IBM WebSphere Business Integration Adapters

Adapter for CORBA ユーザーズ・ガイド

バージョン 1.2.x

IBM WebSphere Business Integration Adapters

Adapter for CORBA ユーザーズ・ガイド

バージョン 1.2.x

お願い

本書および本書で紹介する製品をご使用になる前に、 115 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM WebSphere Business Integration Adapter for CORBA バージョン 1.2.x に適用されます。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原　典： IBM WebSphere Business Integration Adapters
Adapter for CORBA User Guide
Version 1.2.x

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2004.7

この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財）日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

本書について

IBM(R) WebSphere(R) Business Integration Adapter ポートフォリオは、主要な e-business テクノロジー、エンタープライズ・アプリケーション、レガシー、およびメインフレーム・システムに統合コネクティビティーを提供します。本製品には、コンポーネントをカスタマイズ、作成、および管理するためのツールとテンプレートが含まれており、これにより、ビジネス・プロセスの統合を実現します。

本書では、IBM WebSphere Business Integration Adapter for CORBA のインストール、構成、ビジネス・オブジェクトの開発、およびトラブルシューティングについて説明します。

対象読者

本書は、WebSphere Business Integration システムをお客様のサイトでサポートおよび管理する、コンサルタント、開発者、およびシステム管理者を対象としています。

本書の前提条件

本書のユーザーは、WebSphere Business Integration システム、ビジネス・オブジェクトおよびコラボレーション開発、および CORBA テクノロジーに精通している必要があります。

関連文書

この製品に付属する資料の完全セットで、すべての WebSphere Business Integration Adapters のインストールに共通な機能とコンポーネントについて説明します。また、特定のコンポーネントに関する参考資料も含まれています。

以下のサイトから、関連資料をインストールすることができます。

- 一般的なアダプター情報が必要な場合、アダプターを WebSphere Message Broker (WebSphere MQ Integrator、WebSphere MQ Integrator Broker、WebSphere Business Integration Message Broker) とともに使用する場合、およびアダプターを WebSphere Application Server とともに使用する場合は、以下のサイトを参照してください。
  - http://www.ibm.com/websphere/integration/wbiadapters/infocenter
- アダプターを InterChange Server とともに使用する場合は、以下のサイトを参照してください。
  - http://www.ibm.com/websphere/integration/wicserver/infocenter
  - http://www.ibm.com/websphere/integration/wbicollaborations/infocenter
- Message Broker (WebSphere MQ Integrator Broker、 WebSphere MQ Integrator、および WebSphere Business Integration Message Broker) の詳細については、以下のサイトを参照してください。

- http://www.ibm.com/software/integration/mqfamily/library/manualsa/

- WebSphere Application Server の詳細については、以下をご覧ください。
  - http://www.ibm.com/software/webservers/appserv/library.html

これらのサイトには、資料をダウンロード、インストールして参照するための簡単な説明が提供されています。

注: 本書の発行後に公開されたテクニカル・サポートの技術情報や速報に、本書の対象製品に関する重要な情報が記載されている場合があります。これらの情報は、WebSphere Business Integration Support Web サイト (http://www.ibm.com/software/integration/websphere/support/) にあります。関心のあるコンポーネント・エリアを選択し、「Technotes」セクションと「Flashes」セクションを参照してください。また、IBM Redbooks (http://www.redbooks.ibm.com/) にもその他の有効な情報があることがあります。

表記上の規則

本書では、以下のような規則を使用しています。

`Courier フォント`	コマンド名、ファイル名、入力情報、システムが画面に出力した情報など、記述されたとおりの値を示します。
太字	初出語を示します。
イタリック、*イタリック*	変数名または相互参照を示します。
青いアウトライン	オンラインで表示したときのみに見られる青いアウトラインは、相互参照用のハイパーリンクです。アウトライン内をクリックすることにより、参照先オブジェクトにジャンプすることができます。
{ }	構文の記述行の場合、中括弧 {} で囲まれた部分は、選択対象のオプションです。1 つのオプションのみを選択する必要があります。
[ ]	構文の記述行の場合、大括弧 [] で囲まれた部分は、オプションのパラメーターです。
...	構文の記述行の場合、省略符号 ... は直前のパラメーターが繰り返されることを示します。例えば、`option[,...]` は、複数のオプションをコンマで区切って指定できることを意味します。
< >	命名規則では、不等号括弧は名前の個々の要素を囲み、各要素を区別します。 (例: `<server_name><`*`connector_name`*`>tmp.log`)
/、¥	本書では、ディレクトリー・パスに円記号 (¥) を使用します。UNIX システムの場合には、円記号をスラッシュ (/) に置き換えてください。すべての製品のパス名は、使用システムで製品がインストールされたディレクトリーを基準とした相対パス名です。
%text% および *$text*	% 記号で囲まれたテキストは、Windows™ の `text` システム変数またはユーザー変数の値を示します。UNIX 環境における同等の表記は *$text* であり、UNIX の text 環境変数の値を示します。
ProductDir	製品がインストールされているディレクトリーを示します。

本リリースの新機能

リリース 1.2.x の新機能

2004 年 6 月更新。Adapter for CORBA バージョン 1.2.x では、このリリースから以下の項目が新しくなっています。

- バージョン 1.2.x より、Adapter for CORBA は Solaris 7 ではサポートされなくなりました。したがって、このプラットフォーム・バージョンに関する言及がこのガイドから削除されました。
- バージョン 1.2.x より、Windows ユーザーおよび AIX ユーザーは、Adapter for CORBA のインストール中に IBM JDK 1.3.1 SR5 の置かれているディレクトリーのパス名を指定しなければならなくなりました (3 ページの『JDK ソフトウェア』で説明しているように、このソフトウェアは Connector for CORBA をインストールするための前提条件です)。詳細については、 19 ページの『Windows および AIX へのインストール』を参照してください。
- クライアントとして実行するコネクターおよびサーバーとして実行するコネクターの構成シナリオのサンプルが、 23 ページの『第 3 章 アダプターの構成』にあります。詳細については、 31 ページの『構成シナリオのサンプル』を参照してください。

リリース 1.1.x の新機能

2003 年 12 月更新。Adapter for CORBA バージョン 1.1.x では、このリリースから以下の項目が新しくなっています。

- アダプターのインストール情報は、本書から移動しました。この情報の新たな入手先については、 19 ページの『Adapter for CORBA と関連ファイルのインストール』を参照してください。
- バージョン 1.1.x 以降の Adapter for CORBA は Microsoft Windows NT ではサポートされなくなりました。
- バージョン 1.1.x 以降の Adapter for CORBA は IBM Java Object Request Broker (ORB) をサポートしています。詳細については、 4 ページの『Object Request Broker (ORB)』を参照してください。
- バージョン 1.1.x 以降の Connector for CORBA の Object Discovery Agent (ODA) コンポーネントは、IDLJ コンパイラー・ツールを使用して、IDL ファイルから Java プロキシー・クラス定義を生成します。ODA は、その他の IDL コンパイラー・ツールを一切使用しません。詳細については、 4 ページの『IDLJ to Java コンパイラー・ツール』を参照してください。

  Adapter for CORBA バージョン 1.0.x のユーザーは、IDL ファイルを再コンパイルして、ODA バージョン 1.1.x と互換性のある Java プロキシー・クラス定義を再生成する必要があります。この移行手順の詳細については、 57 ページの『IDL ファイル互換性』を参照してください。
- バージョン 1.1.x 以降の Adapter for CORBA は、IBM ORB Transient Naming Server とのみ互換性があります。このアダプターは、その他のネーミング・サー

ビスとは互換性がありません。詳細については、 5 ページの『IBM ORB Transient Naming Server』を参照してください。

- バージョン 1.1.x 以降の Adapter for CORBA には、poa_name コネクター固有構成プロパティーがありません。有効なコネクター固有構成プロパティーの詳細については、 24 ページの『コネクター固有のプロパティー』を参照してください。

第 1 章 概要

Connector for CORBA (Common Object Request Broker Architecture) は、WebSphere Business Integration Adapter for CORBA のランタイム・コンポーネントです。CORBA Adapter には、コネクター、メッセージ・ファイル、構成ツール、および Object Discovery Agent (ODA) が含まれます。コネクターにより、WebSphere 統合ブローカーはビジネス・オブジェクトと、CORBA サーバー上で稼働している対応する CORBA オブジェクトの間でデータを交換できます。また、CORBA オブジェクトがコネクター (CORBA サーバーとして機能する場合) に対してクライアント要求を実行依頼できます。

コネクターは、コネクター・フレームワークとアプリケーション固有のコンポーネントという 2 つのコンポーネントで構成されています。コネクター・フレームワークのコードはすべてのコネクターに共通なので、コネクター・フレームワークは、統合ブローカーとアプリケーション固有のコンポーネントとの仲介役の機能を果たします。アプリケーション固有のコンポーネントには、特定のテクノロジー (この場合は CORBA) またはアプリケーション用に調整されたコードが含まれています。コネクター・フレームワークは、統合ブローカーとアプリケーション固有のコンポーネントとの間で以下のようなサービスを提供します。

- ビジネス・オブジェクトの受信と送信
- 始動メッセージや管理メッセージの交換の管理

本書では、コネクター・フレームワークおよびアプリケーション固有のコンポーネントの両方について解説しています。ここでは、これらの両方のコンポーネントを「コネクター」と呼んでいます。

WebSphere Business Integration Adapters は、いずれも統合ブローカーとともに動作します。Connector for CORBA は、WebSphere InterChange Server、WebSphere MQ Integrator Broker、または WebSphere Application Server とともに動作します。詳細については、ご使用のブローカーに関するインストールおよびインプリメンテーション資料を参照してください。

アダプター環境

アダプターをインストール、構成、使用する前に、環境要件を理解しておく必要があります。

ブローカーの互換性

アダプターが使用するアダプター・フレームワークは、アダプターと通信する統合ブローカーのバージョンとの互換性を備えている必要があります。Adapter for CORBA のバージョン 1.2.x は、以下のバージョンのアダプター・フレームワークおよび以下の統合ブローカーでサポートされています。

- アダプター・フレームワーク:
  - WebSphere Business Integration Adapter Framework バージョン 2.4.0
- 統合ブローカー:
  - WebSphere InterChange Server バージョン 4.2.2
  - WebSphere MQ Integrator バージョン 2.1.0
  - WebSphere MQ Integrator Broker バージョン 2.1.0
  - WebSphere Business Integration Message Broker バージョン 5.0
  - WebSphere Application Server Enterprise バージョン 5.0.2 (WebSphere Studio Application Developer Integration Edition バージョン 5.0.2 と併用)

例外については、「*リリース情報*」を参照してください。

注: 統合ブローカーのインストール手順およびその前提条件については、次の資料を参照してください。

WebSphere InterChange Server (ICS) については、「*システム・インストール・ガイド (UNIX 版)*」または「*システム・インストール・ガイド (Windows 版)*」を参照してください。

Message Brokers (WebSphere MQ Integrator Broker、WebSphere MQ Integrator、および WebSphere Business Integration Message Broker) については、「*WebSphere Message Brokers 使用アダプター・インプリメンテーション・ガイド*」およびそれぞれの Message Brokers のインストールに関する資料を参照してください。一部の資料は次の Web サイトにあります。

http://www.ibm.com/software/integration/mqfamily/library/manualsa/

WebSphere Application Server については、「*アダプター実装ガイド (WebSphere Application Server)*」および次の資料を参照してください。

http://www.ibm.com/software/webservers/appserv/library.html

アダプターの規格

このアダプターは CORBA 2.3.1 仕様に従って記述されているため、この規格に従って設計されている CORBA アプリケーションと互換性があります。

アダプターのプラットフォーム

アダプターは以下のプラットフォーム上で稼働します。

- Windows 2000
- Solaris 8
- HP-UX 11.i
- AIX(R) 5.1、5.2

アダプターの依存関係

Connector for CORBA には、実際のアダプター・プラットフォームに従って、それぞれが独自のインストール要件を持つ以下の依存関係があります。

- 『JDK ソフトウェア』
- 4 ページの『Object Request Broker (ORB)』
- 4 ページの『IDLJ to Java コンパイラー・ツール』
- 5 ページの『JavaC』
- 5 ページの『IBM ORB Transient Naming Server』

JDK ソフトウェア

Java Development Kit (JDK) バージョン 1.3.1 は Adapter for CORBA をインストールする前提条件となります。

Windows 2000: WebSphere Business Integration Adapter Framework バージョン 2.4.0 では、IBM JDK バージョン 1.3.1 SR5 は、別にインストールするようになっています。IBM JDK バージョン 1.3.1 SR5 は WebSphere Business Integration Adapter Framework のインストールの一部としてインストールされるのではないことに注意してください。 JDK をインストールするためには、別にインストール作業が必要です。JDK を WebSphere Business Integration Adapter Framework からインストールする方法の詳細は、該当のソフトウェア・パッケージを参照してください。

Solaris: インストールされた WebSphere Business Integration Adapter Framework バージョン 2.4.0 によって指定されている Sun JDK 1.3.1 をインストールします。JDK は、WebSphere Business Integration Adapter Framework のインストールの一部としてはインストールされません。 JDK のインストールには、Sun Microsystems の提供する別個のインストール・ソフトウェアを実行する必要があります。

AIX: IBM JDK バージョン 1.3.1 SR5 をインストールします。このインストール・ソフトウェアの入手方法の詳細については、IBM テクニカル・サポートにご連絡ください。

HP-UX: インストールされた WebSphere Business Integration Adapter Framework バージョン 2.4.0 によって指定されている HP JDK 1.3.1 をインストールします。JDK は、WebSphere Business Integration Adapter Framework のインストールの一部

としてはインストールされません。 JDK のインストールには、HP の提供する別個のインストール・ソフトウェアを実行する必要があります。

Object Request Broker (ORB)

Adapter for CORBA は、IBM Java Object Request Broker (ORB) をサポートするオブジェクト・リクエスト・ブローカー環境を前提とします。

Windows 2000: 必須の IBM Java ORB は、IBM JDK 1.3.1 SR5 に用意されています。ORB のインストール方法の説明については、JDK インストール・パッケージを参照してください。

Solaris: WebSphere Business Integration Adapter Framework バージョン 2.4.0 には、必須の IBM Java ORB ソフトウェアが用意されています。ORB は、アダプター・フレームワークのインストールの一部としてインストールされます。

AIX: 必須の IBM Java ORB は、IBM JDK 1.3.1 SR5 に用意されています。ORB のインストール方法の説明については、JDK インストール・パッケージを参照してください。

HP-UX: WebSphere Business Integration Adapter Framework バージョン 2.4.0 には、必須の IBM Java ORB ソフトウェアが用意されています。ORB は、アダプター・フレームワークのインストールの一部としてインストールされます。

IDLJ to Java コンパイラー・ツール

Connector for CORBA の Object Discovery Agent (ODA) コンポーネントは、IDLJ コンパイラー・ツールを使用します。このツールは、Java プロキシー・クラス定義を生成するために ODA が使用するコンパイラーであり、このクラス定義を使用して、コネクターは、CORBA クライアントまたはサーバーとして機能することができます。

CORBA struct、interface、メソッド、その他のプログラマチック・エンティティーは、IDL (Interface Definition Language) ファイル内で定義されます。ODA により実行される IDLJ コンパイラー・ツールは、IDL ファイル内で定義されている CORBA プログラマチック・エンティティーをプロキシー・クラスに変換します。実行時に、コネクターはプロキシー・クラスからプロキシー・オブジェクトを作成し、プロキシー・オブジェクトを使用して、そのオブジェクトに対応する CORBA クラス (IDL ファイルに定義されている) のメソッドを呼び出します。

Adapter for CORBA バージョン 1.0.x のユーザーは、IDL ファイルを再コンパイルして、ODA バージョン 1.2.x と互換性のある Java プロキシー・クラス定義を再生成する必要があります。この移行手順の詳細については、57 ページの『IDL ファイル互換性』を参照してください。

注: 一部の CORBA サーバーのアプリケーション・プロバイダーは、プロキシー・クラスを定義する `.jar` ファイルを提供して、コネクターの ODA により (IDLJ コンパイラー・ツールを使用して) 生成 (コンパイル) されるオブジェクト定義の必要性を回避します。CORBA サーバー・アプリケーションのプロバイダーからプロキシー・クラス定義の `.jar` ファイルが提供される場合には、このファイルを *ProductDir*¥connectors¥CORBA¥ext ディレクトリー、またはプロキシー・クラス .jar ファイル保管ディレクトリーとして指定したディレクト

リーにコピーしてください。この *ProductDir* は、コネクター製品のインストール先ディレクトリーです。 Business Object Designer を使用して、OutputFileDir 構成エージェント・プロパティーに適切なディレクトリー名を指定します。このプロパティーとその設定方法の詳細については、59 ページの『エージェントの構成』を参照してください。 ODA のプロパティーの定義の詳細については、57 ページの『第 5 章 ビジネス・オブジェクトの作成および変更』を参照してください。

Windows 2000: 必須の IDLJ コンパイラー・ツールは、IBM JDK 1.3.1 SR5 に用意されています。IDLJ のインストール方法の説明については、JDK インストール・パッケージを参照してください。

Solaris: WebSphere Business Integration Adapter Framework バージョン 2.4.0 には、必須の IDLJ ソフトウェアが用意されています。IDLJ は、アダプター・フレームワークのインストールの一部としてインストールされます。

AIX: 必須の IDLJ コンパイラー・ツールは、IBM JDK 1.3.1 SR5 に用意されています。IDLJ のインストール方法の説明については、JDK インストール・パッケージを参照してください。

HP-UX: WebSphere Business Integration Adapter Framework バージョン 2.4.0 には、必須の IBM IDLJ ソフトウェアが用意されています。IDLJ は、アダプター・フレームワークのインストールの一部としてインストールされます。

JavaC

Java プログラミング言語コンパイラーである JavaC は、Adapter for CORBA の前提条件です。JavaC は JDK 1.3.1 の一部であり、したがって、JDK をインストールすると必ず、その一部としてインストールされます。JDK の前提条件の詳細については、3 ページの『JDK ソフトウェア』を参照してください。

IBM ORB Transient Naming Server

IBM ORB Transient Naming Server は、Connector for CORBA の前提条件です。このサービスは、オブジェクト参照に名前を結合することにより、CORBA オブジェクトの命名を可能にする、必須の CORBA ネーミング・サービスを提供します。クライアントとして実行される場合は、コネクターは、IBM Java Object Request Broker (ORB) のみを使用する CORBA サーバーと接続できます。サーバーとして実行される場合は、他の ORB ベンダーの ORB を使用する CORBA クライアントは、コネクターと接続できます。IBM Java ORB の詳細については、4 ページの『Object Request Broker (ORB)』を参照してください。

WebSphere Business Integration Adapter Framework バージョン 2.4.0 には、必須の IBM ORB Transient Naming Server ソフトウェアが用意されています。このソフトウェアは、アダプター・フレームワークのインストールの一部としてインストールされます。 IBM ORB Transient Naming Server を起動するには、実際のプラットフォームに応じて、以下のいずれかのコマンドを ¥connectors¥CORBA¥ ディレクトリーから実行します。

Windows 2000	Solaris、AIX、HP-UX
NamingService.bat	NamingService.sh

このコマンドは、ユーザー定義ポート番号を必要とします。例えば、Windows 2000 環境で実行される以下のコマンドは、ポート 1100 で IBM ORB Transient Naming Server のインスタンスを起動します。

```
NamingService.bat 1100
```

WebSphere Business Integration Adapter Framework に用意されている必須の IBM ORB Transient Naming Server ソフトウェアを使用するには、NamingService.bat/NamingService.sh コマンドによって起動されるサービスに、使用している既存の CORBA サービスの再登録を行うことが必要になる場合があることに注意してください。

ロケール依存データ

コネクターは国際化され、2 バイト文字セットをサポートする CORBA インターフェースを記述する 2 バイト文字セットの配信をサポートし、指定した言語でメッセージ・テキストを配信できるようになっています。ある文字コードを使用するロケーションから別のコード・セットを使用するロケーションへ、コネクターがデータを転送するとき、コネクターはデータの意味を保存するため、文字変換を実行します。

Java 仮想マシン (JVM) 内部の Java ランタイム環境では、Unicode 文字コード・セットでデータを表現します。Unicode は、既知の文字コード・セットのほとんど (単一バイトおよびマルチバイトの両方) に対応するエンコード方式を含んでいます。IBM WebSphere Business Integration システムのほとんどのコンポーネントは Java で書かれています。そのため、統合コンポーネント間でデータを転送するときは、ほとんどの場合文字変換は必要ありません。

用語

本書で使用する用語は、以下のとおりです。

- **ASI (アプリケーション固有情報)** 特定のアプリケーションまたはテクノロジー用に調整されたメタデータ。ASI は、ビジネス・オブジェクトの属性レベル、動詞レベル、およびビジネス・オブジェクト・レベルの両方にあります。『**動詞 ASI**』も参照してください。
- **BO (ビジネス・オブジェクト)** ビジネス・エンティティー (従業員など) およびデータ上のアクション (作成操作や更新操作) を表す一連の属性。WebSphere Business Integration システムのコンポーネントは、ビジネス・オブジェクトを使用して、情報を交換したり、アクションを起動したりします。
- **BO (ビジネス・オブジェクト) ハンドラー** アプリケーションと対話するメソッドを含み、要求ビジネス・オブジェクトをアプリケーション操作に変換するコネクター・コンポーネント。
- **接続オブジェクト** 接続クラスのインスタンスである特殊なプロキシー・オブジェクト。接続とは、状態情報を含むアプリケーションを参照することです。アダプター・サイドにおける接続のインスタンスごとに、CORBA サイドに対応するオブジェクトがあります。バッチでの接続インスタンスの生成、任意での接続の検索、接続プールへ接続を戻す操作、および他のスレッドによる接続再利用が可能です。

- **接続プール** 接続オブジェクトを保管および検索するために使用するリポジトリー。
- **CORBA オブジェクト** コネクターは CORBA サーバーと対話するため、ビジネス・オブジェクトと CORBA オブジェクトの間で処理を行います。コネクターの処理時に、CORBA オブジェクト (アプリケーション) はプロキシー・オブジェクトによってコネクター内に表されます。**プロキシー**は、CORBA オブジェクトを表す Java クラスです。
- **ファクトリー** アプリケーションを参照する特殊なプロキシー・オブジェクト。適切なコネクター・プロパティーが設定されている場合、ファクトリー・オブジェクト (コネクターとともに永続的) は接続プール内に置かれる接続を作成したり、CORBA アプリケーションで使用される CORBA オブジェクトを作成することができます。作成される接続数は、`PoolSize` プロパティーで指定される値によって異なります。
- **外部キー** 一意的に子ビジネス・オブジェクトを識別する値を持つ単純属性。通常、この属性は、子の基本キー値を持つことで子ビジネス・オブジェクトとその親を識別します。Connector for CORBA は、外部キーを使用して、プール可能な接続オブジェクトを指定します。
- **IDLJ** Connector for CORBA は、IDLJ コンパイラー・ツールを使用する IBM Java Object Request Broker (ORB) をサポートします。このツールにより、Java プログラムは、実行時にコネクターがプロキシー・オブジェクトを生成するために必要とする Java プロキシー・クラスを生成することによって CORBA オブジェクトと通信し、次に CORBA オブジェクトを呼び出すことができます。CORBA オブジェクトのプロパティー、構造、およびメソッドは、IDL (Interface Definition Language) ファイルで定義されます。IDLJ コンパイラー・ツールで ODA を使用して作成されたプロキシー・オブジェクト・クラス定義を使用することにより、コネクターは IDL で定義されているオブジェクトの CORBA メソッドを呼び出すことができます。
- **ODA (Object Discovery Agent)** アプリケーション内で指定されたエンティティーを検査し、そのエンティティーの中からビジネス・オブジェクト属性に対応するエレメントを「発見」することで、自動的にビジネス・オブジェクト定義を生成するツール。ODA は、アダプターをインストールすると、自動的にインストールされます。Business Object Designer では、ODA にアクセスして対話式にやりとりするグラフィカル・ユーザー・インターフェースを提供しています。
- **ORB (オブジェクト・リクエスト・ブローカー)** クライアントとサーバー間のミドルウェアとして機能する、CORBA プログラミング・モデルのコンポーネント。CORBA モデルでは、クライアントはネットワークに接続されているサーバーを認識していなくてもサービスを要求できます。さまざまな ORB が要求を受け取り、それを適切なサーバーに転送して、結果をクライアントに戻します。
- **呼び出しごとのオブジェクト・プール** 単一の `doVerbFor` メソッドを呼び出す際に、あるメソッドがら次のメソッドへ渡す必要のあるオブジェクトを保管するためのプログラマチック・エンティティー。保管されるオブジェクトは、プロキシー・オブジェクトの場合もあれば、単純属性の場合もあります。
- **プロキシー・クラス** コネクター内の CORBA オブジェクトを表す Java クラス。コネクターは、ビジネス・オブジェクトの ASI で指定されたプロキシー・クラス名のプロキシー・オブジェクト・インスタンスを作成します。

- **動詞 ASI (アプリケーション固有情報)** 動詞 ASI を使用し、特定の動詞について、その動詞がアクティブな場合にコネクターがビジネス・オブジェクトを処理する方法を指定します。動詞 ASI には、現在の要求ビジネス・オブジェクトを処理するために呼び出すメソッドの名前を含むことができます。

クライアントとして機能するコネクターのアーキテクチャー

コネクターは、次の 2 つの方法で要求を処理できます。

- コネクターがクライアントとして機能して、ビジネス・オブジェクト要求を CORBA サーバーに送信します (9 ページの図 1)。これらの要求は、外部 CORBA サーバー上で実行されているオブジェクトのメソッドを呼び出します。
- コネクターがサーバーとして機能して、外部 CORBA クライアントから要求を受け取ります (13 ページの図 3)。これらの要求は、統合ブローカー上のコラボレーションを呼び出します。コラボレーションは、例えば、外部アプリケーション内のデータを更新できます。

このセクションでは、クライアントとして機能する場合の CORBA コネクターのアーキテクチャーについて説明します。サーバーとして機能する CORBA コネクターの詳細については、12 ページの『サーバーとして機能するコネクターのアーキテクチャー』を参照してください。

要求フロー

9 ページの図 1 は、コネクターがクライアントとして機能する場合の要求フローを示しています。このシナリオで、コネクターは外部 CORBA サーバー上にあるオブジェクトのメソッドを呼び出します。コネクターはクライアントとして機能し、ORB を介して CORBA サーバーと通信します。コネクターは、CORBA オブジェクトがあるサーバーにオブジェクト要求を送信することにより、CORBA サーバーと通信します。

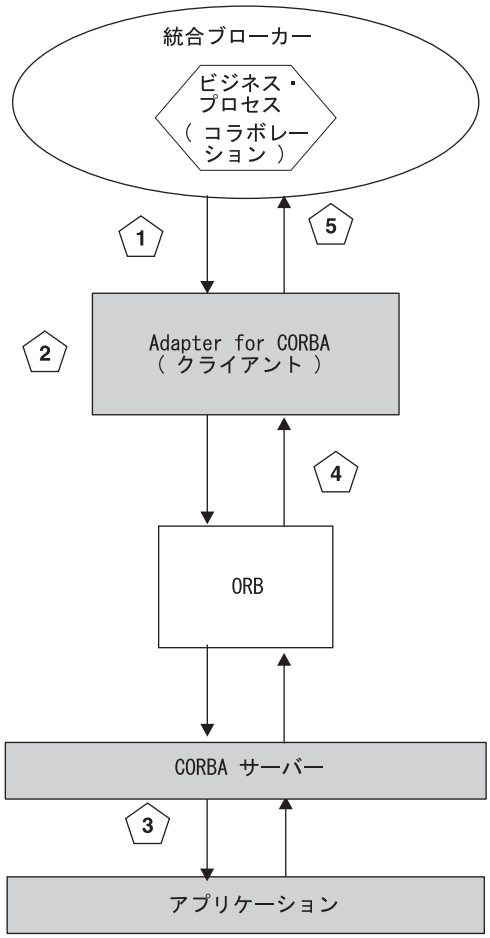


図 1. コネクターがクライアントとして機能する場合の要求プロセス

1. コネクターが、統合ブローカーからビジネス・オブジェクト要求を受け取ります。
2. コネクターが、ビジネス・オブジェクトのプロキシー・オブジェクト・インスタンスを作成します。プロキシー・オブジェクト・インスタンスは、コネクターが要求を送信する先の CORBA オブジェクトの代理として機能します。コネクターがプロキシー・オブジェクトを作成および処理する方法の詳細については、10 ページの『コネクターがクライアントとして機能する仕組み』を参照してください。
3. コネクターが、プロキシー・オブジェクトを使用して、CORBA サーバー上で実行中の対応する CORBA オブジェクトにアクセスし、アプリケーション (オブジェクト) にデータを書き込むことによって、プロキシー・オブジェクトを処理します。コネクターは CORBA オブジェクトのメソッドを呼び出すこともできます。
4. コネクターが CORBA サーバー・オブジェクトからデータを読み取りまたは取得して、プロキシー・オブジェクトを更新します。
5. コネクターが、元のオブジェクト要求が成功したのか、または失敗 (FAIL 状況) したのかを示すメッセージを統合ブローカーに戻します。要求が成功した場合は、コネクターも統合ブローカーに更新されたビジネス・オブジェクトを戻します。

コネクターがクライアントとして機能する仕組み

このセクションでは、図 2 に示すように、コネクターがクライアントとして機能する場合に、コネクターのさまざまな部分がビジネス・オブジェクトを処理する仕組みについて説明します。

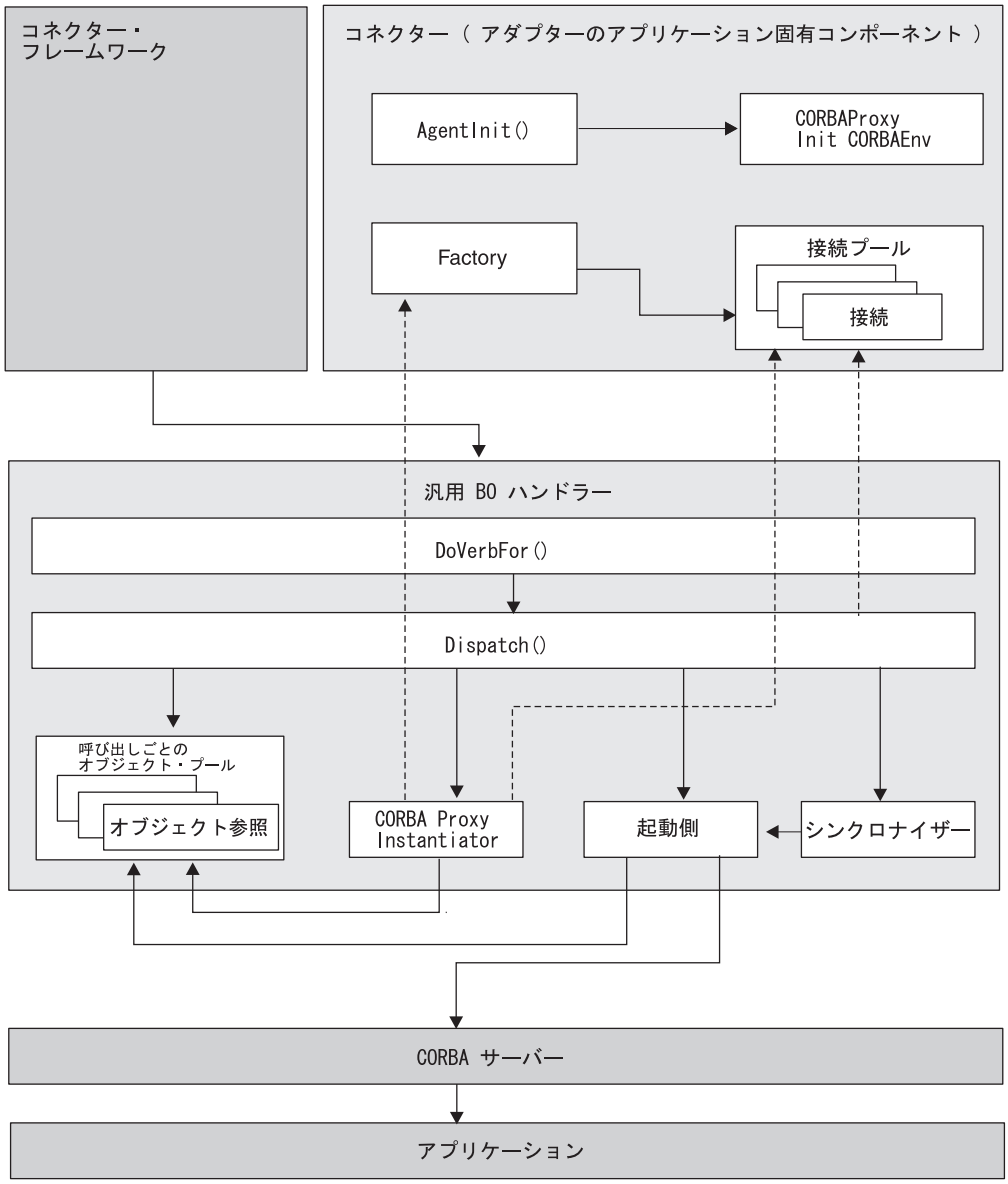


図 2. クライアントとして機能する Connector for CORBA

1. コネクターを初めて始動し、それがクライアントとして機能する場合、コネクターの Agent クラスは以下の初期設定プロセスを実行します。
   - ORB を初期化することによって CORBA 環境をインスタンス化します。
   - コネクター・プロパティーの設定方法に応じて、以下のいずれかを実行します。コネクター・プロパティー、および以下の各シナリオでコネクター・プロパティーがどのように作用するかについての詳細は、 24 ページの『コネクター固有のプロパティー』を参照してください。

