BEA Tuxedo

BEA Tuxedo CORBA idltojava コンパイラ

BEA Tuxedo 8.0
8.0 版
2001 年 10 月 31 日

Copyright

Copyright © 2001, BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement.It is against the law to copy the software except as specifically allowed in the agreement.This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems, Inc. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.FURTHER, BEA Systems, Inc. DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, WebLogic, Tuxedo, and Jolt are registered trademarks of BEA Systems, Inc. How Business Becomes E-Business, BEA WebLogic E-Business Platform, BEA Builder, BEA Manager, BEA eLink, BEA WebLogic Commerce Server, BEA WebLogic Personalization Server, BEA WebLogic Process Integrator, BEA WebLogic Collaborate, BEA WebLogic Enterprise, and BEA WebLogic Server are trademarks of BEA Systems, Inc.

All other company names may be trademarks of the respective companies with which they are associated.

BEA Tuxedo CORBA idltojava コンパイラ

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

このマニュアルでは、インターフェイス定義言語 (IDL) について概説し、idltojava コンパイラを使用して BEA Tuxedo 環境で CORBA Java クライアントおよび CORBA Java 共同クライアント / サーバを開発する方法について説明します。CORBA Java クライアントおよび CORBA Java 共同クライアント / サーバは、インターネット ORB 間プロコトル (IIOP) を使用して、BEA Tuxedo 製品のアプリケーション・トランザクション・モニタ・インターフェイス (ATMI) と通信します。BEA 製品の ATMI 環境は、CORBA Java サーバ・オブジェクトのホスティングをサポートしていません。

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 idltojava コンパイラの概要」では、Java IDL と CORBA の関係について説明し、Java IDL を使用して CORBA オブジェクトと相互運用できる CORBA Java クライアントおよび CORBA Java 共同クライアント / サーバを作成する方法について説明します。また、この章では、BEA idltojava コンパイラの入手先、および BEA idltojava コンパイラと Sun Microsystems, Inc から入手できる idltojava コンパイラとの相違点についても説明します。
- 「第 2 章 idltojava コマンドの使用」では、idltojava コンパイラの実行方法、および `idltojava` コマンドのオプションとフラグについて説明します。
- 「第 3 章 Java IDL の例」では、いくつかのコード例を示して、idltojava コンパイラの使い方について説明します。最初に、簡単なコード例として、Java SimpApp サンプル・アプリケーションを示します。次のコード例では、Callback オブジェクトの使い方を説明します。
- 「第 4 章 Java IDL プログラミングの概念」では、例外、初期化、および FactoryFinder オブジェクトの使用などの、関連するプログラミング概念について説明します。
- 「第 5 章 idltojava コンパイラで使用される IDL から Java へのマッピング」では、idltojava コンパイラでインプリメントされる CORBA IDL から Java へのマッピングについて説明します。

対象読者

このマニュアルは、CORBA オブジェクトと相互運用できる CORBA Java クライアントおよび CORBA Java 共同クライアント / サーバのビルドに関心がある開発者を主な対象としています。CORBA プログラミング環境および Java プログラミングに読者が精通していることを前提として書かれています。

e-docs Web サイト

BEA 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、C++ プログラミング、および Java プログラミングの詳細については、BEA Tuxedo オンライン・マニュアルの「Bibliography」を参照してください。

Java IDL および Java CORBA アプリケーションに関する一般情報については、以下のソースを参照してください。

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

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　idltojava コンパイラの概要

BEA Tuxedo 製品の CORBA 環境では、Object Management Group (OMG) によって定義されている業界標準の Object Management Group (OMG) インターフェイス定義言語 (IDL) とインターネット ORB 間プロトコル (IIOP) を使用して、CORBA Java クライアントおよび CORBA Java 共同クライアント / サーバが BEA Tuxedo ドメイン内の CORBA オブジェクトのオペレーションを呼び出すことができます。

CORBA オブジェクトにアクセスできる CORBA Java クライアントおよび CORBA Java 共同クライアント / サーバを構築するには、OMG IDL ファイルを Java クライアント・スタブおよびスケルトン・ファイルに変換するツールである BEA idltojava コンパイラが必要です。idltojava コンパイラは、BEA Tuxedo ソフトウェアに含まれています。

ここでは、次の内容について説明します。

- BEA idltojava コンパイラの入手先
- BEA 版 idltojava コンパイラと Sun Microsystems, Inc. 版の相違
- IDL とは
- Java IDL とは
- CORBA オブジェクトへのアクセス

BEA idltojava コンパイラの入手先

idltojava コンパイラの BEA 版は、BEA Tuxedo CD-ROM に含まれています。BEA Tuxedo ソフトウェアをインストールすると、idltojava コンパイラは `TUXDIR\bin` に格納されます。

BEA 版 idltojava コンパイラと Sun Microsystems, Inc. 版の相違

BEA Tuxedo 製品で提供される idltojava コンパイラには、Sun Microsystems, Inc. によるオリジナルの idltojava コンパイラにはない機能の改良、拡張、および追加が複数含まれています。ここでは、BEA Tuxedo 製品に含まれているコンパイラの変更点の概要を説明します。BEA Tuxedo 製品で提供される idltojava コンパイラを使用する方法の詳細については、「idltojava コマンドの使用」を参照してください。

