリケーションをビルドする makefile を実行すると、InterceptorData インターセプタを含むサンプル・インターセプター式全部が、同様にビルドされます。この節では、実行時に PersonQuery アプリケーションと共に機能するように InterceptorData インターセプタを登録する方法を説明します。

InterceptorData クライアント・インターセプタおよびサーバ・インターセプタを登録して実行するには、次の手順に従います。

1. InterceptorData サンプルのディレクトリに移動します。ここで *workdirectory* は、「第 4 章 PersonQuery サンプル・アプリケーション」に示す手順でインターセプタのサンプル・アプリケーションをコピーしたディレクトリの名前を表します。

   Windows 2000

   ```
   > cd <workdirectory>\cxx\data_cxx
   ```

   UNIX

   ```
   $ cd <workdirectory>/cxx/data_cxx
   ```

2. インターセプタを登録します。

   Windows 2000

   ```
   > nmake -f makefile.nt config
   ```

UNIX

```
$ make -f makefile.mk config
```

3. CORBA サーバをブートして CORBA クライアントを実行します。

Windows 200

```
> cd <workdirectory>\cxx\app_cxx
> tmboot -y
> PersonQueryClient
```

UNIX

```
> cd <workdirectory>/cxx/app_cxx
> tmboot -y
> PersonQueryClient
```

4. 「第 4 章 PersonQuery サンプル・アプリケーション」で説明したコマンド構文を使用して、PersonQuery クライアント・アプリケーションで呼び出しを任意の回数行います。

5. PersonQuery アプリケーションを停止します。

```
> tmshutdown -y
```

インターセプタ出力の検証

InterceptorData クライアント・インターセプタおよびターゲット・インターセプタは、各呼び出しをログに記録します。PersonQuery アプリケーションの各セッションについて、クライアント・インターセプタは `InterceptorDataClient`*xxx*`.out` という名前のファイルを作成し、ターゲット・インターセプタは `InterceptorDataTarget`*xxx*`.out` という名前のファイルを作成します。ここでは、各インターセプタのログ・ファイルのサンプルを示します。

クライアント・インターセプタのログ出力例

```
InterceptorDataClientInit called
ClientInterceptorData::id called

ClientInterceptorData::client_invoke called
ClientInterceptorData::client_response called
   Request Id:     1
   unable to find request for this reply (must not be one we care about)
```

```
ClientInterceptorData::client_invoke called
   Request Id:     2
   Interface:      IDL:beasys.com/PersonQuery:1.0
   Operation:      findPerson
   Parameters:
                           name:           ALISTER LANCASHIRE
                           address:        3 PENNY LANE
                                           LONDON GB UK
                           ss:             999-99-9999
                           sex:            can't tell
                           age(yrs.):      85
                           marital status: single
                           hobby:          stamp collecting
                           date-of-birth:  11/25/1913
                           height(in.):    32
                           weight(lbs.):   57
                           hair color:     unknown
                           eye color:      blue
                           skin color:     white
                           other markings: missing limb
```

ターゲット・インターセプタのログ出力例

```
InterceptorDataTargetInit called
TargetInterceptorData::id called

TargetInterceptorData::target_response called
   Request Id:     2
   ReplyStatus:    GIOP::NO_EXCEPTION
   Interface:      IDL:beasys.com/PersonQuery:1.0
   Operation:      findPerson
   Method Result:  TRUE
   Parameters:
                   Maximum: 8
                   Length:  8

                   Item 0
                           name:           ALISTER LANCASHIRE
                           address:        3 PENNY LANE
                                           LONDON GB UK
                           ss:             NO PRIVILEDGE
                           sex:            NO PRIVILEDGE
                           age (years):    NO PRIVILEDGE
                           marital status: NO PRIVILEDGE
```

```
hobby:           stamp collecting
date-of-birth:   NO PRIVILEDGE
height (in.):    32
weight (lbs.):   57
hair color:      unknown
eye color:       blue

skin color:      NO PRIVILEDGE
other markings: missing limb
```

インターセプタの登録解除

InterceptorData サンプル・インターセプタで PersonQuery サンプル・アプリケーションを実行後に、次の手順でこれらのインターセプタの登録を解除できます。

1. 次のコマンドを入力して、実行中の CORBA アプリケーションをすべてシャットダウンします。

   ```
   > tmshutdown -y
   ```

2. InterceptorData サンプルのディレクトリに移動します。ここで *workdirectory* は、「第 4 章 PersonQuery サンプル・アプリケーション」に示す手順でインターセプタのサンプル・アプリケーションをコピーしたディレクトリの名前を表します。

   Windows 2000

   ```
   > cd <workdirectory>\cxx\data_cxx
   ```

   UNIX

   ```
   $ cd <workdirectory>/cxx/data_cxx
   ```

3. インターセプタの登録を解除します。

   Windows 2000

   ```
   > nmake -f makefile.nt unconfig
   ```

   UNIX

   ```
   $ make -f makefile.mk unconfig
   ```

8 要求レベルのインターセプタのAPI

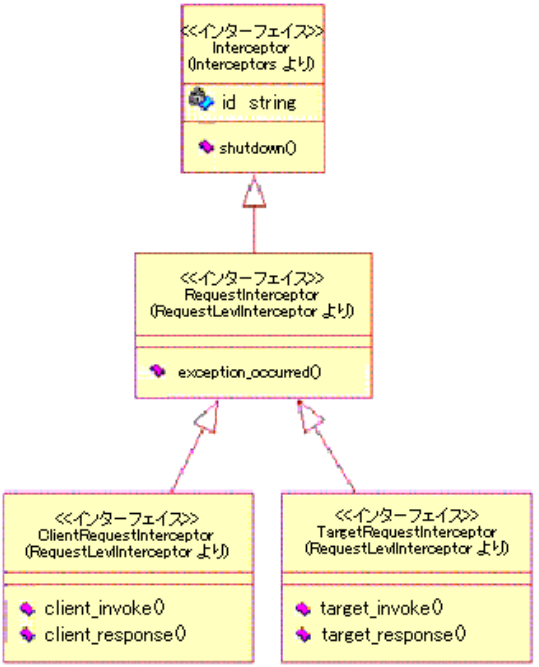
この章では、要求レベルのインターセプタのインプリメントに使用する次のインターフェイスについて説明します。

- `Interceptors::Interceptor`
- `RequestLevelInterceptor::RequestInterceptor`
- `RequestLevelInterceptor::ClientRequestInterceptor`
- `RequestLevelInterceptor::TargetRequestInterceptor`
- `CORBA::DataInputStream`

これらのインターフェイスは、**位置制約付きオブジェクト**です。リファレンスをその位置 (プロセス) の外部に渡そうとしたり、CORBA ORB `object_to_string` オペレーションを使用して、このインターフェイスをサポートしているオブジェクトを外部化しようとしたりすると、CORBA `MARSHAL` システム例外 (`CORBA::MARSHAL`) が発生します。

インターセプタの階層構造

要求レベルのインターセプタは、2 つのインターフェイスに分かれており、クライアント側とターゲット側で別個に機能します。次の図では、BEA Tuxedo 製品でサポートしている要求レベルのインターセプタの継承の階層構造を示します。



未使用インターフェイスについての注意事項

`RequestLevelInterceptor` インターフェイスから派生したクラスのオペレーションのメソッド・シグニチャには、次のインターフェイスのパラメータが含まれます。

- `RequestLevelInterceptor::DataOutputStream`

- RequestLevelInterceptor::ServiceContextList

これらのインターフェイスは BEA Tuxedo 製品では使用しません。しかし BEA Tuxedo 製品では、将来のリリースでこれらのインターフェイスのインプリメンテーションが提供されたときに CORBA アプリケーションを再コンパイルしなくてよいように、これらを定義してあります。ORB は常に実際の引数に対するニル・オブジェクトを渡します。この引数を使用しようとしないでください。プロセスが深刻なエラーにより停止する可能性があります。

Interceptors::Interceptor インターフェイス

Interceptors::Interceptor インターフェイスは、要求レベルのインターセプタをはじめとする、すべての種類のインターセプタの基本インターフェイスとして定義されます。このインターフェイスには、全種類のインターセプタでサポートされる、オペレーションおよび属性のセットが含まれます。
Interceptors::Interceptor インターフェイスは、抽象インターフェイスとして定義されるため、インターフェイスのインスタンスはインスタンス化できません。

リスト 8-1 Interceptors::Interceptor インターフェイスの OMG IDL

```
// ファイル : Interceptors.idl
#ifndef _INTERCEPTORS_IDL
#define _INTERCEPTORS_IDL

#pragma prefix "beasys.com"

module Interceptors
  {
  native ExceptionValue;

  local Interceptor
    {
    readonly attribute string   id; // インターセプタの識別子

    // インターセプタのシャットダウン時に ORB によって呼び出される
    ShutdownReturnStatus shutdown(
                                  in  ShutdownReason   reason,
                                  out ExceptionValue   excep_val
```

```
                                                    );
        };  // 位置制約付き
    };
#endif /* _INTERCEPTORS_IDL */
```

オペレーション _duplicate、_narrow、および _nil のインプリメンテーションは、BEA Tuxedo 製品の CORBA ORB によって提供される CORBA::LocalBase インターフェイスのインプリメンテーションから継承されます。