- **シナリオ 1**: アプリケーションを参照するファクトリー・オブジェクト・インスタンスの作成。コネクターとともに永続的なファクトリー・オブジェクトは、接続プールに置かれる接続を作成します。作成される接続数は、コネクターの PoolSize プロパティーで指定された値によって異なります。
- **シナリオ 2**: 接続プールに配置される接続オブジェクトのみの作成。接続数は、PoolSize プロパティーで指定した値によって異なります。このシナリオでは、ファクトリー・オブジェクトは作成されません。
- **シナリオ 3**: ビジネス・オブジェクトによるメソッド呼び出しの対象となるファクトリー・プロキシー・オブジェクトの作成 (ファクトリー・クラスは BO のプロキシー・クラス ASI と一致)。このシナリオでは、接続は作成されません。

2. 統合ブローカーが、ビジネス・オブジェクト形式で、要求をコネクターに送信します。
3. コネクターの BO ハンドラーがオブジェクトを受け取ります。
4. BO ハンドラーの doVerbFor() メソッドが、BO の ASI を読み取ってプロキシー・クラス名を取得する Dispatch() メソッドを呼び出します。Dispatch() メソッドがプロキシー・クラス名を取得し、それを CORBA Proxy Instantiator に送信します。
5. CORBA Proxy Instantiator が、プロキシー・クラス名を使用して (有効な Java クラス表記、例えば xxxxx.myclass を使用して修飾された) プロキシー・クラスをロードし、プロキシー・オブジェクト・インスタンスを作成して、それを呼び出しごとのオブジェクト・プールにロードします。CORBA Proxy Instantiator は、オブジェクトが以下のいずれかであるかどうか検査します。
   - 接続であるか。接続の場合は、接続オブジェクトとして接続プールから検索します。
   - ファクトリー・オブジェクトであるか。ファクトリー・オブジェクトの場合は、静的オブジェクトとしてファクトリーから検索します。CORBA Proxy Instantiator は、ファクトリー・メソッドがビジネス・オブジェクト ASI に指定されているかどうかも検査します。指定されている場合は、ファクトリー・オブジェクトのファクトリー・メソッドを使用します。
6. ディスパッチを実行して BO の動詞 ASI を読み通し、メソッドのリストを作成します。動詞 ASI は、属性名が配列されたリストです。各属性は、プロキシー・オブジェクトに対するメソッドを表します。つまり、動詞 ASI とは、メソッドのリストではなく、それぞれがプロキシー・オブジェクト・メソッドを表す値を持つ属性のリストです。
7. 動詞 ASI リストの各メソッドについて、BO ハンドラーの InvokeMethods() メソッドは、InvokeMethod() を呼び出し、以下のいずれかを実行します。
   - メソッドが通常のメソッドである場合は、起動側を呼び出します。引き数が外部キーとしてマークされている場合は、呼び出しごとのオブジェクト・プールに引き数を保管します。属性が取り込まれていない場合は、use_attribute_value の属性 ASI を検査します。use_attribute_value の ASI が存在する場合は、呼び出しごとのオブジェクト・プールからオブジェクトのプルを試みます。

- プロキシー・オブジェクトのすべての属性に対して、シンクロナイザーのロード操作および保管操作 (BO ハンドラーのオブジェクト同期化処理) を呼び出します。呼び出される操作は、動詞 ASI の内容によって異なります。LoadFromProxy (ロード) および WriteToProxy (保管) は、動詞 ASI に組み込み可能な定義済み関数です。これらの関数は、ビジネス・オブジェクトの単純属性を CORBA オブジェクトのパブリック・プロパティーに同期させることを目的としています。
- 特定の単純属性に対して、ロード操作 (LoadFromProxy 関数) または保管操作 (WriteToProxy 関数) を呼び出します (LoadFromProxy を呼び出すと、プロキシー・プロパティーが取得され、取得された値に BO プロパティーが設定されます。WriteToProxy を呼び出すと、BO の値を使用してプロキシー・プロパティーが設定されます)。

注: 動詞 ASI が空の場合、BO ハンドラーは、設定済みのパラメーターを指定して、BO に対するメソッドを検索し、呼び出します。1 つのメソッドのみがパラメーターを設定できます。それ以外、つまり複数のメソッドを設定すると、動詞 ASI が空であっても、コネクターはエラーをログに記録し、FAIL コードを戻します。

8. プロキシー・オブジェクトの各メソッドについて、起動側は、以下を実行して、メソッドのパラメーターおよび引き数を構成します。
   - 起動側は、属性に (String などの単一データ型ではなく) BO 型を発見すると、アクティブな BO ハンドラーに対して再帰的に `Dispatch()` メソッドを呼び出します。
   - `Dispatch()` は、親メソッドがそのメソッド呼び出しを起動するために使用するプロキシー・オブジェクトを戻します。
   - シンクロナイザーと呼ばれる、BO ハンドラーの同期化処理によって WriteToProxy が起動され、CORBA オブジェクト (プロキシー・オブジェクト) の各プロパティーに値が保管 (設定) されます。これにより、CORBA サーバーのデータが更新されます。保管される値は、CORBA オブジェクトが対応するビジネス・オブジェクトの対応する属性に基づきます。
9. 値が CORBA サーバーから戻されると、LoadFromProxy 関数がプロキシー・オブジェクトから戻されたデータを BO にロードします。(戻りパラメーター用に、コネクターは戻りプロキシー・オブジェクトを作成し、`in/out` パラメーターを更新します。)
10. コネクターが、ビジネス・オブジェクトを統合ブローカーに戻します。

サーバーとして機能するコネクターのアーキテクチャー

クライアントとして要求を処理するほかに、コネクターは、外部 CORBA クライアントからの要求を受け取るサーバーとして機能することもできます。これらの要求は、統合ブローカー上のコラボレーションを呼び出します。コラボレーションは、例えば、外部アプリケーション内のデータを更新できます。

このセクションでは、サーバーとして機能する場合の CORBA コネクターのアーキテクチャーについて説明します。クライアントとして機能するコネクターの詳細については、8 ページの『クライアントとして機能するコネクターのアーキテクチャー』を参照してください。

要求フロー

図 3 は、コネクターがサーバーとして機能する場合の要求フローを示しています。このシナリオでは、コネクターは CORBA クライアントの代理としてコラボレーションを実行します。

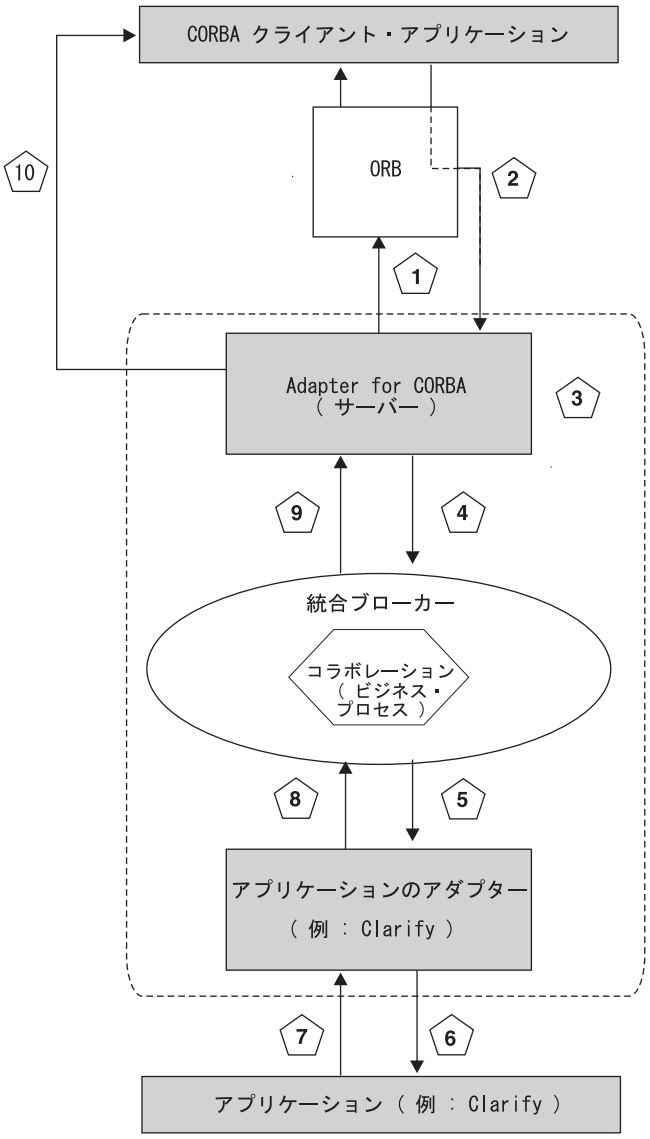


図 3. コネクターがサーバーとして機能する場合の要求プロセス

1. コネクターを初めて始動するときに、サーバー・サイド・オブジェクトとして構成されたすべてのビジネス・オブジェクトが、ORB を使用してコネクターによって登録されます。ビジネス・オブジェクトをサーバーとして構成する方法については、39 ページの『第 4 章 ビジネス・オブジェクトについて』を参照してください。
2. CORBA クライアント・アプリケーションが、登録済みのサーバー・オブジェクトに対してコネクターを介してメソッド呼び出し要求を送信します。

3. コネクターが、メソッド呼び出しを登録済み CORBA サーバー・オブジェクトの属性 ASI に指定された動詞に変換します。ビジネス・オブジェクトの属性 ASI の詳細については、 47 ページの『属性レベル ASI』を参照してください。
4. コネクターが、アダプターの Agent クラスの excecuteCollaboration() メソッドを呼び出します。ビジネス・オブジェクト動詞 (ステップ 3) を特定のコラボレーションに関連付ける、コネクターの BO_COLLAB_MAPPING プロパティーに基づいて、executeCollaboration() は適切なコラボレーション (ビジネス・プロセス) を呼び出します。BO_COLLAB_MAPPING プロパティーの詳細については 23 ページの『コネクターの構成』を参照してください。
5. 統合ブローカーが、コラボレーションまたは実行されたビジネス・プロセスから、外部アプリケーション (つまり、コラボレーションがデータを交換するアプリケーション。 13 ページの図 3 で、この外部アプリケーションの例は Clarify です) のコネクターにデータを送信します。
6. 外部アプリケーションのコネクターが、そのアプリケーションにデータを送信します。
7. アプリケーションが、外部アプリケーションのコネクターにメッセージを戻します (例えば、 13 ページの図 3 では Clarify)。
8. 外部アプリケーションのコネクターが、統合ブローカーにメッセージを転送します。
9. 統合ブローカーが、CORBA サーバーとして機能するコネクター (ステップ 4 で excecuteCollaboration() メソッドを呼び出したコネクター) にメッセージを転送します。
10. 例外が発生した場合、元の要求を開始した CORBA クライアント・アプリケーション (ステップ 2 (13 ページ)) に例外が送信されます。

コネクターがサーバーとして機能する仕組み

サーバーとして機能する場合、コネクターはクライアントとして機能する場合とは異なる方法でビジネス・オブジェクト処理を実行します。このセクションでは、 15 ページの図 4 に示すように、サーバーとして機能するコネクターのさまざまな部分がビジネス・オブジェクトを処理する仕組みについて説明します。

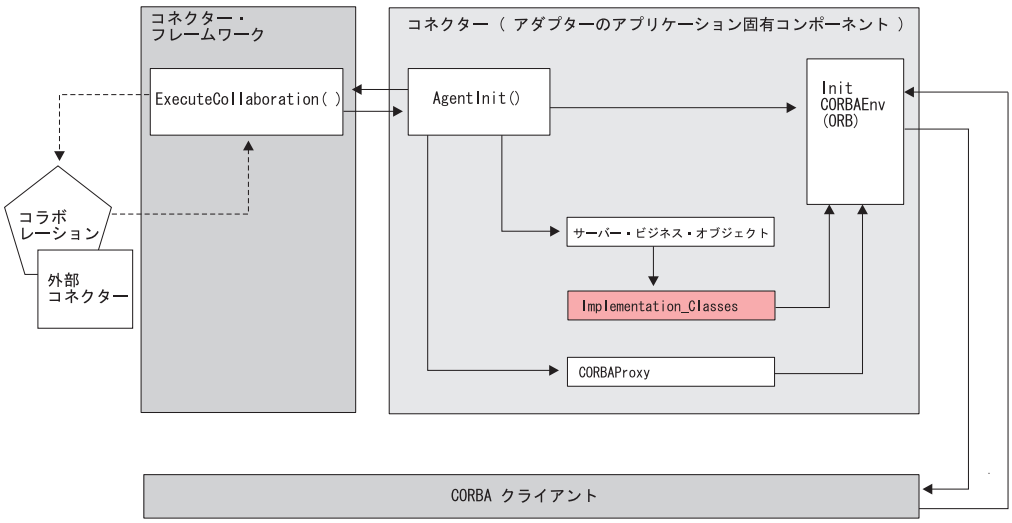


図 4. サーバーとして機能する Connector for CORBA

1. コネクターを始動するとき、コネクターの Agent クラスは以下の初期設定プロセスを実行します。
   - ORB を初期化することによって CORBA 環境をインスタンス化します。
   - サーバー・オブジェクトとして処理されるビジネス・オブジェクトに関する情報をインプリメンテーション・クラスに受け渡します。ビジネス・オブジェクトがサーバー・オブジェクトであるかどうか判断するため、コネクターは BO ASI に属性 `object_type=CorbaImplObject` が含まれるか検査します。
   - ビジネス・オブジェクトに対応するインプリメンテーション・クラスを ORB に登録します。任意のビジネス・オブジェクトのインプリメンテーション・クラスを `implementation_class` ASI に指定してください。クラス定義は、ビジネス・オブジェクトを作成するときに ODA によって生成される `.jar` ファイルに保管されます。詳細については、57 ページの『第 5 章 ビジネス・オブジェクトの作成および変更』を参照してください。
2. Agent クラスが、コネクター・プロパティーの設定内容に応じて、ファクトリー、ファクトリー・プロキシー・オブジェクトのインスタンス、または接続オブジェクトをセットアップします。コネクター・プロパティーの詳細については、24 ページの『コネクター固有のプロパティー』を参照してください。
3. 外部 CORBA クライアントが、コネクター内の CORBA サーバー・ビジネス・オブジェクトに要求を送信して、コラボレーションを呼び出します。ORB を介して送信される要求は、メソッドの入力引き数の形式を取ります。
4. ステップ 3 (14 ページ) およびステップ 4 (14 ページ) で説明したように、以下のようになります。
   - コネクターが、メソッドの引き数値を使用してビジネス・オブジェクトを作成します。
   - コネクターは、CORBA サーバー・ビジネス・オブジェクトの属性 ASI に指定されているようにビジネス・オブジェクトに動詞を設定します。
   - 次に、コネクターはコネクターの Agent クラスの `executeCollab()` メソッドを呼び出します。

- executeCollab() メソッドは、BO_COLLAB_MAPPING プロパティー (ビジネス・オブジェクト動詞をコラボレーションにマップするコネクター・プロパティー) に指定されているコラボレーションを検索します。コラボレーションが見つかった場合、コネクターはそのコラボレーションを実行します。コラボレーションが見つからない場合、例外が発生します。(戻りパラメーター用に、コネクターは戻りプロキシー・オブジェクトを作成し、in/out パラメーターを更新します。)

5. コラボレーション処理の結果のデータが、ORB を介して CORBA クライアント・アプリケーションに戻されます。

ビジネス・オブジェクト要求

ビジネス・オブジェクト要求は、統合ブローカーがビジネス・オブジェクトをコネクターに送信する際に処理されます。ビジネス・オブジェクトの唯一の要件は、対応する CORBA オブジェクト (プロキシー・オブジェクトが代行) にマップされる必要があるということです。プロキシー・クラスは、コネクター内の CORBA オブジェクトを表す Java クラスです。実行時に、コネクターはビジネス・オブジェクトの ASI で指定されたプロキシー・クラス名のプロキシー・オブジェクト・インスタンスを作成します。

動詞の処理

注: このセクションは、クライアントとして機能する場合のコネクターに関連しています。ここで説明する動詞の処理の問題は、コネクターがサーバーとして機能する場合には適用されません。

コネクターは、各ビジネス・オブジェクトの動詞を基にブローカーによってコネクターへに渡されるビジネス・オブジェクトを処理します。

コネクター・フレームワークはブローカーから要求を受け取ると、要求ビジネス・オブジェクトのビジネス・オブジェクト定義に関連するビジネス・オブジェクト・ハンドラー・クラスの doVerbFor() メソッドを呼び出します。doVerbFor() メソッドの役割は、要求ビジネス・オブジェクトのアクティブな動詞に基づいて、実行する動詞処理を決定することです。また、要求ビジネス・オブジェクトから情報を取得し、操作要求を作成してアプリケーションに送信するという役割もあります。

コネクター・フレームワークが要求ビジネス・オブジェクトを doVerbFor() に渡すと、このメソッドは、ビジネス・オブジェクト ASI を検索して、BO ハンドラーを起動します。起動された BO ハンドラーは、動詞 ASI を読み取り、それを一連の呼び出し可能関数に変換します。動詞 ASI は、対象の動詞について呼び出す必要があるメソッドが配列されたリストです。呼び出しが作成される順序は、オブジェクトの処理を成功させる上で重要です。

動詞 ASI が空の場合、BO ハンドラーは、設定済みのパラメーターを指定してメソッドを検索し、呼び出します。設定できるのは 1 つのメソッドのみです。それ以外、つまり、複数のメソッドを設定すると、動詞 ASI が空であっても、コネクターはエラーをログに記録し、FAIL コードを戻します。エラー処理の詳細については、71 ページの『エラー処理』を参照してください。

コネクターでは特定の動詞をサポートしていませんが、ユーザーは ODA を使用することにより、カスタム動詞を構成できます。事前に用意されている標準の動詞は、Create、Retrieve、Update、および Delete です。 Business Object Designer で実行している Object Discovery Agent (ODA) を使用して指定したセマンティックが、これらの動詞に使用されます。ODA を使用してメソッド呼び出しシーケンスを動詞に割り当てる方法の詳細については、57 ページの『第 5 章 ビジネス・オブジェクトの作成および変更』を参照してください。

注: ユーザーは、動詞 ASI に 2 つの定義済み関数 (`LoadFromProxy` および `WriteToProxy`) を指定することができます。これらの関数は、ビジネス・オブジェクトの単純属性を CORBA オブジェクトのパブリック・プロパティーに同期させることを目的としています。

カスタム・ビジネス・オブジェクト・ハンドラー

ビジネス・オブジェクトを作成する場合、BO の動詞 ASI に `CBOH` キーワードを指定すると、デフォルトの BO ハンドラーをオーバーライドできます。コネクターの実行時に、`doVerbFor()` メソッドを使用して、ビジネス・オブジェクト ASI を検索します。`CBOH` キーワードが検出されると、`doVerbFor()` によってカスタム BO ハンドラーが起動されます。

コネクターでは、親レベルのビジネス・オブジェクトのみでカスタム BO ハンドラーをサポートします。カスタム BO ハンドラー作成の詳細については、「コネクター*開発ガイド*」を参照してください。

第 2 章 アダプターのインストール

インストール作業の概要

Adapter for CORBA をインストールするには、必要なアダプター前提条件がご使用の環境に存在するかを確認し、統合ブローカーをインストールしてから、アダプターのインストールを実行する必要があります。

アダプターの前提条件の確認

アダプターをインストールする前に、アダプターをインストールおよび実行するための環境の前提条件がご使用のシステムですべて満たされていることを確認してください。詳細については、2 ページの『アダプター環境』を参照してください。

統合ブローカーのインストール

統合ブローカーのインストール、つまり、WebSphere Business Integration システムをインストールし、ブローカーを始動するタスクについては、ご使用のブローカーに関する資料に説明されています。Connector for CORBA がサポートするブローカーの詳細については、2 ページの『ブローカーの互換性』 を参照してください。

ブローカーのインストール方法についての詳細は、ご使用のブローカーの実装に関する資料を参照してください。

Adapter for CORBA と関連ファイルのインストール

WebSphere Business Integration アダプター製品のインストールについては、次のサイトで WebSphere Business Integration Adapters Infocenter にある「*WebSphere Business Integration Adapters インストール・ガイド*」を参照してください。

http://www.ibm.com/websphere/integration/wbiadapters/infocenter

Windows および AIX へのインストール

Windows または AIX プラットフォームにコネクターをインストールする場合、インストーラーによって IBM JDK 1.3.1 SR5 の置かれているディレクトリーのパス名を指定するよう求められることに注意してください (3 ページの『JDK ソフトウェア』 で説明しているように、このソフトウェアは Connector for CORBA をインストールするための前提条件です)。

インストール時にこのディレクトリー情報を提供できるように、この情報を用意しておいてください。そうしないと、インストーラーを途中で終了し、ご使用のマシンの JDK ディレクトリーを探し出してから、インストーラーを最初からやり直さなければならなくなります。

アダプターには、次のスクリプトが提供されており、これを実行して JDK ホーム・ディレクトリーのパス名を取得することができます。

- ¥connectors¥CORBA¥BIA_CORBAEnv.bat (Windows)
- ¥connectors¥CORBA¥BIA_CORBAEnv.sh (Unix)

コネクター・ファイルの構造

アダプターは、コネクターに関連付けられた標準ファイルをご使用のシステムにコピーします。

このユーティリティーは、コネクターを *ProductDir*¥connectors¥CORBA ディレクトリーにインストールして、「スタート」メニューにコネクターへのショートカットを追加します。*ProductDir* は、製品がインストールされているディレクトリーを指します。

表 1 に、コネクターが使用するファイル構造を示します。また、インストーラーでコネクターのインストールを選択すると自動的にインストールされるファイルも示します。

表 1. コネクターのファイル構造

***ProductDir* のサブディレクトリー**	**説明**
¥connectors¥CORBA¥BIA_CORBA.jar	CORBA コネクターのみに使用されるクラスを含みます。
¥connectors¥CORBA¥start_CORBA.bat	汎用コネクター (Windows) の始動スクリプト。
¥connectors¥CORBA¥start_CORBA.sh	汎用コネクター (UNIX) の始動スクリプト。
¥connectors¥CORBA¥ext¥	ODA が生成した .jar ファイルが保管できるディレクトリー。このディレクトリーに保管する場合は、始動スクリプト (start_CORBA.bat または start_CORBA.sh) にこのディレクトリーを指定します。
¥connectors¥CORBA¥NamingService.bat	IBM Transient Naming Server 始動ファイル (Windows)。
¥connectors¥CORBA¥NamingService.sh	IBM Transient Naming Server 始動ファイル (Unix)。
¥connectors¥messages¥BIA_CORBAConnector.txt	コネクターのメッセージ・ファイル。
¥connectors¥CORBA¥BIA_CORBAEnv.bat	JDK ホーム・ディレクトリーのパス名を追跡するスクリプト・ファイル。
¥connectors¥CORBA¥BIA_CORBAEnv.sh	JDK ホーム・ディレクトリーのパス名を追跡するスクリプト・ファイル (Unix)。
¥ODA¥CORBA¥BIA_CORBAODA.jar	CORBA ODA。
¥ODA¥CORBA¥start_CORBAODA.bat	ODA 始動ファイル (Windows)。
¥ODA¥CORBA¥BIA_CORBAODA.sh	ODA 始動ファイル (UNIX)。
¥ODA¥messages¥BIA_CORBAODAAgent_de_DE.txt	ODA のメッセージ・ファイル (ドイツ語テキスト・ストリング)。
¥ODA¥messages¥BIA_CORBAODAAgent_en_US.txt	ODA のメッセージ・ファイル (米国英語テキスト・ストリング)。
¥ODA¥messages¥BIA_CORBAODAAgent_es_ES.txt	ODA のメッセージ・ファイル (スペイン語テキスト・ストリング)。
¥ODA¥messages¥BIA_CORBAODAAgent_fr_FR.txt	ODA のメッセージ・ファイル (フランス語テキスト・ストリング)。

表 1. コネクターのファイル構造 (続き)

***ProductDir* のサブディレクトリー**	**説明**
¥ODA¥messages¥BIA_CORBAODAAgent_it_IT.txt	ODA のメッセージ・ファイル (イタリア語テキスト・ストリング)。
¥ODA¥messages¥BIA_CORBAODAAgent_ja_JP.txt	ODA のメッセージ・ファイル (日本語テキスト・ストリング)。
¥ODA¥messages¥BIA_CORBAODAAgent_ko_KR.txt	ODA のメッセージ・ファイル (韓国語テキスト・ストリング)。
¥ODA¥messages¥BIA_CORBAODAAgent_pt_BR.txt	ODA のメッセージ・ファイル (ポルトガル語 (ブラジル) テキスト・ストリング)。
¥ODA¥messages¥BIA_CORBAODAAgent_zh_CN.txt	ODA のメッセージ・ファイル (中国語 (簡体字) テキスト・ストリング)。
¥ODA¥messages¥BIA_CORBAODAAgent_zh_TW.txt	ODA のメッセージ・ファイル（中国語 (繁体字) テキスト・ストリング)。
¥repository¥CORBA¥BIA_CN_CORBA.txt	コネクターのリポジトリー定義。デフォルトの名前は BIA_CN_CORBA.txt です。

注: すべての製品のパス名は、使用システムで製品がインストールされたディレクトリーを基準とした相対パス名です。

インストール後の作業

インストールが完了したら、始動する前に、アダプターを構成する必要があります。詳細については、23 ページの『第 3 章 アダプターの構成』を参照してください。

第 3 章 アダプターの構成

構成タスクの概要

インストールが完了したら、始動する前に、このセクションで説明するコンポーネントを構成する必要があります。

コネクターの構成

コネクターの構成とは、コネクターをセットアップして構成することです。詳細については、『コネクターの構成』を参照してください。

ビジネス・オブジェクトの構成

Business Object Designer を使用してビジネス・オブジェクトを作成および構成します。Business Object Designer で ODA (Object Discovery Agent) を使用してビジネス・オブジェクトを作成できます。 ODA を使用すると、ビジネス・オブジェクト定義を生成できます。ビジネス・オブジェクト定義とは、ビジネス・オブジェクトのテンプレートです。ODA は、指定したアプリケーション・オブジェクトの検証、ビジネス・オブジェクト属性に対応するビジネス・オブジェクトのエレメントの「発見」、および情報を示すビジネス・オブジェクト定義の生成を実行します。Business Object Designer を使用すると、グラフィカル・インターフェースを使用して Object Discovery Agent にアクセスし、対話式に操作できます。既存のビジネス・オブジェクトを編集するには Business Object Designer を使用します。

ビジネス・オブジェクトの詳細については、39 ページの『第 4 章 ビジネス・オブジェクトについて』を参照してください。

ODA の使用に関する詳細については、57 ページの『第 5 章 ビジネス・オブジェクトの作成および変更』を参照してください。

コネクターの構成

コネクターの構成プロパティーには、標準構成プロパティーとアダプター固有の構成プロパティーという 2 つのタイプがあります。アダプターを実行する前に、Connector Configurator を使用して、これらのプロパティーの値を設定する必要があります。詳細については、97 ページの『付録 B. Connector Configurator』を参照してください。

コネクターは、始動時に構成値を取得します。実行時セッション中に、1 つ以上のコネクター・プロパティーの値の変更が必要になることがあります。AgentTraceLevel など一部のコネクター構成プロパティーへの変更は、即時に有効になります。他のコネクター・プロパティーを変更する場合は、変更後に、コネクター・コンポーネントの再始動またはシステムの再始動が必要です。プロパティーが動的 (変更が即時に有効化) であるか、または静的 (コネクター・コンポーネントの再始動またはシステムの再始動が必要) であるかを判別するには、System Manager の「コネクターのプロパティー」ウィンドウにある「更新メソッド」列を参照します。

標準コネクター・プロパティー

標準コネクター構成プロパティーでは、すべてのアダプターが使用する情報を提供します。これらのプロパティーの詳細については、77 ページの『付録 A. コネクターの標準構成プロパティー』を参照してください。

以下のプロパティーは 77 ページの『付録 A. コネクターの標準構成プロパティー』にリストされていますが、Connector for CORBA ではこれらを使用していません。

- DuplicateEvent Elimination
- PollEndTime
- PollFrequency
- PollStartTime

コネクターを稼働させる前に、`ApplicationName` 構成プロパティーの値を設定する必要があります。

コネクター固有のプロパティー

コネクター固有の構成プロパティーは、コネクターが実行時に必要とする情報を提供します。これらのプロパティーを使用すれば、コネクター内の静的情報やロジックを、再コーティングや再ビルドせずに変更できるようになります。

コネクター固有のプロパティーを構成するには、Connector Configurator を使用します。**「アプリケーション構成プロパティー」**タブをクリックして、構成プロパティーを追加または変更します。詳細については、97 ページの『付録 B. Connector Configurator』を参照してください。

特に指定のないかぎり、すべてのコネクター固有のプロパティーはオプションです。つまり、これらのプロパティーは、ユーザー固有のコネクター構成要件に基づいて選択して設定できます。例えば、コネクターがクライアントとして機能している場合に、コネクターにファクトリー・オブジェクトと接続の両方を作成させるか、ファクトリー・オブジェクトのみ、または接続のみを作成させるか選択します。選択した構成によって、設定する必要があるプロパティーが決定します。

25 ページの表 2 に、コネクターのコネクター固有構成プロパティーと、その説明および指定可能な値を示します。**+** 文字は、プロパティー階層内でのエントリーの位置を示します。プロパティーの詳細については、これらのプロパティーの階層を示す26 ページの図 5 を含む後続のセクションを参照してください。

表 2. コネクター固有の構成プロパティー

名前	指定可能な値	デフォルト値
+ Factory	*なし。これは、階層内のカテゴリーです。*	なし
+ + FactoryClass	*クラス名*	なし
+ + FactoryMethod	*メソッド名*	なし
+ + + Arguments	*暗号化ストリングまたは非暗号化ストリング*	
+ + FactoryInitializer	*初期化指定子のメソッド名*	なし
+ + + ior_file_name	*IOR ファイルのディレクトリーとファイル名*	なし
+ + + name	*Factory の取得に使用されるオブジェクトの名前。例:* `HelloServerServerObject`	なし
+ ConnectionPool	*なし。これは、階層内のカテゴリーです。*	なし
+ + ConnectionClass	*クラス名*	なし
+ + ConnectionInitializer	*初期化指定子のメソッド名*	なし
+ + + Arguments	*暗号化ストリングまたは非暗号化ストリング*	なし
+ + PoolSize	*整数*	0
+ UseNamingContext	`True`、`False`	なし
+ ClientOnly	`True`、`False`	なし
+ BO_COLLAB_MAPPING	*コラボレーションにマップされたビジネス・オブジェクトと動詞。構造は次のようになります。* *<businessObject.verb>* マップ先: *<collaborationName>*	なし
+ CORBAServerName	*有効な CORBA サーバー名。例:* `CORBAAdapter`	なし

+ BO_CONNECTION_PROP	*ior_file_name、または name にマップされたビジネス・オブジェクトの名前。例:*
	
	`Customer` マップ先:
	`ior_file_name=c:¥psrserver.ior;`
	`name=;`
		なし

+ ORBInitialHost	*CORBA ネーム・サーバーのホスト名。IBM ORB Transient Naming Server とします。*	なし
+ ORBInitialPort	*CORBA ネーム・サーバーのポート番号。IBM ORB Transient Naming Server とします。*	なし