次に、BEA Tuxedo 版の idltojava コンパイラの相違点を示します。

- フラグの振る舞いとデフォルトが Sun Microsystems, Inc. のマニュアルの記述とは異なります。「idltojava コマンドのフラグ」を参照してください。
- 新しい #pragma タグ `#pragma ID <name> <Repostitory_id>` が追加されています。「IDL ファイルでの #pragma の使用」を参照してください。
- 新しい #pragma タグ `#pragma version <name> <m.n>` が追加されています。「IDL ファイルでの #pragma の使用」を参照してください。
- `#pragma prefix` が拡張されて、内部スコープを扱えるようになりました。空白の接頭辞が戻されます。「IDL ファイルでの #pragma の使用」を参照してください。
- 論理区別子を利用した共用体が認められています。
- 複合型の内部で宣言を入れ子にできます。

IDL とは

インターフェイス定義言語 (IDL) は、何らかのプログラミング言語で記述されたプログラムまたはオブジェクトと未知の言語で記述されたほかのプログラムとを通信できるようにする言語を指す一般的な用語です。分散オブジェクト技術では、新しいオブジェクトは任意のプラットフォーム環境に送信可能であるだけでなく、その環境で実行されるための方法を発見する機能を備えている必要があり

ます。たとえば、ORB はインターフェイス定義言語を使用して 1 つのオブジェクト・プログラムと別のオブジェクト・プログラム間の通信を「仲介 (ブローカ) する」プログラムの一例です。

すべての CORBA オブジェクトは IDL インターフェイスをサポートします。IDL インターフェイスはオブジェクトの型を定義します。1 つのインターフェイスは、1 つまたは複数のほかのインターフェイスから継承することができます。IDL の構文は Java または C++ の構文と類似しており、IDL ファイルは機能的には C++ のヘッダ・ファイルに相当しますが CORBA でのプログラミング言語に依存しません。IDL は各プログラミング言語にマップされて、その言語からオブジェクト・インターフェイスへのアクセスを可能にします。Java IDL を使用すれば、IDL のインターフェイスは idltojava コンパイラで Java に変換可能になります。各 IDL インターフェイスについて、idltojava コンパイラは Java インターフェイスおよび必要な `.java` ファイルを、クライアント・スタブおよびサーバ・スケルトンも含めて生成します。

IDL インターフェイスは、クライアントからアクセス可能なオペレーション、例外、および型付き属性 (値) のセットを宣言します。各オペレーションには、名前、パラメータ、結果、および例外を定義するシグニチャがあります。リスト 1-1 に、BEA Tuxedo 製品に含まれている Simpapp サンプル・アプリケーションの Simple インターフェイスの IDL を示します。

リスト 1-1 Simpapp サンプル・アプリケーションの IDL インターフェイス

```
#pragma prefix "beasys.com"

interface Simple
{
     // 文字列を小文字に変換 （新しい文字列を返す）
     string to_lower(in     string val);

     // 文字列を大文字に変換 （置換）
     void to_upper(inout string val);
};

interface SimpleFactory
{
     Simple find_simple();
};
```

Java IDL とは

Java IDL は、OMG IDL と特別に異なる種類のインターフェイス定義言語 (IDL) ではありません。同じ IDL を idltojava コンパイラでコンパイルすれば CORBA 互換の Java ファイルが生成され、C++ ベースのコンパイラを使用すれば CORBA 互換の C++ ファイルが生成されます。IDL に対して使用するコンパイラによって、相違点が生じます。OMG によって、IDL と C++ のマッピングだけでなく IDL と Java のマッピングも確立されています。特定のプログラミング言語ベースのコンパイラは、OMG CORBA によるプログラミング言語へのマッピングに基づいてコードを生成します。

BEA Tuxedo 製品では、独自の「ブランド」の Java IDL が用意されています。つまり、BEA Tuxedo 製品では、CORBA オブジェクトにアクセスする機能を備えた CORBA Java クライアントおよび CORBA Java 共同クライアント / サーバの構築に必要なすべてのコンポーネントが提供されています。BEA Tuxedo 製品の主要なコンポーネントについては、「CORBA オブジェクトへのアクセス」を参照してください。

CORBA オブジェクトへのアクセス

idltojava コンパイラを使用すると、2 種類のアプリケーションを構築できます。

- `idltojava` コマンドからのファイルを自身のクライアント・スタブに利用する CORBA Java クライアント。
- `idltojava` コマンドからのファイルを自身のクライアント・スタブおよびサーバ・スケルトンに利用する CORBA Java 共同クライアント / サーバ。

CORBA Java クライアントおよび CORBA Java 共同クライアント / サーバは、インターネット ORB 間プロトコル (IIOP) を使用して BEA Tuxedo 製品のアプリケーション・トランザクション・モニタ・インターフェイス (ATMI) 環境と通信します。BEA 製品の ATMI 環境は、CORBA Java サーバ・オブジェクトのホスティングをサポートしていません。

注記 共同クライアント / サーバは、コールバック・オブジェクトとして利用される CORBA サーバ・オブジェクトをインプリメントするクライアントです。リモート共同クライアント / サーバのサーバ機能は、BEA Tuxedo CORBA サーバのサーバ機能よりも大幅にぜい弱です。共同クライアント / サーバの詳細については、『BEA Tuxedo CORBA サーバ間通信』を参照してください。

BEA Tuxedo 製品では、CORBA オブジェクトにアクセスする機能を備えた CORBA Java クライアントおよび CORBA Java 共同クライアント / サーバの構築に必要なすべてのコンポーネントが提供されています。主要なコンポーネントは、次のとおりです。

- idltojava コンパイラ ー IDL のインターフェイス定義を Java クライアント・スタブおよびスケルトン・ファイルに変換するツールです。`idltojava` コマンドは、標準的な CORBA IDL ソース・コードを Java のソース・コードにコンパイルします。その後で、`javac` コンパイラを使用して Java のソース・コードを Java のバイトコードにコンパイルできます。idltojava コンパイラの詳細については、「idltojava コマンドの使用」を参照してください。

