

**IBM WebSphere Business Integration
Adapters**



Adapter for JMS ユーザーズ・ガイド

V 2.5.x

**IBM WebSphere Business Integration
Adapters**



Adapter for JMS ユーザーズ・ガイド

V 2.5.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、95 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 JMS バージョン 2.5.x、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

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 JMS User Guide V 2.5.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 2004.1

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

© Copyright International Business Machines Corporation 2000, 2003. All rights reserved.

Translation: © Copyright IBM Japan 2004

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 テクノロジーやエンタープライズ・アプリケーション向けに統合コネクティビティを提供します。

本書では、IBM WebSphere Business Integration Adapter for JMS 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

本書の読者は、WebSphere Business Integration システム、ビジネス・オブジェクトとコラボレーションの開発、および JMS アプリケーションについて十分な知識と経験を持っている必要があります。

関連資料

本製品とともに提供される一連の資料では、WebSphere アダプターのどのインストールにも共通の機能とコンポーネントを説明するとともに、特定のコンポーネントに関する参考資料についても記載しています。

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。

- 一般のアダプター情報について、WebSphere メッセージ・ブローカー (WebSphere MQ Integrator、WebSphere MQ Integrator Broker、WebSphere Business Integration Message Broker) を搭載したアダプターを使用する場合、および WebSphere Application Server を搭載したアダプターを使用する場合は、次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/wicsserver/infocenter>
 - <http://www.ibm.com/websphere/integration/wbicollaborations/infocenter>

上記のサイトには資料のダウンロード、インストール、および表示に関する簡単な説明が記載されています。

表記上の規則

本書は下記の規則に従って編集されています。

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。
斜体	変数名または相互参照を示します。
青のアウトライン	青のアウトラインは、マニュアルをオンラインで表示するときのみ見られるもので、相互参照用のハイパーリンクを示します。アウトラインの内側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入力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)
ProductDir	製品のインストール先ディレクトリーを表します。
/, \	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 (¥) はスラッシュ (/) に置き換えてください。すべての WebSphere Business Integration システムのパス名は、ご使用のシステムで WebSphere Business Integration システム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。
UNIX: および Windows:	これらのいずれかで始まる段落は、オペレーティング・システム間の相違を列挙した注記です。
u	この記号は、 UNIX/Windows の段落の終わりを示します。また、複数段落にわたる注記の終わりを示します。
%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 環境での同等の表記は \$text です。これは、UNIX 環境変数 text の値を示します。

目次

本書について	iii
対象読者	iii
本書の前提条件	iii
関連資料	iii
表記上の規則	iv
本リリースの新機能	vii
リリース 2.5 の新機能	vii
リリース 2.4.x の新機能	vii
リリース 2.3.x の新機能	vii
リリース 2.2.x の新機能	viii
リリース 2.1.x の新機能	viii
リリース 1.3.x の新機能	viii
リリース 1.2.x の新機能	ix
リリース 1.1.x の新機能	ix
第 1 章 Adapter for JMS の概要	1
Adapter for JMS の環境	2
Adapter for JMS の用語	3
Connector for JMS のアーキテクチャー	4
メッセージ処理	4
第 2 章 アダプターのインストールおよび構成	19
インストール・タスク	19
アダプターと関連ファイルのインストール	19
インストール済みファイルの構造	19
コネクタ構成	21
コネクタ・プロパティの構成	22
メタオブジェクトの構成	29
開始スクリプトの構成	40
複数のコネクタ・インスタンスの作成	40
コネクタの始動	42
コネクタの停止	43
第 3 章 ビジネス・オブジェクトの作成または変更	45
コネクタのビジネス・オブジェクトの構造	45
第 4 章 トラブルシューティング	47
エラー処理	47
トレース	48
開始に関する問題の修正	49
付録 A. コネクタの標準構成プロパティ	51
新規プロパティと削除されたプロパティ	51
標準コネクタ・プロパティの構成	51
標準プロパティの要約	53
標準構成プロパティ	56
付録 B. Connector Configurator	69
Connector Configurator の概要	69

Connector Configurator の始動	70
System Manager からの Configurator の実行	71
コネクタ固有のプロパティ・テンプレートの作成	71
新しい構成ファイルを作成	74
既存ファイルの使用	75
構成ファイルの完成	76
構成ファイル・プロパティの設定	77
構成ファイルの保管	84
構成ファイルの変更	84
構成の完了	85
グローバル化環境における Connector Configurator の使用	85
付録 C. チュートリアル.	87
チュートリアルの概要	87
環境のセットアップ	88
シナリオの実行	90
付録 D. トピック・ベースおよびキュー・ベースのメッセージングの構成	93
キュー・ベース・メッセージングの構成	93
トピック・ベース・メッセージングの構成	94
特記事項	95
プログラミング・インターフェース情報	96
商標	97

本リリースの新機能

リリース 2.5 の新機能

アダプターは、WebSphere Integration Message Broker を統合ブローカーとして使用できるようになりました。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。

2.5.x バージョンから、Adapter for JMS は Microsoft Windows NT 上ではサポートされなくなりました。

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、第 2 章を参照してください。

アダプターは、JMS 規格で定義されているパブリッシュ/サブスクライブ (トピック・ベース) のメッセージング・スタイル、および point-to-point (キュー・ベース) のメッセージング・スタイルをサポートするようになりました。アダプターの 1 つのインスタンスは、1 つのメッセージング・スタイルしかサポートしません。トピックとキューの両方を同じ構成で指定することはできません。ただし、トピック・ベースのメッセージング・スタイルをインプリメントする 1 つ以上のインスタンスおよびキュー・ベースのスタイルをインプリメントする 1 つ以上のインスタンスを使用し、アダプターの複数のインスタンスを実行することによって、両方のメッセージング・スタイルをサポートすることができます。

リリース 2.4.x の新機能

アダプターは、WebSphere Application Server を統合ブローカーとして使用できるようになりました。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。

コネクタは、以下のプラットフォーム上で実行されます。

- Microsoft Windows NT 4.0 Service Pack 6A または Windows 2000
- Solaris 7、8 または AIX 5.1、5.2 または HP UX 11.i

リリース 2.3.x の新機能

2003 年 3 月更新。「CrossWorlds」という名称は、システム全体を指し示すため、あるいはほとんど変更されていないコンポーネントまたはツールの名称だけを変更するためには使用されなくなりました。例えば、「CrossWorlds System Manager」はこのリリースでは「System Manager」と表記され、「CrossWorlds InterChange Server」はこのリリースでは「WebSphere InterChange Server」と表記されます。

データ・ハンドラーを入力キューと関連付けることができるようになりました。詳しくは、35 ページの『入力宛先へのデータ・ハンドラーのマッピング』を参照してください。

保証付きイベント・デリバリー機能が拡張されました。詳細については、「コネクタ開発ガイド (Java 用)」を参照してください。

リリース 2.2.x の新機能

InProgress キューが不要となり、使用不可になりました。詳細については、26 ページの『InProgressDestination』を参照してください。

ReplyToQueue を、ReplyToQueue コネクタ・プロパティではなく、動的子メタオブジェクトで指定できるようになりました。詳細については、38 ページの『JMS ヘッダーおよび動的子メタオブジェクト属性』を参照してください。