26 ページの図 5 に、コネクター固有プロパティーの階層関係を示します。

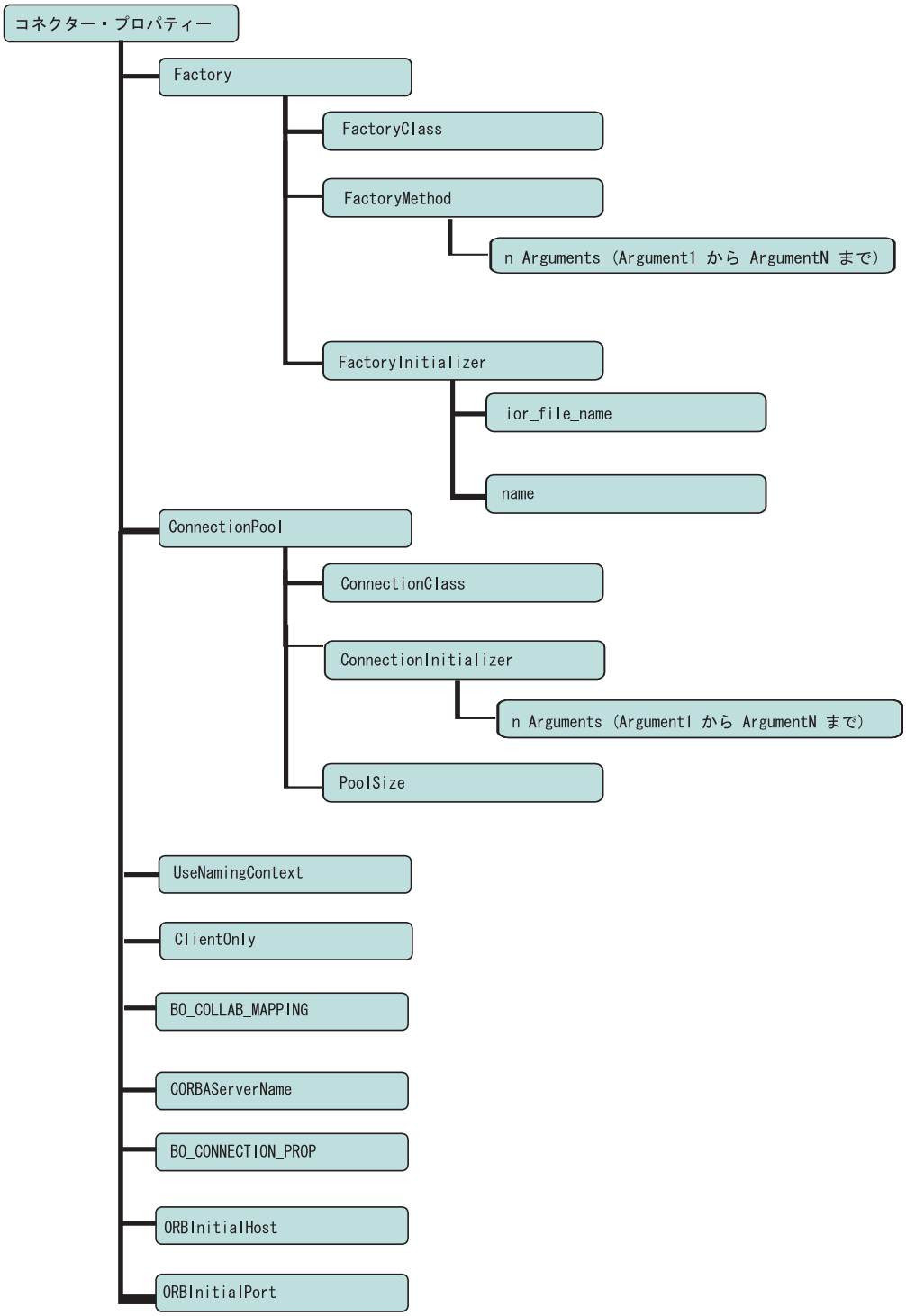


図 5. コネクター固有プロパティーの階層

Factory

Factory クラス情報のカテゴリーを表す階層プロパティー。

FactoryClass

Factory クラスの名前。

- FactoryClass および ConnectionClass を指定すると、コネクターは、ファクトリー・プロキシー・オブジェクトおよび接続のインスタンスを生成します (ステップ 1 (10 ページ) の**シナリオ 1** を参照してください)。
- FactoryClass を指定せずに、ConnectionClass を指定すると、指定された接続クラスおよびサイズを持つ接続プールが、コネクターの初期化時に作成されます (ステップ 1 (10 ページ) の**シナリオ 2** を参照してください)。
- FactoryClass のみを指定すると、コネクターは、ファクトリー・プロキシー・オブジェクトのインスタンスを生成しますが (ステップ 1 (10 ページ) の**シナリオ 3** を参照してください)、接続は使用しません。

このプロパティーを使用できるのは、コネクターがクライアントとして機能する場合です。

FactoryMethod

FactoryClass の FactoryMethod 名を表す階層プロパティー。FactoryMethod を指定すると、Factory メソッドから接続オブジェクトが呼び出されて作成され、作成された接続オブジェクトに対して、ConnectionInitializer が呼び出されます。

このプロパティーを使用できるのは、コネクターがクライアントとして機能する場合です。

Arguments

FactoryMethod のパラメーターは、FactoryClass に対する引き数 (Argument1、Argument2 など) である必要があります。また、適切な順序でリストされなければなりません。プロパティー名には、メソッドが実行するパラメーターの数に応じて、Argument1、Argument2 などの名前を指定します。各引き数の値は、暗号化ストリングまたは非暗号化ストリングになります。

このプロパティーを使用できるのは、コネクターがクライアントとして機能する場合です。

FactoryInitializer

Factory クラスの初期化方法を表す階層プロパティー。

このプロパティーを使用できるのは、コネクターがクライアントとして機能する場合です。

ior_file_name

Factory オブジェクトの ior ファイルの名前。コネクターはこのプロパティーを使用して、Factory オブジェクトをインスタンス化します。このプロパティー、name、または両方を指定できます。

このプロパティーを使用できるのは、コネクターがクライアントとして機能する場合です。

name

Factory の取得に使用されるオブジェクトの名前。このプロパティーは Factory オブジェクトをインスタンス化するために使用されます。ior_file_name に値を指定していない場合、このプロパティーは必須です。また、アダプターがネーミング・コンテキストを使用する場合 (useNamingContext プロパティーが true に設定されている場合) にも、このプロパティーは必須です。

このプロパティーを使用できるのは、コネクターがクライアントとして機能する場合です。

ConnectionPool

Connection クラス情報のカテゴリーを表す階層プロパティー。

ConnectionClass

プール可能な接続クラスの名前。

- ConnectionClass および FactoryClass を指定すると、コネクターは、ファクトリー・プロキシー・オブジェクトおよび接続のインスタンスを生成します (ステップ 1 (10 ページ) の**シナリオ 1** を参照してください)。
- FactoryClass は指定せずに ConnectionClass を指定すると、コネクターの初期化時に、接続プール・インスタンスが作成され、接続が保管されます (ステップ 1 (10 ページ) の**シナリオ 2** を参照してください)。

プール・サイズ (接続数) は、PoolSize プロパティーで指定した値に基づきます。

マルチ使用サーバーで接続をプールする場合は (サーバー・オブジェクトのインスタンスの 1 つが接続を確立するために再利用される場合あり)、ファクトリーおよびファクトリー・メソッド呼び出しをセットアップして、接続プールを作成する必要があります。この場合、各 BO ハンドラー・スレッドは、処理時に必要となる離散的接続を接続プールからプルします。

このプロパティーを使用できるのは、コネクターがクライアントとして機能する場合です。

ConnectionInitializer

プール可能な ConnectionClass 初期化指定子メソッドの名前。

FactoryMethod メソッドを指定すると、FactoryMethod から接続オブジェクトが呼び出されて作成され、作成された接続オブジェクトに対して、ConnectionInitializer が呼び出されます。

このプロパティーを使用できるのは、コネクターがクライアントとして機能する場合です。

Arguments

ConnectionInitializer のパラメーターは ConnectionClass に対する引き数 (Argument1、Argument2 など) である必要があります。また、適切な順序でリストされなければなりません。プロパティー名には、初期化指定子が実行するパラメーターの数に応じて、Argument1、Argument2 などの名前を指定します。各引き数の値は、暗号化ストリングまたは非暗号化ストリングになります。

このプロパティーを使用できるのは、コネクターがクライアントとして機能する場合です。

PoolSize

接続プールのサイズを決定します。デフォルト値は 0 です。

ConnectionClass が指定されている場合、このプロパティーは必須です。このプロパティーを使用できるのは、コネクターがクライアントとして機能する場合です。

UseNamingContext

true/false ブール・フラグ。コネクターが CORBA ネーミング・サービスを使用して、name に対して指定された値を持つオブジェクトを検索するかどうかを示します。ネーミング・サービスの詳細については、5 ページの『IBM ORB Transient Naming Server』を参照してください。

このプロパティーは、コネクターがクライアントおよびサーバーのいずれとして機能する場合でも必須です。

ClientOnly

コネクターが CORBA クライアントとして機能するかどうかを示すブール・フラグ true/false。フラグが false に設定されると、コネクターは CORBA サーバーおよびクライアントの両方として機能します。

これは、必須のプロパティーです。

BO_COLLAB_MAPPING

コネクターが CORBA サーバーとして機能する (ClientOnly プロパティーが false に設定される) 場合、この階層プロパティーを使用してビジネス・オブジェクトおよびその動詞 (*businessObject.verb*) がコラボレーションにマップされます。外部 CORBA クライアントがサーバー・オブジェクトに対してメソッド呼び出しを行うときに、コラボレーションが実行されます。メソッド呼び出しを受け取ると、コネクターはメソッドを適切な動詞 (オブジェクトの属性 ASI に指定されている) に変換します。次に、BO_COLLAB_MAPPING プロパティーに指定されている動詞とコラボレーションのマッピングを使用して、サーバー・オブジェクトが executeCollaboration() メソッドを呼び出し、それを受けてその動詞に対応するコラボレーションが実行されます。

このプロパティーが必要なのは、コネクターが CORBA サーバーにマップするビジネス・オブジェクトを処理する場合のみです。このプロパティーを使用できるのは、コネクターがサーバーとして機能する場合のみです。

CORBAServerName

コネクターを CORBA サーバーとして登録するときに使用される名前。

このプロパティーが必要なのは、コネクターが CORBA サーバーにマップするビジネス・オブジェクトを処理する場合のみです。このプロパティーを使用できるのは、コネクターがサーバーとして機能する場合のみです。

BO_CONNECTION_PROP

ビジネス・オブジェクトを接続情報にマップするのに必要な情報。このプロパティーを設定するのは、コネクターがクライアントとして機能する場合です。情報には、サポートされるクライアント・ビジネス・オブジェクトの名前、ior_file_name、および name が含まれます。例えば、コネクターが 2 つのビジネス・オブジェクト CORBACustomer および CORBAAccount をサポートする場合、BO_CONNECTION_PROP 設定は以下のようになります。

```
CORBACustomer=ior_file_name=<fileName>;name=<name>
CORBAAccount=ior_file_name=<fileName>;name=<name>
```

コネクターは、プロパティー設定に指定された値に応じて、以下のいずれかの方法でこのプロパティーを使用します。

- ior_file_name に値が設定されている場合、コネクターは、その値を使用してビジネス・オブジェクトの検索を行います。
- name に値が指定され、UseNamingContext プロパティーが true に設定されている場合、コネクターはネーミング・コンテキストを使用して、指定された name 値の CORBA オブジェクトを検索します。

これらのプロパティー設定シナリオがいずれも true でない場合 (つまり、ior_file_name および name がブランクの場合)、コネクターはログ・ファイルにエラーを記録し、ビジネス・オブジェクトの必要な接続情報が欠落していることを示します。また、BO ハンドラーは所定のビジネス・オブジェクト用に初期化されません。エラー・ロギングの詳細については、71 ページの『第 6 章 トラブルシューティングおよびエラー処理』を参照してください。

このプロパティーが必要なのは、コネクターが CORBA クライアントとして機能するビジネス・オブジェクトを処理する場合のみです。

ORBInitialHost

IBM ORB Transient Name Server のホスト名。クライアントとして実行されているかまたはサーバーとして実行されているかに関係なく、コネクターは、IBM ORB Transient Naming Server のネーミング・サービスを使用する必要があります。ネーミング・サーバーの詳細については、5 ページの『IBM ORB Transient Naming Server』を参照してください。

このプロパティーは必須です。コネクターが CORBA サーバーに接続する場合、またはコネクター自体がサーバーとして機能する場合に、コネクターによって使用されます。ネーム・サーバーが、コネクターと同じホスト上で動作している場合にも、このプロパティーを指定する必要があります。

ORBInitialPort

IBM ORB Transient Name Server のポート番号。クライアントとして実行されているかまたはサーバーとして実行されているかに関係なく、コネクターは、IBM ORB Transient Naming Server のネーミング・サービスを使用する必要があります。ネーミング・サーバーの詳細については、5 ページの『IBM ORB Transient Naming Server』を参照してください。

このプロパティーは必須です。コネクターが CORBA サーバーに接続する場合、またはコネクター自体がサーバーとして機能する場合に、コネクターによって使用されます。ネーム・サーバーが、デフォルト・ポート (900) 上で動作している場合にも、このプロパティーを指定する必要があります。

構成シナリオのサンプル

このセクションでは、次の構成シナリオの例を示します。

- 『FactoryClass および ConnectionClass を使用するクライアントとして実行するコネクターの構成』
- 32 ページの『FactoryClass および ConnectionClass を使用せずに単純なクライアントとして実行するコネクターの構成』
- 33 ページの『サーバーとして実行するコネクターの構成』

FactoryClass および ConnectionClass を使用するクライアントとして実行するコネクターの構成

外部の CORBA サーバーに対する呼び出しを行うクライアントとしてコネクターを実行する場合は、サーバーが使用するサービス (Transient Name Server (TNS) または相互運用オブジェクト参照 (IOR)) をあらかじめ決定し、これに合わせてコネクターのプロパティーを設定します。TNS の場合、アダプター呼び出しはネットワークを経由して行われますが、IOR の場合、呼び出しはファイル・ベースであり、サーバーを見つけるために必要なネットワーク情報はアダプターが読み込む IOR ファイルに保管されます。

以下に示すサンプルの IDL ファイル (`HelloFactory.idl`) は、TNS を使用するサーバーに対して、`FactoryClass` および `ConnectionClass` を使用するクライアントとしてコネクターを実行する際に、コネクターによって必要とされるビジネス・オブジェクトを生成するために使用します。`FactoryClass` および `ConnectionClass` を *使用しない* 単純なクライアントとして実行するコネクターの例については、32 ページの『FactoryClass および ConnectionClass を使用せずに単純なクライアントとして実行するコネクターの構成』を参照してください。

```
module HelloAppFactory
{
  interface HelloApp
  {
    string sayHello();
  };

  interface HelloFactory
  {
    HelloApp getHello();
    HelloApp getHelloUsingName(in string name);
  };
};
```

サンプル・コードには、次のエレメントが含まれています。

- `interface HelloApp` は ConnectionClass を参照します。
- `interface HelloFactory` は FactoryClass を参照します。
- `HelloApp getHello()` は `HelloApp` ConnectionClass のインスタンスを戻す FactoryMethod を参照します。

表 3 は、HelloFactory.idl シナリオの一部のコネクター固有プロパティーの構成設定例を示しています。コネクター固有プロパティーの詳細なリストについては、 25 ページの表 2 を参照してください。

表 3. HelloFactory.idl シナリオのコネクター固有プロパティー (サーバー構成)

コネクター固有プロパティー	値
UseNamingContext	true
ClientOnly	true
FactoryClass	HelloAppFactory.HelloFactory
FactoryInitializer > name	HelloApp_HelloFactoryServerNaming
FactoryMethod	getHello
ConnectionClass	HelloAppFactory.Hello
PoolSize	10
BO_CONNECTION_PROP	Client_HelloAppFactory_HelloFactory
Client_HelloAppFactory_HelloFactory	name=HelloApp_HelloFactoryServerNaming

コネクターがクライアントとして実行している場合、TNS ではなく IOR サービスを使用する CORBA サーバーへの呼び出しを行うときは、次の要件があることに注意してください。

- BO_CONNECTION_PROP > Client_HelloAppFactory_HelloFactory プロパティーで IOR ファイル名 (ior_file_name= ;) を指定する。
  - Client_HelloAppFactory_HelloFactory は CORBA クライアントをサーバーに接続するために使用するクライアント・ビジネス・オブジェクトの名前です。
  - IOR ファイル名 (ior_file_name= ;) は、ビジネス・オブジェクトの検索に使用します。
- UseNamingContext プロパティーを false に設定する。

FactoryClass および ConnectionClass を使用せずに単純なクライアントとして実行するコネクターの構成

以下に示す module corbaadapter_sample の IDL ファイルのサンプルは、TNS を使用するが FactoryClass および ConnectionClass を*使用しない* で、クライアントとしてコネクターを実行する際に、コネクターによって必要とされるビジネス・オブジェクトの生成に使用するインターフェース (Hello) を定義します。
FactoryClass および ConnectionClass を使用するクライアントとして実行するコネクターの例については、 31 ページの『FactoryClass および ConnectionClass を使用するクライアントとして実行するコネクターの構成』を参照してください。

```
//code for simple client and server config
module corbaadapter_sample
{

    typedef sequence<long> LongSeq;
    typedef sequence<string> StringSeq;

    interface Hello
    {

        //simple type tests
        string simpleIn(in LongSeq in_long_val, in double in_amount,
        in boolean in_istrue ,in string in_firstNm)
        raises (ProcessingFailureException);
        StringSeq simpleOut(out LongSeq out_long_val,
```

```
        out double out_amount,out boolean out_istrue ,
        out string out_firstNm) raises (ProcessingFailureException);
    };
};
```

表 4 は、interface Hello を定義する module corbaadapter_sample を使用する場合に構成するコネクター固有プロパティーの構成設定例を示しています。コネクター固有プロパティーの詳細なリストについては、25 ページの表 2 を参照してください。

表 4. interface Hello シナリオのコネクター固有プロパティー (FactoryClass および ConnectionClass を使用しない単純なクライアント)

コネクター固有プロパティー	値
UseNamingContext	true
ClientOnly	true

このシナリオでは、ClientOnly を true に設定し、FactoryClass、FactoryInitializer、FactoryMethod、ConnectionClass、および ConnectionPool プロパティーは使用していません。これらのプロパティーは、FactoryClass および ConnectionClass を使用するクライアントとしてコネクターを構成する場合にのみ関連します。

サーバーとして実行するコネクターの構成

コネクターをサーバーとして実行するように構成すると、コネクターは外部の CORBA クライアントから要求を受信できます。これらの要求は、統合ブローカー上のコラボレーションを呼び出します。

次の module corbaadapter_sample の IDL ファイルのサンプルは、コネクターが34 ページの表 5 に示すプロパティー設定で構成されている場合に CORBA クライアントに公開されるインターフェース (interface Hello) を定義しています。コネクターをサーバーとして実行する場合、コネクターは、この IDL ファイルから生成されたビジネス・オブジェクトを使用して CORBA クライアントからの要求に対してサービスを提供します。

```
//code for simple client and server config
module corbaadapter_sample
{

    typedef sequence<long> LongSeq;
    typedef sequence<string> StringSeq;

    interface Hello
    {

        //simple type tests
        string simpleIn(in LongSeq in_long_val, n double in_amount,
        in boolean in_istrue ,in string in_firstNm) raises
        (ProcessingFailureException);
        StringSeq simpleOut(out LongSeq out_long_val,
        out double out_amount,out boolean out_istrue ,
        out string out_firstNm) raises (ProcessingFailureException);
    };
};
```

34 ページの表 5 は、interface Hello を定義する module corbaadapter_sample を使用する場合に設定するコネクター固有プロパティーの構成設定例を示しています。コネクター固有プロパティーの詳細なリストについては、25 ページの表 2 を参照してください。

表 5. interface Hello シナリオのコネクター固有プロパティー (サーバー構成)

コネクター固有プロパティー	値
UseNamingContext	true
ClientOnly	false
BO_COLLAB_MAPPING	Server_corbaadapter_sample_Hello.Create

このシナリオでは、FactoryClass、FactoryInitializer、FactoryMethod、ConnectionClass、および ConnectionPool プロパティーは使用していません。その理由は、これらのプロパティーは FactoryClass および ConnectionClass を使用するクライアントとしてコネクターを構成する場合にのみ関連するためです。

BO_COLLAB_MAPPING プロパティーは、サーバーとして動作するコネクターがコラボレーションにマップする (*businessObjectName.verb* としてフォーマットされた) サーバー・オブジェクトを表します。コラボレーションは、外部の CORBA クライアントが (module corbaadapter_sample に定義されているインターフェースを作成する) このサーバー・オブジェクトのインスタンスに対してメソッド呼び出しを行ったときに実行されます。

複数のコネクター・インスタンスの作成

コネクターの複数のインスタンスを作成する作業は、いろいろな意味で、カスタム・コネクターの作成と同じです。以下に示すステップを実行することによって、コネクターの複数のインスタンスを作成して実行するように、ご使用のシステムを設定することができます。次のようにする必要があります。

- コネクター・インスタンス用に新規ディレクトリーを作成します。
- 必要なビジネス・オブジェクト定義が設定されていることを確認します。
- 新規コネクター定義ファイルを作成します。
- 新規始動スクリプトを作成します。

新規ディレクトリーの作成

それぞれのコネクター・インスタンスごとにコネクター・ディレクトリーを作成する必要があります。このコネクター・ディレクトリーには、次の名前を付けなければなりません。

```
ProductDir¥connectors¥connectorInstance
```

ここで connectorInstance は、コネクター・インスタンスを一意的に示します。

コネクターに、コネクター固有のメタオブジェクトがある場合、コネクター・インスタンス用のメタオブジェクトを作成する必要があります。メタオブジェクトをファイルとして保管する場合は、次のディレクトリーを作成して、ファイルをそこに格納します。

```
ProductDir¥repository¥connectorInstance
```

ビジネス・オブジェクト定義の作成

各コネクター・インスタンスのビジネス・オブジェクト定義がプロジェクト内にまだ存在しない場合は、それらを作成する必要があります。

1. 初期コネクターに関連付けられているビジネス・オブジェクト定義を変更する必要がある場合は、適切なファイルをコピーし、Business Object Designer を使用してそれらのファイルをインポートします。初期コネクターの任意のファイルをコピーできます。変更を加えた場合は、名前を変更してください。
2. 初期コネクターのファイルは、次のディレクトリーに入っていなければなりません。

   ```
   ProductDir¥repository¥initialConnectorInstance
   ```

   作成した追加ファイルは、ProductDir¥repository の適切な connectorInstance サブディレクトリー内に存在している必要があります。

コネクター定義の作成

Connector Configurator 内で、コネクター・インスタンスの構成ファイル (コネクター定義) を作成します。これを行うには、以下のステップを実行します。

1. 初期コネクターの構成ファイル (コネクター定義) をコピーし、名前変更します。
2. 各コネクター・インスタンスが、サポートされるビジネス・オブジェクト (および関連メタオブジェクト) を正しくリストしていることを確認します。
3. 必要に応じて、コネクター・プロパティーをカスタマイズします。

始動スクリプトの作成

始動スクリプトは以下のように作成します。

1. 初期コネクターの始動スクリプトをコピーし、コネクター・ディレクトリーの名前を含む名前を付けます。

   dirname
2. この始動スクリプトを、34 ページの『新規ディレクトリーの作成』で作成したコネクター・ディレクトリーに格納します。
3. 始動スクリプトのショートカットを作成します (Windows のみ)。
4. 初期コネクターのショートカット・テキストをコピーし、新規コネクター・インスタンスの名前に一致するように (コマンド行で) 初期コネクターの名前を変更します。

これで、ご使用の統合サーバー上でコネクターの両方のインスタンスを同時に実行することができます。

カスタム・コネクター作成の詳細については、「*コネクター開発ガイド (C++ 用)*」または「*コネクター開発ガイド (Java 用)*」を参照してください。

コネクターの始動

コネクターは、**コネクター始動スクリプト**を使用して明示的に始動する必要があります。始動スクリプトは、次に示すようなコネクターのランタイム・ディレクトリーに存在していなければなりません。

ProductDir¥connectors¥*connName*

ここで、*connName* はコネクターを示します。始動スクリプトの名前は、表 6 に示すように、オペレーティング・システム・プラットフォームによって異なります。

表 6. コネクターの始動スクリプト

オペレーティング・システム	起動スクリプト
UNIX ベースのシステム	`connector_manager_connName`
Windows	`start_connName.bat`

コネクター始動スクリプトは、以下に示すいずれかの方法で起動することができます。

- Windows システムで**「スタート」**メニューから。

  「プログラム」>「IBM WebSphere Business Integration Adapters」>「アダプター」>「コネクター」 を選択します。デフォルトでは、プログラム名は「IBM WebSphere Business Integration Adapters」となっています。ただし、これはカスタマイズすることができます。あるいは、ご使用のコネクターへのデスクトップ・ショートカットを作成することもできます。
- コマンド行から。
  - Windows システム:

    `start_connName connName brokerName [-cconfigFile ]`
  - UNIX ベースのシステム:

    `connector_manager_connName -start`

  ここで、*connName* はコネクターの名前であり、*brokerName* は以下のようにご使用の統合ブローカーを表します。
  - WebSphere InterChange Server の場合は、*brokerName* に ICS インスタンスの名前を指定します。
  - WebSphere Message Brokers (WebSphere MQ Integrator、WebSphere MQ Integrator Broker、または WebSphere Business Integration Message Broker) または WebSphere Application Server の場合は、*brokerName* にブローカーを示すストリングを指定します。

  注: Windows システム上の WebSphere Message Broker または WebSphere Application Server の場合は、`-c` オプションに続いてコネクター構成ファイルの名前を指定しなければなりません。ICS の場合は、`-c` はオプションです。
- Adapter Monitor から (WebSphere Business Integration Adapters 製品のみ)。Adapter Monitor は System Manager 始動時に起動されます。

  このツールを使用して、コネクターのロード、アクティブ化、非アクティブ化、休止、シャットダウン、または削除を行うことができます。
- System Monitor から (WebSphere InterChange Server 製品のみ)。

  このツールを使用して、コネクターのロード、アクティブ化、非アクティブ化、休止、シャットダウン、または削除を行うことができます。
- Windows システムでは、Windows サービスとして始動するようにコネクターを構成することができます。この場合、Windows システムがブートしたとき (自動サービスの場合)、または Windows サービス・ウィンドウを通じてサービスを始動したとき (手動サービスの場合) に、コネクターが始動します。

コマンド行の始動オプションなどのコネクターの始動方法の詳細については、以下の資料のいずれかを参照してください。

- WebSphere InterChange Server については、「*システム管理ガイド*」を参照してください。
- WebSphere Message Brokers については、「*WebSphere Message Brokers 使用アダプター・インプリメンテーション・ガイド*」を参照してください。
- WebSphere Application Server については、「*アダプター実装ガイド (WebSphere Application Server)*」を参照してください。

コネクターの停止

コネクターを停止する方法は、以下に示すように、コネクターが始動された方法によって異なります。

- コマンド行からコネクターを始動した場合は、コネクター始動スクリプトを用いて、以下の操作を実行します。
  - Windows システムでは、始動スクリプトを起動すると、そのコネクター用の別個の「コンソール」ウィンドウが作成されます。このウィンドウで、「Q」と入力して Enter キーを押すと、コネクターが停止します。
  - UNIX ベースのシステムでは、コネクターはバックグラウンドで実行されるため、別ウィンドウはありません。代わりに、次のコマンドを実行してコネクターを停止します。

    ```
    connector_manager_connName -stop
    ```

    ここで、*connName* はコネクターの名前です。
- Adapter Monitor から (WebSphere Business Integration Adapters 製品のみ)。Adapter Monitor は System Manager 始動時に起動されます。

  このツールを使用して、コネクターのロード、アクティブ化、非アクティブ化、休止、シャットダウン、または削除を行うことができます。
- System Monitor から (WebSphere InterChange Server 製品のみ)

  このツールを使用して、コネクターのロード、アクティブ化、非アクティブ化、休止、シャットダウン、または削除を行うことができます。
- Windows システムでは、Windows サービスとして始動するようにコネクターを構成することができます。この場合、Windows システムのシャットダウン時に、コネクターは停止します。

ログ・ファイルとトレース・ファイルの使用

アダプターのコンポーネントは、複数レベルのメッセージ・ロギングとトレースを提供します。コネクターはエラー・メッセージ、通知メッセージ、およびトレース・メッセージを記録するために、アダプター・フレームワークを使用します。エラー・メッセージと通知メッセージがログ・ファイルに記録され、トレース・メッセージと対応するトレース・レベル (0 から 5) がトレース・ファイルに記録されます。ロギングおよびトレース・レベルの詳細については、71 ページの『第 6 章 トラブルシューティングおよびエラー処理』を参照してください。

ログ・ファイルとトレース・ファイルの名前、およびトレース・レベルをConnector Configurator で構成してください。このツールの詳細については、 97 ページの『付録 B. Connector Configurator』を参照してください。

ODA にはロギング機能がない点に注意してください。エラー・メッセージは直接ユーザー・インターフェースに送られます。トレース・ファイルとトレース・レベルは Business Object Designer 内で構成します。構成の手順は 59 ページの『エージェントの構成』に記載されています。ODA トレース・レベルはコネクター・トレース・レベルと同一であり、 75 ページの『トレース』 に定義されています。

第 4 章 ビジネス・オブジェクトについて

本章では、ビジネス・オブジェクトの構造、アダプターによるビジネス・オブジェクトの処理方法、およびアダプターで想定される前提事項について説明します。

本章の内容は、次のとおりです。

メタデータの定義

Connector for CORBA は、メタデータ主導型です。WebSphere Business Integration システムにおいて、メタデータは、CORBA アプリケーション・オブジェクトのデータ構造を記述するアプリケーション固有の情報として定義されています。メタデータは、ビジネス・オブジェクト定義を構成するために使用します。コネクターは、実行時にこの定義を使用して、ビジネス・オブジェクトを作成します。

コネクターのインストール後、コネクターを実行する前に、ビジネス・オブジェクト定義を作成する必要があります。コネクターが処理するビジネス・オブジェクトには、統合ブローカーで許可された任意の名前を付けることができます。命名規則の詳細については、「*コンポーネント命名ガイド*」を参照してください。

メタデータ主導型コネクターは、サポートしている各ビジネス・オブジェクトを、ビジネス・オブジェクト定義でエンコードされたメタデータに従って処理します。これにより、コードに変更を加えなくても、コネクターは新規または変更されたビジネス・オブジェクト定義を処理できるようになります。Business Object Designer では、オブジェクトを新規に作成するときに ODA を使用せずにオブジェクトを作成することができます。既存のオブジェクトを変更するには、Business Object Designer を直接使用します (ODA は既存のビジネス・オブジェクトの変更には使用できません)。

アプリケーション固有のメタデータには、ビジネス・オブジェクトの構造と、属性プロパティーの設定が記述されています。各ビジネス・オブジェクトの実際のデータ値は、実行時にメッセージ・オブジェクトによって伝達されます。

コネクターは、サポートするビジネス・オブジェクトの構造、親ビジネス・オブジェクトと子ビジネス・オブジェクト間の関係、およびデータの形式を推測します。そのため、ビジネス・オブジェクトの構造に、対応する CORBA オブジェクトに対して定義した構造を正確に反映させる必要があります。構造が一致していないと、アダプターはビジネス・オブジェクトを正しく処理できません。

ビジネス・オブジェクト構造を変更する必要がある場合は、変更する構造を CORBA 内の対応するオブジェクトにしてから、その変更をファイル・システム・リポジトリーにエクスポートして、ODA に入力できるようにします。

ビジネス・オブジェクト定義の変更の詳細については、「*WebSphere Business Integration Adapters ビジネス・オブジェクト開発ガイド*」を参照してください。

コネクター・ビジネス・オブジェクトの構造

コネクターは、次の 2 種類のビジネス・オブジェクトを処理します。どちらも ODA によって生成されます。

- コネクターがクライアントとして機能する場合に、CORBA サーバー・サイドのコンポーネントによって使用されるビジネス・オブジェクト
- コネクターがサーバーとして機能する場合に、CORBA クライアント・サイドのコンポーネントによって使用されるビジネス・オブジェクト

IDL コンポーネントが、クライアント・サイドとサーバー・サイドの両方のオブジェクトとして意図されている場合、クライアント・サイド処理とサーバー・サイド処理のためにそれぞれ 1 つずつ、2 つのビジネス・オブジェクトが ODA により生成されます。この場合、1 つの IDL コンポーネントに対して ODA を 2 回実行します。1 回目は、ODA `CORBAServerImpl` プロパティーを `true` に設定した状態で実行し、サーバー・サイドのビジネス・オブジェクトを生成します。2 回目は、このプロパティーを `false` に設定した状態で実行します。このプロパティーの詳細については、59 ページの『エージェントの構成』を参照してください。

このセクションでは、CORBA コネクターで処理されるビジネス・オブジェクトの構造に関連した主要概念について説明します。

属性

IDL ファイルに定義された CORBA クラス内の属性ごとに、対応するビジネス・オブジェクト属性が ODA によって生成されます。IDL ファイルは、プロキシー・オブジェクト定義をコンパイルするために、ODA によって使用されます。

CORBA クラス内の属性が単純属性ではなく、`struct` (41 ページの図 6)、`union` (42 ページの図 9)、または `sequence` (42 ページの図 8) の場合、BO 属性は、CORBA オブジェクト内の対応する構造型 (construct) と定義が一致する子オブジェクトにマップされます。CORBA `enum` 構造 (42 ページの図 10) は子オブジェクト属性ではなく単純属性にマップされます。

CORBA の基本構造を 41 ページの表 7、および 41 ページの図 6 から 42 ページの図 9 に示します。(コネクターが、`constant` 構造をサポートしていないことに注意してください)。

CORBA 構造とビジネス・オブジェクト間のマッピングに関する完全なリストについては、49 ページの『属性のマッピング: CORBA、Java、およびビジネス・オブジェクト』を参照してください。

表 7. CORBA 構造

CORBA 構造	説明
struct	図 6 に示すような、ビジネス・データを保持するオブジェクト。
interface	図 7 に示すような、ビジネス・オペレーション (メソッド) のリストを保持するオブジェクト。
sequence	42 ページの図 8 に示すような、構造のリストまたは単一データ型を保持するオブジェクト。CORBA sequence は、配列と同じように、カーディナリティー n ビジネス・オブジェクトにマップされます。
union	42 ページの図 9 に示すような、構造のコレクションまたは単一データ型を保持するオブジェクト。union 内で値を保持できるのは、一時に 1 つの属性のみです。
enum	42 ページの図 10 に示すような、順次 ID または 列挙 ID のリストを含むオブジェクト。