リスト 8-2　Interceptors::Interceptor インターフェイスの C++ 宣言

```
#ifndef _INTERCEPTORS_H
#define _INTERCEPTORS_H

#include <string.h>
#include <CORBA.h>

class OBBEXPDLL Interceptors
  {
  public:
  class Interceptor;
  typedef Interceptor * Interceptor_ptr;

  enum InvokeReturnStatus
    {
    INVOKE_NO_EXCEPTION,// 通常どおり進行する
    REPLY_NO_EXCEPTION, // 進行を停止し、応答処理を開始する
    REPLY_EXCEPTION     // 進行を停止し、例外を返す
    };

  enum ResponseReturnStatus
    {
        RESPONSE_NO_EXCEPTION,  // 通常どおり進行する
        RESPONSE_EXCEPTION
    };

  enum ShutdownReturnStatus
    {
    SHUTDOWN_NO_EXCEPTION,
    SHUTDOWN_EXCEPTION
    };

  enum ShutdownReason
    {
```

```
  ORB_SHUTDOWN,
  CONNECTION_ABORTED,
  RESOURCES_EXCEEDED
  };

struct Version
  {
  CORBA::Octet    major_version;
  CORBA::Octet    minor_version;
  };
typedef Version *   Version_ptr;

//+
// すべてのインターセプタの抽象基本インターフェイス
//-
class OBBEXPDLL Interceptor : public virtual CORBA::LocalBase
  {
  public:
    static Interceptor_ptr _duplicate(Interceptor_ptr obj);
    static Interceptor_ptr _narrow(Interceptor_ptr obj);
    static Interceptor_ptr _nil();
    virtual ShutdownReturnStatus
      shutdown( ShutdownReason reason,
                CORBA::Exception_ptr & excep_val) = 0;
    virtual CORBA::String id() = 0;

  protected:
    Interceptor();
    virtual ~Interceptor();
  };
};#endif /* _INTERCEPTORS_H */
```

Interceptor::id

概要　ベンダによって割り当てられたインターセプタの ID を文字列値として取得します。

C++ マッピング

```
virtual CORBA::String id() = 0;
```

パラメータ　特にありません。

例外　特にありません。

説明　id アクセサ・オペレーションは、ベンダによって割り当てられたインターセプタの ID を文字列値として取得するために ORB によって使用されます。この属性は、主に ORB によって呼び出されたインターセプタのオペレーションのデバッグとトレーシングに使用されます。

戻り値　このオペレーションは、インターセプタのインプリメンテーションの提供者によって割り当てられた ID を含む、ヌルで終了する文字列へのポインタを返します。

Interceptor::shutdown

概要 インターセプタのインプリメンテーションに、インターセプタがシャットダウンされていることを通知します。

C++ バインディング

```
virtual ShutdownReturnStatus
        shutdown( ShutdownReason reason,
                  CORBA::Exception_ptr & excep_val) = 0;
```

パラメータ

reason

インターセプタがシャットダウンされている理由を示す ShutdownReason 値です。次の ShutdownReason 値を、オペレーションに渡すことができます。

ステータス値	説明
ORB_SHUTDOWN	ORB がシャットダウンされていることを示します。
RESOURCES_EXCEEDED	プロセスのリソースが限界まで消費されたことを示します。
CONNECTION_ABORTED	この例外は BEA Tuxedo 8.0 では報告されません。

excep_val

ExceptionValue へのリファレンスです。発生した例外はすべて、オペレーションによってこのパラメータに格納されます。このパラメータは、オペレーションから SHUTDOWN_EXCEPTION の値が返された場合のみ有効です。
ExceptionValue は、クラス CORBA::Exception にマッピングされます。

例外 特にありません。

説明 shutdown オペレーションは、インターセプタのインプリメンテーションに、インターセプタがシャットダウンされていることを通知するために、ORB によって使用されます。ORB は、オペレーションから ORB に制御が返されると、インターセプタのインスタンスを破棄します。

戻り値 SHUTDOWN_NO_EXCEPTION

オペレーションが例外を発生していないことを示します。

SHUTDOWN_EXCEPTION
: オペレーションが例外を発生していることを示します。例外の値は、excep_val パラメータに格納されます。

RequestLevelInterceptor:: RequestInterceptor インターフェイス

RequestLevelInterceptor::RequestInterceptor インターフェイスは、すべての要求レベルのインターセプタの基本インターフェイスです。これは Interceptors::Interceptor インターフェイスから直接継承されます。RequestLevelInterceptor::RequestInterceptor インターフェイスの特長は次のとおりです。

- すべての要求レベルのインターセプタでサポートされる、オペレーションおよび属性のセットが含まれます。
- 抽象インターフェイスとして定義されるため、このインターフェイスのインスタンスはインスタンス化できません。

OMG IDL による local キーワードは、RequestInterceptor インターフェイスが通常の CORBA オブジェクトではなく、したがってそのようには使用できないことを示します。

リスト 8-3 RequestLevelInterceptor::RequestInterceptor インターフェイスの OMG IDL

```
#ifndef _REQUEST_LEVEL_INTERCEPTOR_IDL
#define _REQUEST_LEVEL_INTERCEPTOR_IDL

#include <orb.idl>
#include <Giop.idl>
#include <Interceptors.idl>

#pragma prefix "beasys.com"

module RequestLevelInterceptor
  {
  local RequestInterceptor : Interceptors::Interceptor
    {
    void exception_occurred(
                             in  ReplyContext       reply_context,
                             in  ExceptionValue     excep_val
                            );
    };
  };
#endif /* _REQUEST_LEVEL_INTERCEPTOR_IDL */
```

RequestInterceptor インターフェイスのインプリメンテーションは、CORBA::Object ではなく、CORBA::LocalBase から継承されます。CORBA::LocalBase は、CORBA::Object の場合と同様に、オペレーション _duplicate、_narrow、および _nil のインプリメンテーションを提供します。

リスト 8-4　RequestInterceptor インターフェイスの C++ 宣言

```
#ifndef _RequestLevelInterceptor_h
#define _RequestLevelInterceptor_h

#include <CORBA.h>
#include <IOP.h>
#include <GIOP.h>
#include <Interceptors.h>

class OBBEXPDLL RequestLevelInterceptor
  {
  public:
    class   RequestInterceptor;
    typedef RequestInterceptor *    RequestInterceptor_ptr;

    struct  RequestContext
      {
      Interceptors::Version struct_version;
      CORBA::ULong request_id;
      CORBA::Octet response_flags;
      GIOP::TargetAddress target;
      CORBA::String_var interface_id;
      CORBA::String_var operation;
      RequestContext &operator=(const RequestContext obj);
      };

    typedef RequestContext *    RequestContext_ptr;
    typedef GIOP::ReplyStatusType_1_2 ReplyStatus;

    struct  ReplyContext
      {
      Interceptors::Version struct_version;
      CORBA::ULong request_id;
      ReplyStatus reply_status;
      };

    typedef ReplyContext *  ReplyContext_ptr;
```

```
    class OBBEXPDLL RequestInterceptor :
            public virtual  Interceptors::Interceptor
      {
      public:
        static RequestInterceptor_ptr
                 _duplicate(RequestInterceptor_ptr obj);
        static RequestInterceptor_ptr
                 _narrow(RequestInterceptor_ptr obj);
        inline static RequestInterceptor_ptr _nil() { return 0; }

        virtual void
          exception_occurred( const ReplyContext & reply_context,
                              CORBA::Exception_ptr excep_val) = 0;

      protected:
        RequestInterceptor(CORBA::LocalBase_ptr obj = 0) { }
        virtual ~RequestInterceptor(){ }

      private:
        RequestInterceptor( const RequestInterceptor&) { }
        void operator=(const RequestInterceptor&) { }
      };  // クラス RequestInterceptor
#endif /* _RequestLevelInterceptor_h */
```

RequestContext 構造体

概要　要求が処理されるコンテキストを表す情報が格納されます。

C++ バインディング

```
struct  RequestContext
      {
      Interceptors::Version struct_version;
      CORBA::ULong request_id;
      CORBA::Octet response_flags;
      GIOP::TargetAddress target;
      CORBA::String_var interface_id;
      CORBA::String_var operation;
      RequestContext &operator=(const RequestContext obj);
      };
```

メンバ

`struct_version`

形式とメンバを示す RequestContext のバージョン表示です。バージョン情報は、次の 2 つの部分に分かれます。

バージョン・メンバ	説明
`major_version`	メジャー・バージョン値を示します。このメンバの値は、旧バージョンとの下位互換性のない変更が `RequestContext` の内容やレイアウトに加えられた場合に増分されます。
`minor_version`	マイナー・バージョン値を示します。このメンバの値は、旧バージョンと下位互換性のある変更が `RequestContext` の内容やレイアウトに加えられた場合に増分されます。

`request_id`

初期化 ORB によって要求に割り当てられた識別子を指定する、unsigned long 型の値です。

`response_flags`

`response_flags` の最下位ビットは、この要求に対して応答メッセージが返される場合は 1 に設定します。オペレーションが `oneway` と定義されておらず、要求が `INV_NO_RESPONSE` フラグを設定された DII を介して呼び出されるのではない場合、`response_flags` は `\x03` に設定します。

オペレーションが oneway と定義されているか、または要求が INV_NO_RESPONSE フラグを設定された DII を介して呼び出されている場合、response_flags は \x00 または \x01 に設定できます。このフラグが oneway オペレーションについて \x01 に設定されている場合、応答の受け取りは必ずしもオペレーションの完了を意味しません。

target

呼び出しのターゲットであるオブジェクトを識別するための、区別子を利用した共用体です。区別子は、ターゲットのアドレス指定が提示される形式を示します。想定されている区別子の値は、次のとおりです。