メッセージ・セレクターを使用して、要求に対する応答メッセージの識別、フィルター操作、およびアダプターでの識別方法を制御できます。この JMS 機能は、同期要求処理のみに適用されます。詳しくは、14 ページの『同期処理』を参照してください。

リリース 2.1.x の新機能

コネクタは国際化されました。詳細については、3 ページの『ロケール依存データ』および 51 ページの『付録 A. コネクタの標準構成プロパティ』を参照してください。

本書では、このアダプターと ICS の併用について説明します。

注: 保証付きイベント・デリバリー機能を使用するには、ICS のリリース 4.1.1.2 をインストールする必要があります。

リリース 1.3.x の新機能

IBM WebSphere Business Integration Adapter for JMS は Connector for JMS を含みます。このアダプターは、InterChange Server (ICS) 統合ブローカーとともに動作します。異機種のアプリケーション間の統合を実現するアプリケーションである統合ブローカーは、データ・ルーティングなどの各種サービスを提供します。アダプターは以下のものを含みます。

- JMS に固有のアプリケーション・コンポーネント
- ビジネス・オブジェクトのサンプル
- 以下のコンポーネントで構成されている IBM WebSphere Adapter Framework
 - コネクタ・フレームワーク
 - 開発ツール (Business Object Designer、IBM CrossWorlds System Manager など)
 - API (CDK など)

本書では、このアダプターと ICS の併用について説明します。

重要: 本アダプターに含まれるコネクタは国際化に対応していないため、ISO Latin-1 データのみが処理されるという保証がない場合には、コネクタを ICS バージョン 4.1.1 とともに実行しないでください。

コネクタは現在、AIX 4.3.3 Patch Level 7 をサポートしています。

リリース 1.2.x の新機能

Connector for JMS の前のバージョンでは、JMS メッセージと IBM CrossWorlds ビジネス・オブジェクトの間でのデータ変換に使用されるデータ・ハンドラーは、コネクタ・プロパティ `DataHandlerConfigMO` および `DataHandlerMimeType` によって決定されていました。このため、Connector は各種のデータ形式を処理するために複数のインスタンスが必要になるという制約がありました。リリース 1.2.x では、これらのプロパティをコネクタの静的なメタオブジェクト内または要求ビジネス・オブジェクトの動的な子メタオブジェクト内にオプションで指定することが可能になりました。詳細については、29 ページの『メタオブジェクトの構成』を参照してください。

リリース 1.1.x の新機能

このリリースの資料には、以下の新機能と製品拡張についての情報が含まれます。

- コネクタは、コネクタに渡されるビジネス・オブジェクトに子として追加される動的なメタオブジェクトから、変換プロパティを認識し、読み取ります。この動的な子メタオブジェクトの属性値は、前バージョンでコネクタの構成に使用される静的なメタオブジェクトに指定可能であった変換プロパティと重複します。静的なメタオブジェクトを指定するコネクタ・プロパティは不要になりましたが、まだ使用可能です。動的な子メタオブジェクトは、静的なメタオブジェクトとは無関係に使用することができ、その逆もまた同じです。
- コネクタは、コネクタ・プロパティ `InputQueue` の複数のキュー名を受け入れます。コネクタは、ラウンドロビン方式でキューをポーリングして、各キューから最大 `pollQuantity` 個のメッセージを検索します。複数のキュー名はセミコロンで区切られます。

第 1 章 Adapter for JMS の概要

- 2 ページの『Adapter for JMS の環境』
- 3 ページの『Adapter for JMS の用語』
- 4 ページの『Connector for JMS のアーキテクチャー』
- 4 ページの『メッセージ処理』

Connector for JMS は、IBM WebSphere Business Integration Adapter for JMS のランタイム・コンポーネントです。コネクターを使用すると、IBM WebSphere 統合ブローカーは、JMS メッセージ形式のデータを送受信するアプリケーションとの間でビジネス・オブジェクトを交換することができます。

JMS は、企業メッセージング・システムにアクセスするためのオープン・スタンダード API です。これは、ビジネス・アプリケーションがビジネス・データとイベントを送受信できるように設計されています。

コネクターは、アプリケーション固有のコンポーネントおよびコネクター・フレームワークから構成されます。アプリケーション固有のコンポーネントには、特定のアプリケーションに合わせて作成されたコードが含まれます。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の間の仲介役として機能し、そのコードはどのコネクターにも共通です。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の間で、以下のサービスを提供します。

- ビジネス・オブジェクトの送受信
- 始動メッセージおよび管理メッセージの交換の管理

本書では、アプリケーション固有のコンポーネントおよびコネクター・フレームワークについて説明します。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。

注: WebSphere Business Integration Adapter は、いずれも統合ブローカーとともに動作します。JMS 用のコネクターは、以下の統合ブローカーとともに動作します。

- InterChange Server 統合ブローカー。詳細については、「*テクニカル入門 (IBM WebSphere InterChange Server)*」を参照してください。
- WebSphere MQ message brokers。詳細は、「*Implementing Adapters with WebSphere Message Brokers*」を参照してください。
- WebSphere Application Server (WAS) 統合ブローカー。詳細は、「*アダプター実装ガイド (WebSphere Application Server)*」を参照してください。

Adapter for JMS の環境

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』
- 『アダプターの規格』
- 3 ページの『アダプターのプラットフォーム』
- 3 ページの『アダプターの依存関係』
- 3 ページの『ロケール依存データ』

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。JMS のアダプターのバージョン 2.5 は、以下のアダプター・フレームワークと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 の以下のバージョン:
 - 2.2.0
 - 2.3.0
 - 2.3.1
 - 2.4.0
- 統合ブローカー:
 - WebSphere InterChange Server の以下のバージョン:
 - 4.11
 - 4.2
 - 4.2.1
 - 4.2.x
 - 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 5.0.1 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。

アダプターの規格

アダプターの規格は JMS 1.0.2 規格に記載されています。その他のバージョンの規格については、サポートは検証されていませんが、現在、サポートを妨げる既知の問題はありません。

アダプターは、JMS 規格で定義された point-to-point (PTP) メッセージングおよびパブリッシュ/サブスクライブ (Pub/Sub) メッセージングの両方のインターフェースをサポートします。これらのスタイルは一般に、それぞれ、キュー・ベース・メッセージングおよびトピック・ベース・メッセージングと呼ばれています。アダプター

の 1 つのインスタンスは、同時に 1 つのメッセージング・スタイルしかサポートしません (つまり、トピックとキューを混合して構成することはできません)。ただし、PTP または Pub/Sub のいずれかに構成されたインスタンスを使用し、アダプターの複数のインスタンスを平行して実行することにより、両方のメッセージング・スタイルをサポートすることはできます。

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でサポートされます。

- Windows 2000
- AIX 5.1、5.2
- Solaris 7、8
- HP-UX 11i

アダプターの依存関係

アダプターは、データベースを使用しません。また、データベースに依存しません。JMS プロバイダーおよび JNDI プロバイダーが必要とするすべてのクライアント・ライブラリーは、アダプター・クラスパスに入れる必要があります。これらのライブラリーはプロバイダーによって異なります。

ロケール依存データ

コネクターは、2 バイト文字セットをサポートして、指定された言語でメッセージ・テキストを送れるように国際化されています。ある文字コードを使用する場所から別の文字コード・セットを使用する場所へデータを転送する場合、コネクターは、そのデータの意味が伝わるように文字変換を実行します。