- Bootstrap オブジェクトおよび FactoryFinder ー ローカルおよびリモートのオブジェクト・リファレンスを提供する CORBA オブジェクトです。CORBA Java クライアントまたは CORBA Java 共同クライアント / サーバは、Bootstrap オブジェクトを使用して BEA Tuxedo ドメインの複数の CORBA オブジェクト (うち 1 つは FactoryFinder) への初期オブジェクト・リファレンスを取得します。

CORBA Java クライアントおよび CORBA Java 共同クライアント / サーバは、以下のものからオブジェクト・リファレンスを取得します。

- ファクトリ・オブジェクト

  たとえば、クライアントは `DocumentFactory` オブジェクトの `create` メソッドを呼び出して新しい `Document` を作成できます。`DocumentFactory` の `create` メソッドは、`Document` へのオブジェクト・リファレンスをクライアントに返します。

  BEA Tuxedo 製品の今回のリリースに含まれる CORBA Java クライアントでは、オブジェクト・リファレンスの取得にファクトリ・オブジェクトを使用することを推奨します。

  FactoryFinder オブジェクトは、1 つの BEA Tuxedo ドメイン内または複数のドメイン間で CORBA オブジェクトを登録、格納、および検索します。

FactoryFinder オブジェクトの使用の詳細については、CORBA Javadoc を参照してください。

- オブジェクト・リファレンスから特別に作成された文字列

オブジェクト・リファレンスを取得した後で、クライアントはオブジェクト・リファレンスを適切な型にナロー変換する必要があります。IDL は継承をサポートします。継承の階層のルートは、IDL では `Object`、Java では `org.omg.CORBA.Object` です。(`org.omg.CORBA.Object` は、`java.lang.Object` のサブクラスです)。一部のオペレーション、たとえば名前のルックアップや非文字列化では、`org.omg.CORBA.Object` が返されるので、idltojava コンパイラで生成されたヘルパ・クラスを利用して必要な派生型にナロー変換してください。Java ランタイムが CORBA オブジェクトの型を常に正確に認識できるとは限らないので、CORBA オブジェクトは明示的にナロー変換される必要があります。

2 idltojava コマンドの使用

idltojava コンパイラは、OMG によって定義されている IDL と Java のマッピングに基づいて IDL ファイルを Java ソース・コードにコンパイルします。IDL と Java のマッピングの詳細については、「idltojava コンパイラで使用される IDL から Java へのマッピング」を参照してください。

ここでは、次の内容について説明します。

- idltojava コマンドの構文
- idltojava コマンドの説明
- クライアントまたは共同クライアント / サーバの IDL ファイルに対する idltojava の実行
- idltojava コマンドのオプション
- idltojava コマンドのフラグ
- IDL ファイルでの #pragma の使用

BEA Tuxedo の idltjava コンパイラに追加された機能の拡張と更新の概要については、「idltojava コンパイラの概要」を参照してください。

idltojava コマンドの構文

次に、idltojava コマンドの構文の例を示します。

```
idltojava [idltojava Command Flags] [idltojava Command Options] filename ...
```

idltojava コマンドの説明

idltojava コマンドは、IDL のソース・コードを Java のソース・コードにコンパイルします。その後で、javac コンパイラを使用して Java のソースを Java バイトコードにコンパイルします。idltojava コマンドを使用して、IDL のソース・コードを汎用のクライアント・スタブおよび汎用のサーバ・スケルトンに変換し、コールバックに使用できるようにします。

指定された IDL ファイルにある IDL の宣言は、OMG IDL から Java へのマッピングに指定されているマッピングに従って Java の宣言に変換されます。マッピングの詳細については、「idltojava コンパイラで使用される IDL から Java へのマッピング」を参照してください。

クライアントまたは共同クライアント / サーバの IDL ファイルに対する idltojava の実行

CORBA Java クライアントまたは CORBA Java 共同クライアント / サーバのクライアント側 IDL ファイルに idltojava を実行するには、次のコマンドを使用します。

```
idltojava <flags> <options> <idl-files>
```

idltojava コマンドには C++ プリプロセッサが必要で、使用を避ける名前を生成するために使用されます。idltojava コマンドは、Java クライアントの ORB に適した形の Java コードを生成します。

注記 リモート共同クライアント / サーバは、コールバック・オブジェクトとして利用されるサーバ・オブジェクトをインプリメントするクライアントです。リモート共同クライアント / サーバのサーバ機能は、BEA Tuxedo サーバのサーバ機能よりも大幅にぜい弱です。共同クライアント / サーバの詳細については、『BEA Tuxedo CORBA サーバ間通信』を参照してください。

idltojava コマンドのオプション

注記 この節には、Sun Microsystems, Inc. の idltojava コンパイラのマニュアルに記載されていないオプションの説明が追加されています。表 2-1 を参照してください。

表 2-1 idltojava に追加されたオプション

オプション	説明
-j javaDirectory	生成された Java ファイルの書き込み先となるディレクトリを指定します。このディレクトリ指定は、-p オプションが指定されていても影響を受けません。
-J filesFile	idltojava によって生成されたファイルのリストを *filesFile* に書き込みます。

表 2-1 idltojava に追加されたオプション (続き)