区別子	説明
KeyAddr	トランスポート固有の GIOP プロファイル (たとえばターゲット・オブジェクト用 IOR のカプセル化された IIOP プロファイル) からの object_key フィールドです。この値はサーバに対してのみ意味を持ち、クライアント側では解釈も変更もされません。
ProfileAddr	クライアント ORB によって、ターゲットの IOR 用に選択されたトランスポート固有の GIOP プロファイルです。 **注記** BEA Tuxedo 8.0 製品では、この区別子の値はサポートされていませんが、将来 GIOP 1.2 をサポートする場合に備えて提供されています。
ReferenceAddr	ターゲット・オブジェクトの完全な IOR です。selected_profile_index は、クライアント ORB によって選択されたトランスポート固有の GIOP プロファイルを示します。 **注記** BEA Tuxedo 8.0 製品では、この区別子の値はサポートされていませんが、将来 GIOP 1.2 をサポートする場合に備えて提供されています。

interface_id

オブジェクトのインターフェイスに割り当てられるリポジトリ識別子を指定する、NULL で終了する文字列です。

operation
: ターゲット・メンバによって示されるターゲット・オブジェクトに対して要求されているオペレーションの名前を指定し、interface_id メンバの値によって指定されるインターフェイスをサポートする、NULL で終了する文字列です。

説明　RequestContext データ・オブジェクトには、要求が処理されるコンテキストを表す情報が格納されます。RequestContext に含まれるコンテキスト情報は、所定の要求の処理と、対応する応答の間で調整を行うのに必要な情報を提供します。

RequestContext 構造体のコンテキスト情報は、インターセプタのインプリメンテーションでは変更できません。ORB は RequestContext の所有権を維持し、RequestContext が関連リソースの使用を終えると、それを解放する役割を果たします。

ReplyContext 構造体

概要 応答が処理されるコンテキストを表す情報が格納されます。

C++ バインディング

```
struct  ReplyContext
      {
      Interceptors::Version struct_version;
      CORBA::ULong request_id;
      ReplyStatus reply_status;
      };
```

メンバ

`struct_version`

形式とメンバを示す `ReplyContext` のバージョン表示です。バージョン情報は、次の 2 つの部分に分かれます。

バージョン・メンバ	説明
`major_version`	メジャー・バージョン値を示します。このメンバの値は、旧バージョンとの下位互換性のない変更が `ReplyContext` の内容やレイアウトに加えられた場合に増加します。
`minor_version`	マイナー・バージョン値を示します。このメンバの値は、旧バージョンと下位互換性のある変更が `ReplyContext` の内容やレイアウトに加えられた場合に増加します。

`request_id`

初期化 ORB によって要求に割り当てられた識別子を指定する、unsigned long 型の値です。

`reply_status`

関連の要求の完了ステータスを示します。また、応答本文の内容の一部を決定します。

ステータス値	説明
`NO_EXCEPTION`	要求されたオペレーションが正常に完了したことと、`arg_stream` パラメータの値にオペレーションの戻り値が含まれていることを示します。

ステータス値	説明
USER_EXCEPTION	ターゲット・オブジェクトによって報告された例外のために、要求されたオペレーションが失敗したことを示します。
SYSTEM_EXCEPTION	ターゲット・オブジェクトまたはインフラストラクチャによって報告された例外のために、要求されたオペレーションが失敗したことを示します。
LOCATION_FORWARD	本文にオブジェクト・リファレンス (IOR) が含まれていることを示します。クライアント ORB の役割は、元の要求を、その別の要求に再送信することです。この再送信は、要求を行うクライアント・プログラムにとっては透過的に行われますが、インターセプタに対しては透過的ではありません。
LOCATION_FORWARD_PERM	本文にオブジェクト・リファレンスが含まれていることを示します。使い方は LOCATION_FORWARD の場合と同様ですが、サーバによって使用される場合、この値はクライアントに対して、古い IOR が新しい IOR で置換され得ることをも示します。新旧の IOR はどちらも有効ですが、将来の使用を考えると、新しい IOR のほうが有利です。この再送信は、要求を行うクライアント・プログラムにとっては透過的に行われますが、インターセプタに対しては透過的ではありません。
NEEDS_ADDRESSING_MODE	本文に GIOP::AddressingDisposition が含まれることを示します。クライアント ORB の役割は、要求されたアドレス指定モードで元の要求を再送信することです。この再送信は、要求を行うクライアント・プログラムにとっては透過的に行われますが、インターセプタに対しては透過的ではありません。

説明 ReplyContext データ・オブジェクトには、応答が処理されるコンテキストを表す情報が格納されます。ReplyContext に含まれるコンテキスト情報は、所定の要求の処理と対応する応答の間で調整を行うのに必要な情報を提供します。

ReplyContext のコンテキスト情報は、インターセプタのインプリメンテーションでは変更できません。ORB は ReplyContext の所有権を維持し、ReplyContext が関連リソースの使用を終えると、それを解放する役割を果たします。

RequestInterceptor::exception_occurred

概要　ORB によって呼び出されて、ある要求に固有の、インターセプタが管理していたと考えられる一切の状態を、そのインターセプががクリーンアップすることを許可します。

C++ バインディング

```
virtual void
          exception_occurred( const ReplyContext & reply_context,
                              CORBA::Exception_ptr excep_val) = 0;
```

パラメータ

`reply_context`
応答が行われているコンテキストについての情報が含まれた `ReplyContext` へのリファレンスです。

`excep_val`
ORB またはほかのインターセプタによって報告された例外へのポインタです。

例外　特にありません。

説明　次の 3 つのうちいずれかの場合、要求レベルのインターセプタ・インプリメンテーションの `exception_occurred` オペレーションが呼び出されます。

1. 例外が ORB またはメソッドで生成されたのではなく、別のインターセプタによって設定された場合。
2. ORB がオペレーティング・システムまたは通信関連の問題を検出した場合。
3. クライアントが、遅延同期 DII の初期化に使用された `Request` オブジェクトを削除した場合。そのインターセプタの `client_response` または `target_response` メソッドの代わりに、`exception_occurred` メソッドが呼び出されます。ORB が `exception_occurred` メソッドを呼び出すことにより、インターセプタのインプリメンテーションは、管理していたと考えられる、ある要求に固有の一切の状態を、クリーンアップできるようになります。

戻り値　特にありません。

RequestLevelInterceptor:: ClientRequestInterceptor インターフェイス

すべての要求レベルのインターセプタの基本インターフェイスです。これは RequestLevelInterceptor::RequestInterceptor インターフェイスから直接継承されます。このインターフェイスには、クライアント側のすべての要求レベルのインターセプタでサポートされる、オペレーションおよび属性のセットが含まれます。

リスト 8-5 OMG IDL 定義

```
// ファイル : RequestLevelInterceptor.idl

#ifndef _REQUEST_LEVEL_INTERCEPTOR_IDL
#define _REQUEST_LEVEL_INTERCEPTOR_IDL

#include <orb.idl>
#include <Giop.idl>
#include <Interceptors.idl>

#pragma prefix "beasys.com"

module RequestLevelInterceptor
  {
  local ClientRequestInterceptor : RequestInterceptor
    {
    InvokeReturnStatus
      client_invoke(
             in    RequestContext           request_context,
             in    ServiceContextList       service_context,
             in    CORBA::DataInputStream   request_arg_stream,
             in    CORBA::DataOutputStream  reply_arg_stream,
             out   ExceptionValue           excep_val
                   );

    ResponseReturnStatus
      client_response(
```

```
                    in    ReplyContext           reply_context,
                    in    ServiceContextList     service_context,
                    in    CORBA::DataInputStream arg_stream,
                    out   ExceptionValue         excep_val
                     );
    };
  };
#endif /* _REQUEST_LEVEL_INTERCEPTOR_IDL */
```

オペレーション _duplicate、_narrow、および _nil のインプリメンテーションは、BEA Tuxedo 製品内の CORBA ORB によって提供される CORBA::LocalBase インターフェイスのインプリメンテーションから間接的に継承されます。

リスト 8-6　C++ 宣言

```
#ifndef _RequestLevelInterceptor_h
#define _RequestLevelInterceptor_h

#include <CORBA.h>
#include <IOP.h>
#include <GIOP.h>
#include <Interceptors.h>

class OBBEXPDLL RequestLevelInterceptor
  {
  public:
    class   ClientRequestInterceptor;
    typedef ClientRequestInterceptor *
               ClientRequestInterceptor_ptr;

    class OBBEXPDLL ClientRequestInterceptor :
            public virtual  RequestInterceptor
      {
      public:
        static ClientRequestInterceptor_ptr
          _duplicate(ClientRequestInterceptor_ptr obj);
        static ClientRequestInterceptor_ptr
          _narrow(ClientRequestInterceptor_ptr obj);
        inline static ClientRequestInterceptor_ptr
          _nil() { return 0; }

        virtual Interceptors::InvokeReturnStatus
          client_invoke(
```

```
            const RequestContext & request_context,
            ServiceContextList_ptr service_context,
            CORBA::DataInputStream_ptr request_arg_stream,
            CORBA::DataOutputStream_ptr reply_arg_stream,
            CORBA::Exception_ptr & excep_val ) = 0;

      virtual Interceptors::ResponseReturnStatus
        client_response(
            const ReplyContext & reply_context,
            ServiceContextList_ptr service_context,
            CORBA::DataInputStream_ptr arg_stream,
            CORBA::Exception_ptr & excep_val ) = 0;

    protected:
      ClientRequestInterceptor(CORBA::LocalBase_ptr obj = 0) { }
      virtual ~ClientRequestInterceptor(){ }

    private:
      ClientRequestInterceptor( const ClientRequestInterceptor&)
         { }
      void operator=(const ClientRequestInterceptor&) { }
    };  // クラス ClientRequestInterceptor
  };
#endif /* _RequestLevelInterceptor_h */
```