Java 仮想マシン (JVM) 内での Java ランタイム環境は、Unicode 文字コード・セットでデータを表します。Unicode には、最も一般的な文字コード・セット (単一バイトおよびマルチバイトの両方) の文字のエンコードが含まれています。WebSphere Business Integration システムの大部分のコンポーネントは Java で作成されています。したがって、大部分のインテグレーション・コンポーネント間で行われるデータ転送には、文字変換は必要ありません。

エラー・メッセージや通知メッセージを個々の国や地域に合った適切な言語で記録するには、個々の環境に合わせて Locale 標準構成プロパティを構成する必要があります。構成プロパティの詳細については、51 ページの『付録 A. コネクターの標準構成プロパティ』を参照してください。

Adapter for JMS の用語

- **JMS プロバイダー**。JMS をインプリメントするメッセージング・システム。
- **メッセージ**。エンタープライズ・アプリケーションによって使用されるビジネス・データが入った要求およびイベント。
- **PTP**。point-to-point スタイルまたはキュー・ベースのメッセージング。
- **Pub/Sub**。パブリッシュ/サブスクライブ・スタイルまたはトピック・ベースのメッセージング。

- **JMS 宛先**。メッセージのソースまたはターゲットを表す。PTP メッセージングでは、宛先はキューです。Pub/Sub では、宛先はトピックです。この用語は、特定の状況でキューまたはトピックが使用されるときに、説明および実際のプロパティ名を明記する両方の仕様で、広く使用されます。
- **ASI**。アプリケーション固有情報。— ビジネス・オブジェクトおよびメタオブジェクトにおいて、セミコロンで区切られた `name=value` ペアとして表されるメタデータ。

Connector for JMS のアーキテクチャー

メッセージは、このアダプターとの関係においては、エンタープライズ・アプリケーションによって使用されるビジネス・データが入った要求およびイベントです。Message Oriented Middleware プロダクト (MOM) を使用すると、エンタープライズ・アプリケーションは、お互いに非同期方式でメッセージの送受信を行うことができます。Java Message Service (JMS) API は、Java プログラムがこのようなメッセージング・システムと通信する方法を標準化するために定められました。以前は、しばしば、単一の特定 MOM システムと一緒に動作するメッセージング・クライアントが作成されました。アダプターなどの JMS クライアントは、一般に、JMS サポートを提供するすべてのメッセージング・システムを利用できます。JMS の WBI アダプターを使用すると、JMS 規格をサポートする、急増しているエンタープライズ・メッセージング・システムと統合できるようになります。

メッセージ処理

アダプターは、以下の 2 つの主要な操作をサポートします。

1. JMS 宛先からのメッセージの検索
2. JMS 宛先へのメッセージの配信

アダプターは、JMS プロバイダー (WebSphere MQ など) への接続を確立して両方の操作を行い、次に JMS API を使用して以下のことを行います。

- JMS 宛先の既存メッセージのポーリングおよび検索
- ブローカーが要求した新しいメッセージの生成および配信

これらの 2 つの操作については、『イベント・メッセージの処理』 および 11 ページの『要求メッセージの処理』 で詳しく説明します。

イベント・メッセージの処理

コネクタは、新しいメッセージが、1 つ以上の JMS 宛先に配信されているか定期的に検査します。各ポーリング・サイクルで、コネクタは以下のことを行います。

1. JMS API を使用して、待機メッセージを検索する。
2. 構成済みデータ・ハンドラーを呼び出し、メッセージの内容をビジネス・オブジェクトに変換する。
3. サブスクライブしているビジネス・プロセスが処理できるように、イベント・ビジネス・オブジェクトを構成済み統合ブローカーに配信または公表する。

これらのステップについては、図 1 に示されています。また、以下で詳しく説明されています。

- 『イベント検出』
- 6 ページの『イベント状況およびリカバリー』
- 8 ページの『イベント検索』

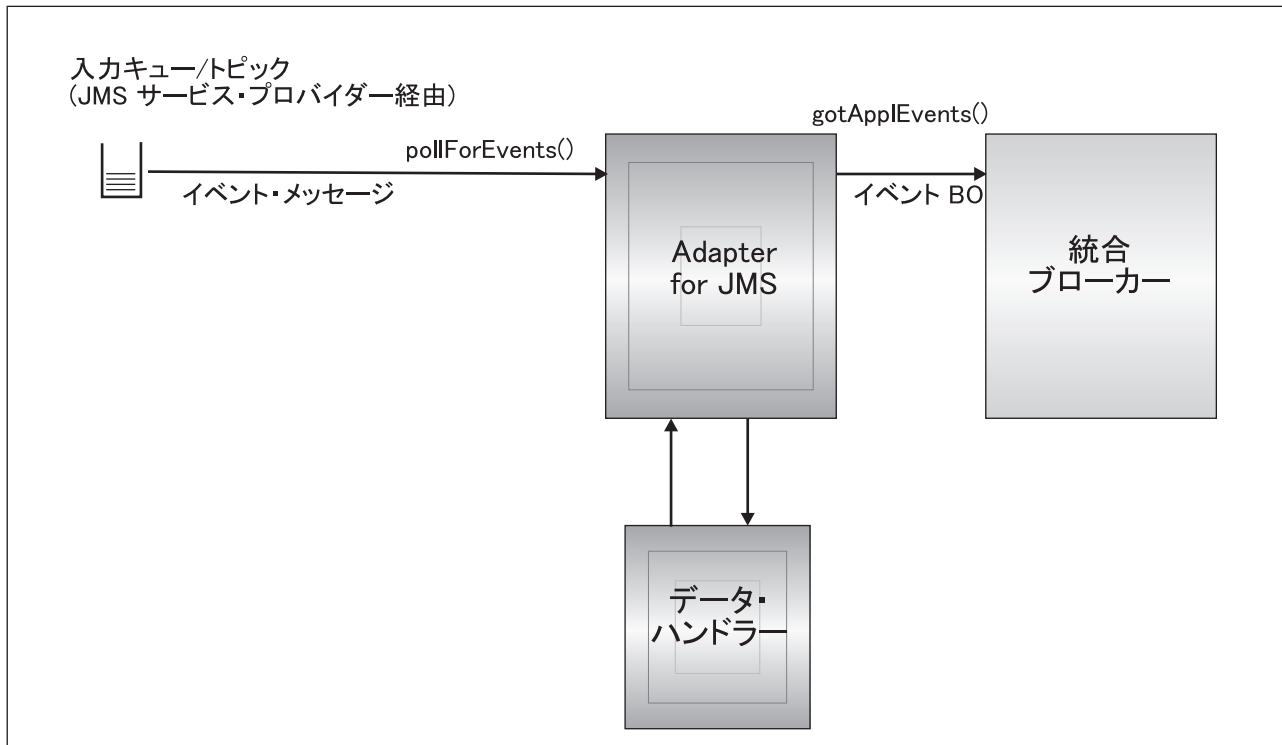


図 1. イベント・メッセージ・フロー

イベント検出

各イベント・ポーリング・サイクル中に、コネクターは、コネクター・プロパティ `InputDestination` によって指定された宛先で、メッセージの非ブロッキング読み取りを行います (コネクター・プロパティの詳細については、22 ページの『コネクター・プロパティの構成』を参照)。コネクターはメッセージを検索してから、ブローカーに公表します。