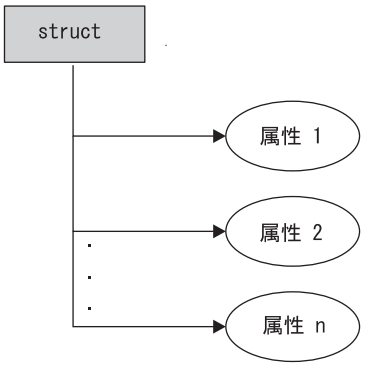


図 6. CORBA 構造: struct

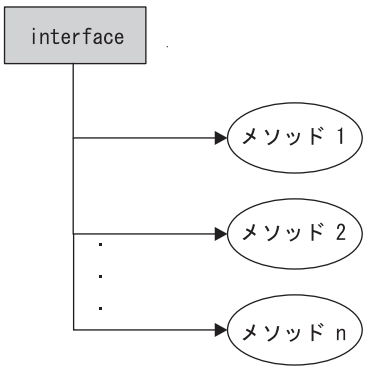


図 7. CORBA 構造: interface

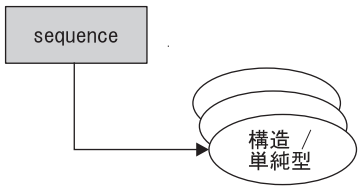


図 8. CORBA 構造: sequence

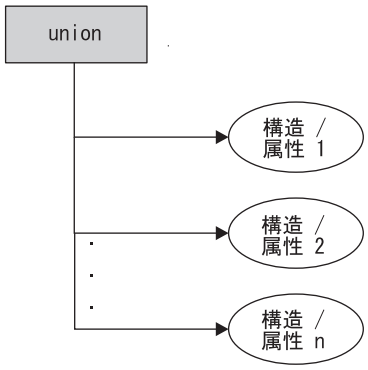


図 9. CORBA 構造: union

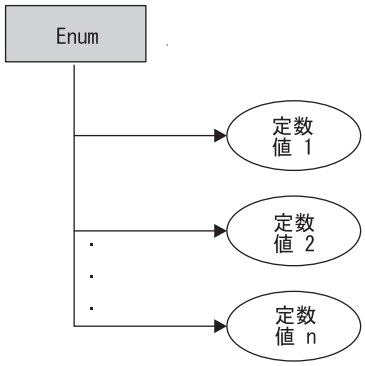


図 10. CORBA 構造: enum

ビジネス・オブジェクト属性の中には、データを含む代わりに、子ビジネス・オブジェクトや、子オブジェクトのデータを含む子ビジネス・オブジェクトの配列を参照する属性もあります。親レコードと子レコード間のデータはキーにより関連付けられます。

ビジネス・オブジェクトにはフラットなものと階層構造のものがあります。フラットなビジネス・オブジェクトには、単純属性、つまり、ストリングなどの単一値を表し、子ビジネス・オブジェクトを参照しない属性のみが含まれます。階層ビジネス・オブジェクトは単純属性、および属性値を含む子ビジネス・オブジェクトまたは子ビジネス・オブジェクトの配列を含みます。

カーディナリティー 1 コンテナー・オブジェクトまたは単一カーディナリティー関係は、親ビジネス・オブジェクトの属性に単一の子ビジネス・オブジェクトが含まれる場合に発生します。この場合、子ビジネス・オブジェクトは、レコードを 1 つのみ含むコレクションになります。属性タイプは子ビジネス・オブジェクトのものと同一です。

カーディナリティー n コンテナー・オブジェクトまたは複数カーディナリティー関係は、親ビジネス・オブジェクトの属性が子ビジネス・オブジェクトの配列を含む場合に発生します。この場合、子ビジネス・オブジェクトは、複数のレコードを含むコレクションになります。属性タイプは、子ビジネス・オブジェクトの配列の属性タイプと同じになります。

メソッド

CORBA IDL ファイルで定義されたメソッドごとに、ビジネス・オブジェクトの属性が作成されます。属性タイプは、メソッド・パラメーターを表す属性を含む子 BO です。子 BO の属性は、CORBA メソッドのパラメーターと同じ順序で表示されます。子 BO は、CORBA メソッド呼び出しの結果を表す `Return_Value` 属性も持ちます。このような子 BO の属性は、メソッド・パラメーターのタイプや戻り値に応じて、単一タイプの場合もあれば、オブジェクト・タイプの場合もあります。

アプリケーション固有の情報

アプリケーション固有の情報を使用して、ビジネス・オブジェクトの処理方法に関するアプリケーション固有の指示をコネクターに提供します。ビジネス・オブジェクト定義を拡張または変更する場合には、定義内のアプリケーション固有の情報と、コネクターが予期する構文とを必ず一致させる必要があります。

各ビジネス・オブジェクト属性と同様にビジネス・オブジェクト全体に対してもアプリケーション固有の情報を指定できます。

ビジネス・オブジェクト・レベルの ASI

オブジェクト・レベルの ASI では、ビジネス・オブジェクトの性質およびビジネス・オブジェクトに含まれるオブジェクトについての基本情報を提供します。ビジネス・オブジェクトに必要な ASI は、サーバーとして機能するコネクターのオブジェクトを生成するか、またはクライアントとして機能するコネクターのオブジェクトを生成するかどうかによって異なります。

注: アプリケーション固有の情報は、メソッド、メソッド・パラメーター、およびメソッド戻り値を表すビジネス・オブジェクトに使用されます。CORBA オブジェクトのメソッドとして作成されるビジネス・オブジェクト属性の詳細については、『メソッド』を参照してください。

44 ページの表 8 に、ビジネス・オブジェクトのビジネス・オブジェクト・レベル ASI を示します。このビジネス・オブジェクトは、Connector for CORBA がクライアントとして機能する際に、クライアント・オブジェクトとして処理されます。

表 8. クライアント・オブジェクトのオブジェクト・レベル *ASI*

オブジェクト・レベル ASI	説明
proxy_class=<*nameOfProxy*>	ビジネス・オブジェクトが表すプロキシー・クラスの名前。この ASI を使用して、プロキシー・クラスをビジネス・オブジェクトにマップします。この値は、有効な Java パッケージ表記 (java.lang.Vector など) を使用して指定する必要があります。
factory_method=<*Name Of Factory Method*>	指定した proxy_class のインスタンスを生成するために使用される Factory クラスのメソッドの名前。今回のリリースでは、引き数を取らないメソッドのみをサポートしています。
object_type=<*leave blank or set to zero-length string*>	ODA エージェントの CORBAServerImpl プロパティー (60 ページの表 15 を参照) に、false が設定されている、すなわち、ODA がクライアント側オブジェクトを生成するように構成されている場合は、この ASI は指定しないか、ゼロ長 String を設定するようにします。
implementation_class=<*leave blank or set to zero-length string*>	ODA エージェントの CORBAServerImpl プロパティー (60 ページの表 15を参照) に、false が設定されている、すなわち、ODA がクライアント側オブジェクトを生成するように構成されている場合は、この ASI は指定しないか、ゼロ長 String を設定するようにします。

外部 CORBA クライアント・オブジェクトがメソッド呼び出しを実行できるサーバーとしてコネクターが機能する際に、サーバー・オブジェクトとして処理されるビジネス・オブジェクトのビジネス・オブジェクト・レベル ASI を表 9 に示します。ODA エージェントの CORBAServerImpl プロパティーに true が設定されている場合は、ODA が、サーバー側ビジネス・オブジェクトを生成することに注意してください。このプロパティーの詳細については、 60 ページの表 15 を参照してください。

表 9. サーバー・オブジェクトのオブジェクト・レベル *ASI*

オブジェクト・レベル ASI	説明
proxy_class=<*nameOfProxy*>	ビジネス・オブジェクトが表すプロキシー・クラスの名前。この ASI を使用して、プロキシー・クラスをビジネス・オブジェクトにマップします。この値は、Java パッケージ表記 (java.lang.Vector など) を使用して指定する必要があります。
object_type=CorbaImplObject	コネクターがサーバーとして機能し、BO が CORBA クライアントに対して意図されて BO へのメソッド呼び出しを行う場合、この ASI を CorbaImplObject に設定します。

表 9. サーバー・オブジェクトのオブジェクト・レベル ASI (続き)

オブジェクト・レベル ASI	説明
implementation_class=<*Name Of Implementation Class*>	ODA で作成中のサーバー・ビジネス・オブジェクトに対応する、インプリメンテーション・クラスの名前。インプリメンテーション・クラス名は、常に次のようになります。 com.ibm.adapters.corbaadapter.impl.<*ClassName*>Impl ここで、<*ClassName*> はインプリメントされている元のクラスの名前です。クラスは、出力 .jar ファイルの形式で次のフォルダーに保管されます。 com¥ibm¥adapters¥corbaadapter¥impl

図 11 に、サーバー・オブジェクトとして機能するサンプル・ビジネス・オブジェクトのオブジェクト・レベル ASI を示します。

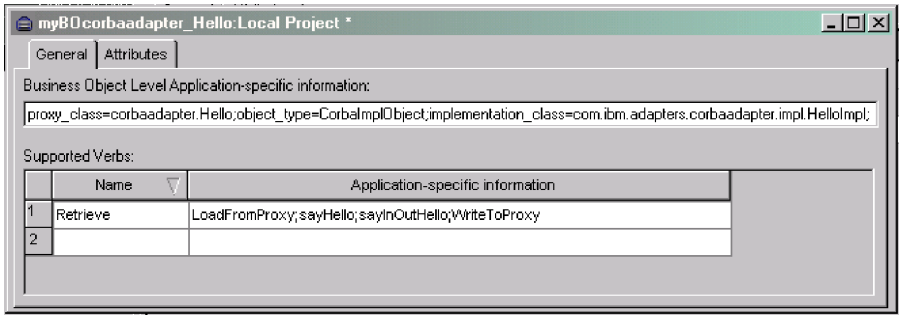


図 11. サーバー・サイド・オブジェクトのサンプル・ビジネス・オブジェクト・レベル ASI

動詞 ASI

コネクターがクライアントとして稼働している場合、コネクターが処理する各クライアント・サイド・ビジネス・オブジェクトに動詞が含まれます。この動詞を使用して、受信アプリケーションがビジネス・オブジェクト内のデータを処理する方法を記述します。

注: サーバー・サイド・オブジェクトは、コネクターがサーバーとして稼働している場合に処理されますが、動詞 ASI を持ちません。

動詞 ASI には、一連の属性名が含まれています。各属性名は、汎用ビジネス・オブジェクト・ハンドラーによる呼び出しの対象となるメソッドが含まれます。一般に、呼び出しの対象となるメソッドは、親ではなくオブジェクト自体に属しています。この場合、オブジェクトの動詞 ASI にこのメソッドを指定できます。例えば、メソッド IncrementCounter が指定されているオブジェクトの場合、対応するビジネス・オブジェクトの動詞 ASI にこのメソッドを指定する必要があります。

呼び出し対象のメソッドがビジネス・オブジェクト階層の親に属する場合は、PARENT タグの付いたメソッド名をプレフィックス交換することにより、親を参照することができます。

例えば、図 12 は、ContactDetails が Contact の子オブジェクトであり、Contact は PSRCustomerAccount の子であるというビジネス・オブジェクト階層を示しています。

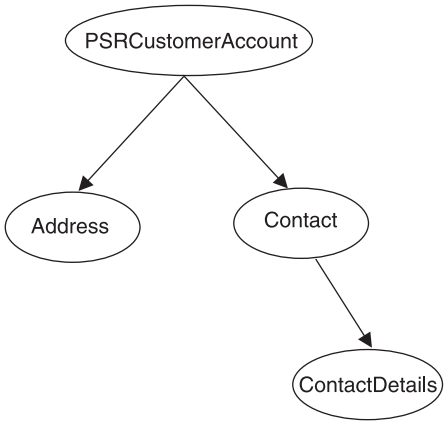


図 12. ビジネス・オブジェクト階層と動詞 ASI

PSRCustomerAccount に属するメソッドを ContactDetails ビジネス・オブジェクトで呼び出す場合、ContactDetails の動詞 ASI は、次のようなビジネス・オブジェクト階層を示します。

PARENT.PARENT.*<methodName>*

メソッドが、Contact ビジネス・オブジェクトに属する場合は、ContactDetails の動詞 ASI を次のように設定します。

PARENT.*<methodName>*

呼び出しが可能なのは、階層内の親オブジェクトに属するメソッドのみです。そのため、親ビジネス・オブジェクトは、子のメソッドを呼び出すことができません。

コネクター開発者が、動詞に割り当てられる CORBA 操作を決定します。サポートされている動詞を以下に示します。ただし、Business Object Designer を使用して他の動詞を手動で追加することもできます。

- Create
- Delete
- Retrieve
- Update

以下のキーワードは、属性名の動詞 ASI シーケンスに使用します。

表 10. 動詞 ASI に使用可能なキーワード

キーワード	説明
LoadFromProxy= *<attributeName>*	プロキシー・オブジェクトから特定の非メソッド属性値をロードします。

表 10. 動詞 ASI に使用可能なキーワード (続き)

キーワード	説明
WriteToProxy = <*attributeName*>	非メソッド属性値をビジネス・オブジェクトからプロキシー・オブジェクトに書き込みます。
LoadFromProxy (属性名以外)	プロキシー・オブジェクトから現在の BO の非メソッド属性をすべてロードします。
WriteToProxy (属性名以外)	現在の BO の非メソッド属性をすべてプロキシー・オブジェクトに書き込みます。
CBOH=<*custom BO handler className*>	カスタム BO ハンドラーのクラス名。この場合、汎用 BO ハンドラーは使用しません。カスタム BO ハンドラーについて詳しくは、17 ページの『カスタム・ビジネス・オブジェクト・ハンドラー』を参照してください。

特定のオブジェクトに対し、サポートされている 4 つの動詞 (Create、 Delete、 Retrieve、および Update) を指定し、各動詞のアクションとして n + 2 個のメソッド (n は対応する CORBA インターフェースのメソッドの数) を割り当てることができます。 2 つの追加メソッドは、コネクターによりサポートされているメソッド LoadFromProxy および WriteToProxy です。詳細については、46 ページの表 10 を参照してください。

属性レベル ASI

ビジネス・オブジェクトの属性レベル ASI は、子オブジェクトが含まれている複合属性と単純属性の両方に使用できます。複合属性の場合、含まれている子がオブジェクトのプロパティー (メソッドではない) またはメソッドのいずれかによって ASI は異なります。元の CORBA IDL ファイルのすべての属性タイプからビジネス・オブジェクトへのマッピングについては、49 ページの表 14 を参照してください。

単純属性の ASI については、表 11 で説明しています。

表 11. 単純属性を含む属性の属性レベル ASI

属性 ASI	説明
Name	ビジネス・オブジェクトのフィールド名です。
Type	ビジネス・オブジェクトのフィールド・タイプです。
MaxLength	デフォルトでは 255
IsKey	各ビジネス・オブジェクトは、少なくとも 1 つのキー属性を持つ必要があります。キー属性は、属性のキー・プロパティーを true に設定することで指定します。
IsForeignKey	コネクターが呼び出しごとのオブジェクト・プールに値を追加するように設定するには、true を設定します。
IsRequired	false を設定します。
AppSpecInfo	元の Java タイプを保持します。この属性は次のようにフォーマット設定します。 property=<*propertyName*>; type=<*typeName*> property は CORBA オブジェクト属性の名前です。 この ASI を使用して元の CORBA オブジェクト属性名を取り込みます。 type は CORBA 単純属性タイプの名前です。この属性を使用して、 元の CORBA タイプ名を取り込みます。

表 11. 単純属性を含む属性の属性レベル ASI (続き)

属性 ASI	説明
DefaultValue	属性が設定されていない場合に、コネクターがインバウンド・ビジネス・オブジェクトの単純属性に対して使用するデフォルト値であり、必須属性です。

非メソッド複合属性の ASI については、表 12 で説明します。これらの属性には、非メソッド子オブジェクト (元の CORBA IDL ファイルのクラスのプロパティーなど) が含まれています。

表 12. 非メソッド子オブジェクトを含む属性の属性レベル ASI

属性	説明
Name	ビジネス・オブジェクトのフィールド名です。
Type	ビジネス・オブジェクトのフィールド・タイプです。
MaxLength	デフォルトでは 255
IsKey	各ビジネス・オブジェクトは、少なくとも 1 つのキー属性を持つ必要があります。キー属性は、属性のキー・プロパティーを true に設定することで指定します。
IsForeignKey	コネクターが呼び出しごとのオブジェクト・プールに値を追加するように設定するには、true を設定します。
IsRequired	false を設定します。
AppSpecInfo	元の Java タイプを保持します。この属性は次のようにフォーマット設定します。 type=<*typeName*>; use_attribute_value=<*BOName.AttributeName*>; property=<*propertyName*>; proxy_class=<*proxyClassName*>; enumeration_class=<*enumerationClassName*>; inout=<*true or false*>; union=true; union_key=<*unionKeyName*>; type は、オブジェクトを参照する場合にプロキシー・クラス名となる値です。 use_attribute_value には <*BOName.AttributeName*> が設定されます。 このエレメントの値を指定すると、実行時にコネクターは呼び出しごとの オブジェクト・プールからその値を取得します。 property は CORBA オブジェクト属性の名前です。 この ASI を使用して元の CORBA オブジェクト属性名を取り込みます。 proxy_class はオプションです。属性のタイプが in/out の場合にのみ使用します。 enumeration_class は、この非メソッド属性に対応する CORBA 列挙型クラスです。 このオプション・エレメントは、属性が CORBA 列挙型構成にマップする場合にのみ 使用します。 inout には true または false が設定されます。 属性のタイプが in/out の場合にのみ、このオプション・エレメントを メソッド・パラメーターとして使用します。 union には true を設定します。オプションは、子ビジネス・オブジェクトが CORBA union 構造にマップする場合にのみ使用します。 union_key は、各メソッドへのマッピング・キーであり、union に値を戻します。 このオプション・エレメントは、ビジネス・オブジェクトが CORBA union 構造に マップする場合にのみ使用します。 CORBA 構造の詳細については、 40 ページの『属性』を参照してください。
DefaultValue	属性が設定されていない場合に、コネクターがインバウンド・ビジネス・オブジェクトの単純属性に対して使用するデフォルト値であり、必須属性です。

メソッドである子オブジェクトを含む複合属性の ASI については、表 13 で説明します。

表 13. メソッドである子オブジェクトを含む属性の属性レベル ASI

属性	説明
Name	ビジネス・オブジェクトのフィールド名です。
Type	ビジネス・オブジェクトのフィールド・タイプです。
Relationship	子がコンテナー属性の場合は、この値を Containment に設定します。
IsKey	使用しません。
IsForeignKey	false を設定します。
Is Required	false を設定します。
AppSpecInfo	元の CORBA アプリケーション・メソッド名を保持します。 この属性は次のようにフォーマット設定します。 method_name=<*methodName*>; verb=<*verbName*>; method_name は、コネクターがクライアントとして機能するときに 外部 CORBA サーバーに対して実行されるメソッド呼び出しの名前です。 verb は、コネクターがサーバーとして機能するときに、コラボレーションの 呼び出し前にコネクターがビジネス・オブジェクトに対し設定する動詞です。 この動詞は、外部 CORBA クライアントからのメソッド呼び出しに対応しています。 メソッド呼び出しとは、コラボレーション実行要求です。 ビジネス・オブジェクトが、外部 CORBA クライアント・オブジェクトからの メソッド呼び出しを受信する CORBA サーバー・オブジェクトとして 機能している場合にのみ使用してください。 有効な動詞のリストについては、 45 ページの『動詞 ASI』を参照してください。
Cardinality	タイプが配列またはベクトルを表す場合は N を設定し、それ以外の場合は 1 を設定します。

属性のマッピング: CORBA、Java、およびビジネス・オブジェクト

このセクションでは、主要な CORBA IDL 構造と、それに対応する Java 構造およびビジネス・オブジェクト属性をリストにして説明します。子ビジネス・オブジェクト以外のビジネス・オブジェクト属性はすべて、データ型が String となります。ビジネス・オブジェクトにおいて、ASI は、属性の実際のデータ型を保持しており、Java プロキシー・オブジェクトに対してメソッドを呼び出す場合に使用されます。

CORBA 構造の詳細については、 40 ページの『属性』を参照してください。

ビジネス・オブジェクト ASI の詳細については、 43 ページの『アプリケーション固有の情報』を参照してください。

表 14. オブジェクト・マッピング: CORBA、Java、およびビジネス・オブジェクト

CORBA IDL 構造	Java 構造	ビジネス・オブジェクト	ASI
モジュール	パッケージ	(適用なし)	(適用なし)

表 14. オブジェクト・マッピング: *CORBA*、*Java*、およびビジネス・オブジェクト *(続き)*

CORBA IDL 構造	Java 構造	ビジネス・オブジェクト	ASI
インターフェース (非抽象)	シグニチャー・インターフェースおよびオペレーション・インターフェース、helper クラス、holder クラス	BO	proxy_class=<*完全修飾 CORBA クラス名*>
インターフェース (抽象)	シグニチャー・インターフェース、helper クラス、holder クラス	BO	proxy_class=<*完全修飾 CORBA インターフェース名*>
ブール値	ブール値	Boolean	type=boolean
char、wchar	char	String	type=char
オクテット	バイト	String	type=byte
string、wstring	java.lang.String	String	type=String
short、符号なし short	short	Integer	type=short
long、符号なし long	int	Integer	type=int
long、long 符号なし long long	long	Integer	type=long
浮動	浮動	Float	type=float
double	double	Double	type=double
固定	java.math.BigDecimal	String	type=BigDecimal
enum	クラス	String	property=<*propertyName*>;type=String; enumeration_class=<*JavaEnumClassName*>
struct、union	クラス	BO	proxy_class=<*完全修飾クラス名*>
シーケンス、配列	配列	複数カーディナリティーを持つ子 BO	proxy_class=<*完全修飾クラス名*>
例外	クラス	(適用なし)	(適用なし)
読み取り専用属性	accessor メソッド	子 BO	method=<メソッド名>
読み取り/書き込み属性	accessor メソッドおよび modifier メソッド	子 BO	method=<メソッド名>
操作	メソッド	子 BO	method=<メソッド名>

ビジネス・オブジェクト・プロパティー例

このセクションでは、WebSphere Business Integration のビジネス・オブジェクトについて、例を挙げて説明します。対応する CORBA クラスおよび Java プロキシー・クラスについても、3 つの構造にまたがるマッピングを説明します。ビジネス・オブジェクトは、一致する CORBA アプリケーション・オブジェクトから名前を継承します。

このセクションで説明する例は、以下のとおりです。

- 51 ページの『IDL ファイル例』

- 『IDLJ によって生成される Java コード例』
- 53 ページの『Java クラスのビジネス・オブジェクト例』
- 54 ページの『BO ハンドラー・メソッド呼び出し例』

IDL ファイル例

以下のサンプル・コードは、大規模 IDL ファイルの一部です。ここに示す部分では、CORBAAccount `struct` の定義、およびメソッドの引き数タイプにそのクラスを使用する `Hello` インターフェースの定義を示します。

```
# Sample IDL File
#
#
module corbaadapter
{
.
.
.
    struct CORBAAccount
    {
        short                       accessCustomerNumber;
        AccountStatusEnum           accountStatus;
        string                      acctSecurity;
        string                      companyNm;
        long                        custAcctID;
        string                      disconnectReasonCd;
        string                      firstNm;
        string                      lastNm;
        char                        middleInitial;
        CORBASicCodeUnion           sicCode;
        CORBAAddressSeq             addresses;
        LongSeq                     custAcctChildrenIds;
        StringSeq                   nameList;
        ShortSeq                    accountList;
        BooleanSeq                  flagList;
        CharSeq                     initialList;
        FloatSeq                    amountList;
        DoubleSeq                   doubleAmtList;
    };


    interface Hello
    {
        CORBAAccount sayHello(in CORBAAccount test, inout double amount);

        CORBAAccount sayInOutHello(inout CORBAAccount test,
 inout string name,
 in long id);
    };
};
```

IDLJ によって生成される Java コード例

以下の例では、IDLJ コンパイラー・ツールが『IDL ファイル例』のコードを使用することによって生成する Java コードを説明します。

- 52 ページの『Java コード例: CORBAAccount クラス』
- 53 ページの『Java コード例: HelloOperations クラス』

Java コード例: CORBAAccount クラス

以下は、 51 ページの『IDL ファイル例』で定義した CORBAAccount struct に対して IDLJ コンパイラー・ツールが生成した Java サンプル・コードの一部です。

```
package corbaadapter;

/**
 * <ul>
 * <li> <b>IDL Source</b>    "d:/corba adapter/sample/hello.idl"
 * <li> <b>IDL Name</b>      ::corbaadapter::CORBAAccount
 * <li> <b>Repository Id</b> IDL:corbaadapter/CORBAAccount:1.0
 * </ul>
 * <b>IDL definition:</b>
 * <pre>
 * struct CORBAAccount {
  ...
};
 * </pre>
 */
public final class CORBAAccount implements org.omg.CORBA.portable.IDLEntity {

  public short accessCustomerNumber;

  public corbaadapter.AccountStatusEnum accountStatus;

  public java.lang.String acctSecurity;

  public java.lang.String companyNm;

  public int custAcctID;

  public java.lang.String disconnectReasonCd;

  public java.lang.String firstNm;

  public java.lang.String lastNm;

  public char middleInitial;

  public corbaadapter.CORBASicCodeUnion sicCode;

  public corbaadapter.CORBAAddress[] addresses;

  public int[] custAcctChildrenIds;

  public java.lang.String[] nameList;

  public short[] accountList;

  public boolean[] flagList;

  public char[] initialList;

  public float[] amountList;

  public double[] doubleAmtList;

  public CORBAAccount () {
  }

.
.
.
}
```

Java コード例: HelloOperations クラス

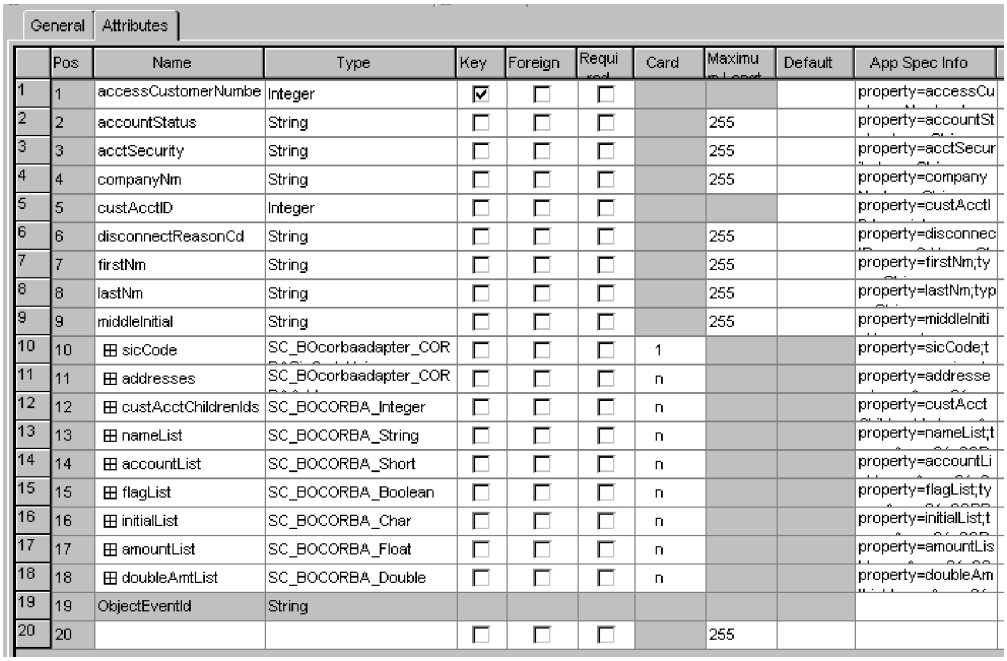
以下は、 51 ページの『IDL ファイル例』で定義した Hello インターフェースに対して、IDLJ コンパイラー・ツールが生成した Java クラスのサンプル・コードです。

```
package corbaadapter;

/**
 * <ul>
 * <li> <b>IDL Source</b>    "d:/corba adapter/sample/hello.idl"
 * <li> <b>IDL Name</b>      ::corbaadapter::Hello
 * <li> <b>Repository Id</b> IDL:corbaadapter/Hello:1.0
 * </ul>
 * <b>IDL definition:</b>
 * <pre>
 * interface Hello {
  ...
};
 * </pre>
 */
public interface HelloOperations {
  /**
   * <pre>
   *   corbaadapter.CORBAAccount sayHello (in corbaadapter.CORBAAccount test,
                                    inout double amount);
   * </pre>
   */
  public corbaadapter.CORBAAccount sayHello (corbaadapter.CORBAAccount test,
                                      org.omg.CORBA.DoubleHolder amount);

  /**
   * <pre>
   *   corbaadapter.CORBAAccount sayInOutHello (inout corbaadapter.
                            CORBAAccount test,inout string name, in long id);
   * </pre>
   */
  public corbaadapter.CORBAAccount sayInOutHello
        (corbaadapter.CORBAAccountHolder test,org.omg.CORBA.StringHolder
         name, int id);
```

Java クラスのビジネス・オブジェクト例

以下は、 51 ページの『IDLJ によって生成される Java コード例』の例で定義した Java クラスについて、Business Object Designer での表示に従ってビジネス・オブジェクト構造を示したサンプル画面です。

54 ページの図 13 に、CORBAAccount クラスのビジネス・オブジェクト構造を示します。



General | Attributes

	Pos	Name	Type	Key	Foreign	Requi red	Card	Maximu m Lengt	Default	App Spec Info
1	1	accessCustomerNumbe	Integer	☑	☐	☐				property=accessCu
2	2	accountStatus	String	☐	☐	☐		255		property=accountSt
3	3	acctSecurity	String	☐	☐	☐		255		property=acctSecur
4	4	companyNm	String	☐	☐	☐		255		property=company
5	5	custAcctID	Integer	☐	☐	☐				property=custAcctI
6	6	disconnectReasonCd	String	☐	☐	☐		255		property=disconnec
7	7	firstNm	String	☐	☐	☐		255		property=firstNm;ty
8	8	lastNm	String	☐	☐	☐		255		property=lastNm;typ
9	9	middleInitial	String	☐	☐	☐		255		property=middleIniti
10	10	⊞ sicCode	SC_BOcorbaadapter_COR	☐	☐	☐	1			property=sicCode;t
11	11	⊞ addresses	SC_BOcorbaadapter_COR	☐	☐	☐	n			property=addresse
12	12	⊞ custAcctChildrenIds	SC_BOCORBA_Integer	☐	☐	☐	n			property=custAcct
13	13	⊞ nameList	SC_BOCORBA_String	☐	☐	☐	n			property=nameList;t
14	14	⊞ accountList	SC_BOCORBA_Short	☐	☐	☐	n			property=accountLi
15	15	⊞ flagList	SC_BOCORBA_Boolean	☐	☐	☐	n			property=flagList;ty
16	16	⊞ initialList	SC_BOCORBA_Char	☐	☐	☐	n			property=initialList;t
17	17	⊞ amountList	SC_BOCORBA_Float	☐	☐	☐	n			property=amountLis
18	18	⊞ doubleAmtList	SC_BOCORBA_Double	☐	☐	☐	n			property=doubleAm
19	19	ObjectEventId	String							
20	20			☐	☐	☐		255		

図 13. CORBAAccount クラスのビジネス・オブジェクト構造

BO ハンドラー・メソッド呼び出し例

51 ページの『IDL ファイル例』 で定義した CORBA オブジェクトに対して、コネクター BO ハンドラーは以下のメソッド呼び出しを実行できます。

```
//Initialize ORB

      ORB orb = ORB.init(args, orbProps);
      System.out.println("ORB initialized");
      byte[] helloId = "HelloServerObject".getBytes();
      Hello helloRef = HelloHelper.bind(orb, "/CORBAServer", helloId);
      // Call the Hello server object and print results
      CORBAAccount customer = new CORBAAccount();
      customer.accessCustomerNumber = 0;
            customer.accountStatus = AccountStatusEnum.asPENDING;
      customer.acctSecurity = "check";
      customer.companyNm = "Hello";
           customer.custAcctID = 100;
      customer.disconnectReasonCd = "Reason";
      customer.firstNm = "Name check";
            customer.lastNm = "Last Name";
      customer.middleInitial = 'D';
      CORBASicCodeUnion sicCodeUnion = new CORBASicCodeUnion();
      CORBASicCode sicCode = new CORBASicCode();
      sicCode.description = "Description";
      sicCode.sicCd = "1000";
      sicCode.stdCdInd = 'N';
      sicCode.subAcctInd = 'S';
      sicCodeUnion.value(sicCode);
      customer.sicCode = sicCodeUnion;
      customer.addresses = new CORBAAddress[0];
      customer.custAcctChildrenIds = new int[0];
```

```
double value = 123;
DoubleHolder dHolder = new DoubleHolder(value);
customer = helloRef.sayHello(customer, dHolder);
```

ビジネス・オブジェクトの生成

実行時にイベントが発生するたびに、CORBA アプリケーションは、オブジェクト・レベルのデータとトランザクション・タイプに関する情報を含むメッセージ・オブジェクトを送信します。コネクターは、このデータを対応するビジネス・オブジェクト定義にマップして、アプリケーション固有のビジネス・オブジェクトを作成します。コネクターは、作成したビジネス・オブジェクトを統合ブローカーに送信して、処理します。また、統合ブローカーから戻されるビジネス・オブジェクトを受け取り、CORBA アプリケーションに渡します。

注: CORBA アプリケーションのオブジェクト・モデルを変更する場合は、ODA を使用して新しい定義を作成します。統合ブローカー・リポジトリーのビジネス・オブジェクト定義が CORBA アプリケーションによって送信されるデータと正確に一致しない場合は、コネクターでビジネス・オブジェクトを作成することはできないためトランザクションは失敗します。