ClientRequestInterceptor::client_invoke

概要 クライアント・アプリケーションがターゲット・オブジェクトに呼び出しを送信するときには必ず、クライアント側 ORB によって呼び出されます。

C++ バインディング

```
virtual Interceptors::InvokeReturnStatus
          client_invoke(
              const RequestContext & request_context,
              ServiceContextList_ptr service_context,
              CORBA::DataInputStream_ptr request_arg_stream,
              CORBA::DataOutputStream_ptr reply_arg_stream,
              CORBA::Exception_ptr & excep_val ) = 0;
```

パラメータ

`request_context`
要求が行われているコンテキストについての情報が含まれた RequestContext へのリファレンスです。

`service_context`
ターゲット・オブジェクトへの要求の一部として送信される、サービス・コンテキスト情報が含まれた `ServiceContextList` へのポインタです。

注記 BEA Tuxedo 8.0 では、このパラメータの値は常にニル・オブジェクトとなります。

`request_arg_stream`
インターセプタのインプリメンテーションが、オペレーションのパラメータ値の取得に使用できる、`DataInputStream` へのポインタです。`DataInputStream` には、すべての `in` パラメータおよび `inout` パラメータが、オペレーションの IDL 定義で指定された順序で左から右へ配列されて含まれます。ニルの `DataInputStream` は、引数が存在しないことを示します。

`reply_arg_stream`
呼び出しのイニシエータに、応答として返されるパラメータを挿入するのに使用可能な、`CORBA::DataOutputStream` へのポインタです。このパラメータの使用は、`REPLY_NO_EXCEPTION` のステータスが返された場合のみ有効です。

注記 BEA Tuxedo 8.0 では、このパラメータの値は常に nil オブジェクトとなります。

`excep_val`
エラーを報告するためにインターセプタが例外を返すことのできる場所に対するリファレンスです。このパラメータの使用は、

REPLY_EXCEPTION のステータスが返された場合のみ有効です。ORB が、excep_val パラメータのためのメモリ管理の役割を持つことに注意してください。

例外 特にありません。

説明 client_invoke オペレーションは、RequestLevelInterceptor::ClientRequestInterceptor インターセプタのインターフェイスをサポートするインターセプタのインプリメンテーションに対して呼び出されます。このオペレーションは、ターゲット・オブジェクトが異なるアドレス領域にあるか、同じアドレス領域にあるかに関係なく、呼び出しがターゲット・オブジェクトに送信されると ORB によって呼び出されます。

戻り値

INVOKE_NO_EXCEPTION
: インターセプタが要求された処理をすべて正常に実行済みであり、ORB は呼び出しの処理を続行して、ターゲット・オブジェクトに送信すべきであることを示します。

REPLY_NO_EXCEPTION
: インターセプタが要求を完全に満たすために必要なすべての処理を正常に実行済みであることを示します。ORB は、完了された要求を検討し、reply_arg_stream に含まれる情報があれば、それをその要求の戻りパラメータ値として、処理し始めなければなりません。

  注記 BEA Tuxedo 8.0 では、インターセプタがこのステータス値を返すことはできません。

REPLY_EXCEPTION
: インターセプタがエラーに遭遇したため、ターゲットへの要求の処理が中断することを示します。ORB に例外を報告するのに、パラメータ excep_val が使用されます。ORB は、クライアント・アプリケーションに戻る過程でのインターセプタの呼び出しを、client_response オペレーションではなく、exception_occurred オペレーションで行います。ORB が、excep_val パラメータのためのメモリ管理の役割を持つことに注意してください。

ClientRequestInterceptor::client_response

概要　RequestLevelInterceptor::ClientRequestInterceptor インターフェイスをサポートするインターセプタのインプリメンテーションに対して呼び出されます。

C++ バインディング

```
virtual Interceptors::ResponseReturnStatus
          client_response(
              const ReplyContext & reply_context,
              ServiceContextList_ptr service_context,
              CORBA::DataInputStream_ptr arg_stream,
              CORBA::Exception_ptr & excep_val ) = 0;
```

パラメータ

reply_context
: 応答が行われているコンテキストについての情報が含まれた ReplyContext へのリファレンスです。

service_context
: ターゲット・オブジェクトによる要求処理の結果として受信される、サービス・コンテキスト情報が含まれた ServiceContextList へのポインタです。

注記　BEA Tuxedo 8.0 では、このパラメータの値は常に nil オブジェクトとなります。

arg_stream
: インターセプタのインプリメンテーションがオペレーションの応答パラメータ値の取得に使用できる、DataInputStream へのポインタです。

次の表は、ReplyContext オブジェクトに含まれるステータスに基づいて DataInputStream オブジェクトに client_response メソッドが返す値を特定したものです。

ステータス値	説明
LOCATION_FORWARD、LOCATION_FORWARD_PERM、または NEEDS_ADDRESSING_MODE	ニルの DataInputStream が指定されます。

NO_EXCEPTION	DataInputStream には、まず任意のオペレーション戻り値、次に inout パラメータと out パラメータが、左から右へ、オペレーションの IDL 定義に登場する順に含まれます。ニルの DataInputStream は、引数が存在しないことを示します。
USER_EXCEPTION または SYSTEM_EXCEPTION	DataInputStream には、オペレーションによって発生した例外が含まれます。

注記 例外には、文字列と、その後に続く任意の例外メンバが含まれます。文字列には、例外のリポジトリ ID が含まれます。例外メンバは、struct と同じように渡されます。システム例外には、2 つの unsigned long 型メンバ、マイナー・コード、および完了ステータスが含まれます。

excep_val
: エラーを報告するためにインターセプタが例外を返すことのできる場所に対するリファレンスです。このパラメータの使用は、REPLY_EXCEPTION のステータスが返された場合のみ有効です。ORB が、excep_val パラメータのためのメモリ管理の役割を持つことに注意してください。

例外 特にありません。

説明 client_response オペレーションは、RequestLevelInterceptor::ClientRequestInterceptor インターフェイスをサポートするインターセプタのインプリメンテーションに対して呼び出されます。このオペレーションは、イニシエータがターゲット・オブジェクトと異なるアドレス領域にあるか、同じアドレス領域にあるかに関係なく、呼び出しへの応答が要求のイニシエータによって受信されると ORB によって呼び出されます。

戻り値 RESPONSE_NO_EXCEPTION
: インターセプタが要求された処理をすべて正常に実行済みであり、ORB は要求に対する応答の処理を続行して、要求のイニシエータに送信すべきであることを示します。

RESPONSE_EXCEPTION
: インターセプタがエラーに遭遇したことを示します。ORB に例外を報告するのに、パラメータ excep_val が使用されます。クライアントに戻る過程でまだ呼び出されていないインターセプタはすべて、

exception_occurred オペレーションを ORB によって呼び出されて、要求の処理が失敗したことを通知されます。

RequestLevelInterceptor:: TargetRequestInterceptor インターフェイス

すべての要求レベルのインターセプタの基本インターフェイスです。これは RequestLevelInterceptor::RequestInterceptor インターフェイスから直接継承されます。このインターフェイスには、ターゲット側のすべての要求レベルのインターセプタでサポートされる、オペレーションおよび属性のセットが含まれます。

リスト 8-7 OMG IDL 定義

```
// ファイル : RequestLevelInterceptor.idl

#ifndef _REQUEST_LEVEL_INTERCEPTOR_IDL
#define _REQUEST_LEVEL_INTERCEPTOR_IDL

#include <orb.idl>
#include <Giop.idl>
#include <Interceptors.idl>

#pragma prefix "beasys.com"

module RequestLevelInterceptor
  {
  local TargetRequestInterceptor : RequestInterceptor
    {
    InvokeReturnStatus
      target_invoke(
                in    RequestContext           request_context,
                in    ServiceContextList       service_context,
                in    CORBA::DataInputStream   request_arg_stream,
                in    CORBA::DataOutputStream  reply_arg_stream,
                out   ExceptionValue           excep_val
                    );

    ResponseReturnStatus
      target_response(
                    in    ReplyContext            reply_context,
                    in    ServiceContextList      service_context,
                    in    CORBA::DataInputStream  arg_stream,
                    out   ExceptionValue          excep_val
                     );
    };
```

```
  };
#endif /* _REQUEST_LEVEL_INTERCEPTOR_IDL */
```

オペレーション _duplicate、_narrow、および _nil のインプリメンテーションは、BEA Tuxedo 製品内の CORBA ORB によって提供される、CORBA::LocalBase インターフェイスのインプリメンテーションから間接的に継承されます。