コネクターは `pollForEvents()` メソッドを使用して、一定の間隔でポーリングしてメッセージの有無を確認します。ポーリング・サイクルごとのメッセージ検索数は、コネクター・プロパティ `PollQuantity` に指定されている最大数に制限されます。指定最大数に達する前に、すべての使用可能なメッセージを検索すると、コネクターは、それ以上のメッセージを待たずに、ポーリング・サイクルから即時に戻ります。

コネクター・プロパティ `InputDestination` で複数の宛先が指定されている場合、コネクターは、指定された各宛先をラウンドロビン方式でポーリングします。各宛先で `PollQuantity` に指定してある最大数のメッセージを検索し、ブローカーに公表し

ます。PollQuantity に指定されている最大数に達する前にすべての宛先が空になると、コネクタはポーリング・サイクルから即時に戻ります。

例えば、次のようなシナリオでは、

- コネクタが、PollQuantity 値を 2、および入力キュー A、B、および C で構成されている
- キュー A にはメッセージが 2 つ、キュー B にはメッセージが 1 つ、キュー C にはメッセージが 5 つ含まれる

アダプターは、単一のポーリング・サイクルにおいて以下の順序でメッセージを検索します。

1. キュー A の次のメッセージ (メッセージが 1 つ残る)
2. キュー B の次のメッセージ (空になる)
3. キュー C の次のメッセージ (メッセージが 4 つ残る)
4. キュー A の次のメッセージ (空になる)
5. コネクタがキュー B をチェックするが、空のまま
6. キュー C の次のメッセージ (メッセージが 3 つ残る)

各キューから最大数 (PollQuantity で指定) の 2 つのメッセージをポーリングしたので、アダプターはポーリング・サイクルから戻ります。

イベント状況およびリカバリー

イベント・メッセージの検索はトランザクションの一部です。トランザクションをコミットする前にコネクタが予期せずに終了する場合、トランザクションはロールバックされ、元のメッセージが復元されます。コネクタ・フレームワークは現在、分散トランザクションをサポートしていないため、コネクタは、ブローカーにイベントを公表しても、ブローカーからの確認通知を受信する前に、予期せずに終了するか通信を切断する場合があります。この場合、イベントがブローカーに受信されたかどうかは、コネクタが入手できる情報からは確認できません。イベント・メッセージを失わないようにするために、ブローカーからイベントの受取を確認する応答をら受信するまでは、コネクタはトランザクションをコミットしません。コネクタがイベントを公表し、確認通知を受信するまでの間に障害が発生すると、トランザクションは自動的にロールバックされ、元のメッセージが復元されます。メッセージがブローカーによって処理されたかどうかはわからないため、そのようなイベントは未確定イベントと呼ばれます。

再始動時に、コネクタは、入力宛先からのメッセージの処理を開始し、未確定イベントを再サブミットします。この方式を使用すると、イベントを失うリスクはなくなりますが、同じイベントが 2 回公表される可能性は避けられません。

重複イベント配信のリスクを軽減したり、なくしたりする方法には、次の 2 つの方法があります。進行中の宛先の使用 (『進行中の宛先によるリカバリー』を参照) または保証付きイベント・デリバリーの使用 (7 ページの『保証付きイベント・デリバリーによるリカバリー』を参照) です。

進行中の宛先によるリカバリー: 未確定イベントの処理を制御するには、コネクタ・プロパティ `InProgressDestination` を指定して、別個の一時宛先を作成します。

注: 進行中の宛先のリカバリーは、Pub/Sub スタイルのメッセージングではサポートされていません。

ブローカーにイベントを公表する前に、コネクタは、イベント・メッセージを入力宛先から進行中の宛先へ移動します。ブローカーから確認通知を受信したら、コネクタは、進行中の宛先からメッセージを削除します。これによって、処理されていない未確定メッセージを分離できます。始動時に進行中の宛先にメッセージがあった場合、コネクタは、これらのメッセージは、予期せず終了したコネクタの以前のインスタンスから残されたものであるとみなすことができるため、安全です。そのようなメッセージに、コネクタが別の処置を行うように指定できます(重複イベント通知が受諾不能な場合)。これを行うには、以下の 4 つのオプションのいずれかを、コネクタ構成プロパティ `InDoubtEvents` に指定します。

- **Fail on startup** 初期設定中に進行中の宛先からメッセージが検出された場合に、コネクタはエラー・ログを記録し、すぐにシャットダウンします。ユーザーまたはシステム管理者は、メッセージを調べ、これらのメッセージをすべて削除するか、それらを別の場所に移動するか、いずれかの適切な処置を行います。
- **Reprocess** 初期設定中に進行中の宛先からメッセージが検出された場合に、コネクタは、以降のポーリングでこれらのメッセージを最初に処理します。進行中の宛先のすべてのメッセージの処理が完了すると、コネクタは入力宛先のメッセージの処理を開始します。
- **Ignore** 初期設定中に進行中の宛先からメッセージが検出された場合に、コネクタはそれらを無視しますが、シャットダウンはしません。
- **Log error** `Log error` オプションを使用すると、初期設定中に進行中の宛先からメッセージが検出された場合に、コネクタはエラー・ログを記録しますが、シャットダウンはしません。

詳細については、25 ページの『`InDoubtEvents`』を参照してください。

保証付きイベント・デリバリーによるリカバリー: 保証付きイベント・デリバリー機能により、コネクタ・フレームワークは、イベントが逸失したり、イベントが 2 度送信されたりするのを防ぐことができます。コネクタ・フレームワークは、コンテナ管理イベント (CME) および重複イベント除去 (DEE) の、2 つの機構によって保証付きイベント・デリバリーをサポートします。

コンテナ管理イベント (CME): コネクタが PTP スタイル・メッセージングに構成されている場合に、CME を使用できます。Pub/Sub スタイル・メッセージングに構成されている場合、コネクタは CME をサポートしません。この保証付きイベント・デリバリー方式の動作方法についての詳細は、「コネクタ開発ガイド (Java 用)」を参照してください。 `ContainerManagedEvents` コネクタ・プロパティの詳細については、58 ページの『`ContainerManagedEvents`』を参照してください。

重複イベント除去 (DEE): JMS アダプターに保証付きイベント・デリバリーをインプリメントするときは、DEE 方式をお勧めします。DEE は Pub/Sub スタイルのメッセージングをサポートする唯一の方法でもあります。

DEE では、コネクタは、ブローカーに公表する各イベントに固有 ID を組み込みます。フレームワークは、コネクタが同じイベント ID を連続してサブミットしていないかチェックします。連続してサブミットされると、フレームワークは、コネクタが同じイベントを 2 度公表しているとみなし、2 番目のサブミットを廃棄

します。PTP スタイルのメッセージングに関しては、DEE は進行中の宛先から、またはその宛先へのメッセージをコピーするオーバーヘッドをかなり削減します。

このコネクタは、ビジネス・オブジェクトをブローカーに公表するときに、すべてのイベントのメッセージ ID を組み込みます。通信障害または予期せぬ終了によって、コネクタがイベントをブローカーに正常に公表できない場合は、前に説明したように、元のメッセージが入力キューにロールバックされます。コネクタは再始動時に、すべての未確定メッセージを含め、キューのイベントの再サブミットを開始します。DEE が使用可能になっていれば、以前ブローカーに正常に到達した未確定メッセージは、すべて廃棄されます。これによって、各メッセージをブローカーに 1 度だけ公表するようになります。