オプション	説明
-p package-name	すべての生成された Java ファイルをまとめる外部パッケージの名前を指定します。この機能は #pragma javaPackage と同じです。 **注記** 外部パッケージを含める必要があります。この作業は、コンパイラでは行われません。外部パッケージがない場合でも、idltojava コンパイラは Java ファイルを生成しますが、*.java ファイルのコンパイルを試みると Java コンパイル・エラーが発生することになります。
以下のオプションは、C/C++ コンパイラ (cpp) の同等のオプションと同じです。	
-Idirectory	IDL ファイルに含まれる (*#include* される) ファイルを検索するディレクトリまたはパスを指定します。このオプションはプリプロセッサに渡されます。
-Dsymbol	IDL ファイルの前処理で定義されるシンボルを指定します。このオプションはプリプロセッサに渡されます。
-Usymbol	IDL ファイルの前処理で未定義にされるシンボルを指定します。このオプションはプリプロセッサに渡されます。

idltojava コマンドのフラグ

フラグを有効にするには、フラグを表示するように指定します。フラグを無効にするには、接頭語として文字列 **no-** を付けます。たとえば、出力 IDL ファイルに対して C プリプロセッサを実行しないようにするには、**-fno-cpp** を使用します。

表 2-2 に、すべてのフラグの説明を示します。

表 2-2 idltojava コマンドのフラグ

フラグ	説明
-flist-flags	すべての **-f** フラグの状態の出力を要求します。このフラグのデフォルト値は off です。
-flist -debug-flags	デバッガ・フラグのリストを表示します。
-fcaseless	キーワードおよび識別子に使用される大文字と小文字の区別を無効にします。このことが、大文字と小文字の違いが無視されることではない点に注意してください。識別子を使用する際は、初出の識別子と同様に大文字と小文字をつづる必要があります。たとえば、「Session」と「session」は同じ識別子とされますが、初出に「Session」を使用してから「session」を使用するとエラーになります。これは、「session」の大文字と小文字のつづりが「Session」とは異なるためです。CORBA では、大文字と小文字の区別をこのように定義することで、大文字と小文字を区別するプログラミング言語間の正確なマッピングを可能にしています。このフラグのデフォルト値は on です。このフラグが指定された状態で発生した識別子の不整合の重要度は warning です (デフォルト)。詳細については、-fstrict フラグを参照してください。
-fclient	提供された IDL ファイルのクライアント側の生成を要求します。このフラグのデフォルト値は on です。
-fcpp	IDL ソースが idltojava コンパイラでコンパイルされる前に、C/C++ プリプロセッサで処理されるようにします。このフラグのデフォルト値は on です。
-fignore-duplicates	重複した定義が無視されるように指定します。複数の IDL ファイルを一度にコンパイルする場合に便利です。このフラグのデフォルト値は off です。
-flist-options	コマンド行に指定されているオプションをリストします。このフラグのデフォルト値は off です。
-fmap-included-files	#include プリプロセッサ・ディレクティブにより含まれた定義に合わせて Java ファイルを生成するように指定します。このフラグのデフォルト値は off で、この場合は定義に合わせて Java ファイルが生成されないように指定されています。

表 2-2 idltojava コマンドのフラグ (続き)

フラグ	説明
-fserver	提供された IDL ファイルのサーバ側の生成を要求します。このフラグのデフォルト値は on です。
-fverbose	コンパイラがコンパイルの進行についてコメントするように指定します。このフラグのデフォルト値は off です。
-fversion	コンパイラが自身のバージョンおよびタイム・スタンプを表示するように指定します。このフラグのデフォルト値は off です。
-fwarn-pragma	未知の #pragma、または指定が不適切な #pragma について警告メッセージを発行するように指定します。このフラグのデフォルト値は on です。
-fwrite-files	派生した Java ファイルが書き込まれるように指定します。このフラグのデフォルト値は on です。ファイルを実際には書き込まないでエラーを調べる場合は、-fno-write-files を指定できます。
-fstrict	以前のリリースでは、-fcaseless フラグが指定された状態で発生した識別子の不整合の重要度は error でした。今回のリリースでは、-fcaseless フラグは以前とまったく同様に識別子の比較を実行しますが、不整合の重要度が error から warning に変更されました。以前のリリースのように、識別子の不整合を error とする CORBA への厳密な準拠が必要な場合は、-fcaseless フラグと共にこのフラグを指定してください。

IDL ファイルでの #pragma の使用

注記 BEA Tuxedo の idltojava コンパイラによる #pragma の処理は、Sun Microsystems, Inc. の idltojava コンパイラによる処理とは一部異なります。

RepositoryPrefix="prefix"

デフォルトのリポジトリ接頭語は、IDL ファイル自身の最高レベルに #pragma prefix "requested prefix" のように行を記述して要求することもできます。次の行について説明します。

```
#pragma javaPackage "package"
```

この行は、デフォルトのパッケージを、呼び出された 1 つのパッケージにラップします。たとえば、M という IDL モジュールをコンパイルすれば、通常は M という名前の Java パッケージが 1 つ作成されます。このモジュール宣言の前に次の行があるとします。

```
#pragma javaPackage browser
```

この場合、コンパイラは、パッケージ M を browser というパッケージの内部に作成します。このプラグマは、1 つの IDL モジュールにある定義を複数の製品で利用する場合に便利です。コマンド行オプション -p も、同じ結果を得るために使用できます。次の行について説明します。

```
#pragma ID scoped-name "IDL:<path>:<version>"
```

この行は、識別子でスコープが設定された名前のリポジトリ ID を指定します。このプラグマは、IDL ファイルのどこにでも記述できます。このプラグマが複合型、たとえば構造体や共用体の内部に現れる場合は、指定する要素の数だけスコープ指定された名前を定義する必要があります。スコープ指定された名前の形式は、outer_name::name::inner_name です。リポジトリ ID の コンポーネントは、スラッシュ (/) によって区切られた一連の識別子です。コンポーネントは十進数 MM.mm で、MM はメジャー・バージョン番号を、mm はマイナー・バージョン番号を表します。