リスト 8-8　C++ 宣言

```
#ifndef _RequestLevelInterceptor_h
#define _RequestLevelInterceptor_h

#include <CORBA.h>
#include <IOP.h>
#include <GIOP.h>
#include <Interceptors.h>

class OBBEXPDLL RequestLevelInterceptor
  {
  public:
    class   TargetRequestInterceptor;
    typedef TargetRequestInterceptor *
               TargetRequestInterceptor_ptr;

    class OBBEXPDLL TargetRequestInterceptor :
            public virtual  RequestInterceptor
      {
      public:
        static TargetRequestInterceptor_ptr
            _duplicate(TargetRequestInterceptor_ptr obj);
        static TargetRequestInterceptor_ptr
            _narrow(TargetRequestInterceptor_ptr obj);
        inline static TargetRequestInterceptor_ptr
            _nil() { return 0; }

        virtual Interceptors::InvokeReturnStatus           target_invoke(
               const RequestContext & request_context,
               ServiceContextList_ptr service_context,
               CORBA::DataInputStream_ptr request_arg_stream,
               CORBA::DataOutputStream_ptr reply_arg_stream,
```

```
                CORBA::Exception_ptr & excep_val ) = 0;

        virtual Interceptors::ResponseReturnStatus
          target_response(
                const ReplyContext & reply_context,
                ServiceContextList_ptr service_context,
                CORBA::DataInputStream_ptr arg_stream,
                CORBA::Exception_ptr & excep_val ) = 0;

      protected:
        TargetRequestInterceptor(CORBA::LocalBase_ptr obj = 0) { }
        virtual ~TargetRequestInterceptor(){ }

      private:
        TargetRequestInterceptor( const TargetRequestInterceptor&)
           { }
        void operator=(const TargetRequestInterceptor&) { }
      };  // クラス TargetRequestInterceptor
  };
#endif /* _RequestLevelInterceptor_h */
```

TargetRequestInterceptor::target_invoke

概要　ターゲット・オブジェクトで呼び出しが受信されると、ターゲット側 ORB によって呼び出されます。

C++ バインディング

```
virtual Interceptors::InvokeReturnStatus
          target_invoke(
              const RequestContext & request_context,
              ServiceContextList_ptr service_context,
              CORBA::DataInputStream_ptr request_arg_stream,
              CORBA::DataOutputStream_ptr reply_arg_stream,
              CORBA::Exception_ptr & excep_val ) = 0;
```

パラメータ

`request_context`
　要求が行われているコンテキストについての情報が含まれた `RequestContext` へのリファレンスです。

`service_context`
　ターゲット・オブジェクトへの要求の一部として受信される、サービス・コンテキスト情報が含まれた `ServiceContextList` へのポインタです。
　BEA Tuxedo 8.0 では、このパラメータの値は常にニル・オブジェクトとなります。

`request_arg_stream`
　インターセプタのインプリメンテーションが、オペレーションのパラメータ値の取得に使用できる、`DataInputStream` へのポインタです。`DataInputStream` には、すべての `in` パラメータおよび `inout` パラメータが、オペレーションの IDL 定義で指定された順序で左から右へ配列されて含まれます。ニルの `DataInputStream` は、引数が存在しないことを示します。

`reply_arg_stream`
　呼び出しのイニシエータに、応答として返されるパラメータを挿入するのに使用可能な、`DataOutputStream` へのポインタです。このパラメータの使用は、`REPLY_NO_EXCEPTION` のステータスが返された場合のみ有効です。

　BEA Tuxedo 8.0 では、このパラメータの値は常にニル・オブジェクトとなります。

`excep_val`
　エラーを報告するためにインターセプタが例外を返すことのできる場所に対するリファレンスです。このパラメータの使用は、`REPLY_EXCEPTION` のステータスが返された場合のみ有効です。ORB

が、excep_val パラメータのためのメモリ管理の役割を持つことに注意してください。

例外 特にありません。

説明 target_invoke オペレーションは、RequestLevelInterceptor::TargetRequestInterceptor インターフェイスをサポートするインターセプタのインプリメンテーションに対して呼び出されます。このオペレーションは、ターゲット・オブジェクトが異なるアドレス領域にあるか、同じアドレス領域にあるかに関係なく、呼び出しがターゲット・オブジェクトによって受信されると ORB によって呼び出されます。

戻り値

INVOKE_NO_EXCEPTION
: インターセプタが要求された処理をすべて正常に実行済みであり、ORB は呼び出しの処理を続行して、ターゲット・オブジェクトに送信すべきであることを示します。

REPLY_NO_EXCEPTION
: インターセプタが要求を完全に満たすために必要なすべての処理を正常に実行済みであることを示します。ORB は、完了された要求を検討し、reply_arg_stream に含まれる情報があれば、それをその要求の戻りパラメータ値として、処理し始めなければなりません。

  注記 BEA Tuxedo 8.0 では、インターセプタがこのステータス値を返すことはできません。

REPLY_EXCEPTION
: インターセプタがエラーに遭遇したため、それにより要求をターゲット・オブジェクトに送信するための処理が中断することを示します。ORB に例外を報告するのに、パラメータ excep_val が使用されます。ORB は、クライアントに戻る過程でのインターセプタの呼び出しを、target_response オペレーションではなく、exception_occurred オペレーションで行います。ORB が、excep_val パラメータのためのメモリ管理の役割を持つことに注意してください。

TargetRequestInterceptor::target_response

概要　呼び出しに対する応答が呼び出しのイニシエータに送信されると、ターゲット側 ORB によって呼び出されます。

C++ バインディング

```
virtual Interceptors::ResponseReturnStatus
          target_response(
              const ReplyContext & reply_context,
              ServiceContextList_ptr service_context,
              CORBA::DataInputStream_ptr arg_stream,
              CORBA::Exception_ptr & excep_val ) = 0;
```

パラメータ

`reply_context`
応答が行われているコンテキストについての情報が含まれた `ReplyContext` へのリファレンスです。

`service_context`
ターゲット・オブジェクトによる要求処理の結果として送信される、サービス・コンテキスト情報が含まれた `ServiceContextList` へのポインタです。

注記　BEA Tuxedo 8.0 では、このパラメータの値は常に nil オブジェクトとなります。

`arg_stream`
インターセプタのインプリメンテーションがオペレーションの応答パラメータ値の取得に使用できる、`DataInputStream` へのポインタです。

次の表は、`ReplyContext` オブジェクトに含まれるステータスに基づいて `DataInputStream` オブジェクトに `target_response` メソッドが返す値を特定したものです。

ステータス値	説明
`LOCATION_FORWARD`、`LOCATION_FORWARD_PERM`、または `NEEDS_ADDRESSING_MODE`	nil の `DataInputStream` が指定されます。

NO_EXCEPTION	DataInputStream には、まず任意のオペレーション戻り値、次に inout パラメータと out パラメータが、左から右へ、オペレーションの IDL 定義に登場する順に含まれます。ニルの DataInputStream は、引数が存在しないことを示します。
USER_EXCEPTION または SYSTEM_EXCEPTION	DataInputStream には、オペレーションによって発生した例外が含まれます。

注記 例外には、文字列と、その後に続く任意の例外メンバが含まれます。文字列には、例外のリポジトリ ID が含まれます。例外メンバは、struct と同じように渡されます。システム例外には、2 つの unsigned long 型メンバ、マイナー・コード、および完了ステータスが含まれます。

excep_val
: エラーを報告するためにインターセプタが例外を返すことのできる場所に対するリファレンスです。このパラメータの使用は、REPLY_EXCEPTION のステータスが返された場合のみ有効です。ORB が、excep_val パラメータのためのメモリ管理の役割を持つことに注意してください。

例外 特にありません。

説明 target_response オペレーションは、RequestLevelInterceptor::TargetRequestInterceptor インターフェイスをサポートするインターセプタのインプリメンテーションに対して呼び出されます。このオペレーションは、イニシエータがターゲット・オブジェクトと異なるアドレス領域にあるか、同じアドレス領域にあるかに関係なく、呼び出しへの応答が要求のイニシエータに送信されると、ターゲット側 ORB によって呼び出されます。

戻り値

RESPONSE_NO_EXCEPTION
: インターセプタが要求された処理をすべて正常に実行済みであり、ORB は要求に対する応答の処理を続行して、要求のイニシエータに送信すべきであることを示します。

RESPONSE_EXCEPTION
: インターセプタがエラーに遭遇したことを示します。ORB に例外を報告するのに、パラメータ excep_val が使用されます。クライアントに

戻る過程でまだ呼び出されていないインターセプタはすべて、exception_occurred オペレーションを ORB によって呼び出されて、要求の処理が失敗したことを通知されます。ORB が、excep_val パラメータのためのメモリ管理の役割を持つことに注意してください。

AppRequestInterceptorInit

概要 クライアント側およびターゲット側のインターセプタをインスタンス化および初期化します。

C++ バインディング

```
typedef void (*AppRequestInterceptorInit)(
    CORBA::ORB_ptr  TheORB,
    RequestLevelInterceptor::ClientRequestInterceptor ** ClientPtr,
    RequestLevelInterceptor::TargetRequestInterceptor ** TargetPtr,
    CORBA::Boolean * RetStatus);
```

パラメータ

`TheORB`
: インターセプタのインプリメンテーションが関連付けられた ORB オブジェクトへのポインタです。

`ClientPtr`
: ORB で使用するためにインスタンス化された `RequestLevelInterceptor::ClientRequestInterceptor` のインスタンスへのポインタが返される、ポインタです。

`TargetPtr`
: ORB で使用するためにインスタンス化された `RequestLevelInterceptor::TargetRequestInterceptor` のインスタンスへのポインタが返される、ポインタです。

`RetStatus`
: インターセプタのインスタンス化と初期化が正常に実行されたかどうかが、インターセプタのインプリメンテーションによって示される、場所へのポインタです。`CORBA::TRUE` の値は、インターセプタのインスタンス化と初期化が正常に実行されたことを示すために使用されます。`CORBA::FALSE` の値は、インターセプタのインスタンス化と初期化が、何らかの理由により正常に実行されなかったことを示すために使用されます。