DEE を使用する場合、コネクタがオフラインの間は、宛先内のメッセージの順序を変更しないようにする必要があります。DEE は、アダプターによって検索された最後のメッセージ ID のみを記録します。アダプターが再始動する前に、高い優先順位を持った新しいメッセージが、最後の未確定メッセージの順序をキュー内で押し下げるなどの場合に、DEE は失敗します。

DEE およびその使用可能化については、「コネクタ開発ガイド (Java 用)」を参照してください。DuplicateEventElimination コネクタ・プロパティの詳細については、61 ページの『DuplicateEventElimination』を参照してください。

イベント検索

イベント検索には、コネクタによるイベントの典型的処理が含まれます。イベント検索は、着信イベントが検出されると始まり、それがターゲット・アプリケーションに適したフォーマットに変換され、指定された統合ブローカーに正常に配信されると終了します。コネクタは、すべてのイベントをブローカーに非同期的に（「応答不要送信」で）配信します。

以下のセクションでイベント検索について説明します。

- 『メタデータおよびメタオブジェクト』
- 9 ページの『ビジネス・オブジェクトのマッピング』
- 10 ページの『メッセージ・ヘッダー・マッピングの理解』
- 10 ページの『アーカイブ』
- 10 ページの『エラー・リカバリー』

メタデータおよびメタオブジェクト: メッセージを正常にビジネス・オブジェクトに変換したり、その逆を行ったりするには、コネクタは、メタデータと呼ばれる追加情報を必要とします。メタデータは、オブジェクト、メッセージ、またはアプリケーション内のデータをどのように表現するか、またはそれらをどのように処理するかを説明します。メタデータには、コネクタが宛先 XYZ からメッセージを検索した場合、どのビジネス・オブジェクトを作成するか、または動詞 Create を持ったタイプ Customer の要求ビジネス・オブジェクトをシリアルライズするときに、どのデータ・ハンドラーを使用するか、などの詳細が含まれています。

属性、プロパティ、動詞、およびアプリケーション固有情報が、ビジネス・オブジェクト定義のメタデータを構成します。さらに、宛先、データ・フォーマット、データ・ハンドラーなどのメタデータを含んだ、1 つ以上のメタオブジェクトを指定できます。

メタオブジェクトには、静的タイプおよび動的タイプの 2 つのタイプがあります。インプリメンテーションのときには、静的メタオブジェクトを作成します。静的メタオブジェクトには、コネクタがサポートしなければならない各ビジネス・オブジェクト・タイプにメタデータを提供する属性が含まれます。静的メタオブジェクトはコネクタ固有プロパティで指定され、初期設定のときにコネクタによって読み取られます。メタオブジェクト・プロパティの概要、およびそれらがメッセージ変換にどのような影響を与えるかについては、『ビジネス・オブジェクトのマッピング』および 10 ページの『メッセージ・ヘッダー・マッピングの理解』を参照してください。

もう一つのメタオブジェクト・タイプは動的メタオブジェクトです。このメタオブジェクトを使用すると、要求処理のときに、要求ごとに、アダプターが使用するメタデータを変更してビジネス・オブジェクトを処理できます。イベント処理のときに、動的メタオブジェクトは、イベントに関するトランスポート固有情報 (メッセージ ID、優先順位など) を保持するコンテナーとして機能します。このため、ダウンストリーム・ビジネス・プロセスはそれらのビジネス・ロジックでその情報を使用できます。動的メタオブジェクトは、イベント (または要求) のトップレベル・オブジェクトで定義される、特別にマークされた子オブジェクトとして表されます。

メタオブジェクトは、同じインプリメンテーションにおいて、どちらか 1 つを使用することも、両方を使用することもできます。一般に、動的メタオブジェクトで指定された値は、静的メタオブジェクトで指定された値に優先します。メタデータの詳細については、「コネクタ開発ガイド (Java 用)」を参照してください。静的および動的メタオブジェクトの構成については、29 ページの『メタオブジェクトの構成』を参照してください。

ビジネス・オブジェクトのマッピング: メッセージの検索のときに、コネクタは、メッセージをどのビジネス・オブジェクトにマップすべきかを確認しようとします。

デフォルトでは、コネクタは、コネクタ・プロパティに設定されているデータ・ハンドラーが、ビジネス・オブジェクト・タイプを決定できるようにします。コネクタは、メッセージ本文をデータ・ハンドラーに渡し、データ・ハンドラーが戻したビジネス・オブジェクトをブローカーに公表します。データ・ハンドラーが適切なビジネス・オブジェクトを決定できない場合、コネクタはイベントに失敗します。

コネクタ構成プロパティ `ConfigurationMetaObject` に静的メタオブジェクトが指定されている場合、コネクタはこのオブジェクトを検索して、入力フォーマットまたは入力宛先に関してメッセージに一致するルールを見付けます。メタオブジェクトで指定されたルールが入力フォーマットおよび入力宛先の両方を指定している場合、メッセージがそれらのプロパティの両方に一致する場合にのみ、コネクタはこのルールに従います。これらのプロパティのどちらかが欠落している場合、コネクタは指定されたプロパティのみを使用します。

例えば、入力フォーマット `Cust_In` を持った、入力宛先 `MyInputDest` のメッセージは、以下の静的メタオブジェクト・ルールに一致します。

1. `InputFormat=Cust_In;InputDestination=MyInputDest`
2. `InputFormat=Cust_In`

3. InputDestination=MyInputDest

イベント・メッセージを 1 つのルールに一致させることができる場合、コネクタは、ビジネス・オブジェクトの新しいインスタンスを作成し、それをメッセージ本文と一緒に、ルールで指定されたデータ・ハンドラーに渡すことによって、このビジネス・オブジェクトを書き取らせます。データ・ハンドラーがルールに指定されていない場合、コネクタは、コネクタ構成プロパティで指定されたデフォルトのデータ・ハンドラーを使用します。

アダプターが、イベント・メッセージを複数のルールに一致させることができる場合、または 1 つのルールにも一致させることができない場合、コネクタは、コネクタ構成プロパティで指定されたデータ・ハンドラーにメッセージ本文のみを渡すことによって、データ・ハンドラーが、ビジネス・オブジェクト・タイプを決定できるようにします。

メッセージ・ヘッダー・マッピングの理解: イベント・メッセージをビジネス・オブジェクトに変換するために、コネクタは、ビジネス・オブジェクトに関するメタデータとメッセージに関するメタデータを比較し、それらをマッピングします。8 ページの『メタデータおよびメタオブジェクト』で説明したように、ビジネス・オブジェクトに関するメタデータは、ビジネス・オブジェクト定義 (アプリケーション固有情報および子動的メタオブジェクト)、コネクタ・プロパティ、および静的メタオブジェクトの中にあります。メッセージ・メタデータはメッセージ・ヘッダーの中にあります。