3 Java IDL の例

ここでは、次の内容について説明します。

- IDL の簡単な例
- Callback オブジェクトの IDL の例

IDL の簡単な例

リスト 3-1 の OMG IDL に記述されている CORBA オブジェクトでは、`to_lower()` および `to_upper()` オペレーションによって、ユーザ入力の大文字と小文字がそれぞれ変更された文字列が返されます (大文字の入力は小文字に変更され、小文字の入力は大文字に変更されます)。

リスト 3-1 Simpapp サンプル・アプリケーションの IDL インターフェイス

```
#pragma prefix "beasys.com"

interface Simple
{
      // 文字列を小文字に変換 （新しい文字列を返す）
      string to_lower(in      string val);

      // 文字列を大文字に変換 （置換）
      void to_upper(inout string val);
};

interface SimpleFactory
{
      Simple find_simple();
};
```

このアプリケーションを最初からインプリメントする場合は、この IDL インターフェイスを次のコマンドでコンパイルします。

```
idltojava Simple.idl
```

この結果、スタブおよびスケルトンと、複数のファイルが生成されます。

idltojava コンパイラのオプションおよびフラグの詳細については、「idltojava コマンドの使用」を参照してください。

Callback オブジェクトの IDL の例

リスト 3-2 の OMG IDL は、Callback サンプル・アプリケーションの Callback、Simple、および SimpleFactory インターフェイスを定義します。

リスト 3-2 Callback サンプル・アプリケーションの IDL 定義

```
#pragma prefix "beasys.com"

      interface Callback

      // このメソッドは、渡されたデータを大文字および小文字で
      // 出力する
      {
          void print_converted(in string message);
      };

      interface Simple

      // 共同クライアント / サーバ・アプリケーションの
      // コールバック・オブジェクトを呼び出す
      {
          void call_callback(in string val, in Callback
                                        callback_ref);
      };

      interface SimpleFactory
      };
          Simple find_simple();
      };
```

Java CORBA コールバックの例の完全な説明、およびこの例を構築し、実行する方法の詳細については、BEA Tuxedo のオンライン・マニュアルの『BEA Tuxedo CORBA サーバ間通信』を参照してください。

4　Java IDL プログラミングの概念

ここでは、次の内容について説明します。

- 例外
- 初期化
- FactoryFinder インターフェイス

例外

CORBA には 2 種類の例外があります。OMG によって完全に指定されている標準的なシステム例外と、個々のアプリケーション・プログラマによって定義されるユーザ例外です。CORBA の例外は Java の例外オブジェクトとわずかに異なりますが、こうした相違は主に IDL と Java のマッピングで処理されます。

ここでは、次の内容について説明します。

- CORBA の例外と Java の例外の相違
- システム例外
- ユーザ例外
- マイナー・コードの意味

CORBA の例外と Java の例外の相違

IDL で例外を指定するために、インターフェイス設計者は *raises* キーワードを使用します。これは、Java での *throws* の指定と同様です。IDL で exception キーワードを使用すると、ユーザ定義の例外を作成することになります。標準的なシステム例外は、このような指定をする必要がありません (このような指定はできません)。

システム例外

CORBA では、一連の標準システム例外を定義しています。これらの例外は一般に ORB ライブラリによって生成され、以下のようなシステムのエラー状態を知らせます。

- サーバ側システム例外。たとえば、リソース不足や活性化の失敗などです。
- 通信システム例外。たとえば、オブジェクトとの通信切断、ホストの停止、あるいは ISL や ISH との通信不能などです。
- クライアント側システム例外。たとえば、無効なオペランド型、あるいは要求の送信前や結果が返された後に発生した何らかの問題です。

すべての IDL オペレーションは、呼び出されたときにシステム例外をスローできます。インターフェイス設計者がインターフェイス内のオペレーションでシステム例外をスローするよう指定しなくても、処理は自動で行われます。

これは、オペレーションのインプリメンテーションが単純であっても、別のプロセス（多くは別のマシン上のプロセス）であるクライアントからのオペレーション呼び出しによって、あらゆる種類のエラーが発生する可能があるためです。

このため、CORBA Java クライアントは常に CORBA システム例外をキャッチできる必要があります。さらに、開発者は、キャッチする必要のあるシステム例外の通知を idltojava コンパイラに期待できません。これは、CORBA のシステム例外が `java.lang.RuntimeException` の下位クラスであるためです。

システム例外の構造

すべての CORBA システム例外は同じ構造です。

```
exception <SystemExceptionName> { // エラーの説明
    unsigned long minor;          // エラーの詳細
    CompletionStatus completed;   // 値は、yes、no、maybe のいずれか
}
```

システム例外は `java.lang.RuntimeException` のサブタイプで `org.omg.CORBA.SystemException` の下位のクラスです。

```
java.lang.Exception
 |
 +--java.lang.RuntimeException
      |
      +--org.omg.CORBA.SystemException
           |
```

```
+--BAD_PARAM
|
+--//etc.
```

完了ステータス

すべての CORBA システム例外には、例外をスローしたオペレーションのステータスを示す完了ステータス・フィールドがあります。完了コードは、以下のとおりです。

- COMPLETED_YES

  オブジェクトのインプリメンテーションは、例外が起きる前に処理を完了しています。

- COMPLETED_NO

  オブジェクトのインプリメンテーションは、例外が起きる前に呼び出されていませんでした。

- COMPLETED_MAYBE

  呼び出しのステータスは不明です。

ユーザ例外

CORBA ユーザ例外は `java.lang.Exception` のクラスのサブタイプで `org.omg.CORBA.UserException` の下位クラスです。

```
java.lang.Exception
 |
 +--org.omg.CORBA.UserException
       |
       +-- Stocks.BadSymbol
       |
       +--//etc.
```