Business Object Designer の提供するグラフィカル・インターフェースを使用すると、実行時に使用するビジネス・オブジェクト定義を作成および変更できます。詳細については、57 ページの『第 5 章 ビジネス・オブジェクトの作成および変更』を参照してください。

第 5 章 ビジネス・オブジェクトの作成および変更

この章では、Object Discovery Agent (ODA) for CORBA について、およびそれを使用して IBM WebSphere Business Integration Adapter for CORBA のビジネス・オブジェクト定義を生成する方法について説明します。

本章の内容は、次のとおりです。

ODA for CORBA の概要

ODA (Object Discovery Agent) を使用すると、ビジネス・オブジェクト定義を生成できます。ビジネス・オブジェクト定義とは、ビジネス・オブジェクトのテンプレートです。ODA は、指定したアプリケーション・オブジェクトの検証、ビジネス・オブジェクト属性に対応するビジネス・オブジェクトのエレメントの「発見」、および情報を示すビジネス・オブジェクト定義の生成を実行します。Business Object Designer では、Object Discovery Agent にアクセスし、ODA と対話的に連動するグラフィカル・インターフェースを提供しています。

Object Discovery Agent (ODA) for CORBA は、IDL ファイルのメタデータからビジネス・オブジェクト定義を生成します。Business Object Designer ウィザードを使用すると、ビジネス・オブジェクト定義の作成が自動的に処理されます。ODAを使用して、ビジネス・オブジェクトおよび Connector Configurator を作成し、それらをサポートするコネクターを構成します。Connector Configurator について詳しくは、 97 ページの『付録 B. Connector Configurator』を参照してください。

IDL ファイル互換性

Adapter for CORBA バージョン 1.0.x のユーザーが、バージョン 1.2.x に移行する場合、IDL ファイルを再コンパイルし、ODA コンポーネントのバージョン 1.2.x と互換性のあるプロキシー・クラス・ファイルを生成する必要があります。そのため、以下の 2 つのオプションがあります。

- **オプション 1:** 58 ページの『ビジネス・オブジェクト定義の生成』の説明に従って、ODA を使用して、必要な実行可能ファイルを再生成します
- **オプション 2:** 既存のビジネス・オブジェクトを維持しながら、IBM IDLJ Java コンパイラー・ツール (`idlj` または `idlj.exe`) を使用して、手動で Java 実行可能ファイルを再生成します。このためには、以下のステップを実行します。
  1. IBM IDLJ Java コンパイラーを使用して、IDL ファイルを再コンパイルします。このコンパイラーは、Window 2000 と AIX プラットフォームの場合、IBM JDK ディレクトリーの bin ディレクトリーから使用することができま

す。Solaris と HP-UX プラットフォームの場合は、コンパイラーは以下のディレクトリーで使用することができます。

<adapter runtime directory>/jre/ibm_bin

ここで、*<adapter runtime directory>* は、アダプターのランタイム・ファイルが格納されるディレクトリーです。

サーバーとして動作するアダプターに対して Java ファイルを生成する場合は、`-fserverTIE` オプションを使用して、IBM IDLJ Java コンパイラーを起動してください。

クライアントとして動作するアダプターに対して Java ファイルを生成する場合は、`-fclient` オプションを使用して、IBM IDLJ Java コンパイラーを起動してください。

2. JDK バージョン 1.3.1 で提供されている Java コンパイラーを使用して、Java ファイルをコンパイルします。
3. ステップ 2 で生成した Java 実行可能ファイル (`.class` 拡張子を持つファイル) を、適切な `.jar` ファイルにパッケージします。

ビジネス・オブジェクト定義の生成

このセクションでは、Business Object Designer の CORBA ODA を使用して、ビジネス・オブジェクト定義を生成する方法について説明します。 Business Object Designer の起動方法および使用方法について詳しくは、「*IBM WebSphere Business Integration Adapters ビジネス・オブジェクト開発ガイド*」を参照してください。

ODA の始動

ODA は、メタデータ・リポジトリー (IDL ファイル) を持つファイル・システムをマウントできる任意のマシンから、始動ファイル start_CORBAODA.bat (Windows) または start_CORBAODA.sh (UNIX) を使用して実行できます。この始動ファイルには、必須の CORBA およびコネクター .jar ファイルをへのパスなどの始動パラメーターが含まれます。必須の .jar ファイルには、ODA を実行しているマシンからもアクセスできます。

ODA for CORBA のデフォルトの名前は、CORBAODA です。この名前は、始動スクリプトで `AGENTNAME` 変数の値を変更することにより、変更できます。

ODA を始動するには、以下のコマンドを実行します。

- **Windows**: `start_CORBAODA`
- **UNIX**: `start_CORBAODA.sh`

Business Object Designer の実行

Business Object Designer では、ODA を使用してビジネス・オブジェクト定義を生成するためのステップをガイドするウィザードを提供しています。ビジネス・オブジェクト定義を生成するステップは、以下のとおりです。

エージェントの選択

最初に、ODA エージェントを選択しておく必要があります。

1. Business Object Designer を始動します。
2. **「ファイル」 >「ODA を使用して新規作成」**を選択します。*「ビジネス・オブジェクト・ウィザード - ステップ 1/6 - エージェントの選択」*画面が表示されます。
3. **「検索されたエージェント」**リストで、(start_CORBAODA スクリプトから) ODA/AGENTNAME を選択して、**「次へ」**をクリックします。(使用するエージェントがリストにない場合には、**「エージェントの検索」**をクリックします。)

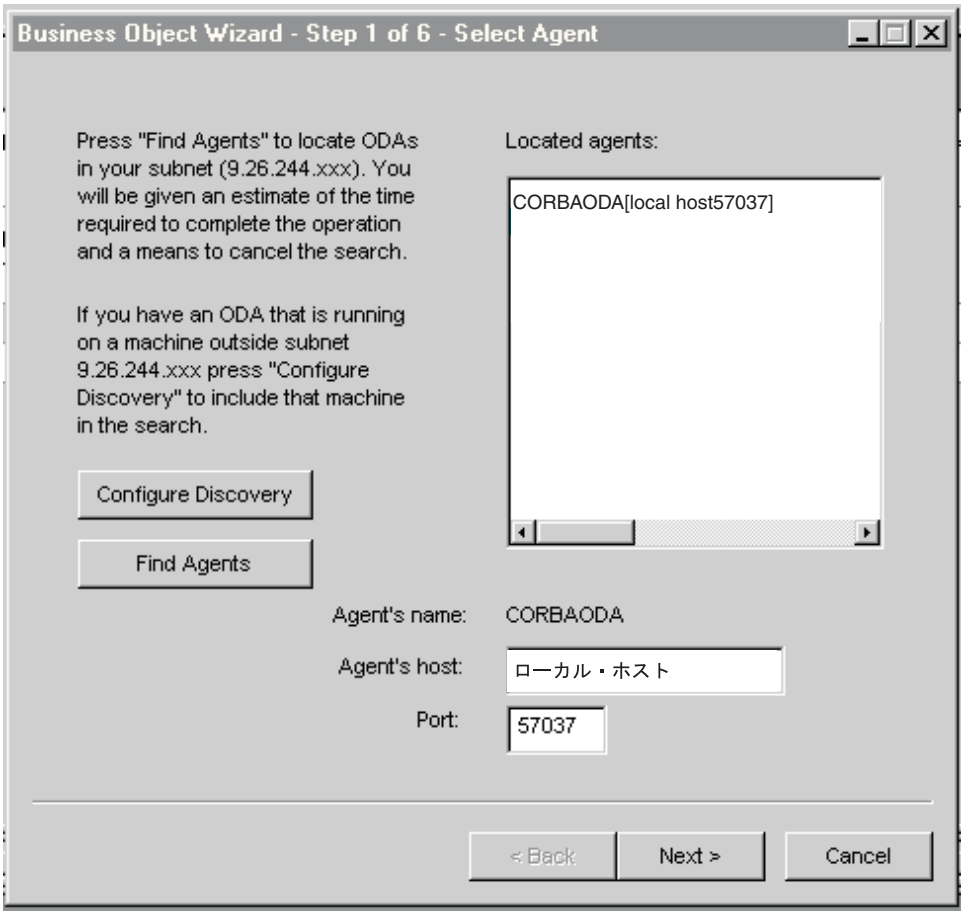


図 14. 「エージェントの選択」画面

エージェントの構成

「エージェントの選択」画面の**「次へ」**をクリックすると、*「ビジネス・オブジェクト・ウィザード - ステップ 2/6 - エージェントの構成」*画面が表示されます。60 ページの図 15 は、サンプル値が指定されたエージェントの構成画面です。

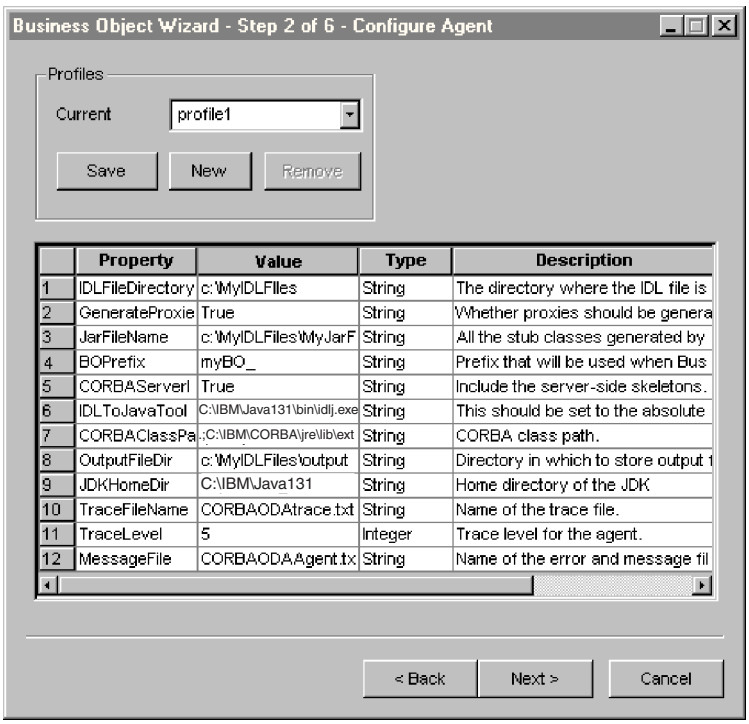


図 15. 「エージェントの構成」画面

表 15 に、この画面で設定するプロパティーを示します。この画面で入力した値はすべて、プロファイルに保管できます。そのため、次回 ODA を実行する際、プロパティー・データを再入力しなくても、ドロップダウン・メニューからプロファイルを選択するだけで保管した値を再利用できます。それぞれが指定した値で構成された異なるセットを持つ複数のプロファイルを保管できます。

表 15. エージェント・プロパティーの構成

プロパティー名	デフォルト値	タイプ	説明
IDLFileDirectory	なし	String	(必須) IDL ファイルが配置されているディレクトリー。CORBA インターフェースを定義する IDL ファイルはすべて、このディレクトリーに配置されている必要があります。

表 15. エージェント・プロパティーの構成 (続き)

プロパティー名	デフォルト値	タイプ	説明
GenerateProxies	True	Boolean	(必須) 設定を true にすると、必要なプロキシー・オブジェクト・クラスが生成されます。設定を false にすると、アダプターは、プロキシー・オブジェクト・クラスの .jar ファイルを生成しないで、CORBA アプリケーション・ベンダーの .jar ファイルを使用します。start_CORBA.bat ファイル (Windows) または start_CORBA.sh ファイル (Unix) 内の JCLASSES 設定に、CORBA アプリケーション・ベンダーのファイル名を必ず指定します。 サーバーとして機能するコネクターのサーバー・サイド・オブジェクトを作成する場合、このプロパティーは無視され、ODA はサーバー・インプリメンテーション・クラスを生成します。これによって、ユーザーが独自のプロキシー・クラス定義を生成するのではなく、CORBA アプリケーション・ベンダーの .jar ファイル内にあるプロキシー・クラス定義を使用することができます。
JarFileName	なし	String	ODA によって生成されるクラスが格納される .jar ファイルの名前。(パスなしで) ファイル名のみを指定すると、ODA では、このファイルを出力するディレクトリーとして、OutputFileDir に指定した値が使用されます。絶対パス (ディレクトリーと .jar ファイル名) をこのプロパティーに指定すると、ODA では、OutputFileDir に指定した値が無視されます。 GenerateProxies プロパティーを true に設定する場合に、このプロパティーを指定する必要があります。
BOPrefix	なし	String	ODA によって生成されたビジネス・オブジェクト名に追加するプレフィックス。 60 ページの図 15 では、ODA によって生成されたすべてのビジネス・オブジェクト名が、myBO_ で始まっています。
CORBAServerImpl	False	Boolean	(必須) true に設定すると、コネクターがサーバーとして機能する場合にサーバー・サイド処理に使用されるサーバー・サイド・ビジネス・オブジェクトと実装クラス定義が、ODA により生成されます。同じ CORBA IDL オブジェクトに対してクライアント・サイドのビジネス・オブジェクトを生成するには、このプロパティーを false に設定して ODA をもう一度実行します。
IDLToJavaTool	なし	String	IDLJ コンパイラー・ツールの絶対パス。

表 15. エージェント・プロパティーの構成 (続き)

プロパティー名	デフォルト値	タイプ	説明
CORBAClassPath	なし	String	(オプション) セミコロン (Windows) またはコロン (UNIX) で区切られたストリング。外部 CORBA インフラストラクチャー・クラス・ファイルのパス名を含みます。このような .jar ファイルは、プロキシー・クラス定義を正常にコンパイルするために必要です。このプロパティーの値は、ODA を実行すると必ず、ODA によって使用されるクラス・パスに一時的に追加されます。すべての汎用 CORBA クラス、例えば、ibmorb.jar (IBM Java ORB によって要求される jar ファイル) に定義されているクラスを始めとし、CORBA クラスのコンパイルに必要な JAR ファイルまたは ディレクトリーのすべてを組み込むように、この値を明示的に設定します。
OutputFileDir	なし	String	(必須) ODA により生成されたすべての出力ファイルを保管するディレクトリー。 GenerateProxies プロパティーに false を設定すると、ODA では、ODA 生成クラスが格納される JAR ファイル (JarFileName プロパティー) が作成されません。その代わり、クラスは、ここで指定したディレクトリーに、別々の出力ファイルとして保管されます。これは、作業ディレクトリーであるため、予防措置として、ODA を実行する前に必ず、ここで指定したディレクトリーにファイルが一切格納されていないことを確認してください。
JDKHomeDir	なし	String	(必須) JDK のインストール先となるこのマシン上のディレクトリー。
TraceFileName	なし	String	トレース・メッセージ・ファイルの名前。 CORBAODAtrace.txt など。
TraceLevel	5	Integer	(必須) エージェントのトレース・レベル (0 から 5 まで)。トレース・レベルの詳細については、 75 ページの『トレース』を参照してください。
MessageFile	なし	String	(必須) ODA によって表示されるすべてのメッセージを含むメッセージ・ファイルの名前。CORBA の場合、このファイルの名前は、BIA_CORBAODAAgent.txt になります。メッセージ・ファイルの名前を適切に指定しないと、ODA はメッセージなしで実行されます。

1. ODA により新規プロファイルを作成する場合には、「プロファイル」グループ・ボックスの**「新規」**ボタンと**「保管」**ボタンを使用します。次回以降は、既存のプロファイルを選択できます。
2. 60 ページの表 15 の説明に従い、各プロパティーの値を入力します。

   注: プロファイルを使用する場合は、プロパティー値が自動的に入力されます。必要に応じてこの値を変更することができます。新しい値を保管することもできます。

ビジネス・オブジェクトの選択

63 ページの図 16 のような、「*ビジネス・オブジェクト・ウィザード - ステップ 3/6 - ソースの選択*」画面が表示されます。この画面には、CORBA IDL ファイルで

定義した interfaces または structs がリストされます。この画面を使用して、ODA によるビジネス・オブジェクト定義の生成対象となる CORBA エンティティーの数を選択します。上位の親エンティティーは、常に interface または struct です。上位親のサブオブジェクトは、interface、struct、union、enum、または sequence です。下位の union、enum、または sequence は、その上位 (親またはより上位のレベル) の interface または struct を選択すると、自動的にビジネス・オブジェクトとして生成されます。

注: 下位の interface または struct が生成されるのは、それを明示的に選択する場合のみです。つまり、上位が選択されているというだけで、自動的には生成されません。

この画面にリストされている CORBA オブジェクトのうち、どれが上位オブジェクトの子オブジェクトであるかを判別するには、元の IDL ファイルを参照します。この画面に表示されている CORBA オブジェクトをすべて選択し、対応するビジネス・オブジェクトを生成することもできます。生成されたビジネス・オブジェクトには、親子関係が反映されます。

ビジネス・オブジェクトの作成対象となる CORBA 構造の詳細については、41 ページの表 7 を参照してください。

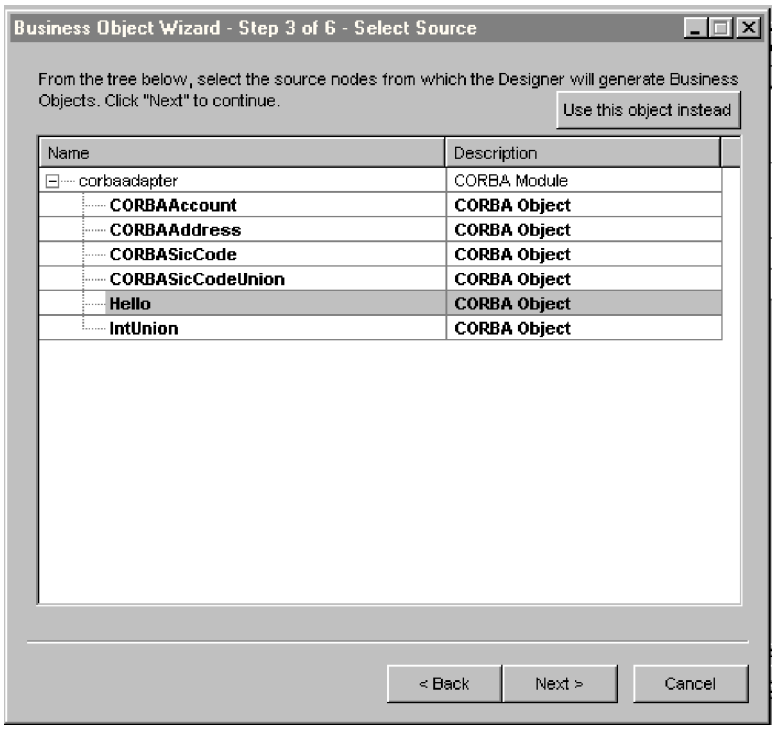


図 16. 「ソースの選択」画面

1. 必要に応じて CORBA モジュールを展開し、サブオブジェクトのリストを表示します。
2. 使用する CORBA オブジェクトを選択します。図 16 では、Hello オブジェクトが選択されています。
3. **「次へ」**をクリックします。

オブジェクト選択の確認

「*ビジネス・オブジェクト・ウィザード - ステップ 4/6 - ビジネス・オブジェクト定義のソース・ノードの確認*」画面が表示されます。この画面には、選択したオブジェクトが表示されます。

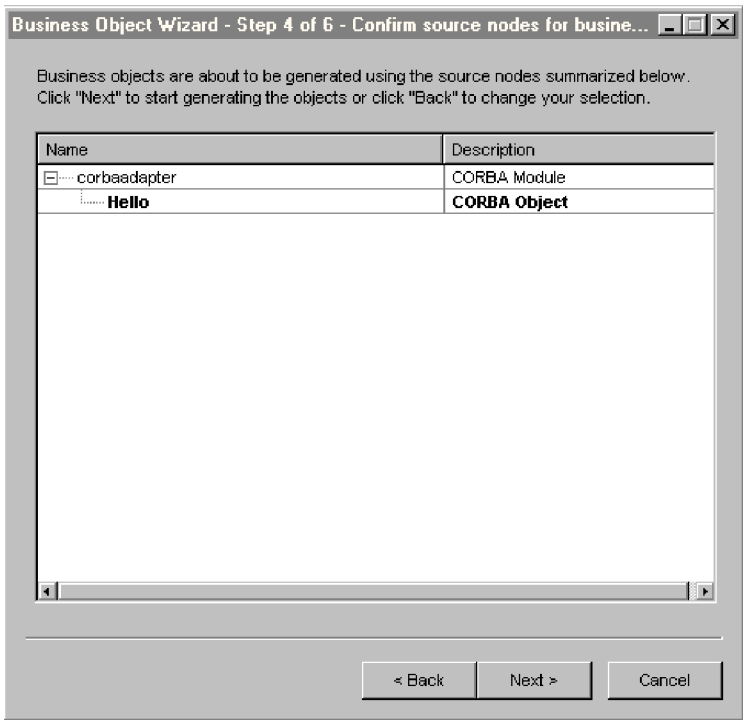


図 17. 「ソース・ノードの確認」画面

「戻る」をクリックして変更を加えるか、**「次へ」**をクリックしてリストが正しいことを確認します。

「*ビジネス・オブジェクト・ウィザード - ステップ 5/6 - ビジネス・オブジェクトの生成中...*」画面が表示されます。画面には、ビジネス・オブジェクトが生成中であることを示すメッセージが表示されます。

ビジネス・オブジェクト情報の指定

ビジネス・オブジェクトを作成したら、そのオブジェクトに対して有効な動詞、オブジェクトで指定した動詞のメソッド・シーケンス、ビジネス・オブジェクト・レベル ASI、および属性レベル ASI を指定します。このセクションでは、これらの情報を Business Object Designer と ODA を使用して指定する方法について説明します。情報のカテゴリーおよび CORBA コネクターのビジネス・オブジェクト構造について詳しくは、39 ページの『第 4 章 ビジネス・オブジェクトについて』を参照してください。

動詞の選択

ビジネス・オブジェクトの作成後、作成したオブジェクトを個別のウィンドウで開いたときに表示される Business Object Designer の最初の画面は、「BO プロパティ

一 - コンポーネントの動詞の選択 (BO Properties - Select Verbs for component)」画面です。図 18 に、 63 ページの図 16 および 64 ページの図 17 で Hello ビジネス・オブジェクトを作成した場合の画面を示します。

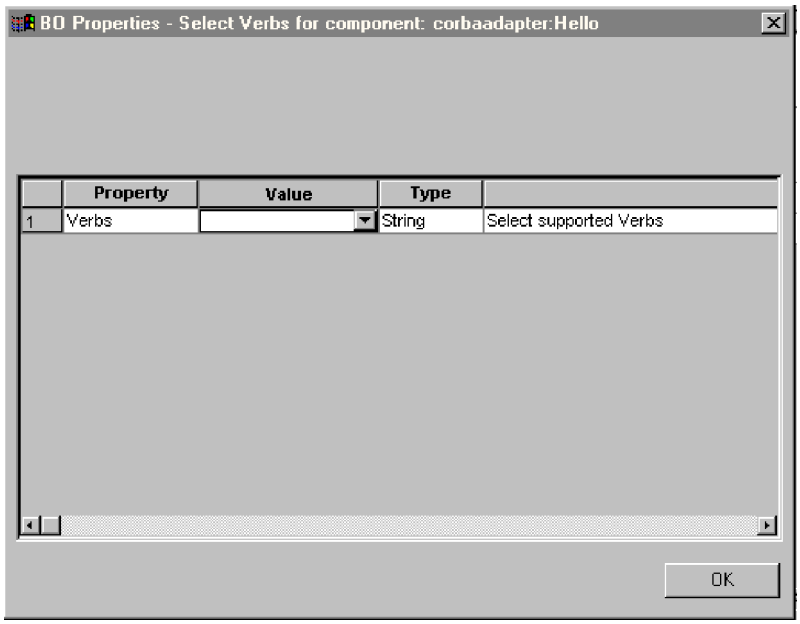
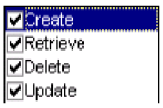


図 18. コンポーネントの動詞選択画面

この画面では、ビジネス・オブジェクトがサポートする動詞を指定します。ODA を使用して、サポートされている 4 つの動詞 (Create、Retrieve、Delete、および Update) を指定し、各動詞にアクションとして n + 2 個のメソッド (n は対応する CORBA インターフェースのメソッド数) を割り当てることができます。 2 つの追加メソッドは、コネクターによりサポートされているメソッド `LoadFromProxy` および `WriteToProxy` です。サポートされている 4 つの動詞以外の動詞を指定する場合、あるいは、ビジネス・オブジェクトの作成後に動詞情報を編集する場合は、Business Object Designer を使用します。

CORBA コネクターのビジネス・オブジェクト動詞の詳細については、 45 ページの『動詞 ASI』を参照してください。

1. Verbs プロパティーの**「値」**リストから、ビジネス・オブジェクトでサポートする動詞を選択します。1 つ以上の動詞を選択してください。選択した動詞はいつでも選択解除できます。



2. **「OK」**をクリックします。

動詞 ASI の指定

64 ページの『動詞の選択』のステップ 1 (65 ページ) で選択した動詞ごとに、個別ウィンドウが表示されます。このウィンドウを使用して、動詞に対して実行するメソッド・シーケンス指定します。

図 19 に、 63 ページの図 16 および 64 ページの図 17 で Hello ビジネス・オブジェクトの Retrieve 動詞を作成した場合の画面を示します。

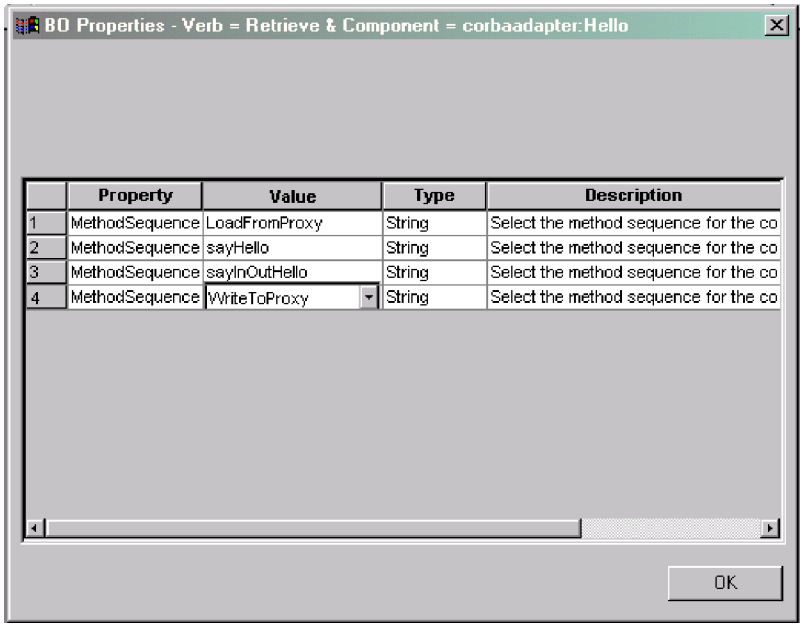


図 19. 動詞メソッド・シーケンスの設定

1. MethodSequence プロパティーの**「値」**リストから、ビジネス・オブジェクトが動詞に対して最初に実行するメソッドを選択します。図 19 でのメソッド・シーケンスは、以下のとおりです。
   - Retrieve 動詞に対するメソッド・シーケンスで最初に実行されるメソッドは、LoadFromProxy です。
   - シーケンスの 2 番目メソッドは、sayHello です。
   - シーケンスの 3 番目メソッドは、sayInOutHello です。
   - シーケンスの最後のメソッドは、WriteToProxy です。

   これらのメソッドは、IDL ファイルで定義される CORBA インターフェースによって提供されます。ただし、LoadFromProxy および WriteToProxy の 2 つのメソッドは ODA によって提供されます。

   動詞に対してメソッド・シーケンスを指定することにより、その動詞に関連した動詞 ASI を作成します。この動詞 ASI は、必要な場合、後で変更することができます。
2. **「OK」**をクリックします。

CORBA 動詞 ASI がサポートするキーワードのリストについては、 46 ページの表 10 を参照してください。

別のウィンドウでビジネス・オブジェクトを開く

「*ビジネス・オブジェクト・ウィザード - ステップ 6/6 - ビジネス・オブジェクトの保管*」画面が表示されます。

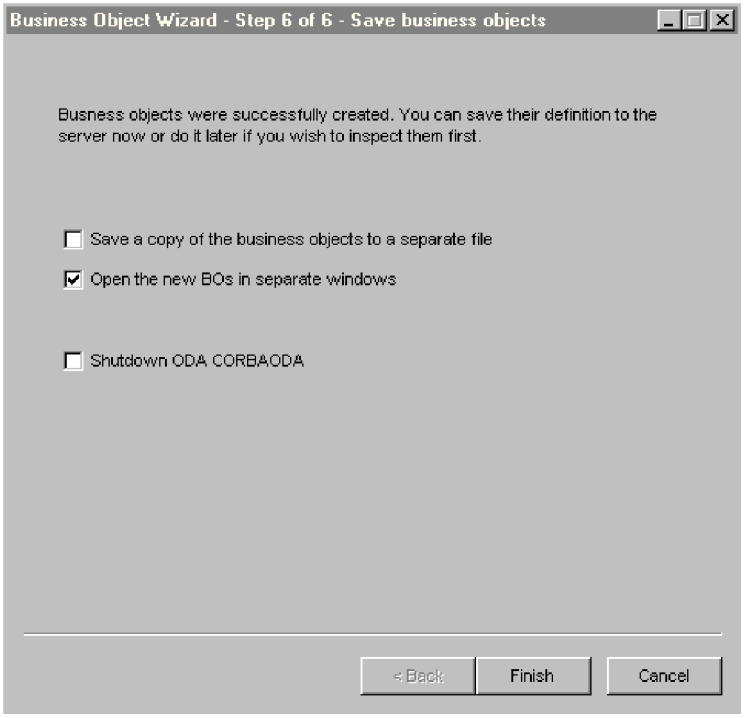


図 20. 「ビジネス・オブジェクトの保管」画面

Business Object Designer では、個別のウィンドウでビジネス・オブジェクトを開くことができます。また、最上位ビジネス・オブジェクトのキーを指定した後に、生成されたビジネス・オブジェクトの定義をファイルに保存できます。

別のウィンドウでビジネス・オブジェクトを開くには、以下のステップを実行します。

1. **「別のウィンドウで新規ビジネス・オブジェクトを開く」**を選択します。ダイアログ・ボックスが表示されます。
2. **「完了」**をクリックします。ビジネス・オブジェクトが別のウィンドウに表示されます。このウィンドウでは、ビジネス・オブジェクトや作成したビジネス・オブジェクト動詞の ASI 情報を参照および設定できます。詳細については、 64 ページの『ビジネス・オブジェクト情報の指定』を参照してください。

ビジネス・オブジェクトをファイルに保管するには、(親レベルのビジネス・オブジェクトのキーを指定した後で) 以下の手順を実行します。

1. **「ビジネス・オブジェクトのコピーを個別のファイルに保管する」**を選択します。ダイアログ・ボックスが表示されます。

2. 新しいビジネス・オブジェクト定義のコピーを保管するロケーションを入力します。

Business Object Designer が指定したロケーションにファイルを保管します。

ODA を使用した作業が完了したら、**「完了」**をクリックする前に、「ODA CORBA ODA をシャットダウン (Shutdown ODA CORBA ODA)」にチェックマークを付けて ODA をシャットダウンします。

属性レベル ASI の指定

各動詞に対して実行するメソッド・シーケンスを指定して、動詞 ASI を定義すると、Business Object Designer にビジネス・オブジェクトの属性が表示されます。CORBA コネクターの属性レベル ASI の詳細については、47 ページの『属性レベル ASI』を参照してください。

属性は、**「位置」**列で定義した数値に従い、ビジネス・オブジェクト構造で表示される順序で**「属性」**タブにリストされます。単一 CORBA オブジェクト属性は単純属性として示され、その ASI には元の CORBA 属性名およびタイプが含まれます。

画面には、属性ごとに属性名、属性タイプ、および ASI 情報が表示されます。ビジネス・オブジェクトの `sayHello` 属性には、元の CORBA IDL メソッド名に属性をマップする ASI が含まれます。この例では、**「アプリケーション固有の情報」**列に、元のメソッド名が `method_name=sayHello` ASI 別に示されています。

「属性」タブでは、ODA によってまだキーが指定されていない各ビジネス・オブジェクトに対して (ビジネス・オブジェクトの妥当性検査、保管を行うために Business Object Designer が必要とする) キーを指定する必要があります。特定の CORBA 型 (例えば、`CORBA_Short`、`CORBA_Boolean`、および `CORBA_Char`) に対しては、ODA がキーを設定することに注意してください。他の CORBA 型に対しては、ユーザーがキーを設定する必要があります。

この画面では、必要に応じて子オブジェクト・キーを設定し、以下の情報を指定できます。

- ビジネス・オブジェクトを処理するコネクターに属性が必要かどうか。必要な場合は、**「必要」**チェック・ボックスにチェックマークを付けます。
- 属性の最大長が**「最大長」**列の値と異なるかどうか。
- 属性のデフォルト値。ある場合は**「デフォルト」**列に値を入力します。

注: ODA (Business Object Designer で実行) を使用してビジネス・オブジェクトを新規に作成できますが、この方法で外部キーを構成することはできません。外部キーは非 ASI メタデータであるため、通常は、ODA を*使用せずに* 構成する必要があります。Business Object Designer の **「ファイル」** > **「新規」**をクリックし、ODA を使用せずに新しいビジネス・オブジェクトを作成します)。