トランスポート固有メッセージ・ヘッダー情報へアクセスしたり、メッセージ・トランスポートの詳細情報を入手したり、それを詳細に制御したりするには、ビジネス・オブジェクト定義の子である動的メタオブジェクトに属性を追加します。属性を追加すると、メッセージ・ヘッダーから読み取りができるようになり、オプションで書き込みもできるようになるため、メッセージ・メタデータを変更できるようになります。そのような変更では、JMS プロパティを変更したり、要求ごとに宛先を制御したり (アダプター・プロパティで指定されたデフォルトの宛先を使用せずに)、メッセージの CorrelationID を再ターゲットしたりできます。ビジネス・オブジェクト定義の子である動的メタオブジェクトにそのようなプロパティを指定すると、コネクタは、メッセージ・ヘッダーでそれらに対応するものをチェックし、メッセージ・ヘッダーの内容に基づいて、動的メタオブジェクトにデータを取り込みます。1 つまたはすべてのサポートされた動的メタオブジェクト属性を定義できます。コネクタはそれに従って、メタオブジェクトにデータを取り込みます。読み取りまたは書き込みができるメッセージ・ヘッダー・プロパティのリストを含め、詳細については、37 ページの『ポーリング中の動的な子メタオブジェクトの含まれるデータ』を参照してください。

アーカイブ: コネクタ固有プロパティ `ArchiveDestination` を指定すると、コネクタは、正常に処理されたすべてのメッセージのコピーをこの宛先に置きます。`ArchiveDestination` が未定義の場合、正常に処理されたメッセージは廃棄されます。詳細については、22 ページの『コネクタ固有プロパティの構成』を参照してください。

エラー・リカバリー: 入力宛先からの読み取りでエラーを検出すると、コネクタは、定数値 `APPRESPONSETIMEOUT` をすぐにブローカーに戻します。これにより、コネクタは終了し、場合により再始動します。一般に、そのようなリカバリ

ー不能エラーは、JMS プロバイダーへの接続が切断されたか、あるいはコネクタ
ーが認識できないか、認識してもリカバリー不能 (トランザクションの失敗など) とみ
なした、JMS プロバイダーによって報告された内部エラーかのいずれかが原因で
す。

インバウンド・メッセージをイベント・ビジネス・オブジェクトに変換している
ときにエラーを検出した場合 (データ・ハンドラーが無効なメッセージ・フォー
マットを報告する場合など)、コネクタはイベントに失敗し、理由を説明する該当
するエラー・メッセージのログを記録します。コネクタ・プロパティ
`ErrorDestination` が定義されており、有効な場合、コネクタは、失敗したメ
ッセージのコピーをこのエラー宛先に置きます。そうでない場合、メッセージは
廃棄されます。

コネクタがイベント・ビジネス・オブジェクトを公表した後にブローカーがエ
ラーを報告する場合、コネクタはイベントに失敗し、ブローカーによって報告
されるエラー・メッセージのログを記録します。コネクタ・プロパティ
`ErrorDestination` が定義されており、有効な場合、コネクタは、失敗したメ
ッセージのコピーをこの宛先に置きます。そうでない場合、メッセージは廃棄
されます。

メッセージのビジネス・オブジェクトを決定できない場合、またはメッセージを
ブローカーに公表しても、ブローカーがそのメッセージはサポートされていない
と報告する場合、コネクタは、メッセージはアンサブスクライブされていると
みなします。コネクタ・プロパティ `UnsubscribedDestination` が定義されて
おり、有効な場合、コネクタは、アンサブスクライブされたメッセージのコ
ピーをこの宛先に置きます。そうでない場合、メッセージは廃棄されます。

要求メッセージの処理

ビジネス・オブジェクト要求がコネクタに送信されると、コネクタは、ター
ゲット宛先で新しいメッセージを作成します。メッセージ・ヘッダーには、要
求メタオブジェクトで指定されたユーザー定義の値とコネクタ・プロパティ
によって指定されたデフォルト・パラメーターを組み合わせたデータが読み
込まれます。メッセージの本文には、構成されたデータ・ハンドラーを介
して要求ビジネス・オブジェクトを渡すことによって生成された結果内容
のデータが取り込まれます。

図 2 に、メッセージ要求通信を示します。doVerbFor() メソッドがブ
ローカーからビジネス・オブジェクトを受信すると、コネクタはビジネス・
オブジェクトをデータ・ハンドラーに渡します。データ・ハンドラーは
ビジネス・オブジェクトを適切なテキストに変換し、コネクタはそれを
メッセージとして宛先に発行します。

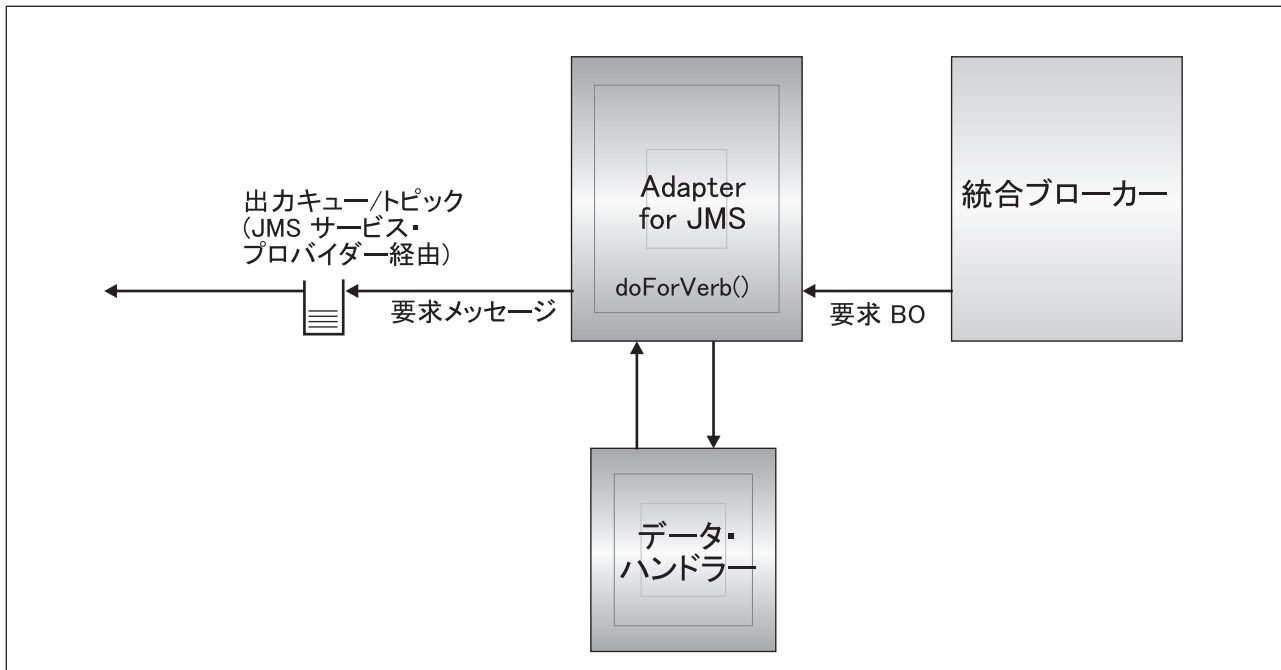


図2. 要求フロー

要求の処理では、コネクタは 2 つのタイプの処置を行うことができます。1 つ目は下記で非同期処理として説明するものであり、コネクタはメッセージをターゲット宛先に置いて、正常に戻ります。一般に、これは「応答不要送信」と呼ばれます。2 つ目は下記で同期処理として説明するものであり、コネクタはメッセージを同じようにターゲット宛先に置きますが、ターゲット・アプリケーションによって応答が戻されるのを待ちます。