IDL に指定されているユーザ定義例外が起きるたびに、結果として Java の例外クラスが生成されます。こうした例外は、すべてプログラマが定義し、インプリメントします。

マイナー・コードの意味

すべてのシステム例外には、例外発生の原因についての付加情報を CORBA ベンダが提供するための [minor] フィールドが用意されています。表 4-1 および表 4-2 に、Java IDL のシステム例外のマイナー・コードと、それらの意味を示します。

表 4-1 ORB のマイナー・コードとその意味

コード	説明
BAD_PARAM 例外のマイナー・コード	
1	Java IDL メソッドに NULL パラメータが渡されました。
COMM_FAILURE 例外のマイナー・コード	
1	オブジェクト・リファレンス、または位置 / オブジェクトの転送の後で取得されたオブジェクト・リファレンスで指定されているホストおよびポートに接続できませんでした。
2	ソケットへの書き込みを試みている間にエラーが発生しました。ソケットは他方の側で閉じられているか、アボートされています。
3	ソケットへの書き込みを試みている間にエラーが発生しました。接続が存在しません。
6	サーバへの接続が、数回試みてもできませんでした。
DATA_CONVERSION 例外のマイナー・コード	
1	ORB の `string_to_object` オペレーション中に不正な 16 進数が見つかりました。
2	`string_to_object()` の指定された IOR の長さの値が奇数です。偶数を指定して下さい。
3	`string_to_object()` に指定された文字列が IOR で始まりません。このため、IOR の文字列化が無効です。

表 4-1 ORB のマイナー・コードとその意味 (続き)

コード	説明
4	ORB の resolve_initial_references オペレーションを実行できません。ホストまたはポートが不正または未指定であるか、リモート・ホストが Java IDL のブートストラップ・プロトコルをサポートしていません。
INTERNAL 例外のマイナー・コード	
3	サーバによる IIOP 応答メッセージで不正なステータスが返されました。
6	マーシャリングの解除時に、ユーザ例外のリポジトリ ID の長さが不正であることが判明しました。
7	Java API の InetAddress.getLocalHost().getHostName() を使用してもローカルなホスト名を判別できません。
8	特定のポートのリスナ・スレッドを作成できません。ポートが既に使用中であるか、デーモン・スレッドの作成でエラーが発生したか、あるいはセキュリティ制限がリスニングを妨げています。
9	IIOP 位置の応答で、不正な位置応答のステータスが見つかりました。
10	オブジェクト・リファレンスの文字列化でエラーが発生しました。
11	不正な GIOP 1.0 メッセージ型の IIOP メッセージが見つかりました。
14	ユーザ例外のマーシャリング解除でエラーが発生しました。
18	内部初期化エラーです。
INV_OBJREF 例外のマイナー・コード	
1	IOR 遭遇のプロファイルがありません。
MARSHAL 例外のマイナー・コード	
4	オブジェクト・リファレンスのマーシャリングの解除でエラーが発生しました。
5	ワイド文字およびワイド文字列といったサポートされていない IDL の種類のマーシャリングまたはマーシャリング解除を行います。

表 4-1 ORB のマイナー・コードとその意味 (続き)

コード	説明
6	文字または文字列のマーシャリングあるいはマーシャリング解除中に、ISO Latin-1 (8859.1) 準拠でない文字が見つかりました。0 ～ 255 の範囲にありません。
NO_IMPLEMENT 例外のマイナー・コード	
1	動的スケルトン・インターフェイスがインプリメントされていません。
OBJ_ADAPTER 例外のマイナー・コード	
1	サーバ側の要求をオブジェクト・アダプタ層へディスパッチするときに、オブジェクト・キーにあるオブジェクト・アダプタと一致するオブジェクト・アダプタが見つかりません。
2	サーバ側のロケート要求をオブジェクト・アダプタ層へディスパッチするときに、オブジェクト・キーにあるオブジェクト・アダプタと一致するオブジェクト・アダプタが見つかりません。
4	ORB のサーバントへの接続を試みてエラーが発生しました。
OBJ_NOT_EXIST 例外のマイナー・コード	
1	ロケート要求の結果、ロケータにとってそのオブジェクトが未知であることを示す応答がありました。
2	要求を受信したサーバのサーバ ID が、呼び出したオブジェクト・リファレンスのオブジェクト・キーに書きこまれたサーバ ID と一致しません。
4	オブジェクト・リファレンスの内部にあるオブジェクト・キーの内容と一致するスケルトンがサーバ側にありません。
UNKNOWN 例外のマイナー・コード	
1	マーシャリングの解除で未知のユーザ例外が見つかりました。サーバは、クライアントが期待したものと一致しないユーザ例外を返しました。
3	未知の実行時例外がサーバのインプリメンテーションによってスローされました。

表 4-2 ネーム・サーバのマイナー・コードとその意味

コード	説明
INITIALIZE 例外のマイナー・コード	
150	一時ネーム・サービスの初期化中に SystemException が発生しました。
151	一時ネーム・サービスの初期化中に Java の例外が発生しました。
INTERNAL 例外のマイナー・コード	
100	rebind オペレーションで AlreadyBound 例外がスローされました。
101	rebind_context オペレーションで AlreadyBound 例外がスローされました。
102	内部のバインドのインプリメンテーションに渡されたバインドの型が、BindingType.nobject または BindingType.ncontext ではありません。
103	オブジェクト・リファレンスはコンテキストとしてバインドされていますが、CosNaming.NamingContext にナロー変換できません。
200	バインド・オペレーションのインプリメンテーションが、以前のバインディングに遭遇しました。
201	リスト・オペレーションのインプリメンテーションがリスト・イテレータを作成する際に Java の例外をキャッチしました。
202	new_context オペレーションのインプリメンテーションが新しい NamingContxt サーバントを作成する際に Java の例外をキャッチしました。
203	destroy オペレーションのインプリメンテーションが、ORB の切り離し中に、Java の例外をキャッチしました。