ビジネス・オブジェクト・レベル ASI の指定

属性レベル ASI を指定すると、ビジネス・オブジェクト・レベル ASI を表示および変更できるようになります。ビジネス・オブジェクト・レベル ASI の詳細については、43 ページの『ビジネス・オブジェクト・レベルの ASI』を参照してください。

ビジネス・オブジェクト・レベル ASI は、**「一般」**タブにリストされます。**「ビジネス・オブジェクト・レベル・アプリケーション固有の情報」**フィールドに表示される ASI 値には、このビジネス・オブジェクトを表すプロキシー・クラスの名前が含まれます。コネクターは、この情報を使用してプロキシー・クラスをビジネス・オブジェクトにマップします。また、サーバー・サイド・ビジネス・オブジェクトの場合 (コネクターがサーバーとしても機能する場合)、コネクターはこの情報を使用して、インプリメンテーション・クラスをビジネス・オブジェクトにマップします。

この画面には、ビジネス・オブジェクトがサポートする動詞もすべてリストされます。また、66 ページの『動詞 ASI の指定』で定義したような、動詞ごとの ASI も提供されます。

図 21 に、Hello ビジネス・オブジェクトのビジネス・オブジェクト・レベル ASI を示します。このビジネス・オブジェクトのメソッド・シーケンスを実行する動詞は Retrieve のみです。 Retrieve には、以下に示すような (66 ページの図 19 で最初に指定した) メソッド・シーケンスを持つ動詞 ASI が含まれています。

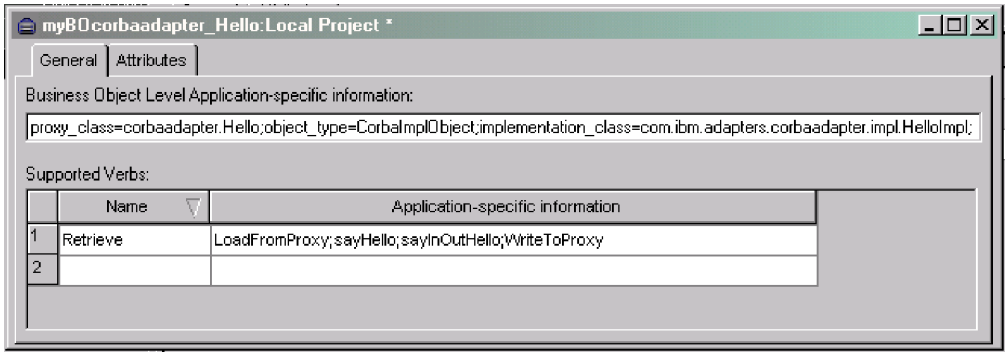


図 21. ビジネス・オブジェクト・レベル ASI の設定

この画面では、ビジネス・オブジェクトの ASI およびサポートされる動詞を変更できます。

ビジネス・オブジェクト・ファイルのアップロード

新たに作成するビジネス・オブジェクト定義ファイルは、作成後に、統合ブローカーにアップロードする必要があります。アップロード・プロセスは、WebSphere InterChange Server、WebSphere MQ Integrator Broker、WebSphere Application Server のいずれを実行しているかによって異なります。

- **WebSphere InterChange Server** を実行している場合: ビジネス・オブジェクト定義ファイルをローカル・マシンに保管しており、ファイルをサーバーのリポジトリーにアップロードする必要がある場合は、「*WebSphere InterChange Server システム・インプリメンテーション・ガイド*」を参照してください。
- **WebSphere MQ Integrator Broker** を実行している場合: ビジネス・オブジェクト定義を Business Object Designer から統合ブローカーにエクスポートする必要があります。詳細については、「*WebSphere MQ Integrator Broker 用インプリメンテーション・ガイド*」を参照してください。
- **WebSphere Application Server** を実行している場合: 詳細については、「*アダプター実装ガイド (WebSphere Application Server)*」を参照してください。

第 6 章 トラブルシューティングおよびエラー処理

本章では、Adapter for CORBA によるエラーの処理方法について説明します。このアダプターによって、ロギング・メッセージおよびトレース・メッセージが生成されます。本章では、これらのメッセージと、トラブルシューティングのヒントについて説明します。本章の内容は、次のとおりです。

- 『エラー処理』
- 75 ページの『トラブルシューティングのヒント』
- 75 ページの『ロギング』
- 75 ページの『トレース』

エラー処理

コネクターが生成するすべてのメッセージは、BIA_CORBAConnector.txt という名前のメッセージ・ファイルに保管されます(ファイル名は、LogFileName 標準コネクター構成プロパティーによって決定されます)。各メッセージには、メッセージ番号が付けられています。メッセージ番号は、以下のように、メッセージの次に表示されます。

```
Message number
Message text
```

コネクターは、以下のセクションで説明する追加の特定のエラーを処理します。

コネクターのエラー処理

CORBA 例外

CORBA アプリケーションがダウンした場合や、CORBA 呼び出しが失敗した場合、コネクターは CORBA 例外をスローします。

コネクターは、エラーをログに記録して FAIL コードを戻すことにより、このような例外を処理します。デバッグ時に役立つように、コネクターは CORBA 例外の詳細を記録し、それを VerbProcessingFailed 例外のメッセージ・フィールドに戻します。例外には、シーケンスにおける呼び出し失敗に関する情報も含まれます。

プロキシーにおける ClassNotFound

ローダーがプロキシー・クラス名を受け取り、そのクラスのプロキシー・オブジェクトを作成しようとする際、クラスが見つからないと例外が発生します。コネクターはエラーをログに記録し、FAIL コードを戻します。ログには、見つからなかったクラス名が記載されます。

ローダーにおける InstantiationException

ローダーがプロキシー・クラス名を受け取り、そのクラスのプロキシー・オブジェクトを作成しようとする際、オブジェクト・インスタンスを作成できないと例外が発生します。コネクターはエラーをログに記録し、FAIL コードを戻します。ログには、インスタンス生成できなかったオブジェクトのクラス名が記載されます。

ローダーまたは起動側における IIlegal AccessException

コードが無効であったり、IDLJ コンパイラー・ツールがメソッドに対して不適切にアクセス (パブリックまたはプライベート) したりすると、コネクターは例外を発生します。

コネクターはエラーをログに記録し、FAIL コードを戻します。

起動側における NoSuchMethodException

対応するプロキシー・オブジェクトに存在しないビジネス・オブジェクトに対してメソッドが指定されると、コネクターは例外を発生します。

コネクターはエラーをログに記録し、FAIL コードを戻します。

起動側における InvocationTargetException

CORBA アプリケーション (コネクターがビジネス・オブジェクトを交換する対象) で例外が発生すると、コネクターは例外を発生します。

コネクターはエラーをログに記録し、FAIL コードを戻します。

起動側における InParameter オブジェクトの無効な引き数 (CXIgnore)

ビジネス・オブジェクトの動詞 ASI にメソッドは含まれているものの、メソッドの引き数が設定されていない場合、コネクターは例外を発生します。ビジネス・オブジェクト構造および動詞 ASI の詳細については、39 ページの『第 4 章 ビジネス・オブジェクトについて』を参照してください。

コネクターはエラーをログに記録し、FAIL コードを戻します。

キャストの失敗または属性タイプの間違い

プロキシー・オブジェクト・メソッドが、ビジネス・オブジェクトで指定したものと異なるデータ型を実行または戻すと、コネクターは例外を発生します。

コネクターはエラーをログに記録し、FAIL コードを戻します。

無効な動詞 ASI

コネクターに渡されるビジネス・オブジェクトの動詞 ASI が、不適切にフォーマット設定されていたり、不適切な構文を使用していたりすると、コネクターは例外を発生します。例えば、適切なメソッド・シーケンスを含まない動詞 ASIや、アクティブな動詞対して CBOH (カスタム BO ハンドラー) を指定する子ビジネス・オブジェクトなどです。

コネクターはエラーをログに記録し、FAIL コードを戻します。

ODA のエラー処理

バッファーありリーダーで生成されたエラー: {1}

javac コンパイラーでエラーが発生した場合に、この例外が発生します。エラーの説明は {1} に示されます。

ユーザーが選択したコンポーネントまたはメソッドはありません。

ユーザーが、対応するビジネス・オブジェクトを作成する際に Business Object Designer で CORBA メソッドまたはコンポーネントを選択していない場合に、ODA がこの例外を発生します。

{1} でエラーが発生しました: {2}

ODA ルーチンで予期しないエラーが発生した場合に、この例外が発生します。エラーが発生したモジュールまたは関数は {1} に示され、エラーの説明は {2} に示されます。

指定のディレクトリー {1} に Java ファイルはありません。

IDLJ コンパイラー・ツールが、Java プロキシー・ファイルのディレクトリーにいかなるファイルも出力しない場合、ODA がこの例外を発生します。ディレクトリーの名前は {1} に示されます。このディレクトリーは、ODA Configure Agent の OutputFileDir プロパティーで指定されます。このプロパティーの詳細については、60 ページの表 15 を参照してください。

指定のディレクトリー {1} が存在しません。

ユーザーが ODA Configure Agent の OutputFileDir プロパティーに値を指定するが、IDLJ コンパイラー・ツールによるプロキシー .jar ファイルの出力先であるマシンにディレクトリーが存在しない場合、ODA がこの例外を発生します。ディレクトリーの名前は {1} に示されます。このプロパティーの詳細については、60 ページの表 15 を参照してください。

{1} 属性の IDL タイプ情報を取得できませんでした。

ODA が IDL ファイルに指定された CORBA 属性のデータ型を判別できない場合、この例外を発生します。属性の名前は {1} に示されます。CORBA 属性とビジネス・オブジェクト属性は同じ名前です (ただし、ビジネス・オブジェクト名に追加されている接頭部は除きます)。

{1}: IDL コンパイラーへのパスが誤っています: {2} ({1}: Path to IDL Compiler is incorrect: {2})

Configure Agent の IDLToJavaTool プロパティーに誤ったパス名が指定された場合、ODA がこの例外を発生します。このプロパティーの詳細については、60 ページの表 15 を参照してください。エラーが発生したモジュールまたは関数は {1} に示され、誤ったパス名は {2} に示されます。

{1}: IDL コンパイラーへのパスが指定されていません。({1}: Path to IDL Compiler is unspecified)

Configure Agent の IDLToJavaTool プロパティーにパス名が指定されていない場合、ODA がこの例外を発生します。このプロパティーの詳細については、60 ページの表 15 を参照してください。エラーが発生したモジュールまたは関数は {1} に示されます。

{1}: IDLJ を実行できませんでした。({1}: Could not execute IDLJ.) PATH に指定されているか確認してください。(Make sure it is in your PATH)

ODA が IDLJ コンパイラー・ツールを実行できない場合、この例外を発生します。エラーが発生したモジュールまたは関数は {1} に示されます。

IDLJ コンパイラー・ツールを格納しているディレクトリーが、システム・パスに指定されているか、あるいは IDLToJavaTool プロパティーに、IDLJ コンパイラー・ツールの絶対パスが指定されているかを確認します。このプロパティーの詳細については、60 ページの表 15 を参照してください。

無効な入力ファイルが見つかりました: <Type>Operations.java が必要です。

ODA が IDL インターフェースに対応するビジネス・オブジェクトの生成を試行したが、特定の Java ファイルを検出できない場合に、この例外が発生します。

ベース・ディレクトリーを作成できません。

ODA が何らかの理由で、(サーバーとして機能するコネクターで処理される) サーバー・サイド・ビジネス・オブジェクトに使用されるインプリメンテーション・クラスのディレクトリーを作成できない場合、この例外を発生します。

ディレクトリーが見つからず、絶対パスではありません。

いずれかの Configure Agent プロパティーに絶対ディレクトリー・パスではなく相対パスが指定された場合、ODA がこの例外を発生します。特定のドライブおよびディレクトリーへの絶対パスを指定する必要があります。

Configure Agent プロパティーの詳細については、60 ページの表 15 を参照してください。

ディレクトリー {1} を作成できません。(Unable to create directory {1})

ODA が何らかの理由で、ビジネス・オブジェクトの出力ファイル用の出力ディレクトリーを作成できない場合、この例外を発生します。ディレクトリーの名前は {1} に示されます。

{1}: ファイル {3} に IDL タイプ {2} が見つかりました。({1}: Found IDL type of ″{2} ″ in file {3}.) これは既知の IDL タイプではありません。(This is not a known IDL type.) IDL タイプを判別できませんでした。(Could not determine IDL type)

IDL ファイルに未知のタイプが存在する場合、ODA がこの例外を発生します。エラーが発生したモジュールまたは関数は {1} に示され、未知のデータ型の名前は {2} に示され、IDL ファイル名は {3} に示されます。

通常、この例外が発生するのは、IBM Java ORB の IDLJ コンパイラー・ツール (コネクターが現在サポートする唯一の Java プロキシー・コンパイラーの IDL) 以外の Java プロキシー・コンパイラーを使用している場合です。

トラブルシューティングのヒント

問題	考えられる解決方法/説明
ODA の「ソースの選択」画面 (63 ページの図 16 を参照) に、ビジネス・オブジェクトの生成のために選択する CORBA オブジェクトが 1 つも表示されません。	使用している Java プロキシー・クラス・コンパイラーの IDL を検査し、それがコネクターでサポートされることを確認してください。現在、コネクターは、IBM Java ORB の IDLJ コンパイラー・ツールをサポートしています。IDLJ コンパイラー・ツールのパス名が、Configure Agent の `IDLToJavaTool` プロパティーに正しく指定されていることを確認してください。このプロパティーの詳細については、 60 ページの表 15 を参照してください。
IDL のコンパイル時に、CORBA `fixed` データ型が検出されたため、 74 ページの『{1}: ファイル {3} に IDL タイプ {2} が見つかりました。({1}: Found IDL type of ″{2}″ in file {3}.) これは既知の IDL タイプではありません。(This is not a known IDL type.) IDL タイプを判別できませんでした。(Could not determine IDL type)』というメッセージが表示されます。	IBM IDLJ コンパイラー・ツールは、CORBA `fixed` データ型をサポートしません。そのため、ODA も、`fixed` データ型を表す属性のビジネス・オブジェクトの生成をサポートしていません。使用しているコンパイラーの IDL を検査し、それがコネクターでサポートされることを確認してください。現在、コネクターは、IBM IDLJ コンパイラー・ツールをサポートしています。
ODA が、何らかの理由で CORBA IDL ファイルから Java プロキシー・ファイルを生成できませんでした。	ODA 外部で IDLJ コンパイラー・ツール (`idlj.exe` (Windows 版)、`idlj` (UNIX 版)) を手動で実行し、`javac` を使用して Java ファイルを手動でコンパイルします。

ロギング

71 ページの『エラー処理』に説明されたすべてのメッセージをメッセージ・ファイル (`BIA_CORBAConnector.txt`) から読み取る必要があります。

トレース

トレースはオプションのデバッグ機能であり、この機能をオンにするとコネクターの動作を密着して追跡できます。トレース・メッセージは、デフォルトでは STDOUT に書き込まれます。トレース・メッセージの構成について詳しくは、 23 ページの『コネクターの構成』のコネクター構成プロパティーを参照してください。トレースに関する情報、トレースを有効化して設定する方法について詳しくは、「*コネクター開発ガイド*」を参照してください。

表 16 に、お勧めするコネクター・トレース・メッセージの内容をレベル別に示します。

表 16. トレース・メッセージの内容

レベル	説明
レベル 0	コネクター・バージョンを示すトレース・メッセージには、このレベルを使用します。このレベルで実行されるトレースは他にありません。

表 16. トレース・メッセージの内容 (続き)

レベル	説明
レベル 1	このトレース・メッセージ・レベルでは、以下を実行します。 • 状況情報を提供します。 • 処理対象の各ビジネス・オブジェクトのキー情報を提供します。 • ポーリング・スレッドが入力キューで新規メッセージを検出するたびに記録を取ります。
レベル 2	このトレース・メッセージ・レベルでは、以下を実行します。 • コネクターが処理する各オブジェクトで使用する BO ハンドラーを識別します。 • ビジネス・オブジェクトが統合ブローカーに送られるたびにログに記録します。 • 要求ビジネス・オブジェクトを受け取るたびに指示を出します。
レベル 3	このトレース・メッセージ・レベルでは、以下を実行します。 • 処理対象の外部キーを識別します (該当する場合)。このようなメッセージは、コネクターがビジネス・オブジェクト内で外部キーを検出したり、コネクターがビジネス・オブジェクト内に外部キーを設定したりすると表示されます。 • ビジネス・オブジェクト処理に関連付けます。例えば、ビジネス・オブジェクト間の一致の検出や、子ビジネス・オブジェクトの配列におけるビジネス・オブジェクトの検出などです。
レベル 4	このトレース・メッセージ・レベルでは、以下を実行します。 • アプリケーション固有の情報を識別します。例えば、ビジネス・オブジェクト内のアプリケーション固有情報フィールドを処理するメソッドによって戻される値などです。 • コネクターがいつ関数を呼び出したか、または終了したかを識別します。このようなメッセージは、コネクターの処理フローをトレースするときに役立ちます。 • スレッド固有の処理を記録します。例えば、コネクターが複数のスレッドを作成した場合、新しいスレッドが作成されるたびにメッセージをログに記録します。
レベル 5	このトレース・メッセージ・レベルでは、以下を実行します。 • コネクターの初期化を指示します。このメッセージ・タイプには、例えば、ブローカーから検索された各コネクター構成プロパティーの値などがあります。 • コネクターが作成した各スレッドの、実行中の詳細状況を提供します。 • このアプリケーションで実行されるステートメントを表示します。該当する場合、宛先アプリケーションで実行されるすべてのステートメント、および置換されるすべての変数の値がコネクター・ログ・ファイルに記述されます。 • ビジネス・オブジェクト・ダンプを記録します。コネクターは、オブジェクト処理を開始する前と後にビジネス・オブジェクトのテキスト表記を出力します (処理開始前はコネクターがコラボレーションから受け取るオブジェクトが出力され、処理後はコネクターがコラボレーションに戻すビジネス・オブジェクトが出力されます)。

付録 A. コネクターの標準構成プロパティー

この付録では、WebSphere Business Integration アダプターのコネクター・コンポーネントの標準構成プロパティーについて説明します。この付録の内容は、次の統合ブローカーで実行されるコネクターを対象としています。

- WebSphere InterChange Server (ICS)
- WebSphere MQ Integrator、WebSphere MQ Integrator Broker、および WebSphere Business Integration Message Broker (WebSphere Message Brokers (WMQI) と総称)
- WebSphere Application Server (WAS)

コネクターによっては、一部の標準プロパティーが使用されないことがあります。Connector Configurator から統合ブローカーを選択すると、そのブローカーで実行されるアダプターについて構成する必要のある標準プロパティーのリストが表示されます。

コネクター固有のプロパティーの詳細については、該当するアダプターのユーザーズ・ガイドを参照してください。

注: 本書では、ディレクトリー・パスの規則として円記号 (¥) を使用します。UNIX システムを使用している場合は、円記号をスラッシュ (/) に置き換えてください。また、各オペレーティング・システムの規則に従ってください。

新規プロパティーと削除されたプロパティー

本リリースには、次の標準プロパティーが追加されました。

新規プロパティー

- XMLNameSpaceFormat

削除されたプロパティー

- RestartCount

標準コネクター・プロパティーの構成

アダプター・コネクターには 2 つのタイプの構成プロパティーがあります。

- 標準構成プロパティー
- コネクター固有の構成プロパティー

このセクションでは、標準構成プロパティーについて説明します。コネクター固有の構成プロパティーについては、該当するアダプターのユーザーズ・ガイドを参照してください。

Connector Configurator の使用

Connector Configurator からコネクター・プロパティーを構成します。Connector Configurator には、System Manager からアクセスします。Connector Configurator の使用法の詳細については、本書の Connector Configurator に関する付録を参照してください。

注: Connector Configurator と System Manager は、Windows システム上でのみ動作します。コネクターを UNIX システム上で稼働している場合でも、これらのツールがインストールされた Windows マシンが必要です。UNIX 上で動作するコネクターのコネクター・プロパティーを設定する場合は、Windows マシン上で System Manager を起動し、UNIX の統合ブローカーに接続してから、コネクター用の Connector Configurator を開く必要があります。

プロパティー値の設定と更新

プロパティー・フィールドのデフォルトの長さは 255 文字です。

コネクターは、以下の順序に従ってプロパティーの値を決定します (最も番号の大きい項目が他の項目よりも優先されます)。

1. デフォルト
2. リポジトリー (WebSphere InterChange Server が統合ブローカーである場合のみ)
3. ローカル構成ファイル
4. コマンド行

コネクターは、始動時に構成値を取得します。実行時セッション中に 1 つ以上のコネクター・プロパティーの値を変更する場合は、プロパティーの**更新メソッド**によって、変更を有効にする方法が決定されます。標準コネクター・プロパティーには、以下の 4 種類の更新メソッドがあります。

- **動的**
  変更を System Manager に保管すると、変更が即時に有効になります。例えば WebSphere Message Broker で稼動している場合など、コネクターがスタンドアロン・モードで (System Manager から独立して) 稼動している場合は、構成ファイルでのみプロパティーを変更できます。この場合、動的更新は実行できません。
- **エージェント再始動 (ICS のみ)**
  アプリケーション固有のコンポーネントを停止して再始動しなければ、変更が有効になりません。
- **コンポーネント再始動**
  System Manager でコネクターを停止してから再始動しなければ、変更が有効になりません。アプリケーション固有コンポーネントまたは統合ブローカーを停止、再始動する必要はありません。
- **サーバー再始動**
  アプリケーション固有のコンポーネントおよび統合ブローカーを停止して再始動しなければ、変更が有効になりません。

特定のプロパティーの更新方法を確認するには、「Connector Configurator」ウィンドウ内の**「更新メソッド」**列を参照するか、次に示す 79 ページの表 17 の「更新メソッド」列を参照してください。

標準プロパティーの要約

表 17 は、標準コネクター構成プロパティーの早見表です。標準プロパティーの依存関係は RepositoryDirectory に基づいているため、コネクターによっては使用されないプロパティーがあり、使用する統合ブローカーによってプロパティーの設定が異なる可能性があります。

コネクターを実行する前に、これらのプロパティーの一部の値を設定する必要があります。各プロパティーの詳細については、次のセクションを参照してください。

注: 表 17 の「注」列にある「Repository Directory は REMOTE」という句は、ブローカーが InterChange Server であることを示します。ブローカーが WMQI または WAS の場合には、リポジトリー・ディレクトリーは LOCAL に設定されます。

表 17. 標準構成プロパティーの要約

プロパティー名	指定可能な値	デフォルト値	更新メソッド	注
AdminInQueue	有効な JMS キュー名	*CONNECTORNAME* /ADMININQUEUE	コンポーネント再始動	Delivery Transport は JMS
AdminOutQueue	有効な JMS キュー名	*CONNECTORNAME*/ADMINOUTQUEUE	コンポーネント再始動	Delivery Transport は JMS
AgentConnections	1 から 4	1	コンポーネント再始動	Delivery Transport は MQ および IDL: Repository Directory は <REMOTE> (ブローカーは ICS)
AgentTraceLevel	0 から 5	0	動的	
ApplicationName	アプリケーション名	コネクター・アプリケーション名として指定された値	コンポーネント再始動	
BrokerType	ICS、WMQI、WAS		コンポーネント再始動	
CharacterEncoding	ascii7、ascii8、SJIS、Cp949、GBK、Big5、Cp297、Cp273、Cp280、Cp284、Cp037、Cp437 **注:** これは、サポートされる値の一部です。	ascii7	コンポーネント再始動	
ConcurrentEventTriggeredFlows	1 から 32,767	1	コンポーネント再始動	Repository Directory は <REMOTE> (ブローカーは ICS)
ContainerManagedEvents	値なし、または JMS	値なし	コンポーネント再始動	Delivery Transport は JMS

表 17. 標準構成プロパティーの要約 (続き)

プロパティー名	指定可能な値	デフォルト値	更新メソッド	注
ControllerStoreAndForwardMode	true または false	true	動的	Repository Directory は <REMOTE> (ブローカーは ICS)
ControllerTraceLevel	0 から 5	0	動的	Repository Directory は <REMOTE> (ブローカーは ICS)
DeliveryQueue		*CONNECTORNAME*/DELIVERYQUEUE	コンポーネント再始動	JMS トランスポートのみ
DeliveryTransport	MQ、IDL、または JMS	JMS	コンポーネント再始動	Repository Directory がローカルの場合は、値は JMS のみ
DuplicateEventElimination	true または false	false	コンポーネント再始動	JMS トランスポートのみ: Container Managed Events は <NONE> でなければならない
FaultQueue		*CONNECTORNAME*/FAULTQUEUE	コンポーネント再始動	JMS トランスポートのみ
jms.FactoryClassName	CxCommon.Messaging.jms.IBMMQSeriesFactory または CxCommon.Messaging.jms.SonicMQFactory または任意の Java クラス名	CxCommon.Messaging.jms.IBMMQSeriesFactory	コンポーネント再始動	JMS トランスポートのみ
jms.MessageBrokerName	FactoryClassName が IBM の場合は crossworlds.queue.manager を使用。FactoryClassName が Sonicの場合 localhost:2506 を使用。	crossworlds.queue.manager	コンポーネント再始動	JMS トランスポートのみ
jms.NumConcurrentRequests	正整数	10	コンポーネント再始動	JMS トランスポートのみ
jms.Password	任意の有効なパスワード		コンポーネント再始動	JMS トランスポートのみ
jms.UserName	任意の有効な名前		コンポーネント再始動	JMS トランスポートのみ

表 17. 標準構成プロパティーの要約 (続き)

プロパティー名	指定可能な値	デフォルト値	更新メソッド	注
JvmMaxHeapSize	ヒープ・サイズ (メガバイト単位)	128m	コンポーネント再始動	Repository Directory は <REMOTE> (ブローカーは ICS)
JvmMaxNativeStackSize	スタックのサイズ (キロバイト単位)	128k	コンポーネント再始動	Repository Directory は <REMOTE> (ブローカーは ICS)
JvmMinHeapSize	ヒープ・サイズ (メガバイト単位)	1m	コンポーネント再始動	Repository Directory は <REMOTE> (ブローカーは ICS)
ListenerConcurrency	1 から 100	1	コンポーネント再始動	Delivery Transport は MQ でなければならない
Locale	en_US、ja_JP、ko_KR、zh_CN、zh_TW、fr_FR、de_DE、it_IT、es_ES、pt_BR **注:** これは、サポートされるロケールの一部です。	en_US	コンポーネント再始動	
LogAtInterchangeEnd	true または false	false	コンポーネント再始動	Repository Directory は <REMOTE> でなければならない (ブローカーは ICS)
MaxEventCapacity	1 から 2147483647	2147483647	動的	Repository Directory は <REMOTE> でなければならない (ブローカーは ICS)
MessageFileName	パスまたはファイル名	CONNECTORNAMEConnector.txt	コンポーネント再始動	
MonitorQueue	任意の有効なキュー名	*CONNECTORNAME*/MONITORQUEUE	コンポーネント再始動	JMS トランスポートのみ: DuplicateEvent Elimination は true でなければならない

表 17. 標準構成プロパティーの要約 (続き)

プロパティー名	指定可能な値	デフォルト値	更新メソッド	注
OADAutoRestartAgent	`true` または `false`	`false`	動的	Repository Directory は <REMOTE> でなければならない (ブローカーは ICS)
OADMaxNumRetry	正数	`1000`	動的	Repository Directory は <REMOTE> でなければならない (ブローカーは ICS)
OADRetryTimeInterval	正数 (単位: 分)	`10`	動的	Repository Directory は <REMOTE> でなければならない (ブローカーは ICS)
PollEndTime	`HH:MM`	`HH:MM`	コンポーネント再始動	
PollFrequency	正整数 (単位: ミリ秒) `no` (ポーリングを使用不可にする) `key` (コネクターのコマンド・プロンプト・ウィンドウで文字 `p` が入力された場合にのみポーリングする)	`10000`	動的	
PollQuantity	`1` から `500`	`1`	エージェント再始動	JMS トランスポートのみ: Container Managed Events を指定
PollStartTime	`HH:MM` (HH は 0 から 23、MM は 0 から 59)	`HH:MM`	コンポーネント再始動	
RepositoryDirectory	メタデータ・リポジトリーの場所		エージェント再始動	ICS の場合は <REMOTE> に設定する。WebSphere MQ Message Brokers および WAS の場合: C:¥crossworlds¥repository に設定する

表 17. 標準構成プロパティーの要約 (続き)

プロパティー名	指定可能な値	デフォルト値	更新メソッド	注
RequestQueue	有効な JMS キュー名	*CONNECTORNAME*/REQUESTQUEUE	コンポーネント再始動	Delivery Transport は JMS
ResponseQueue	有効な JMS キュー名	*CONNECTORNAME*/RESPONSEQUEUE	コンポーネント再始動	Delivery Transport が JMS の場合: Repository Directory が <REMOTE> の場合のみ必要
RestartRetryCount	0 から 99	3	動的	
RestartRetryInterval	適切な正数 (単位: 分): 1 から 2147483547	1	動的	
RHF2MessageDomain	mrm、xml	mrm	コンポーネント再始動	Delivery Transport が JMS であり、かつ WireFormat が CwXML である。
SourceQueue	有効な WebSphere MQ 名	*CONNECTORNAME*/SOURCEQUEUE	エージェント再始動	Delivery Transport が JMS であり、かつ Container Managed Events が指定されている場合のみ
SynchronousRequestQueue		*CONNECTORNAME*/SYNCHRONOUSREQUESTQUEUE	コンポーネント再始動	Delivery Transport は JMS
SynchronousRequestTimeout	0 以上の任意の数値 (ミリ秒)	0	コンポーネント再始動	Delivery Transport は JMS
SynchronousResponseQueue		*CONNECTORNAME*/SYNCHRONOUSRESPONSEQUEUE	コンポーネント再始動	Delivery Transport は JMS
WireFormat	CwXML、CwBO	CwXML	エージェント再始動	Repository Directory が <REMOTE> でない場合は CwXML。Repository Directory が <REMOTE> であれば CwBO
WsifSynchronousRequestTimeout	0 以上の任意の数値 (ミリ秒)	0	コンポーネント再始動	WAS のみ
XMLNameSpaceFormat	short、long	short	エージェント再始動	WebSphere MQ Message Brokers および WAS のみ

標準構成プロパティー

このセクションでは、各標準コネクター構成プロパティーの定義を示します。

AdminInQueue

統合ブローカーからコネクターへ管理メッセージが送信されるときに使用されるキューです。

デフォルト値は CONNECTORNAME/ADMININQUEUE です。

AdminOutQueue

コネクターから統合ブローカーへ管理メッセージが送信されるときに使用されるキューです。

デフォルト値は CONNECTORNAME/ADMINOUTQUEUE です。

AgentConnections

RepositoryDirectory が <REMOTE> の場合のみ適用可能です。

AgentConnections プロパティーは、orb.init[] により開かれる ORB (オブジェクト・リクエスト・ブローカー) 接続の数を制御します。

このプロパティーのデフォルト値は 1 に設定されます。必要に応じてこの値を変更できます。

AgentTraceLevel

アプリケーション固有のコンポーネントのトレース・メッセージのレベルです。デフォルト値は 0 です。コネクターは、設定されたトレース・レベル以下の該当するトレース・メッセージをすべてデリバリーします。

ApplicationName

コネクターのアプリケーションを一意的に特定する名前です。この名前は、システム管理者が WebSphere Business Integration システム環境をモニターするために使用されます。コネクターを実行する前に、このプロパティーに値を指定する必要があります。

BrokerType

使用する統合ブローカー・タイプを指定します。オプションは ICS、WebSphere Message Brokers (WMQI、WMQIB または WBIMB) または WAS です。

CharacterEncoding

文字 (アルファベットの文字、数値表現、句読記号など) から数値へのマッピングに使用する文字コード・セットを指定します。

注: Java ベースのコネクターでは、このプロパティーは使用しません。C++ ベースのコネクターでは、現在、このプロパティーに ascii7 という値が使用されています。

デフォルトでは、ドロップダウン・リストには、サポートされる文字エンコードの一部のみが表示されます。ドロップダウン・リストに、サポートされる他の値を追加するには、製品ディレクトリーにある ¥Data¥Std¥stdConnProps.xml ファイルを手動で変更する必要があります。詳細については、本書の Connector Configurator に関する付録を参照してください。

ConcurrentEventTriggeredFlows

RepositoryDirectory が <REMOTE> の場合のみ適用可能です。

コネクターがイベントのデリバリー時に並行処理できるビジネス・オブジェクトの数を決定します。この属性の値を、並行してマップおよびデリバリーできるビジネス・オブジェクトの数に設定します。例えば、この属性の値を 5 に設定すると、5 個のビジネス・オブジェクトが並行して処理されます。デフォルト値は 1 です。

このプロパティーを 1 よりも大きい値に設定すると、ソース・アプリケーションのコネクターが、複数のイベント・ビジネス・オブジェクトを同時にマップして、複数のコラボレーション・インスタンスにそれらのビジネス・オブジェクトを同時にデリバリーすることができます。これにより、統合ブローカーへのビジネス・オブジェクトのデリバリーにかかる時間、特にビジネス・オブジェクトが複雑なマップを使用している場合のデリバリー時間が短縮されます。ビジネス・オブジェクトのコラボレーションに到達する速度を増大させると、システム全体のパフォーマンスを向上させることができます。