例外 特にありません。

説明 `AppRequestInterceptorInit` 関数は、クライアント側およびターゲット側のインターセプタをインスタンス化および初期化するために ORB によって使用される、ユーザ指定の関数です。

戻り値 特にありません。

CORBA::DataInputStream インターフェイス

DataInputStream に適用される、IDL 中の abstract valuetype キーワードは、それがインターフェイスと同じでないことを示します。

リスト 8-9　OMG IDL 定義

```
module CORBA  {
//... ほかのすべて

// DataInputStream で使用される定義
  typedef sequence<any>             AnySeq;
  typedef sequence<boolean>         BooleanSeq;
  typedef sequence<char>            CharSeq;
  typedef sequence<octet>           OctetSeq;
  typedef sequence<short>           ShortSeq;
  typedef sequence<unsigned short>  UShortSeq;
  typedef sequence<long>            LongSeq;
  typedef sequence<unsigned long>   ULongSeq;
  typedef sequence<float>           FloatSeq;
  typedef sequence<double>          DoubleSeq;

// DataInputStream - ストリームからのデータの読み込みに使用する
  abstract valuetype DataInputStream
          {
        any                         read_any();  // NO_IMPLEMENT を発生する
        boolean                     read_boolean();
        char                        read_char();
        octet                       read_octet();
        short                       read_short();
        unsigned short              read_ushort();
        long                        read_long();
        unsigned long               read_ulong();
        float                       read_float();
        double                      read_double();
        string                      read_string ();
        Object                      read_Object();
        TypeCode                    read_TypeCode();
```

```
        void                        read_any_array( inout AnySeq seq,
                                                    in unsigned long offset,
                                                    in unsigned long length);
                                                // NO_IMPLEMENT を発生する
        void                        read_boolean_array(inout BooleanSeq seq,
                                                    in unsigned long offset,
                                                    in unsigned long length);
        void                        read_char_array( inout CharSeq seq,
                                                    in unsigned long offset,
                                                    in unsigned long length);
        void                        read_octet_array(inout OctetSeq seq,
                                                    in unsigned long offset,
                                                    in unsigned long length);
        void                        read_short_array(inout ShortSeq seq,
                                                    in unsigned long offset,
                                                    in unsigned long length);
        void                        read_ushort_array(inout UShortSeq seq,
                                                    in unsigned long offset,
                                                    in unsigned long length);
        void                        read_long_array( inout LongSeq seq,
                                                    in unsigned long offset,
                                                    in unsigned long length);
        void                        read_ulong_array(inout ULongSeq seq,
                                                    in unsigned long offset,
                                                    in unsigned long length);
        void                        read_float_array(inout FloatSeq seq,
                                                    in unsigned long offset,
                                                    in unsigned long length);
        void                        read_double_array(inout DoubleSeq seq,
                                                    in unsigned long offset,
                                                    in unsigned long length);
            };
};
```

CORBA::DataInputStream のインプリメンテーションは、CORBA::Object ではなく CORBA::ValueBase から継承されます。たとえば、_duplicate、_narrow、および _nil の各オペレーションは、CORBA::Object にのみ適用されるため、使用できません。このとき、CORBA::ValueBase インターフェイスには、ユーザの関心の対象となるものはありません。

リスト 8-10　C++ 宣言

```
class  CORBA
{
 public:

  class      AnySeq {/* Normal sequence definition */};
  typedef    AnySeq *    AnySeq_ptr;

  class      BooleanSeq {/* Normal sequence definition */};
  typedef    BooleanSeq *    BooleanSeq_ptr;
  static const CORBA::TypeCode_ptr _tc_BooleanSeq;

  class      CharSeq {/* Normal sequence definition */};
  typedef    CharSeq *    CharSeq_ptr;
  static const CORBA::TypeCode_ptr _tc_CharSeq;

  class      OctetSeq {/* Normal sequence definition */};
  typedef    OctetSeq *    OctetSeq_ptr;
  static const CORBA::TypeCode_ptr _tc_OctetSeq;

  class      ShortSeq {/* Normal sequence definition */};
  typedef    ShortSeq *    ShortSeq_ptr;
  static const CORBA::TypeCode_ptr _tc_ShortSeq;

  class      UshortSeq {/* Normal sequence definition */};
  typedef    UShortSeq *    UShortSeq_ptr;
  static const CORBA::TypeCode_ptr _tc_UShortSeq;

  class      LongSeq {/* Normal sequence definition */};

  typedef    LongSeq *    LongSeq_ptr;
  static const CORBA::TypeCode_ptr _tc_LongSeq;

  class      UlongSeq {/* Normal sequence definition */};
  typedef    ULongSeq *    ULongSeq_ptr;
  static const CORBA::TypeCode_ptr _tc_ULongSeq;

  class      FloatSeq {/* Normal sequence definition */};
  typedef    FloatSeq *    FloatSeq_ptr;
  static const CORBA::TypeCode_ptr _tc_FloatSeq;

  class      DoubleSeq {/* Normal sequence definition */};
  typedef    DoubleSeq *    DoubleSeq_ptr;
  static const CORBA::TypeCode_ptr _tc_DoubleSeq;
```

```
class OBBEXPDLL DataInputStream : public virtual ValueBase
{
 public:
  static DataInputStream_ptr _downcast(ValueBase_ptr obj);

  virtual Any *          read_any (); // NO_IMPLEMENT を発生する
  virtual Boolean        read_boolean ();
  virtual Char           read_char ();
  virtual Octet          read_octet ();
  virtual Short          read_short ();
  virtual UShort         read_ushort ();
  virtual Long           read_long ();
  virtual ULong          read_ulong ();
  virtual Float          read_float ();
  virtual Double         read_double ();
  virtual Char *         read_string ();
  virtual Object_ptr     read_Object ();
  virtual TypeCode_ptr   read_TypeCode ();

  virtual void read_any_array  (    AnySeq & seq,
                       ULong offset, ULong length);
                       // NO_IMPLEMENT を発生する
  virtual void read_boolean_array(BooleanSeq & seq,
                       ULong offset, ULong length);
  virtual void read_char_array (    CharSeq & seq,
                       ULong offset, ULong length);
  virtual void read_octet_array (    OctetSeq & seq,
                       ULong offset, ULong length);
  virtual void read_short_array (    ShortSeq & seq,
                       ULong offset, ULong length);
  virtual void read_ushort_array (UShortSeq & seq,
                       ULong offset, ULong length);
  virtual void read_long_array (    LongSeq & seq,
                       ULong offset, ULong length);
  virtual void read_ulong_array (    ULongSeq & seq,
                       ULong offset, ULong length);
  virtual void read_float_array (    FloatSeq & seq,
                       ULong offset, ULong length);
  virtual void read_double_array (DoubleSeq & seq,
                       ULong offset, ULong length);

 protected:
  DataInputStream(){ };
  virtual ~DataInputStream(){ }
```

```
    private:
    void operator=(const DataInputStream&) { }
  };

typedef    DataInputStream *    DataInputStream_ptr;
};
```

DataInputStream::read_<primitive>

概要 ストリームから値を返します。

C++ バインディング

```
<primitive> read_<primitive>();
```

パラメータ 特にありません。

例外 特にありません。

説明 DataInputStream からプリミティブ・エレメント (<*primitive*>) を読み込むためのオペレーションはすべて、同じ形式です。各オペレーションは、ストリームから値を返します。

注記 メモリ管理には、String_var、TypeCode_var、または Object_var を使用できます。これらを使用しない場合は、CORBA オブジェクトの string_free() オペレーションによって文字列を解放し、CORBA オブジェクトの release() オペレーションによって TypeCode または Object ポインタを解放する必要があります。

次のプリミティブがあります。

```
AnySeq  (インプリメントされていません。)
BooleanSeq
CharSeq
OctetSeq
ShortSeq
UshortSeq
LongSeq
UlongSeq
FloatSeq
DoubleSeq
```

戻り値 特にありません。

DataInputStream::read_array_<primitive>

概要　ストリームから CORBA シーケンスにプリミティブ値の配列を返します。

C++ バインディング

```
void read_array_<primitive>(     <primitive>Seq & seq,
                                 ULong offset,
                                 ULong length);
```

パラメータ

*<primitive>*Seq

読み込まれた配列エレメントを受信する、適切な型のシーケンスです。

このシーケンスに、追加のエレメントを格納できるだけの長さがない場合、この長さは合計値 offset + length に設定されます (長さは、下方調整されることはありません)。

Offset

エレメントを読み込むための、配列へのオフセットです。つまり、配列は、配列インデックス offset から配列インデックス offset + length - 1 までの新しいエレメントを持つようになります。

Length

seq パラメータに返される配列のエレメント数です。

例外　特にありません。

説明　DataInputStream からプリミティブ・エレメント (*<primitive>*) の配列を読み込むためのオペレーションはすべて、同じ形式です。各オペレーションは、ストリームから、同じプリミティブ型の CORBA シーケンスに、プリミティブ値の配列を返します。

次のプリミティブがあります。

AnySeq (インプリメントされていません。)
BooleanSeq
CharSeq
OctetSeq
ShortSeq
UshortSeq
LongSeq
UlongSeq
FloatSeq
DoubleSeq