初期化

CORBA Java クライアントまたは CORBA Java 共同クライアント / サーバで CORBA オブジェクトを使用可能にするには、次の方法で自身を初期化する必要があります。

- ORB オブジェクトを作成します。
- 1 つまたは複数の初期オブジェクト・リファレンスを取得します。通常は、FactoryFinder オブジェクトを使用します。

ORB オブジェクトの作成

CORBA Java クライアントまたは CORBA Java 共同クライアント / サーバが CORBA オブジェクトを作成または呼び出せるようになるには、最初に ORB オブジェクトを作成する必要があります。ORB オブジェクトを作成することで、クライアントまたは共同クライアント / サーバは ORB に自身の存在を知らせ、ORB オブジェクトで定義されている重要なオペレーションへのアクセスを取得します。

クライアントと共同クライアント / サーバでは、ORB のインスタンスを作成する方法がわずかに異なります。これは、`ORB.init` の呼び出しに渡される必要があるパラメータの配列が異なるためです。

アプリケーション用 ORB の作成

次のコードは、CORBA Java クライアントが ORB を作成する方法の一例です。

```
import org.omg.CORBA.ORB;

public static void main(String args[])
{
  try{
    ORB orb = ORB.init(args, null);
// コード続く
```

アプレット用 ORB の作成

Java アプレットでは、次のようにして ORB を作成します。

```
import org.omg.CORBA.ORB;

public void init() {
  try {
    ORB orb = ORB.init(this, null);
// コード続く
```

一部の Web ブラウザには、ORB が組み込まれています。このため、ブラウザの ORB が標準に完全に準拠していないと問題が生じます。そのような場合は、Java IDL ORB を確実に初期化するための特別なステップが必要になります。たとえば、Netscape Communicator 4.01 の ORB ではクラスが不足しているために、このブラウザで表示するアプレットの `init()` メソッドには、次に示すようなコードが必要です。

```
import java.util.Properties;
import org.omg.CORBA.*;

public class MyApplet extends java.applet.Applet {
  public void init()
  {
    // Java IDL ORB のインスタンス化。このアプレットに渡して
    // ORB がアプレットのプロパティを取得できるようにする
    Properties p = new Properties();
    p.put("org.omg.CORBA.ORBClass", "com.sun.CORBA.iiop.ORB");
    p.put("org.omg.CORBA.ORBSingletonClass","com.sun.CORBA.idl.ORBSingleton");
    System.setProperties(p);
    ORB orb = ORB.init(args, p);
    ...
  }
}
```

ORB.init() の引数

アプリケーションとアプレットの両方について、初期化メソッドの引数は次のとおりです。

- `args` または `this`

  アプリケーションの引数またはアプレットのパラメータへの ORB アクセスを提供します。

- null

  java.util.Properties のオブジェクトです。

init() オペレーションはこれらのパラメータをシステムのプロパティとともに使用して、ORB のコンフィギュレーションに必要な情報を取得します。ORB コンフィギュレーションのプロパティの検索は、以下の場所および順序で行われます。

1. アプリケーションまたはアプレットのパラメータ (最初の引数)。
2. java.util.Properties オブジェクト (2 番目の引数)。提供されている場合。
3. System.getProperties() によって返された java.util.Properties オブジェクト。

いずれかのプロパティで最初に見つかった値が、init() オペレーションで使用される値です。上記の場所のいずれにもコンフィギュレーションのプロパティがない場合、init() オペレーションはインプリメンテーションに固有の値を推定します。ORB のインプリメンテーション間で移植性を最大にするために、アプレットおよびアプリケーションはオペレーションに影響するコンフィギュレーションのプロパティの値を明示的に指定する必要があります。自身が実行されている ORB による推定には依存しないようにしてください。

システムのプロパティ

BEA Tuxedo 製品は Sun Microsystems, Inc の Java 仮想マシンを使用しています。この仮想マシンは、コマンド行引数 -D を追加しています。ほかの Java 仮想マシンの中には、Sun の仮想マシンと同様のものもあれば、異なるものもあります。

現在、すべての ORB インプリメンテーションについて以下のコンフィギュレーション・プロパティが定義されています。

- org.omg.CORBA.ORBClass

  org.omg.CORBA.ORB インターフェイスをインプリメントする Java クラスの名前です。アプレットおよびアプリケーションは、特定の ORB インプリメンテーションが必要でなければ、このプロパティを提供しなくてもかまいません。Java IDL ORB 用の値は com.sun.CORBA.iiop.ORB です。

- `org.omg.CORBA.ORBSingletonClass`

  `org.omg.CORBA.ORB` インターフェイスをインプリメントする Java クラスの名前です。このオブジェクトは、`orb.init()` への引数なしの呼び出しによって返されます。主な用途は、安全な環境にある信頼性のないコード (署名のないアプレットなど) の間で共有できるタイプ・コードのインスタンスを作成することです。

アプレットのパラメータには、完全なプロパティ名を指定する必要があります。アプリケーションの規則はアプレットとは異なるので、コマンド行呼び出しでの言語固有の詳細項目がエクスポーズされることはありません。

初期オブジェクト・リファレンスの取得

CORBA オブジェクトを呼び出すには、アプレットまたはアプリケーションで CORBA オブジェクトへのリファレンスが必要です。CORBA オブジェクトへのリファレンスを取得する方法は、次の 3 つです。

- オブジェクト・リファレンスから特別に作成された文字列を使用します。
- たとえば FactoryFinder のような、ほかのオブジェクトを使用します。
- `bootstrap` メソッドを使用します。