ソース・アプリケーションから宛先アプリケーションまでのフロー全体に並行処理を実装するには、次のようにする必要があります。

- Maximum number of concurrent events プロパティーの値を増加して、コラボレーションが複数のスレッドを使用できるように構成します。
- 宛先アプリケーションのアプリケーション固有コンポーネントが複数の要求を並行して実行できることを確認します。つまり、このコンポーネントがマルチスレッド化されているか、またはコネクター・エージェント並列処理を使用でき、複数プロセスに対応するよう構成されている必要があります。Parallel Process Degree 構成プロパティーに、1 より大きい値を設定します。

ConcurrentEventTriggeredFlows プロパティーは、順次に実行される単一スレッド処理であるコネクターのポーリングでは無効です。

ContainerManagedEvents

このプロパティーにより、JMS イベント・ストアを使用する JMS 対応コネクターが、保証付きイベント・デリバリーを提供できるようになります。保証付きイベント・デリバリーでは、イベントはソース・キューから除去され、単一 JMS トランザクションとして宛先キューに配置されます。

デフォルト値はありません。

ContainerManagedEvents を JMS に設定した場合には、保証付きイベント・デリバリーを使用できるように次のプロパティーも構成する必要があります。

- PollQuantity = 1 から 500
- SourceQueue = /SOURCEQUEUE

また、MimeType、DHClass (データ・ハンドラー・クラス)、および DataHandlerConfigMOName (オプションのメタオブジェクト名) プロパティーを設定したデータ・ハンドラーも構成する必要があります。これらのプロパティーの値を設定するには、Connector Configurator の**「データ・ハンドラー」**タブを使用します。

これらのプロパティーはアダプター固有ですが、**例**の値は次のようになります。

- MimeType = text¥xml
- DHClass = com.crossworlds.DataHandlers.text.xml
- DataHandlerConfigMOName = MO_DataHandler_Default

「データ・ハンドラー」タブのこれらの値のフィールドは、ContainerManagedEvents を JMS に設定した場合にのみ表示されます。

注: ContainerManagedEvents を JMS に設定した場合、コネクターはその pollForEvents() メソッドを*呼び出*さなくなる ため、そのメソッドの機能は使用できなく なります。

このプロパティーは、DeliveryTransport プロパティーが値 JMS に設定されている場合にのみ表示されます。

ControllerStoreAndForwardMode

RepositoryDirectory が <REMOTE> の場合のみ適用可能です。

宛先側のアプリケーション固有のコンポーネントが使用不可であることをコネクター・コントローラーが検出した場合に、コネクター・コントローラーが実行する動作を設定します。

このプロパティーを true に設定した場合、イベントが ICS に到達したときに宛先側のアプリケーション固有のコンポーネントが使用不可であれば、コネクター・コントローラーはそのアプリケーション固有のコンポーネントへの要求をブロックします。アプリケーション固有のコンポーネントが作動可能になると、コネクター・コントローラーはアプリケーション固有のコンポーネントにその要求を転送します。

ただし、コネクター・コントローラーが宛先側のアプリケーション固有のコンポーネントにサービス呼び出し要求を**転送した後で**このコンポーネントが使用不可になった場合、コネクター・コントローラーはその要求を失敗させます。

このプロパティーを false に設定した場合、コネクター・コントローラーは、宛先側のアプリケーション固有のコンポーネントが使用不可であることを検出すると、ただちにすべてのサービス呼び出し要求を失敗させます。

デフォルト値は true です。

ControllerTraceLevel

RepositoryDirectory が <REMOTE> の場合のみ適用可能です。

コネクター・コントローラーのトレース・メッセージのレベルです。デフォルト値は 0 です。

DeliveryQueue

DeliveryTransport が JMS の場合のみ適用されます。

コネクターから統合ブローカーへビジネス・オブジェクトが送信されるときに使用されるキューです。

デフォルト値は CONNECTORNAME/DELIVERYQUEUE です。

DeliveryTransport

イベントのデリバリーのためのトランスポート機構を指定します。指定可能な値は、WebSphere MQ の MQ、CORBA IIOP の IDL、Java Messaging Service の JMS です。

- RepositoryDirectory がリモートの場合は、DeliveryTransport プロパティーの指定可能な値は MQ、IDL、または JMS であり、デフォルトは IDL になります。
- RepositoryDirectory がローカル・ディレクトリーの場合は、指定可能な値は JMS のみです。

DeliveryTransport プロパティーに指定されている値が、MQ または IDL である場合、コネクターは、CORBA IIOP を使用してサービス呼び出し要求と管理メッセージを送信します。

WebSphere MQ および IDL

イベントのデリバリー・トランスポートには、IDL ではなく WebSphere MQ を使用してください (1 種類の製品だけを使用する必要がある場合を除きます)。
WebSphere MQ が IDL よりも優れている点は以下のとおりです。

- 非同期 (ASYNC) 通信:
  WebSphere MQ を使用すると、アプリケーション固有のコンポーネントは、サーバーが利用不能である場合でも、イベントをポーリングして永続的に格納することができます。
- サーバー・サイド・パフォーマンス:
  WebSphere MQ を使用すると、サーバー・サイドのパフォーマンスが向上します。最適化モードでは、WebSphere MQ はイベントへのポインターのみをリポジトリー・データベースに格納するので、実際のイベントは WebSphere MQ キュー内に残ります。これにより、サイズが大きい可能性のあるイベントをリポジトリー・データベースに書き込む必要がありません。
- エージェント・サイド・パフォーマンス:
  WebSphere MQ を使用すると、アプリケーション固有のコンポーネント側のパフォーマンスが向上します。WebSphere MQ を使用すると、コネクターのポーリング・スレッドは、イベントを選出した後、コネクターのキューにそのイベントを入れ、次のイベントを選出します。この方法は IDL よりも高速で、IDL の場合、コネクターのポーリング・スレッドは、イベントを選出した後、ネットワーク経由でサーバー・プロセスにアクセスしてそのイベントをリポジトリー・データベースに永続的に格納してから、次のイベントを選出する必要があります。

JMS

Java Messaging Service (JMS) を使用しての、コネクターとクライアント・コネクター・フレームワークとの間の通信を可能にします。

JMS をデリバリー・トランスポートとして選択した場合は、jms.MessageBrokerName、jms.FactoryClassName、jms.Password、jms.UserName などの追加の JMS プロパティーが Connector Configurator 内に表示されます。このうち最初の 2 つは、このトランスポートの必須プロパティーです。

重要: 以下の環境では、コネクターに JMS トランスポート機構を使用すると、メモリー制限が発生することもあります。

- AIX 5.0
- WebSphere MQ 5.3.0.1
- ICS が統合ブローカーの場合

この環境では、WebSphere MQ クライアント内でメモリーが使用されるため、(サーバー側の) コネクター・コントローラーと (クライアント側の) コネクターの両方を始動するのは困難な場合があります。ご使用のシステムのプロセス・ヒープ・サイズが 768M 未満である場合には、次のように設定することをお勧めします。

- CWSharedEnv.sh スクリプト内で LDR_CNTRL 環境変数を設定する。

  このスクリプトは、製品ディレクトリー配下の ¥bin ディレクトリーにあります。テキスト・エディターを使用して、CWSharedEnv.sh スクリプトの最初の行として次の行を追加します。

  ```
  export LDR_CNTRL=MAXDATA=0x30000000
  ```

  この行は、ヒープ・メモリーの使用量を最大 768 MB (3 セグメント * 256 MB) に制限します。プロセス・メモリーがこの制限値を超えると、ページ・スワッピングが発生し、システムのパフォーマンスに悪影響を与える場合があります。
- IPCCBaseAddress プロパティーの値を 11 または 12 に設定する。このプロパティーの詳細については、「*システム・インストール・ガイド (UNIX 版)*」を参照してください。

DuplicateEventElimination

このプロパティーを true に設定すると、JMS 対応コネクターによるデリバリー・キューへの重複イベントのデリバリーが防止されます。この機能を使用するには、コネクターに対し、アプリケーション固有のコード内でビジネス・オブジェクトの **ObjectEventId** 属性として一意のイベント ID が設定されている必要があります。これはコネクター開発時に設定されます。

このプロパティーは、false に設定することもできます。

注: DuplicateEventElimination を true に設定する際は、MonitorQueue プロパティーを構成して保証付きイベント・デリバリーを使用可能にする必要があります。

FaultQueue

コネクターでメッセージを処理中にエラーが発生すると、コネクターは、そのメッセージを状況表示および問題説明とともにこのプロパティーに指定されているキューに移動します。

デフォルト値は CONNECTORNAME/FAULTQUEUE です。

JvmMaxHeapSize

エージェントの最大ヒープ・サイズ (メガバイト単位)。このプロパティーは、RepositoryDirectory の値が <REMOTE> の場合にのみ適用されます。

デフォルト値は 128M です。

JvmMaxNativeStackSize

エージェントの最大ネイティブ・スタック・サイズ (キロバイト単位)。このプロパティーは、RepositoryDirectory の値が <REMOTE> の場合にのみ適用されます。

デフォルト値は 128K です。

JvmMinHeapSize

エージェントの最小ヒープ・サイズ (メガバイト単位)。このプロパティーは、RepositoryDirectory の値が <REMOTE> の場合にのみ適用されます。

デフォルト値は 1M です。

jms.FactoryClassName

JMS プロバイダーのためにインスタンスを生成するクラス名を指定します。JMS をデリバリー・トランスポート機構 (DeliveryTransport) として選択する際は、このコネクター・プロパティーを必ず 設定してください。

デフォルト値は CxCommon.Messaging.jms.IBMMQSeriesFactory です。

jms.MessageBrokerName

JMS プロバイダーのために使用するブローカー名を指定します。JMS をデリバリー・トランスポート機構 (DeliveryTransport) として選択する際は、このコネクター・プロパティーを必ず 設定してください。

デフォルト値は crossworlds.queue.manager です。ローカル・メッセージ・ブローカーに接続する場合は、デフォルト値を使用します。

リモート・メッセージ・ブローカーに接続すると、このプロパティーは次の (必須) 値をとります。
QueueMgrName:<Channel>:<HostName>:<PortNumber>
各変数の意味は以下のとおりです。
QueueMgrName: キュー・マネージャー名です。
Channel: クライアントが使用するチャネルです。
HostName: キュー・マネージャーの配置先のマシン名です。
PortNumber: キュー・マネージャーが listen に使用するポートの番号です。

例えば、次のようにします。

```
jms.MessageBrokerName = WBIMB.Queue.Manager:CHANNEL1:RemoteMachine:1456
```

jms.NumConcurrentRequests

コネクターに対して同時に送信することができる並行サービス呼び出し要求の数 (最大値) を指定します。この最大値に達した場合、新規のサービス呼び出し要求はブロックされ、既存のいずれかの要求が完了した後で処理されます。

デフォルト値は 10 です。

jms.Password

JMS プロバイダーのためのパスワードを指定します。このプロパティーの値はオプションです。

デフォルトはありません。

jms.UserName

JMS プロバイダーのためのユーザー名を指定します。このプロパティーの値はオプションです。

デフォルトはありません。

ListenerConcurrency

このプロパティーは、統合ブローカーとして ICS を使用する場合の MQ Listener でのマルチスレッド化をサポートしています。このプロパティーにより、データベースへの複数イベントの書き込み操作をバッチ処理できるので、システム・パフォーマンスが向上します。デフォルト値は 1 です。

このプロパティーは、MQ トランスポートを使用するコネクターにのみ適用されます。`DeliveryTransport` プロパティーには `MQ` を設定してください。

Locale

言語コード、国または地域、および、希望する場合には、関連した文字コード・セットを指定します。このプロパティーの値は、データの照合やソート順、日付と時刻の形式、通貨記号などの国/地域別情報を決定します。

ロケール名は、次の書式で指定します。

ll_TT.codeset

ここで、以下のように説明されます。

ll	2 文字の言語コード (普通は小文字)
TT	2 文字の国または地域コード (普通は大文字)
codeset	関連文字コード・セットの名前。名前のこの部分は、通常、オプションです。

デフォルトでは、ドロップダウン・リストには、サポートされるロケールの一部のみが表示されます。ドロップダウン・リストに、サポートされる他の値を追加する

には、製品ディレクトリーにある ¥Data¥Std¥stdConnProps.xml ファイルを手動で変更する必要があります。詳細については、本書の Connector Configurator に関する付録を参照してください。

デフォルト値は en_US です。コネクターがグローバル化に対応していない場合、このプロパティーの有効な値は en_US のみです。特定のコネクターがグローバル化に対応しているかどうかを判別するには、以下の Web サイトにあるコネクターのバージョン・リストを参照してください。

http://www.ibm.com/software/websphere/wbiadapters/infocenter、または
http://www.ibm.com/websphere/integration/wicserver/infocenter

LogAtInterchangeEnd

RepositoryDirectory が <REMOTE> の場合のみ適用可能です。

統合ブローカーのログ宛先にエラーを記録するかどうかを指定します。ブローカーのログ宛先にログを記録すると、電子メール通知もオンになります。これにより、エラーまたは致命的エラーが発生すると、InterchangeSystem.cfg ファイルに指定された MESSAGE_RECIPIENT に対する電子メール・メッセージが生成されます。

例えば、LogAtInterChangeEnd を true に設定した場合にコネクターからアプリケーションへの接続が失われると、指定されたメッセージ宛先に、電子メール・メッセージが送信されます。デフォルト値は false です。

MaxEventCapacity

コントローラー・バッファー内のイベントの最大数。このプロパティーはフロー制御が使用し、RepositoryDirectory プロパティーの値が <REMOTE> の場合にのみ適用されます。

値は 1 から 2147483647 の間の正整数です。デフォルト値は 2147483647 です。

MessageFileName

コネクター・メッセージ・ファイルの名前です。メッセージ・ファイルの標準位置は、製品ディレクトリーの ¥connectors¥messages です。メッセージ・ファイルが標準位置に格納されていない場合は、メッセージ・ファイル名を絶対パスで指定します。

コネクター・メッセージ・ファイルが存在しない場合は、コネクターは InterchangeSystem.txt をメッセージ・ファイルとして使用します。このファイルは、製品ディレクトリーに格納されています。

注: 特定のコネクターについて、コネクター独自のメッセージ・ファイルがあるかどうかを判別するには、該当するアダプターのユーザーズ・ガイドを参照してください。

MonitorQueue

コネクターが重複イベントをモニターするために使用する論理キューです。このプロパティーは、DeliveryTransport プロパティー値が JMS であり、かつ DuplicateEventElimination が TRUE に設定されている場合にのみ使用されます。

デフォルト値は CONNECTORNAME/MONITORQUEUE です。

OADAutoRestartAgent

RepositoryDirectory が <REMOTE> の場合のみ有効です。

コネクターが自動再始動およびリモート再始動機能を使用するかどうかを指定します。この機能では、MQ により起動される Object Activation Daemon (OAD) を使用して、異常シャットダウン後にコネクターを再始動したり、System Monitor からリモート・コネクターを始動したりします。

自動再始動機能およびリモート再始動機能を使用可能にするには、このプロパティーを true に設定する必要があります。MQ によりトリガーされる OAD 機能の構成方法については、「システム・インストール・ガイド *(Windows 版)*」または「システム・インストール・ガイド *(UNIX 版)*」を参照してください。

デフォルト値は false です。

OADMaxNumRetry

RepositoryDirectory が <REMOTE> の場合のみ有効です。

異常シャットダウンの後で MQ によりトリガーされる OAD がコネクターの再始動を自動的に試行する回数の最大数を指定します。このプロパティーを有効にするためには、OADAutoRestartAgent プロパティーを true に設定する必要があります。

デフォルト値は 1000 です。

OADRetryTimeInterval

RepositoryDirectory が <REMOTE> の場合のみ有効です。

MQ によりトリガーされる OAD の再試行時間間隔の分数を指定します。コネクター・エージェントがこの再試行時間間隔内に再始動しない場合は、コネクター・コントローラーはコネクター・エージェントを再び再始動するように OAD に要求します。OAD はこの再試行プロセスを OADMaxNumRetry プロパティーで指定された回数だけ繰り返します。このプロパティーを有効にするためには、OADAutoRestartAgent プロパティーを true に設定する必要があります。

デフォルト値は 10 です。

PollEndTime

イベント・キューのポーリングを停止する時刻です。形式は HH:MM です。ここで、*HH* は 0 から 23 時を表し、*MM* は 0 から 59 分を表します。

このプロパティーには必ず有効な値を指定してください。デフォルト値は HH:MM ですが、この値は必ず変更する必要があります。

PollFrequency

これは、前回のポーリングの終了から次のポーリングの開始までの間の間隔です。PollFrequency は、あるポーリング・アクションの終了から次のポーリング・アク

ションの開始までの時間をミリ秒単位で指定します。これはポーリング・アクション間の間隔ではありません。この論理を次に説明します。

- ポーリングし、PollQuantity の値により指定される数のオブジェクトを取得します。
- これらのオブジェクトを処理します。一部のアダプターでは、これは個別のスレッドで部分的に実行されます。これにより、次のポーリング・アクションまで処理が非同期に実行されます。
- PollFrequency で指定された間隔にわたって遅延します。
- このサイクルを繰り返します。

PollFrequency は以下の値のいずれかに設定します。

- ポーリング・アクション間のミリ秒数 (整数)。
- ワード key。コネクターは、コネクターのコマンド・プロンプト・ウィンドウで文字 p が入力されたときにのみポーリングを実行します。このワードは小文字で入力します。
- ワード no。コネクターはポーリングを実行しません。このワードは小文字で入力します。

デフォルト値は 10000 です。

重要: 一部のコネクターでは、このプロパティーの使用が制限されています。このようなコネクターが存在する場合には、アダプターのインストールと構成に関する章で制約事項が説明されています。

PollQuantity

コネクターがアプリケーションからポーリングする項目の数を指定します。アダプターにコネクター固有のポーリング数設定プロパティーがある場合、標準プロパティーの値は、このコネクター固有のプロパティーの設定値によりオーバーライドされます。

電子メール・メッセージもイベントと見なされます。コネクターは、電子メールに関するポーリングを受けたときには次のように動作します。

コネクターは、1 回目のポーリングを受けると、メッセージの本文を選出します。これは、本文が添付とも見なされるからです。本文の MIME タイプにはデータ・ハンドラーが指定されていないので、コネクターは本文を無視します。
コネクターは PO の最初の添付を処理します。この添付の MIME タイプには対応するデータ・ハンドラーがあるので、コネクターはビジネス・オブジェクトを Visual Test Connector に送信します。
2 回目のポーリングを受けると、コネクターは PO の 2 番目の添付を処理します。この添付の MIME タイプには対応するデータ・ハンドラーがあるので、コネクターはビジネス・オブジェクトを Visual Test Connector に送信します。
これが受け入れられると、PO の 3 番目の添付が届きます。

PollStartTime

イベント・キューのポーリングを開始する時刻です。形式は *HH:MM* です。ここで、*HH* は 0 から 23 時を表し、*MM* は 0 から 59 分を表します。

このプロパティーには必ず有効な値を指定してください。デフォルト値は HH:MM ですが、この値は必ず変更する必要があります。

RequestQueue

統合ブローカーが、ビジネス・オブジェクトをコネクターに送信するときに使用されるキューです。

デフォルト値は CONNECTOR/REQUESTQUEUE です。

RepositoryDirectory

コネクターが XML スキーマ文書を読み取るリポジトリーの場所です。この XML スキーマ文書には、ビジネス・オブジェクト定義のメタデータが含まれています。

統合ブローカーが ICS の場合はこの値を <REMOTE> に設定する必要があります。これは、コネクターが InterChange Server リポジトリーからこの情報を取得するためです。

統合ブローカーが WebSphere Message Broker または WAS の場合には、この値を <*local directory*> に設定する必要があります。

ResponseQueue

DeliveryTransport が JMS の場合のみ適用可能で、RepositoryDirectory が <REMOTE> の場合のみ必須です。

JMS 応答キューを指定します。JMS 応答キューは、応答メッセージをコネクター・フレームワークから統合ブローカーへデリバリーします。統合ブローカーが ICS の場合、サーバーは要求を送信し、JMS 応答キューの応答メッセージを待ちます。

RestartRetryCount

コネクターによるコネクター自体の再始動の試行回数を指定します。このプロパティーを並列コネクターに対して使用する場合、コネクターのマスター側のアプリケーション固有のコンポーネントがスレーブ側のアプリケーション固有のコンポーネントの再始動を試行する回数が指定されます。

デフォルト値は 3 です。

RestartRetryInterval

コネクターによるコネクター自体の再始動の試行間隔を分単位で指定します。このプロパティーを並列コネクターに対して使用する場合、コネクターのマスター側のアプリケーション固有のコンポーネントがスレーブ側のアプリケーション固有のコンポーネントの再始動を試行する間隔が指定されます。指定可能な値の範囲は 1 から 2147483647 です。

デフォルト値は 1 です。

RHF2MessageDomain

WebSphere Message Brokers および WAS でのみ使用されます。

このプロパティーにより、JMS ヘッダーのドメイン名フィールドの値を構成できます。JMS トランスポートを介してデータを WMQI に送信するときに、アダプター・フレームワークにより JMS ヘッダー情報、ドメイン名、および固定値 mrm が書き込まれます。構成可能なドメイン名により、ユーザーは WMQI ブローカーによるメッセージ・データの処理方法を追跡できます。

サンプル・ヘッダーを以下に示します。

```
<mcd><Msd>mrm</Msd><Set>3</Set><Type>
Retek_POPhyDesc</Type><Fmt>CwXML</Fmt></mcd>
```

デフォルト値は mrm ですが、このプロパティーには xml も設定できます。このプロパティーは、DeliveryTransport が JMS に設定されており、かつ WireFormat が CwXML に設定されている場合にのみ表示されます。

SourceQueue

DeliveryTransport が JMS で、ContainerManagedEvents が指定されている場合のみ適用されます。

JMS イベント・ストアを使用する JMS 対応コネクターでの保証付きイベント・デリバリーをサポートするコネクター・フレームワークに、JMS ソース・キューを指定します。詳細については、85 ページの『ContainerManagedEvents』を参照してください。

デフォルト値は CONNECTOR/SOURCEQUEUE です。

SynchronousRequestQueue

DeliveryTransport が JMS の場合のみ適用されます。

同期応答を要求する要求メッセージを、コネクター・フレームワークからブローカーに配信します。このキューは、コネクターが同期実行を使用する場合にのみ必要です。同期実行の場合、コネクター・フレームワークは、SynchronousRequestQueue にメッセージを送信し、SynchronousResponseQueue でブローカーから戻される応答を待機します。コネクターに送信される応答メッセージには、元のメッセージの ID を指定する相関 ID が含まれています。

デフォルト値は CONNECTORNAME/SYNCHRONOUSREQUESTQUEUE です。

SynchronousResponseQueue

DeliveryTransport が JMS の場合のみ適用されます。

同期要求に対する応答として送信される応答メッセージを、ブローカーからコネクター・フレームワークに配信します。このキューは、コネクターが同期実行を使用する場合にのみ必要です。

デフォルト値は CONNECTORNAME/SYNCHRONOUSRESPONSEQUEUE です。

SynchronousRequestTimeout

DeliveryTransport が JMS の場合のみ適用されます。

コネクターが同期要求への応答を待機する時間を分単位で指定します。コネクターは、指定された時間内に応答を受信できなかった場合、元の同期要求メッセージをエラー・メッセージとともに障害キューに移動します。

デフォルト値は 0 です。

WireFormat

トランスポートのメッセージ・フォーマットです。

- RepositoryDirectory がローカル・ディレクトリーの場合は、設定は CwXML になります。
- RepositoryDirectory の値が <REMOTE> の場合には、設定値は CwBO です。

WsifSynchronousRequestTimeout

WAS 統合ブローカーでのみ使用されます。

コネクターが同期要求への応答を待機する時間を分単位で指定します。コネクターは、指定された時間内に応答を受信できなかった場合、元の同期要求メッセージをエラー・メッセージとともに障害キューに移動します。

デフォルト値は 0 です。

XMLNameSpaceFormat

WebSphere Message Brokers および WAS 統合ブローカーでのみ使用されます。

ビジネス・オブジェクト定義の XML 形式でネーム・スペースを short と long のどちらにするかをユーザーが指定できるようにするための、強力なプロパティーです。

デフォルト値は short です。

付録 B. Connector Configurator

この付録では、Connector Configurator を使用してアダプターの構成プロパティー値を設定する方法について説明します。

Connector Configurator を使用して次の作業を行います。

- コネクターを構成するためのコネクター固有のプロパティー・テンプレートを作成する
- 構成ファイルを作成する
- 構成ファイル内のプロパティーを設定する

注:

本書では、ディレクトリー・パスの規則として円記号 (¥) を使用します。UNIX システムを使用している場合は、円記号をスラッシュ (/) に置き換えてください。また、各オペレーティング・システムの規則に従ってください。

この付録では、次のトピックについて説明します。

Connector Configurator の概要

Connector Configurator では、次の統合ブローカーで使用するアダプターのコネクター・コンポーネントを構成できます。

- WebSphere InterChange Server (ICS)
- WebSphere MQ Integrator、WebSphere MQ Integrator Broker、および WebSphere Business Integration Message Broker (WebSphere Message Brokers (WMQI) と総称)
- WebSphere Application Server (WAS)

Connector Configurator を使用して次の作業を行います。

- コネクターを構成するための**コネクター固有のプロパティー・テンプレート** を作成する。
- **コネクター構成ファイル**を作成します。インストールするコネクターごとに構成ファイルを 1 つ作成する必要があります。
- 構成ファイル内のプロパティーを設定する。
  場合によっては、コネクター・テンプレートでプロパティーに対して設定されているデフォルト値を変更する必要があります。また、サポートされるビジネス・オブジェクト定義と、ICS の場合はコラボレーションとともに使用するマップを

指定し、必要に応じてメッセージング、ロギング、トレース、およびデータ・ハンドラー・パラメーターを指定する必要があります。

Connector Configurator の実行モードと使用する構成ファイルのタイプは、実行する統合ブローカーによって異なります。例えば、使用している統合ブローカーが WMQI の場合、Connector Configurator を System Manager から実行するのではなく、直接実行します (『スタンドアロン・モードでの Configurator の実行』を参照)。

コネクター構成プロパティーには、標準の構成プロパティー (すべてのコネクターがもつプロパティー) と、コネクター固有のプロパティー (特定のアプリケーションまたはテクノロジーのためにコネクターで必要なプロパティー) とが含まれます。

標準プロパティーはすべてのコネクターにより使用されるので、標準プロパティーを新規に定義する必要はありません。ファイルを作成すると、Connector Configurator により標準プロパティーがこの構成ファイルに挿入されます。ただし、Connector Configurator で各標準プロパティーの値を設定する必要があります。

標準プロパティーの範囲は、ブローカーと構成によって異なる可能性があります。特定のプロパティーに特定の値が設定されている場合にのみ使用できるプロパティーがあります。Connector Configurator の「標準のプロパティー」ウィンドウには、特定の構成で設定可能なプロパティーが表示されます。

ただし**コネクター固有プロパティー**の場合は、最初にプロパティーを定義し、その値を設定する必要があります。このため、特定のアダプターのコネクター固有プロパティーのテンプレートを作成します。システム内で既にテンプレートが作成されている場合には、作成されているテンプレートを使用します。システム内でまだテンプレートが作成されていない場合には、100 ページの『新規テンプレートの作成』のステップに従い、テンプレートを新規に作成します。

注: Connector Configurator は、Windows 環境内でのみ実行されます。UNIX 環境でコネクターを実行する場合には、Windows でConnector Configurator を使用して構成ファイルを変更し、このファイルを UNIX 環境へコピーします。

Connector Configurator の始動

以下の 2 種類のモードで Connector Configurator を開始および実行できます。

- スタンドアロン・モードで個別に実行
- System Manager から

スタンドアロン・モードでの Configurator の実行

どのブローカーを実行している場合にも、Connector Configurator を個別に実行し、コネクター構成ファイルを編集できます。

これを行うには、以下のステップを実行します。

- **「スタート」>「プログラム」**から、**「IBM WebSphere InterChange Server」>「IBM WebSphere Business Integration Tools」>「Connector Configurator」**をクリックします。
- **「ファイル」>「新規」>「コネクター構成」**を選択します。

- **「システム接続: Integration Broker」**の隣のプルダウン・メニューをクリックします。使用しているブローカーに応じて、ICS、WebSphere Message Brokers、または WAS を選択します。

Connector Configurator を個別に実行して構成ファイルを生成してから、System Manager に接続してこの構成ファイルを System Manager プロジェクトに保存することもできます (105 ページの『構成ファイルの完成』を参照)。

System Manager からの Configurator の実行

System Manager から Connector Configurator を実行できます。

Connector Configurator を実行するには、以下のステップを実行します。

1. System Manager を開きます。
2. 「System Manager」ウィンドウで、**「統合コンポーネント・ライブラリー」**アイコンを展開し、**「コネクター」**を強調表示します。
3. System Manager メニュー・バーから、**「ツール」>「Connector Configurator」**をクリックします。「Connector Configurator」ウィンドウが開き、**「新規コネクター」**ダイアログ・ボックスが表示されます。
4. **「システム接続: Integration Broker」**の隣のプルダウン・メニューをクリックします。使用しているブローカーに応じて、ICS、WebSphere Message Brokers、または WAS を選択します。

既存の構成ファイルを編集するには、以下のステップを実行します。

- 「System Manager」ウィンドウの「コネクター」フォルダーでいずれかの構成ファイルを選択し、右クリックします。Connector Configurator が開き、この構成ファイルの統合ブローカー・タイプおよびファイル名が上部に表示されます。
- Connector Configurator で**「ファイル」>「開く」**を選択します。プロジェクトまたはプロジェクトが保管されているディレクトリーからコネクター構成ファイルを選択します。
- 「標準のプロパティー」タブをクリックし、この構成ファイルに含まれているプロパティーを確認します。

コネクター固有のプロパティー・テンプレートの作成

コネクターの構成ファイルを作成するには、コネクター固有プロパティーのテンプレートとシステム提供の標準プロパティーが必要です。

コネクター固有プロパティーのテンプレートを新規に作成するか、または既存のコネクター定義をテンプレートとして使用します。

- テンプレートの新規作成については、100 ページの『新規テンプレートの作成』を参照してください。
- 既存のファイルを使用する場合には、既存のテンプレートを変更し、新しい名前でこのテンプレートを保管します。既存のテンプレートは ¥WebSphereAdapters¥bin¥Data¥App ディレクトリーにあります。

新規テンプレートの作成

このセクションでは、テンプレートでプロパティーを作成し、プロパティーの一般特性および値を定義し、プロパティー間の依存関係を指定する方法について説明します。次にそのテンプレートを保管し、新規コネクター構成ファイルを作成するためのベースとして使用します。

Connector Configurator でテンプレートを作成するには、以下のステップを実行します。

1. **「ファイル」>「新規」>「コネクター固有プロパティー・テンプレート」**をクリックします。
2. **「コネクター固有プロパティー・テンプレート」** ダイアログ・ボックスが表示されます。
   - **「新規テンプレート名を入力してください」**の下の**「名前」**フィールドに、新規テンプレートの名前を入力します。テンプレートから新規構成ファイルを作成するためのダイアログ・ボックスを開くと、この名前が再度表示されます。
   - テンプレートに含まれているコネクター固有のプロパティー定義を調べるには、**「テンプレート名」**表示でそのテンプレートの名前を選択します。そのテンプレートに含まれているプロパティー定義のリストが**「テンプレートのプレビュー」**表示に表示されます。
3. テンプレートを作成するときには、ご使用のコネクターに必要なプロパティー定義に類似したプロパティー定義が含まれている既存のテンプレートを使用できます。ご使用のコネクターで使用するコネクター固有のプロパティーが表示されるテンプレートが見つからない場合は、自分で作成する必要があります。
   - 既存のテンプレートを変更する場合には、**「変更する既存のテンプレートを選択してください: 検索テンプレート」**の下の**「テンプレート名」**テーブルのリストから、テンプレート名を選択します。
   - このテーブルには、現在使用可能なすべてのテンプレートの名前が表示されます。テンプレートを検索することもできます。

一般特性の指定

「次へ」をクリックしてテンプレートを選択すると、**「プロパティー: コネクター固有プロパティー・テンプレート」**ダイアログ・ボックスが表示されます。このダイアログ・ボックスには、定義済みプロパティーの「一般」特性のタブと「値」の制限のタブがあります。「一般」表示には以下のフィールドがあります。

- **一般:**
  プロパティー・タイプ
  更新されたメソッド
  説明
- **フラグ**
  標準フラグ
- **カスタム・フラグ**
  フラグ

プロパティーの一般特性の選択を終えたら、**「値」**タブをクリックします。

値の指定

「値」タブを使用すると、プロパティーの最大長、最大複数値、デフォルト値、または値の範囲を設定できます。編集可能な値も許可されます。これを行うには、以下のステップを実行します。

1. **「値」**タブをクリックします。「一般」のパネルに代わって「値」の表示パネルが表示されます。
2. **「プロパティーを編集」**表示でプロパティーの名前を選択します。
3. **「最大長」**および**「最大複数値」**のフィールドに値を入力します。

新規プロパティー値を作成するには、以下のステップを実行します。

1. **「プロパティーを編集」**リストでプロパティーを選択し、右マウス・ボタンでクリックします。
2. ダイアログ・ボックスから**「追加」**を選択します。
3. 新規プロパティー値の名前を入力し、「OK」をクリックします。右側の**「値」**パネルに値が表示されます。