戻り値　特にありません。

A　要求レベルのインターセプタのスタータ・ファイル

この付録では、インターセプタのインプリメントを開始する際に使用できる次のコードを示します。

- スタータ・インプリメンテーション・コード
- スタータ・ヘッダ・ファイル・コード

このコードを使用する場合は、*YourInterceptor* をインプリメントしているインターセプタの名前に置き換えてください。

スタータ・インプリメンテーション・コード

```
#if defined(WIN32)
#include <windows.h>
#endif

#include <ctype.h>


#include "YourInterceptor.h"

// クラスのクリーンアップ -- 推奨
class Cleanup
{
    public:
    Cleanup() {}
    ~Cleanup()
    {
        // <<< ここにコードを記入する >>>

    }
```

```
};
static Cleanup CleanupOnImageExit;

#define SECURITY_BUFFSIZE   100

#if defined(WIN32)
// 標準 DLL 処理の提案

BOOL WINAPI DllMain( HANDLE hDLL,
             DWORD  dwReason,
             LPVOID lpReserved )
{
    switch( dwReason )
    {
    case DLL_PROCESS_ATTACH:
            break;
    case DLL_PROCESS_DETACH:
            break;
    case DLL_THREAD_ATTACH:
            break;
    case DLL_THREAD_DETACH:
            break;
    }

    // オペレーションが正常に実行されたという情報を返す
    return( TRUE );
    }
#endif  /* WIN32 */

/*************************************************************

   関数名        YourInterceptorInit

   機能説明

        初期化中に ORB によって呼び出される初期化ルーチンです。
        このルーチンは、サポートしている RequestLevelInterceptor
        クラスのインスタンスを作成して返します。

        注記　インターセプタ・ライブラリは、複数の初期化エントリ・ポイントを
        指定することによって、複数セットのインターセプタをサポートできます。
        この場合、各初期化エントリを個別に ORB に登録する必要があります。
        また、1 種類のインターセプタのみを指定することもできます。つまり、
        クライアントのみ、またはターゲットのみを指定できます。
```

```
*************************************************************/
#ifdef  WIN32
extern "C" __declspec(dllexport) void __cdecl
#else
extern "C" void
#endif
YourInterceptorInit(
    CORBA::ORB_ptr                                          TheORB,
    RequestLevelInterceptor::ClientRequestInterceptor ** ClientPtr,
    RequestLevelInterceptor::TargetRequestInterceptor ** TargetPtr,
    CORBA::Boolean *                                        RetStatus)
{
    // <<< ここにコードを記入する >>>

}

/*************************************************************

   関数名 YourInterceptor コンストラクタ

   機能説明

*************************************************************/
YourInterceptorClient::YourInterceptorClient(CORBA::ORB_ptr TheOrb)
{
    // この次の行は有用だが、絶対に必要なものではない

    m_orb = TheOrb;

    // <<< ここにコードを記入する >>>
}

/*************************************************************

   関数名        YourInterceptorClient::shutdown

   機能説明

        shutdown オペレーションは、インターセプタの
        インプリメンテーションに、インターセプタがシャットダウンされている
        ことを通知するために、ORB によって使用されます。
        ORB は、オペレーションから ORB に制御が返されると、
        インターセプタのインスタンスを破棄します。

*************************************************************/

Interceptors::ShutdownReturnStatus YourInterceptorClient::shutdown(
```

```
    Interceptors::ShutdownReason    reason,
    CORBA::Exception_ptr &          excep_val)

{
    // 次の各行は、単なる提案例。必要に応じて置換する

    Interceptors::ShutdownReturnStatus ret_status =
Interceptors::SHUTDOWN_NO_EXCEPTION;
    switch (reason)
    {
    case Interceptors::ORB_SHUTDOWN:
        // <<< ここにコードを記入する >>>
        break;
    case Interceptors::CONNECTION_ABORTED:
        // <<< ここにコードを記入する >>>

        break;
    case Interceptors::RESOURCES_EXCEEDED:
        // <<< ここにコードを記入する >>>

        break;
    }
    return ret_status;
}

/************************************************************

   関数名        YourInterceptorClient::id

   機能説明
        id アクセサ・オペレーションは、ベンダによって割り当てられたインターセプタの
        ID を文字列値として取得するために ORB によって使用されます。この属性は、
        主に ORB によって呼び出されたインターセプタに対するオペレーションのデバッグ
        とトレーシングに使用されます。

************************************************************/
CORBA::String YourInterceptorClient::id()
{
    // <<< ここにコードを記入する >>>

    // 次の行は、想定される有用なインプリメンテーション
    return CORBA::string_dup("YourInterceptorClient");
}

/************************************************************
```

関数名　　*YourInterceptor*Client::exception_occurred

機能説明

例外が発生すると、exception_occurred オペレーションが要求レベルのインターセプタのインプリメンテーションに対して呼び出されます。

これは、そのインターセプタの <xxx>_response メソッドの代わりに呼び出されるものです。ORB がこのオペレーションを呼び出すことにより、インターセプタのインプリメンテーションは、管理していた、ある要求に固有の一切の状態を、クリーンアップできるようになります。

```
************************************************************/
void YourInterceptorClient::exception_occurred (
        const RequestLevelInterceptor::ReplyContext &   reply_context,
        CORBA::Exception_ptr                            excep_val)
{
    // <<< ここにコードを記入する >>>
}

/************************************************************
```

関数名　　*YourInterceptor*Client::client_invoke

機能説明

このオペレーションは、ターゲット・オブジェクトが異なるアドレス領域にあるか、同じアドレス領域にあるかに関係なく、呼び出しがターゲット・オブジェクトに送信されると ORB によって呼び出されます。

```
************************************************************/
Interceptors::InvokeReturnStatus YourInterceptorClient::client_invoke (
        const RequestLevelInterceptor::RequestContext &   request_context,
        RequestLevelInterceptor::ServiceContextList_ptr   service_context,
        CORBA::DataInputStream_ptr                        request_arg_stream,
        CORBA::DataOutputStream_ptr                       reply_arg_stream,
        CORBA::Exception_ptr &                            excep_val)
{
    // 次の行は提案例。以下の最終行と関連して機能する

    Interceptors::InvokeReturnStatus ret_status =
Interceptors::INVOKE_NO_EXCEPTION;


    // <<< ここにコードを記入する >>>
```

```
    return ret_status;
}

/************************************************************

   関数名        YourInterceptorClient::client_response

   機能説明

         このオペレーションは、イニシエータがターゲット・
         オブジェクトと異なるアドレス領域にあるか、同じアドレス
         領域にあるかに関係なく、呼び出しへの応答が要求の
         イニシエータによって受信されると ORB によって
         呼び出されます。

************************************************************/
Interceptors::ResponseReturnStatus YourInterceptorClient::client_response (
        const RequestLevelInterceptor::ReplyContext &     reply_context,
        RequestLevelInterceptor::ServiceContextList_ptr   service_context,
        CORBA::DataInputStream_ptr                        arg_stream,
        CORBA::Exception_ptr &                            excep_val)
{
    // 次の行は提案例。以下の最終行と関連して機能する

    // これ以外の一般的な使用方法の提案については、例を参照

    Interceptors::ResponseReturnStatus ret_status =
Interceptors::RESPONSE_NO_EXCEPTION;



    // <<< ここにコードを記入する >>>



    return ret_status;

}

/************************************************************

   関数名        YourInterceptorTarget constructor

   機能説明
```

```
        この関数は、ターゲット・インターセプタのインスタンスを構成します。
        この例では、セキュリティ・インターセプタのインプリメントに
        使用できるデータ・メンバを示します。

*************************************************************/
YourInterceptorTarget::YourInterceptorTarget(CORBA::ORB_ptr TheOrb) :
    m_orb(TheOrb),           // 提案例
    m_security_current(0),   // セキュリティ・インターセプタの提案例
    m_attributes_to_get(0)   // セキュリティ・インターセプタの提案例
{

    // <<< ここにコードを記入する >>>

}


/*************************************************************

   関数名        YourInterceptorTarget::shutdown

   機能説明

        shutdown オペレーションは、インターセプタの
        インプリメンテーションに、インターセプタがシャットダウンされている
        ことを通知するために、ORB によって使用されます。
        ORB は、オペレーションから ORB に制御が返されると、
        インターセプタのインスタンスを破棄します。

*************************************************************/
Interceptors::ShutdownReturnStatus YourInterceptorTarget::shutdown(
        Interceptors::ShutdownReason    reason,
        CORBA::Exception_ptr &          excep_val)
{
    // <<< ここにコードを記入する >>>



    // 次の各行は、単なる提案例。必要に応じて置換する

    Interceptors::ShutdownReturnStatus ret_status =
Interceptors::SHUTDOWN_NO_EXCEPTION;
    switch (reason)
    {
    case Interceptors::ORB_SHUTDOWN:
        // <<< ここにコードを記入する >>>
```

```
        break;
    case Interceptors::CONNECTION_ABORTED:
        // <<< ここにコードを記入する >>>
        break;
    case Interceptors::RESOURCES_EXCEEDED:
        // <<< ここにコードを記入する >>>
        break;
    }
    return ret_status;
}

/************************************************************

   関数名        YourInterceptorTarget::id

   機能説明

        id アクセサ・オペレーションは、ベンダによって割り当てられたインターセプタの
        ID を文字列値として取得するために ORB によって使用されます。この属性は、
        主に ORB によって呼び出されたインターセプタのオペレーションのデバッグ
        とトレーシングに使用されます。

************************************************************/
CORBA::String YourInterceptorTarget::id()
{
    // <<< ここにコードを記入する >>>

    // 次の行は、想定される有用なインプリメンテーション

    return CORBA::string_dup("YourInterceptorTarget");
}

/************************************************************

   関数名        YourInterceptorTarget::exception_occurred

   機能説明

        例外が発生すると、exception_occurred オペレーションが要求レベルの
        インターセプタのインプリメンテーションに対して呼び出されます。
        これは、そのインターセプタの <xxx>_response メソッドの代わりに
        呼び出されるものです。ORB がこのオペレーションを呼び出すことにより、
        インターセプタのインプリメンテーションは、管理していた、ある要求に
        固有の一切の状態を、クリーンアップできるようになります。
```

```
*************************************************************/
void YourInterceptorTarget::exception_occurred (
        const RequestLevelInterceptor::ReplyContext &   reply_context,
        CORBA::Exception_ptr                            excep_val)
{
    // <<< ここにコードを記入する >>>

}

/*************************************************************

   関数名        YourInterceptorTarget::target_invoke

   機能説明

        このオペレーションは、ターゲット・オブジェクトが
        異なるアドレス領域にあるか、
        同じアドレス領域にあるかに関係なく、
        呼び出しがターゲット・オブジェクトによって

        受信されると ORB によって呼び出されます。

*************************************************************/
Interceptors::InvokeReturnStatus YourInterceptorTarget::target_invoke (
        const RequestLevelInterceptor::RequestContext &     request_context,
        RequestLevelInterceptor::ServiceContextList_ptr     service_context,
        CORBA::DataInputStream_ptr                          request_arg_stream,
        CORBA::DataOutputStream_ptr                         reply_arg_stream,
        CORBA::Exception_ptr &                              excep_val)
{
    // 次の行は提案例。以下の最終行と関連して機能する

    Interceptors::InvokeReturnStatus ret_status =
Interceptors::INVOKE_NO_EXCEPTION;



    // <<< ここにコードを記入する >>>



    return ret_status;

}

/*************************************************************
```

```
   関数名        YourInterceptorTarget::target_response

   機能説明

         このオペレーションは、イニシエータがターゲット・
         オブジェクトと異なるアドレス領域にあるか、同じアドレス
         領域にあるかに関係なく、呼び出しへの応答が要求の
         イニシエータに送信されると ORB によって
         呼び出されます。

************************************************************/
Interceptors::ResponseReturnStatus YourInterceptorTarget::target_response (
        const RequestLevelInterceptor::ReplyContext &       reply_context,
        RequestLevelInterceptor::ServiceContextList_ptr     service_context,
        CORBA::DataInputStream_ptr                          arg_stream,
        CORBA::Exception_ptr &                              excep_val)
{
    // 次の行は提案例。以下の最終行と関連して機能する

    Interceptors::ResponseReturnStatus ret_status =

Interceptors::RESPONSE_NO_EXCEPTION;



    // <<< ここにコードを記入する >>>



    return ret_status;

}

/************************************************************

   関数名        YourInterceptorTarget destructor

   機能説明

************************************************************/
YourInterceptorTarget::~YourInterceptorTarget()
{
    // <<< ここにコードを記入する >>>

}
```