文字列化されたオブジェクト・リファレンス

第一のテクニックである、文字列化されたリファレンスを実際のオブジェクト・リファレンスに変換する方法は、ORB のインプリメンテーションに依存しません。アプレットまたはアプリケーションがどの ORB 上で実行されていても、文字列化されたオブジェクト・リファレンスを変換することができます。ただし、アプレットまたはアプリケーションの開発者は、以下の点に留意してください。

- 参照されるオブジェクトが、アプレットやアプリケーションの実行場所から実際にアクセス可能であること。
- 文字列化されたリファレンスを、ファイルやパラメータから取得すること。

ORB からのリファレンスの取得

初期 CORBA オブジェクトを取得するために文字列化されたリファレンスを使用しない場合は、ORB 自身を使用して初期オブジェクト・リファレンスを生成します。

Bootstrap オブジェクトは、新しく開始されたアプリケーションまたはアプレットにオブジェクト・リファレンスをブートストラップ処理するための `resolve_initial_references()` というオペレーションを定義します。このオペレーションは、認識されるいくつかのオブジェクトのうち 1 つの名前を指定する文字列引数を取ります。結果として 1 つの CORBA オブジェクトが返され、このオブジェクトはアプレットやアプリケーションが認識する型にナロー変換される必要があります。

Bootstrap オブジェクトを使用すると、オブジェクト・リファレンス (SecurityCurrent、TransactionCurrent、FactoryFinder、NotificationService、Tobj_SimpleEventsService、NameService、および InterfaceRepository) を取得できます。ここでは、FactoryFinder オブジェクトについて説明します。

FactoryFinder インターフェイスは、BEA Tuxedo ドメインへの唯一の入り口となるオブジェクト・リファレンスをクライアントに提供します。FactoryFinder オブジェクトは特定のファクトリ・オブジェクトの取得に使用されます。取得したファクトリ・オブジェクトで、必要なオブジェクトを作成することができます。

Bootstrap オブジェクトの使用の詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』を参照してください。

FactoryFinder インターフェイス

FactoryFinder インターフェイスは、BEA Tuxedo ドメインへの唯一の入り口となるオブジェクト・リファレンスをクライアントに提供します。複数の FactoryFinder によって、可用性と信頼性が高まります。複数のドメインにわたるマッピングがサポートされています。

FactoryFinder のインターフェイスと NameManaer は、オブジェクトの登録、格納、および検索のためのメカニズムです。BEA Tuxedo 環境では、次のことができます。

- Bootstrap オブジェクトを使用して、FactoryFinder オブジェクトへのオブジェクト・リファレンスを取得できます。
- FactoryFinder オブジェクトを使用して、必要な Factory オブジェクトを検索できます。
- Factory オブジェクトを使用して、CORBA オブジェクトの新しいインスタンスを作成できます。

FactoryFinder オブジェクトの使用の詳細については、『BEA Tuxedo CORBA プログラミング・リファレンス』を参照してください。

5 idltojava コンパイラで使用される IDL から Java へのマッピング

idltojava コンパイラは OMG IDL インターフェイスを読み込んで、Java のインターフェイスに変換またはマップします。また、idltojava コンパイラは、必要に応じてスタブ、スケルトン、ヘルパ、ホルダなどのファイルを作成します。これらの .java ファイルは、OMG の文書「IDL/Java Language Mapping」に指定されたマッピングに従って IDL ファイルから生成されます。

IDL から Java へのマッピングの詳細については、OMG の Web サイト (http://www.omg.org) を参照してください。

CORBA オブジェクトは、OMG IDL (Object Management Group インターフェイス定義言語) で定義されています。Java 開発者が CORBA オブジェクトを使用するには、CORBA オブジェクトのインターフェイスを Java のクラスおよびインターフェイスにマップする必要があります。idltojava コンパイラは、このマッピングを自動的に実行します。

表 5-1 に、OMG IDL の構造体と Java の構造体の対応を示します。OMG IDL は、その名が示すとおり、インターフェイスを定義します。Java のインターフェイスと同様に、IDL のインターフェイスにはそのオペレーションのためのインプリメンテーション (Java のメソッドに相当) は含まれません。つまり、IDL のインターフェイスでは、オペレーションのためのシグニチャ (オペレーションの名前、オペレーションの戻り値のデータ型、オペレーションのパラメータのデータ型、および発生する例外) のみを定義します。これらのオペレーションで使用するインプリメンテーションは、Java プログラマによって記述された Java クラスの形で提供される必要があります。

表 5-1 IDL の構造体から Java の構造体へのマッピング

IDL の構造体	Java の構造体
module	package
interface	interface、helper class、holder class

表 5-1 IDL の構造体から Java の構造体へのマッピング (続き)

IDL の構造体	Java の構造体
constant	public static final
boolean	boolean
char、wchar	char
octet	byte
string、wstring	`java.lang.String`
short、unsigned short	short
long、unsigned long	int
long long、unsigned long long	long
float	float
double	double
enum、struct、union	class
sequence、array	array
exception	class
readonly attribute	属性値にアクセスするメソッド
readwrite attribute	属性値にアクセスおよび設定するメソッド
operation	method

注記 CORBA オペレーションの型が Java オブジェクトの型と対応するとき (たとえば、string) は、Java の null をパラメータの値として渡すと不正になります。代わりに、指定されたオブジェクト型の空のバージョンを渡してください。たとえば、空の string または空の array)。Java の null をパラメータとして渡すことができるのは、パラメータの型が CORBA オブジェクト・リファレンスであるときだけです。この場合、null は CORBA オブジェクト・リファレンスの nil として解釈されます。

索引