「値」パネルには、3 つの列からなるテーブルが表示されます。

「値」の列には、**「プロパティー値」**ダイアログ・ボックスで入力した値と、以前に作成した値が表示されます。

「デフォルト値」の列では、値のいずれかをデフォルトとして指定することができます。

「値の範囲」の列には、**「プロパティー値」**ダイアログ・ボックスで入力した範囲が表示されます。

値が作成されて、グリッドに表示されると、そのテーブルの表示内から編集できるようになります。

テーブルにある既存の値の変更を行うには、その行の行番号をクリックして行全体を選択します。次に**「値」**フィールドを右マウス・ボタンでクリックし、**「値の編集 (Edit Value)」**をクリックします。

依存関係の設定

「一般」タブと**「値」**タブで変更を行ったら、**「次へ」**をクリックします。**「依存関係 - コネクター固有プロパティー・テンプレート」** ダイアログ・ボックスが表示されます。

依存プロパティーは、別のプロパティーの値が特定の条件に合致する場合にのみ、テンプレートに組み込まれて、構成ファイルで使用されるプロパティーです。例えば、テンプレートに `PollQuantity` が表示されるのは、トランスポート機構が JMS であり、`DuplicateEventElimination` が `True` に設定されている場合のみです。プロパティーを依存プロパティーとして指定し、依存する条件を設定するには、以下のステップを実行します。

1. **「使用可能なプロパティー」**表示で、依存プロパティーとして指定するプロパティーを選択します。

2. **「プロパティーを選択」**フィールドで、ドロップダウン・メニューを使用して、条件値を持たせるプロパティーを選択します。
3. **「条件演算子」**フィールドで以下のいずれかを選択します。

   == (等しい)

   != (等しくない)

   > (より大)

   < (より小)

   >= (より大か等しい)

   <= (より小か等しい)
4. **「条件値」**フィールドで、依存プロパティーをテンプレートに組み込むために必要な値を入力します。
5. **「使用可能なプロパティー」**表示で依存プロパティーを強調表示させて矢印をクリックし、**「依存プロパティー」**表示に移動させます。
6. **「完了」**をクリックします。Connector Configurator により、XML 文書として入力した情報が、Connector Configurator がインストールされている `¥bin` ディレクトリーの `¥data¥app` の下に保管されます。

新規構成ファイルの作成

構成ファイルを新規に作成するには、構成ファイルの名前を指定し、統合ブローカーを選択する必要があります。

- 「System Manager」ウィンドウで**「コネクター」**フォルダーを右クリックし、**「新規コネクターの作成」**を選択します。Connector Configurator が開き、**「新規コネクター」**ダイアログ・ボックスが表示されます。
- スタンドアロン・モードの場合は、Connector Configurator で**「ファイル」>「新規」>「コネクター構成」**を選択します。「新規コネクター」ウィンドウで、新規コネクターの名前を入力します。

また、統合ブローカーも選択する必要があります。選択したブローカーによって、構成ファイルに記述されるプロパティーが決まります。ブローカーを選択するには、以下のステップを実行します。

- **「Integration Broker」**フィールドで、ICS 接続、WebSphere Message Brokers 接続、WAS 接続のいずれかを選択します。
- この章で後述する説明に従って**「新規コネクター」**ウィンドウの残りのフィールドに入力します。

コネクター固有のテンプレートからの構成ファイルの作成

コネクター固有のテンプレートを作成すると、テンプレートを使用して構成ファイルを作成できます。

1. **「ファイル」>「新規」>「コネクター構成」**をクリックします。
2. 以下のフィールドを含む**「新規コネクター」**ダイアログ・ボックス表示されます。

- **名前**

  コネクターの名前を入力します。名前では大文字と小文字が区別されます。入力する名前は、システムにインストールされているコネクターのファイル名に対応した一意の名前でなければなりません。

  重要: Connector Configurator では、入力された名前のスペルはチェックされません。名前が正しいことを確認してください。

- **システム接続**

  ICS 接続、WebSphere Message Brokers 接続、WAS のいずれかをクリックします。

- **「コネクター固有プロパティー・テンプレート」を選択します。**

  ご使用のコネクター用に設計したテンプレートの名前を入力します。**「テンプレート名」**表示に、使用可能なテンプレートが表示されます。「テンプレート名」表示で名前を選択すると、**「プロパティー・テンプレートのプレビュー」**表示に、そのテンプレートで定義されているコネクター固有のプロパティーが表示されます。

  使用するテンプレートを選択し、**「OK」**をクリックします。

3. 構成しているコネクターの構成画面が表示されます。タイトル・バーに統合ブローカーとコネクターの名前が表示されます。ここですべてのフィールドに値を入力して定義を完了するか、ファイルを保管して後でフィールドに値を入力するかを選択できます。
4. ファイルを保管するには、**「ファイル」>「保管」>「ファイルに」**をクリックするか、**「ファイル」>「保管」>「プロジェクトに保管」**をクリックします。プロジェクトに保管するには、System Manager が実行中でなければなりません。
   ファイルとして保管する場合は、**「ファイル・コネクターを保管」**ダイアログ・ボックスが表示されます。`*.cfg` をファイル・タイプとして選択し、「ファイル名」フィールド内に名前が正しいスペル (大文字と小文字の区別を含む) で表示されていることを確認してから、ファイルを保管するディレクトリーにナビゲートし、**「保管」**をクリックします。 Connector Configurator のメッセージ・パネルの状況表示に、構成ファイルが正常に作成されたことが示されます。

   重要: ここで設定するディレクトリー・パスおよび名前は、コネクターの始動ファイルで指定するコネクター構成ファイルのパスおよび名前に一致している必要があります。
5. この章で後述する手順に従って、「Connector Configurator」ウィンドウの各タブにあるフィールドに値を入力し、コネクター定義を完了します。

既存ファイルの使用

使用可能な既存ファイルは、以下の 1 つまたは複数の形式になります。

- コネクター定義ファイル。
  コネクター定義ファイルは、特定のコネクターのプロパティーと、適用可能なデフォルト値がリストされたテキスト・ファイルです。コネクターの配布パッケー

ジの ¥repository ディレクトリー内には、このようなファイルが格納されていることがあります (通常、このファイルの拡張子は .txt です。例えば、XML コネクターの場合は CN_XML.txt です)。

- ICS リポジトリー・ファイル。
  コネクターの以前の ICS インプリメンテーションで使用した定義は、そのコネクターの構成で使用されたリポジトリー・ファイルで使用可能になります。そのようなファイルの拡張子は、通常 .in または .out です。
- コネクターの以前の構成ファイル。
  これらのファイルの拡張子は、通常 *.cfg です。

これらのいずれのファイル・ソースにも、コネクターのコネクター固有プロパティーのほとんど、あるいはすべてが含まれますが、この章内の後で説明するように、コネクター構成ファイルは、ファイルを開いて、プロパティーを設定しない限り完成しません。

既存ファイルを使用してコネクターを構成するには、Connector Configurator でそのファイルを開き、構成を修正し、そのファイルを再度保管する必要があります。

以下のステップを実行して、ディレクトリーから *.txt、*.cfg、または *.in ファイルを開きます。

1. Connector Configurator 内で、**「ファイル」>「開く」>「ファイルから」**をクリックします。
2. **「ファイル・コネクターを開く」**ダイアログ・ボックス内で、以下のいずれかのファイル・タイプを選択して、使用可能なファイルを調べます。
   - 構成 (*.cfg)
   - ICS リポジトリー (*.in、*.out)

     ICS 環境でのコネクターの構成にリポジトリー・ファイルが使用された場合には、このオプションを選択します。リポジトリー・ファイルに複数のコネクター定義が含まれている場合は、ファイルを開くとすべての定義が表示されます。
   - すべてのファイル (*.*)

     コネクターのアダプター・パッケージに *.txt ファイルが付属していた場合、または別の拡張子で定義ファイルが使用可能である場合は、このオプションを選択します。
3. ディレクトリー表示内で、適切なコネクター定義ファイルへ移動し、ファイルを選択し、**「開く」**をクリックします。

System Manager プロジェクトからコネクター構成を開くには、以下のステップを実行します。

1. System Manager を始動します。System Manager が開始されている場合にのみ、構成を System Manager から開いたり、System Manager に保管したりできます。
2. Connector Configurator を始動します。
3. **「ファイル」>「開く」>「プロジェクトから」**をクリックします。

構成ファイルの完成

構成ファイルを開くか、プロジェクトからコネクターを開くと、「Connector Configurator」ウィンドウに構成画面が表示されます。この画面には、現在の属性と値が表示されます。

構成画面のタイトルには、ファイル内で指定された統合ブローカーとコネクターの名前が表示されます。正しいブローカーが設定されていることを確認してください。正しいブローカーが設定されていない場合、コネクターを構成する前にブローカー値を変更してください。これを行うには、以下のステップを実行します。

1. **「標準のプロパティー」**タブで、BrokerType プロパティーの値フィールドを選択します。ドロップダウン・メニューで、値 ICS、WMQI、または WAS を選択します。
2. 選択したブローカーに関連付けられているプロパティーが「標準のプロパティー」タブに表示されます。ここでファイルを保管するか、または 108 ページの『サポートされるビジネス・オブジェクト定義の指定』の説明に従い残りの構成フィールドに値を入力することができます。
3. 構成が完了したら、**「ファイル」>「保管」>「プロジェクトに」**を選択するか、または**「ファイル」>「保管」>「ファイルに」**を選択します。

   ファイルに保管する場合は、*.cfg を拡張子として選択し、ファイルの正しい格納場所を選択して、**「保管」**をクリックします。

   複数のコネクター構成を開いている場合、構成をすべてファイルに保管するには**「すべてファイルに保管」**を選択し、コネクター構成をすべて System Manager プロジェクトに保管するには**「すべてプロジェクトに保管」**をクリックします。

   Connector Configurator では、ファイルを保管する前に、必須の標準プロパティーすべてに値が設定されているかどうかが確認されます。必須の標準プロパティーに値が設定されていない場合、Connector Configurator は、検証が失敗したというメッセージを表示します。構成ファイルを保管するには、そのプロパティーの値を指定する必要があります。

構成ファイル・プロパティーの設定

新規のコネクター構成ファイルを作成して名前を付けるとき、または既存のコネクター構成ファイルを開くときには、Connector Configurator によって構成画面が表示されます。構成画面には、必要な構成値のカテゴリーに対応する複数のタブがあります。

Connector Configurator では、すべてのブローカーで実行されているコネクターで、以下のカテゴリーのプロパティーに値が設定されている必要があります。

- 標準のプロパティー
- コネクター固有のプロパティー
- サポートされるビジネス・オブジェクト
- トレース/ログ・ファイルの値
- データ・ハンドラー (保証付きイベント・デリバリーで JMS メッセージングを使用するコネクターの場合該当する)

注: JMS メッセージングを使用するコネクターの場合は、データをビジネス・オブジェクトに変換するデータ・ハンドラーの構成に関して追加のカテゴリーが表示される場合があります。

ICS で実行されているコネクターの場合、以下のプロパティーの値も設定されている必要があります。

- 関連付けられたマップ
- リソース
- メッセージング (該当する場合)

重要: Connector Configurator では、英語文字セットまたは英語以外の文字セットのいずれのプロパティー値も設定可能です。ただし、標準のプロパティーおよびコネクター固有プロパティー、およびサポートされるビジネス・オブジェクトの名前では、英語文字セットのみを使用する必要があります。

標準プロパティーとコネクター固有プロパティーの違いは、以下のとおりです。

- コネクターの標準プロパティーは、コネクターのアプリケーション固有のコンポーネントとブローカー・コンポーネントの両方によって共用されます。すべてのコネクターが同じ標準プロパティーのセットを使用します。これらのプロパティーの説明は、各アダプター・ガイドの付録 A にあります。変更できるのはこれらの値の一部のみです。
- アプリケーション固有のプロパティーは、コネクターのアプリケーション固有コンポーネント (アプリケーションと直接対話するコンポーネント) のみに適用されます。各コネクターには、そのコネクターのアプリケーションだけで使用されるアプリケーション固有のプロパティーがあります。これらのプロパティーには、デフォルト値が用意されているものもあれば、そうでないものもあります。また、一部のデフォルト値は変更することができます。各アダプター・ガイドのインストールおよび構成の章に、アプリケーション固有のプロパティーおよび推奨値が記述されています。

「標準プロパティー」と**「コネクター固有プロパティー」**のフィールドは、どのフィールドが構成可能であるかを示すために色分けされています。

- 背景がグレーのフィールドは、標準のプロパティーを表します。値を変更することはできますが、名前の変更およびプロパティーの除去はできません。
- 背景が白のフィールドは、アプリケーション固有のプロパティーを表します。これらのプロパティーは、アプリケーションまたはコネクターの特定のニーズによって異なります。値の変更も、これらのプロパティーの除去も可能です。
- 「値」フィールドは構成できます。
- プロパティーごとに**「更新メソッド」**フィールドが表示されます。これは、変更された値をアクティブにするためにコンポーネントまたはエージェントの再始動が必要かどうかを示します。この設定を構成することはできません。

標準コネクター・プロパティーの設定

標準のプロパティーの値を変更するには、以下の手順を実行します。

1. 値を設定するフィールド内でクリックします。

2. 値を入力するか、ドロップダウン・メニューが表示された場合にはメニューから値を選択します。
3. 標準のプロパティーの値をすべて入力後、以下のいずれかを実行することができます。
   - 変更内容を破棄し、元の値を保持したままで Connector Configurator を終了するには、**「ファイル」>「終了」**をクリックし (またはウィンドウを閉じ)、変更内容を保管するかどうかを確認するプロンプトが出されたら**「いいえ」**をクリックします。
   - Connector Configurator 内の他のカテゴリーの値を入力するには、そのカテゴリーのタブを選択します。**「標準のプロパティー」**(またはその他のカテゴリー) で入力した値は、次のカテゴリーに移動しても保持されます。ウィンドウを閉じると、すべてのカテゴリーで入力した値を一括して保管するかまたは破棄するかを確認するプロンプトが出されます。
   - 修正した値を保管するには、**「ファイル」>「終了」**をクリックし (またはウィンドウを閉じ)、変更内容を保管するかどうかを確認するプロンプトが出されたら**「はい」**をクリックします。「ファイル」メニューまたはツールバーから**「保管」>「ファイルに」**をクリックする方法もあります。

アプリケーション固有の構成プロパティーの設定

アプリケーション固有の構成プロパティーの場合、プロパティー名の追加または変更、値の構成、プロパティーの削除、およびプロパティーの暗号化が可能です。プロパティーのデフォルトの長さは 255 文字です。

1. グリッドの左上端の部分で右マウス・ボタンをクリックします。ポップアップ・メニュー・バーが表示されます。プロパティーを追加するときは**「追加」**をクリックします。子プロパティーを追加するには、親の行番号で右マウス・ボタンをクリックし、**「子を追加」**をクリックします。
2. プロパティーまたは子プロパティーの値を入力します。
3. プロパティーを暗号化するには、**「暗号化」**ボックスを選択します。
4. 106 ページの『標準コネクター・プロパティーの設定』の説明に従い、変更内容を保管するかまたは破棄するかを選択します。

各プロパティーごとに表示される「更新メソッド」は、変更された値をアクティブにするためにコンポーネントまたはエージェントの再始動が必要かどうかを示します。

重要: 事前設定のアプリケーション固有のコネクター・プロパティー名を変更すると、コネクターに障害が発生する可能性があります。コネクターをアプリケーションに接続したり正常に実行したりするために、特定のプロパティー名が必要である場合があります。

コネクター・プロパティーの暗号化

「コネクター固有プロパティー」ウィンドウの**「暗号化」**チェック・ボックスにチェックマークを付けると、アプリケーション固有のプロパティーを暗号化することができます。値の暗号化を解除するには、**「暗号化」**チェック・ボックスをクリックしてチェックマークを外し、**「検証」**ダイアログ・ボックスに正しい値を入力し、**「OK」**をクリックします。入力された値が正しい場合は、暗号化解除された値が表示されます。

各プロパティーとそのデフォルト値のリストおよび説明は、各コネクターのアダプター・ユーザーズ・ガイドにあります。

プロパティーに複数の値がある場合には、プロパティーの最初の値に**「暗号化」**チェック・ボックスが表示されます。**「暗号化」**を選択すると、そのプロパティーのすべての値が暗号化されます。プロパティーの複数の値を暗号化解除するには、そのプロパティーの最初の値の**「暗号化」**チェック・ボックスをクリックしてチェックマークを外してから、**「検証」**ダイアログ・ボックスで新規の値を入力します。入力値が一致すれば、すべての複数値が暗号化解除されます。

更新メソッド

付録 A『*コネクターの標準構成プロパティー*』の 78 ページの『プロパティー値の設定と更新』にある更新メソッドの説明を参照してください。

サポートされるビジネス・オブジェクト定義の指定

コネクターで使用するビジネス・オブジェクトを指定するには、Connector Configurator の**「サポートされているビジネス・オブジェクト」**タブを使用します。汎用ビジネス・オブジェクトと、アプリケーション固有のビジネス・オブジェクトの両方を指定する必要があり、またそれらのビジネス・オブジェクト間のマップの関連を指定することが必要です。

注: コネクターによっては、アプリケーションでイベント通知や (メタオブジェクトを使用した) 追加の構成を実行するために、特定のビジネス・オブジェクトをサポートされているものとして指定することが必要な場合もあります。詳細は、「*コネクター開発ガイド (C++ 用)*」または「*コネクター開発ガイド (Java 用)*」を参照してください。

ご使用のブローカーが ICS の場合

ビジネス・オブジェクト定義がコネクターでサポートされることを指定する場合や、既存のビジネス・オブジェクト定義のサポート設定を変更する場合は、**「サポートされているビジネス・オブジェクト」**タブをクリックし、以下のフィールドを使用してください。

ビジネス・オブジェクト名: ビジネス・オブジェクト定義がコネクターによってサポートされることを指定するには、System Manager を実行し、以下の手順を実行します。

1. **「ビジネス・オブジェクト名」**リストで空のフィールドをクリックします。System Manager プロジェクトに存在するすべてのビジネス・オブジェクト定義を示すドロップダウン・リストが表示されます。
2. 追加するビジネス・オブジェクトをクリックします。
3. ビジネス・オブジェクトの**「エージェント・サポート」**(以下で説明) を設定します。
4. 「Connector Configurator」ウィンドウの「ファイル」メニューで、**「プロジェクトに保管」**をクリックします。追加したビジネス・オブジェクト定義に指定されたサポートを含む、変更されたコネクター定義が、System Manager の ICL (Integration Component Library) プロジェクトに保管されます。

サポートされるリストからビジネス・オブジェクトを削除する場合は、以下の手順を実行します。

1. ビジネス・オブジェクト・フィールドを選択するため、そのビジネス・オブジェクトの左側の番号をクリックします。
2. 「Connector Configurator」ウィンドウの**「編集」**メニューから、**「行を削除」**をクリックします。リスト表示からビジネス・オブジェクトが除去されます。
3. **「ファイル」**メニューから、**「プロジェクトの保管」**をクリックします。

サポートされるリストからビジネス・オブジェクトを削除すると、コネクター定義が変更され、削除されたビジネス・オブジェクトはコネクターのこのインプリメンテーションで使用不可になります。コネクターのコードに影響したり、そのビジネス・オブジェクト定義そのものが System Manager から削除されることはありません。

エージェント・サポート: ビジネス・オブジェクトがエージェント・サポートを備えている場合、システムは、コネクター・エージェントを介してアプリケーションにデータを配布する際にそのビジネス・オブジェクトの使用を試みます。

一般に、コネクターのアプリケーション固有ビジネス・オブジェクトは、そのコネクターのエージェントによってサポートされますが、汎用ビジネス・オブジェクトはサポートされません。

ビジネス・オブジェクトがコネクター・エージェントによってサポートされるよう指定するには、**「エージェント・サポート」**ボックスにチェックマークを付けます。「Connector Configurator」ウィンドウでは「エージェント・サポート」の選択の妥当性は検査されません。

最大トランザクション・レベル: コネクターの最大トランザクション・レベルは、そのコネクターがサポートする最大のトランザクション・レベルです。

ほとんどのコネクターの場合、選択可能な項目は「**最大限の努力**」のみです。

トランザクション・レベルの変更を有効にするには、サーバーを再始動する必要があります。

ご使用のブローカーが WebSphere Message Broker の場合

スタンドアロン・モードで作業している (System Manager に接続していない) 場合、手動でビジネス・オブジェクト名を入力する必要があります。

System Manager を実行している場合、**「サポートされているビジネス・オブジェクト」** タブの**「ビジネス・オブジェクト名」** 列の下にある空のボックスを選択できます。コンボ・ボックスが表示され、コネクターが属する統合コンポーネント・ライブラリー・プロジェクトから選択可能なビジネス・オブジェクトのリストが示されます。このリストから必要なビジネス・オブジェクトを選択します。

「メッセージ・セット ID」は、WebSphere Business Integration Message Broker 5.0 のオプションのフィールドです。この ID が提供される場合、一意である必要はありません。ただし、WebSphere MQ Integrator および Integrator Broker 2.1 の場合は、一意の **ID** を提供する必要があります。

ご使用のブローカーが WAS の場合

使用するブローカー・タイプとして WebSphere Application Server を選択した場合、Connector Configurator にメッセージ・セット ID は必要ありません。**「サポートされているビジネス・オブジェクト」**タブには、サポートされるビジネス・オブジェクトの**「ビジネス・オブジェクト名」**列のみが表示されます。

スタンドアロン・モードで作業している (System Manager に接続していない) 場合、手動でビジネス・オブジェクト名を入力する必要があります。

System Manager を実行している場合、「サポートされているビジネス・オブジェクト」タブの「ビジネス・オブジェクト名」列の下にある空のボックスを選択できます。コンボ・ボックスが表示され、コネクターが属する統合コンポーネント・ライブラリー・プロジェクトから選択可能なビジネス・オブジェクトのリストが示されます。このリストから必要なビジネス・オブジェクトを選択します。

関連付けられたマップ (ICS のみ)

各コネクターは、現在 WebSphere InterChange Server でアクティブなビジネス・オブジェクト定義、およびそれらの関連付けられたマップのリストをサポートします。このリストは、**「関連付けられたマップ」**タブを選択すると表示されます。

ビジネス・オブジェクトのリストには、エージェントでサポートされるアプリケーション固有のビジネス・オブジェクトと、コントローラーがサブスクライブ・コラボレーションに送信する、対応する汎用オブジェクトが含まれます。マップの関連によって、アプリケーション固有のビジネス・オブジェクトを汎用ビジネス・オブジェクトに変換したり、汎用ビジネス・オブジェクトをアプリケーション固有のビジネス・オブジェクトに変換したりするときに、どのマップを使用するかが決定されます。

特定のソースおよび宛先ビジネス・オブジェクトについて一意的に定義されたマップを使用する場合、表示を開くと、マップは常にそれらの該当するビジネス・オブジェクトに関連付けられます。ユーザーがそれらを変更する必要はありません (変更できません)。

サポートされるビジネス・オブジェクトで使用可能なマップが複数ある場合は、そのビジネス・オブジェクトを、使用する必要のあるマップに明示的にバインドすることが必要になります。

「関連付けられたマップ」タブには以下のフィールドが表示されます。

- **ビジネス・オブジェクト名**

  これらは、**「サポートされているビジネス・オブジェクト」**タブで指定した、このコネクターでサポートされるビジネス・オブジェクトです。「サポートされているビジネス・オブジェクト」タブでビジネス・オブジェクトを追加指定した場合、その内容は、「Connector Configurator」ウィンドウの**「ファイル」**メニューから**「プロジェクトに保管」**を選択して、変更を保管した後に、このリストに反映されます。
- **関連付けられたマップ**

この表示には、コネクターの、サポートされるビジネス・オブジェクトでの使用のためにシステムにインストールされたすべてのマップが示されます。各マップのソース・ビジネス・オブジェクトは、**「ビジネス・オブジェクト名」**表示でマップ名の左側に表示されます。

- **明示的**

  場合によっては、関連付けられたマップを明示的にバインドすることが必要になります。

  明示的バインディングが必要なのは、特定のサポートされるビジネス・オブジェクトに複数のマップが存在する場合のみです。ICS は、ブート時、各コネクターでサポートされるそれぞれのビジネス・オブジェクトにマップを自動的にバインドしようとします。複数のマップでその入力データとして同一のビジネス・オブジェクトが使用されている場合、サーバーは、他のマップのスーパーセットである 1 つのマップを見付けて、バインドしようとします。

  他のマップのスーパーセットであるマップがないと、サーバーは、ビジネス・オブジェクトを単一のマップにバインドすることができないため、バインディングを明示的に設定することが必要になります。

  以下の手順を実行して、マップを明示的にバインドします。

  1. **「明示的 (Explicit)」**列で、バインドするマップのチェック・ボックスにチェックマークを付けます。
  2. ビジネス・オブジェクトに関連付けるマップを選択します。
  3. 「Connector Configurator」ウィンドウの**「ファイル」**メニューで、**「プロジェクトに保管」**をクリックします。
  4. プロジェクトを ICS に配置します。
  5. 変更を有効にするため、サーバーをリブートします。

リソース (ICS)

「リソース」タブでは、コネクター・エージェントが、コネクター・エージェント並列処理を使用して同時に複数のプロセスを処理するかどうか、またどの程度処理するかを決定する値を設定できます。

すべてのコネクターがこの機能をサポートしているわけではありません。複数のプロセスを使用するよりも複数のスレッドを使用する方が通常は効率的であるため、Java でマルチスレッドとして設計されたコネクター・エージェントを実行している場合、この機能を使用することはお勧めできません。

メッセージング (ICS)

メッセージング・プロパティーは、`DeliveryTransport` 標準プロパティーの値として MQ を設定し、ブローカー・タイプとして ICS を設定した場合にのみ、使用可能です。これらのプロパティーは、コネクターによるキューの使用方法に影響します。

トレース/ログ・ファイル値の設定

コネクター構成ファイルまたはコネクター定義ファイルを開くと、Connector Configurator は、そのファイルのログおよびトレースの値をデフォルト値として使用します。Connector Configurator 内でこれらの値を変更できます。

ログとトレースの値を変更するには、以下の手順を実行します。

1. **「トレース/ログ・ファイル」**タブをクリックします。
2. ログとトレースのどちらでも、以下のいずれかまたは両方へのメッセージの書き込みを選択できます。
   - コンソールに (STDOUT):
     ログ・メッセージまたはトレース・メッセージを STDOUT ディスプレイに書き込みます。

     注: STDOUT オプションは、Windows プラットフォームで実行しているコネクターの**「トレース/ログ・ファイル」**タブでのみ使用できます。
   - ファイルに:
     ログ・メッセージまたはトレース・メッセージを指定されたファイルに書き込みます。ファイルを指定するには、ディレクトリー・ボタン (省略符号) をクリックし、指定する格納場所に移動し、ファイル名を指定し、**「保管」**をクリックします。ログ・メッセージまたはトレース・メッセージは、指定した場所の指定したファイルに書き込まれます。

     注: ログ・ファイルとトレース・ファイルはどちらも単純なテキスト・ファイルです。任意のファイル拡張子を使用してこれらのファイル名を設定できます。ただし、トレース・ファイルの場合、拡張子として `.trc` ではなく `.trace` を使用することをお勧めします。これは、システム内に存在する可能性がある他のファイルとの混同を避けるためです。ログ・ファイルの場合、通常使用されるファイル拡張子は `.log` および `.txt` です。

データ・ハンドラー

データ・ハンドラー・セクションの構成が使用可能となるのは、DeliveryTransport の値に JMS を、また ContainerManagedEvents の値に JMS を指定した場合のみです。すべてのアダプターでデータ・ハンドラーを使用できるわけではありません。

これらのプロパティーに使用する値については、付録 A『コネクターの標準構成プロパティー』の `ContainerManagedEvents` の下の説明を参照してください。その他の詳細は、「*コネクター開発ガイド (C++ 用)*」または「*コネクター開発ガイド (Java 用)*」を参照してください。

構成ファイルの保管

コネクターの構成が完了したら、コネクター構成ファイルを保管します。Connector Configurator では、構成中に選択したブローカー・モードでファイルを保管します。Connector Configurator のタイトル・バーには現在のブローカー・モード (ICS、WMQI、または WAS) が常に表示されます。

ファイルは XML 文書として保管されます。XML 文書は次の 3 通りの方法で保管できます。

- System Manager から、統合コンポーネント・ライブラリーに *.con 拡張子付きファイルとして保管します。
- 指定したディレクトリーに保管します。
- スタンドアロン・モードで、ディレクトリー・フォルダーに *.cfg 拡張子付きファイルとして保管します。デフォルトでは、このファイルは ¥WebSphereAdapters¥bin¥Data¥App に保管されます。
- WebSphere Application Server プロジェクトをセットアップしている場合には、このファイルを WebSphere Application Server プロジェクトに保管することもできます。

System Manager でのプロジェクトの使用法、および配置の詳細については、以下のインプリメンテーション・ガイドを参照してください。

- ICS: 「*WebSphere InterChange Server システム・インプリメンテーション・ガイド*」
- WebSphere Message Brokers: 「*WebSphere Message Brokers 使用アダプター・インプリメンテーション・ガイド*」
- WAS: 「*アダプター実装ガイド (WebSphere Application Server)*」

構成ファイルの変更

既存の構成ファイルの統合ブローカー設定を変更できます。これにより、他のブローカーで使用する構成ファイルを新規に作成するときに、このファイルをテンプレートとして使用できます。

注: 統合ブローカーを切り替える場合には、ブローカー・モード・プロパティーと同様に他の構成プロパティーも変更する必要があります。

既存の構成ファイルでのブローカーの選択を変更するには、以下の手順を実行します (オプション)。

- Connector Configurator で既存の構成ファイルを開きます。
- **「標準のプロパティー」**タブを選択します。
- 「標準のプロパティー」タブの**「BrokerType」**フィールドで、ご使用のブローカーに合った値を選択します。
  現行値を変更すると、プロパティー画面の利用可能なタブおよびフィールド選択がただちに変更され、選択した新規ブローカーに適したタブとフィールドのみが表示されます。

構成の完了

コネクターの構成ファイルを作成し、そのファイルを変更した後で、コネクターの始動時にコネクターが構成ファイルの位置を特定できるかどうかを確認してください。

これを行うには、コネクターが使用する始動ファイルを開き、コネクター構成ファイルに使用されている格納場所とファイル名が、ファイルに対して指定した名前およびファイルを格納したディレクトリーまたはパスと正確に一致しているかどうかを検証します。

グローバル化環境における Connector Configurator の使用

Connector Configurator はグローバル化され、構成ファイルと統合ブローカー間の文字変換を処理できます。 Connector Configurator では、ネイティブなエンコード方式を使用しています。構成ファイルに書き込む場合は UTF-8 エンコード方式を使用します。

Connector Configurator は、以下の場所で英語以外の文字をサポートします。

- すべての値のフィールド
- ログ・ファイルおよびトレース・ファイル・パス (**「トレース/ログ・ファイル」**タブで指定)

CharacterEncoding および Locale 標準構成プロパティーのドロップ・リストに表示されるのは、サポートされる値の一部のみです。ドロップ・リストに、サポートされる他の値を追加するには、製品ディレクトリーの ¥Data¥Std¥stdConnProps.xml ファイルを手動で変更する必要があります。

例えば、Locale プロパティーの値のリストにロケール en_GB を追加するには、stdConnProps.xml ファイルを開き、以下に太文字で示した行を追加してください。

```
<Property name="Locale"
isRequired="true"
updateMethod="component restart">
                <ValidType>String</ValidType>
            <ValidValues>
                                <Value>ja_JP</Value>
                                <Value>ko_KR</Value>
                                <Value>zh_CN</Value>
                                <Value>zh_TW</Value>
                                <Value>fr_FR</Value>
                                <Value>de_DE</Value>
                                <Value>it_IT</Value>
                                <Value>es_ES</Value>
                                <Value>pt_BR</Value>
                                <Value>en_US</Value>
                                <Value>en_GB</Value>

                    <DefaultValue>en_US</DefaultValue>
            </ValidValues>
    </Property>
```

特記事項

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム（本プログラムを含む）との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Burlingame Laboratory Director
IBM Burlingame Laboratory
577 Airport Blvd., Suite 800
Burlingame, CA 94010
U.S.A

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

著作権使用許諾

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

プログラミング・インターフェース情報

プログラミング・インターフェース情報は、プログラムを使用してアプリケーション・ソフトウェアを作成する際に役立ちます。

一般使用プログラミング・インターフェースにより、お客様はこのプログラム・ツール・サービスを含むアプリケーション・ソフトウェアを書くことができます。

ただし、この情報には、診断、修正、および調整情報が含まれている場合があります。診断、修正、調整情報は、お客様のアプリケーション・ソフトウェアのデバッグ支援のために提供されています。

警告: 診断、修正、調整情報は、変更される場合がありますので、プログラミング・インターフェースとしては使用しないでください。

商標

以下は、IBM Corporation の商標です。

IBM
IBM ロゴ
AIX
CrossWorlds
DB2
DB2 Universal Database
Lotus
Lotus Domino
Lotus Notes
MQIntegrator
MQSeries
Tivoli
WebSphere

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

MMX、Pentium および ProShare は、Intel Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

WebSphere Business Integration Adapter Framework V2.4.0

Printed in Japan

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12