スタータ・ヘッダ・ファイル・コード

```
#include <CORBA.h>
#include <RequestLevelInterceptor.h>
#include <security_c.h>      // セキュリティ用

class YourInterceptorClient : public virtual
RequestLevelInterceptor::ClientRequestInterceptor
{
private:
    YourInterceptorClient() {}
    CORBA::ORB_ptr  m_orb;
public:
    YourInterceptorClient(CORBA::ORB_ptr TheOrb);
    ~YourInterceptorClient() {}
    Interceptors::ShutdownReturnStatus shutdown(
        Interceptors::ShutdownReason reason,
        CORBA::Exception_ptr & excep_val);
    CORBA::String id();
    void exception_occurred (
        const RequestLevelInterceptor::ReplyContext & reply_context,
        CORBA::Exception_ptr excep_val);
    Interceptors::InvokeReturnStatus client_invoke (
        const RequestLevelInterceptor::RequestContext & request_context,
        RequestLevelInterceptor::ServiceContextList_ptr service_context,
        CORBA::DataInputStream_ptr request_arg_stream,
        CORBA::DataOutputStream_ptr reply_arg_stream,
        CORBA::Exception_ptr & excep_val);
    Interceptors::ResponseReturnStatus client_response (
        const RequestLevelInterceptor::ReplyContext & reply_context,
        RequestLevelInterceptor::ServiceContextList_ptr service_context,
        CORBA::DataInputStream_ptr arg_stream,
        CORBA::Exception_ptr & excep_val);

};

class YourInterceptorTarget : public virtual
RequestLevelInterceptor::TargetRequestInterceptor
{
private:

    YourInterceptorTarget() {}
    CORBA::ORB_ptr  m_orb;
```

```
    SecurityLevel1::Current_ptr m_security_current;       // セキュリティ用
    Security::AttributeTypeList * m_attributes_to_get;    // セキュリティ用
public:
    YourInterceptorTarget(CORBA::ORB_ptr TheOrb);
    ~YourInterceptorTarget();
    Interceptors::ShutdownReturnStatus shutdown(
        Interceptors::ShutdownReason reason,
        CORBA::Exception_ptr & excep_val);
    CORBA::String id();
    void exception_occurred (
        const RequestLevelInterceptor::ReplyContext & reply_context,
        CORBA::Exception_ptr excep_val);
    Interceptors::InvokeReturnStatus target_invoke (
        const RequestLevelInterceptor::RequestContext & request_context,
        RequestLevelInterceptor::ServiceContextList_ptr service_context,
        CORBA::DataInputStream_ptr request_arg_stream,
        CORBA::DataOutputStream_ptr reply_arg_stream,
        CORBA::Exception_ptr & excep_val);
    Interceptors::ResponseReturnStatus target_response (
        const RequestLevelInterceptor::ReplyContext & reply_context,
        RequestLevelInterceptor::ServiceContextList_ptr service_context,
        CORBA::DataInputStream_ptr arg_stream,
        CORBA::Exception_ptr & excep_val);

};
```

索引

A

AppRequestInterceptorInit オペレーション 8-35

B

Bootstrap オブジェクト
　インターセプタの呼び出し 1-4

C

client_invoke オペレーション 8-22
client_response オペレーション 8-24
ClientRequestInterceptor インターフェイス 8-19
CONNECTION_ABORTED 8-7

D

DataInputStream インターフェイス 8-36
DataOutputStream インターフェイス 8-2

E

exception_occurred オペレーション 8-18

I

id オペレーション 8-6
USER_EXCEPTION 8-15
InterceptorData サンプル・インターセプタ 7-1
InterceptorSec サンプル・インターセプタ
　インターセプタ
　　InterceptorSec サンプル 6-1
InterceptorSimp サンプル・インターセプタ
　インターセプタ
　　InterceptorSimp サンプル 5-1

K

KeyAddr 区別子の値 8-12

L

LOCATION_FORWARD 8-15
LOCATION_FORWARD_PERM 8-15

N

NEEDS_ADDRESSING_MODE 8-15
NO_EXCEPTION 8-15

O

object_to_string オペレーション 8-1
ORB_SHUTDOWN 8-7

P

PersonQuery サンプル・アプリケーション 4-1
　OMG IDL 4-5
　環境変数 4-13
　コマンド行インターフェイス 4-3
　実行 4-14
　ソース・ファイル 4-9
　データベース 4-2
　ビルドおよび実行 4-8
ProfileAddr 区別子の値 8-12

R

read_array プリミティブ 8-42
ReferenceAddr 区別子の値 8-12
ReplyContext 構造体 8-15
RequestContext 構造体 8-12
RequestInterceptor インターフェイス 8-9
RESOURCES_EXCEEDED 8-7

S

SecurityCurrent オブジェクト
　インターセプタの取得 1-4
SYSTEM_EXCEPTION 8-15

T

target_invoke オペレーション 8-30
target_response オペレーション 8-32
TargetRequestInterceptor インターフェイス 8-27

い

インターセプタ
　InterceptorData サンプル 7-1
　アーキテクチャの概要 1-2
　インスタンス化 1-5
　開発 2-1, 3-1
　概要 1-1
　クライアント側での実行 1-6
　クライアントに対する戻りステータス値 1-8
　クラスの階層構造 8-2
　実行 1-5
　シャットダウン 8-7
　ターゲットに対する戻りステータス値 1-12
　デプロイ 2-11
　複数使用 1-15
　目的 1-4
インターセプタ・サンプル
　ビルドと実行 5-2
インターセプタのインターフェイス 8-3
インターセプタのデプロイ 2-11
インターフェイス 2-1
　ClientRequestInterceptor 8-19
　DataInputStream 8-36
　RequestInterceptor 8-9
　TargetRequestInterceptor 8-27
　インターセプタ 8-3

か

カスタマ・サポートへのお問い合わせ情報 ix
関連情報 ix

く

クライアント・インターセプタ
　戻りステータス値 1-8
クライアント側インターセプタ 1-6

さ

サポート
　テクニカル ix

し

シャットダウン・オペレーション 8-7

す

スケルトン・ヘッダ・ファイル
　作成 2-9

せ

製品マニュアルを印刷する viii
セキュリティ・カレント・オブジェクト
　取得 1-4

た

ターゲット・インターセプタの戻りステータス値 1-12
ターゲット側インターセプタ 1-10

と

トランザクション・コンテキスト・オブジェクト 1-4

ふ

複数のインターセプタ 1-15
プリミティブの読み込み 8-41

ま

マニュアルの場所 viii

も

戻りステータス値
　クライアント側 1-8
　ターゲット 1-12

よ

要求レベルのインターセプタ
　インターセプタを参照
　概要 1-1