処理モードは数値プロパティ `ResponseTimeout` によって決まります。これは、ビジネス・オブジェクト要求の動的メタオブジェクトまたは静的メタオブジェクトのいずれかで指定されます。このプロパティが定義されていないか、`-1` に設定されている場合、コネクタは要求を非同期的に配信します。このプロパティが `0` 以上の場合、アダプタは、要求を同期的に処理し、ターゲット・アプリケーションが応答メッセージを戻すのをそのミリ秒以上待ちます。図2 に示した要求処理を、以下で詳しく説明します。

- 『動詞サポート』
- 『非同期処理』
- 14 ページの『同期処理』

動詞サポート

コネクタは、要求ビジネス・オブジェクトで指定される動詞にセマンティック値は置きません。コネクタは同じ処置を行います。つまり、指定される動詞に関係なく、メッセージを JMS 宛先に置きます。

非同期処理

非同期処理では、コネクタは要求ビジネス・オブジェクトをメッセージに変換し、そのメッセージをターゲット宛先に置いてから、すぐブローカーに戻ります。要求が成功するか失敗するかは、すべて、メッセージを JMS 宛先に置くコネクタの能力に基づいています。この配信が成功しても、ターゲット・アプリケーション

ンにメッセージがあるわけでもメッセージを受信するわけでもないことに注意してください。メッセージング・システムが非同期的性質を持つ場合、ターゲット・アプリケーションがメッセージを処理できるようになるまで、または有効期限切れになる (そのように構成されている場合) まで、メッセージは JMS プロバイダーに無期限にとどまる場合があります。

コネクタはまず、構成されたデータ・ハンドラーを使用して、要求ビジネス・オブジェクトをテキストにシリアルライズします。コネクタは、以下に指定されたデータ・ハンドラーを使用します (優先順)。

1. 動的メタオブジェクト
2. 静的メタオブジェクト
3. コネクタ構成プロパティ

コネクタは、シリアルライズされたビジネス・オブジェクト・データのいった新しいメッセージをメッセージの本文として作成します。コネクタは、以下の表の説明に従って、メッセージ・ヘッダーにデータを読み込みます。動的メタオブジェクトまたは静的メタオブジェクトのどちらにおいてもプロパティを指定できるすべての場合で、動的メタオブジェクトで指定された値は、静的メタオブジェクトで指定された値に優先します。メタオブジェクトで指定できるプロパティの説明およびリストについては、29 ページの『メタオブジェクトの構成』を参照してください。

表 1. 非同期要求処理での JMS メッセージ・ヘッダーの読み込み

メタオブジェクト・プロパティ	プロパティが未定義の場合のデフォルト処置	プロパティが定義されている場合に行われる処置
OutputFormat	コネクタはメッセージ・フォーマットを指定しない	コネクタはメッセージ・フォーマットにこの値を指定する
CorrelationID	コネクタはメッセージ・ヘッダーでこの値をブランクにする	コネクタは、要求メッセージ・ヘッダーで相関 ID にこの値を指定する
ReplyToDestination	コネクタはメッセージ・ヘッダーでこの値をブランクにする	コネクタは、要求メッセージ・ヘッダーで応答宛先にこの値を指定する
Priority	コネクタは、JMS プロバイダーがデフォルト優先順位を使用できるようにする	コネクタは、この値を使用してメッセージ優先順位の数値を設定する
JMSProperties	なし	コネクタは、指定された JMS プロパティをメッセージ・ヘッダーの JMS プロパティにマップする

メタオブジェクトの次の属性が、メッセージの配信方法を決めます。

表 2. 宛先への非同期配信

メタオブジェクト・プロパティ	プロパティが未定義の場合のデフォルト処置	プロパティが定義されている場合に行われる処置
OutputDestination	値が必要	メッセージのターゲット宛先
DeliveryMode	コネクタは、JMS プロバイダーがメッセージ・パーシスタンスを書き取らせることができるようにする	コネクタは、ユーザーの指示に従って、メッセージを永続的または非永続的に書き込む

コネクタが、要求メッセージを出力 (ターゲット) 宛先に正常に配信できたかどうかによって、以下のいずれかのコードがブローカーに戻されます。

表 3. 非同期戻りコード

コネクタの処置	ブローカーへの戻りコード
メッセージを正常にターゲット宛先へ配信	SUCCESS
不適切または不完全なメタデータ、データ・ハンドラーの失敗、または一般的な処理問題などのリカバリー可能エラーにより、配信が失敗	FAIL
JMS プロバイダーによって報告される、接続の失敗などのリカバリー不能エラーにより、配信が失敗	APPRESPONSETIMEOUT

同期処理

同期処理では、コネクタは要求をターゲット宛先へ配信してから、2 番目の宛先で応答メッセージを待ちます。要求メッセージの作成は非同期処理の場合と同じです。ただし、コネクタは、メタオブジェクトの以下の追加属性もチェックします。

表 4. 同期メタオブジェクト・プロパティ

メタオブジェクト・プロパティ	プロパティが未定義の場合のデフォルト処置	プロパティが定義されている場合に行われる処置
ResponseTimeout	値が必要	アダプターが、応答メッセージが戻るのを待つ最小時間 (ミリ秒単位)
TimeoutFatal	ResponseTimeout で指定された時間までに応答を受信できない場合、コネクタは APPRESPONSETIMEOUT をブローカーに戻す。これにより、通常、コネクタは終了する。	応答を受信できない場合、コネクタは要求に失敗するが (FAIL をブローカーに戻す)、終了はしない。

ターゲット宛先へのメッセージの配信は、以下の点を除いて、非同期処理の場合と同じです。

表 5. 宛先への同期配信

メタオブジェクト・プロパティ	プロパティが未定義の場合のデフォルト処置	プロパティが定義されている場合に行われる処置
ReplyToDestination	非同期の場合と同じ	コネクタは、要求メッセージのこのフィールドに、コネクタ固有プロパティ ReplyToDestination の値を読み込む

コネクタは、メタオブジェクト属性 ResponseTimeout で指定された時間以上、ReplyToDestination で指定されたターゲット・アプリケーションからの応答メッセージを待ちます。応答がその時間内に戻らない場合、コネクタはタイムアウトになり、エラーを報告します。

応答基準: コネクタは、応答宛先の最初のメッセージが正しい応答メッセージであるとはみなしません。その代わりに、JMS 要求応答規則に従って、要求のメッセージ ID に一致する関連 ID を持つ最初のメッセージを探します。つまり、要求メッ

セージを受信するアプリケーションは、要求メッセージ ID に等しい相関 ID を持つ応答メッセージを作成し、そのメッセージを、要求メッセージによって指定された応答宛先に置く必要があります。

すべてのアプリケーションが相関 ID を使用する規則に従って、要求メッセージと応答メッセージをマップするわけではありません。その場合、コネクターは、応答メッセージを識別するカスタム基準を受け入れます。

コネクターは、同期要求処理の対象となるビジネス・オブジェクトを受信するときに、動詞のアプリケーション固有情報に、名前と値のペア `response_selector=` が存在するかどうかを検査します。そのような名前と値のペアが存在しない場合、コネクターは、既に説明したように、メッセージ相関 ID を使用して応答メッセージを識別します。

応答セレクターの名と値のペアが定義されている場合、コネクターは、応答メッセージを識別できる、JMS メッセージ・セレクター・ストリングを表す値とみなします。以下にいくつかの使用例を示します。JMS メッセージ・セレクター構文の詳細については、JMS API 仕様を参照してください。JMS メッセージ・セレクター構文はコネクターによって解析されないことに注意してください。その代わりに、構文は JMS プロバイダーによって解釈されます。コネクターは、JMS プロバイダーが、メッセージをフィルタリングする手段としてセレクターを使用できるようにします (データベースのクエリーに類似)。

例えば、名前と値のペアが入った、次の動詞アプリケーション固有情報は、

```
response_selector=JMSType = 'xmlResponse'
```

応答メッセージが、セレクター・ストリング `JMSType = 'xmlResponse'` に一致しなければならぬことをコネクターに知らせます。コネクターはこのセレクターを JMS プロバイダーに提供します。次に JMS プロバイダーは、配信された最初のメッセージを、メッセージの JMS タイプ・フィールドが `xmlResponse` に等しい応答宛先に戻します。

すべての場合で、メッセージ・セレクター・ストリングは、1 つだけの応答を一意に識別できなければなりません。応答セレクターの基準を満たした応答宛先に複数のメッセージが配信されると、アダプターは最初のメッセージのみを検索します。基準に一致する可能性のあるその他の応答メッセージは無視されます。

実行時にメッセージ・セレクターが固有なものになるように、コネクターは、メッセージ・セレクター自身への属性値の動的置換をサポートしています。これを行うには、応答セレクターで、整数を中括弧で囲んだ形式 ("`{1}`") のプレースホルダーを指定する必要があります。この後にコロンを置き、置換に使用する属性をコンマで区切ってリストしてください。プレースホルダーの整数は、置換に使用する属性に対する指標として機能します。

例えば、以下のメッセージ・セレクターでは、

```
response_selector=JMSCorrelationID LIKE '{1}':MyDynamicMO.CorrelationID
```

コネクタは、トークン {1} を、子オブジェクト MyDynamicMO の属性 CorrelationID の値で置き換えることを知らせます。属性 CorrelationID が 123ABC の値を持つ場合、コネクタは、次のメッセージ・セレクターを生成し、それを使用します。

```
JMSCorrelation LIKE '123ABC'
```

下に示すように、複数の置換を指定することもできます。

```
response_selector=Name LIKE '{1}'AND Zip LIKE '{2}':PrimaryID,Address[4].AddressID
```

この例では、コネクタは、'{1}' を、トップレベル・ビジネス・オブジェクトの属性 PrimaryID の値で置き換え、'{2}' を、子コンテナ・オブジェクト Address の 5 番目 (ベース 0) にある AddressID の値で置き換えます。この方法では、応答メッセージ・セレクターでビジネス・オブジェクトおよびメタオブジェクトの任意の属性を参照できます。

メッセージ・セレクターでリテラル値「{」を指定するには、その代わりに「{{」を使用します。例えば、以下のセレクターでは、

```
response_selector=PrimaryID LIKE {{1}}
```

アダプターは、これを以下のリテラル値として認識します。

```
PrimaryID LIKE {1}
```

この場合、コネクタは、値 '{1}' での置換は行いません。

コネクタは、属性値で「{」、「}」、「:」、「;」などの特殊文字を検出した場合は、それらの文字を照会ストリングに直接挿入します。これにより、アプリケーション固有情報の区切り文字としても機能する特殊文字を照会ストリングに含めることができます。例えば、以下のセレクターでは、

```
Response_selector=PrimaryID = '{1}':Foo
```

属性 Foo が {A:B};{C:D} の値を持つ場合、以下のようなリテラル・メッセージ・セレクターに変換されます。

```
PrimaryID = '{A:B};{C:D}'
```

応答処理: 応答メッセージを受け取ったときに行う処置を確認するために、コネクタは、コネクタ・プロパティ MessageResponseResultProperty によって指定された JMS 結果プロパティを検査します。この JMS プロパティの値によって、コネクタは、メッセージ本文にビジネス・オブジェクトとエラー・メッセージのどちらが含まれているかを予想します (下の表を参照)。すべての場合に、コネクタは対応する戻りコードをブローカーに戻します。メッセージで JMS 結果プロパティが VALCHANGE に等しい場合、コネクタは、この表で説明された VALCHANGE についての処置を行い、ブローカー定数 VALCHANGE に対応した数値をブローカーに戻します。

表 6. 応答メッセージの処理

JMS 結果プロパティの値	コネクタの処置
SUCCESS	要求ビジネス・オブジェクトを変更せず、単に、正常にブローカーに戻ります。

表 6. 応答メッセージの処理 (続き)

VALCHANGE MULTIPLE_HITS	要求ビジネス・オブジェクトに応答メッセージ本文の内容を再び読み込みます。応答メッセージ本文が空の場合、要求ビジネス・オブジェクトは変更されません。 要求ビジネス・オブジェクトの動的メタオブジェクトに、応答メッセージの JMS ヘッダー・フィールドを再び読み込みます。
FAIL FAIL_RETRIEVE_BY_CONTENT BO_DOES_NOT_EXIST UNABLE_TO_LOGIN VALDUPES	応答にデータが読み込まれている場合、コネクタは応答をエラー・メッセージとみなし、それをブローカーに戻します。応答メッセージ本文が空の場合、コネクタは一般エラー・メッセージをブローカーに戻します。
APPRESPONSETIMEOUT	APPRESPONSETIMEOUT がブローカーに戻ると、通常、アダプター・エージェントが終了する点を除いて、上記と同じです。
未定義または認識できない値	コネクタは要求に失敗します。

エラー処理: ターゲット宛先との間での要求メッセージの読み取りまたは書き込みのとき、または応答メッセージの検査のときに (該当する場合) エラーを検出すると、コネクタは、APPRESPONSETIMEOUT をすぐにブローカーに戻します。これにより、アダプターは終了するか、場合により再始動します。一般に、そのようなリカバリー不能エラーは、JMS プロバイダーへの接続が切断されたか、あるいはコネクタが認識できないか、認識してもリカバリー不能 (トランザクションの失敗など) とみなした、JMS プロバイダーによって報告された内部エラーかのいずれかが原因です。

ビジネス・オブジェクトをメッセージに変換しているとき、またはその逆の変換のときにエラーを検出した場合 (データ・ハンドラーが無効なメッセージ・フォーマットを報告する場合など)、コネクタは要求に失敗し、理由を説明した該当するエラー・メッセージのログを記録します。

イベント失敗のシナリオを含む詳細については、47 ページの『エラー処理』を参照してください。

第 2 章 アダプターのインストールおよび構成

- 『インストール・タスク』
- 『アダプターと関連ファイルのインストール』
- 『インストール済みファイルの構造』
- 22 ページの『コネクタ・プロパティの構成』
- 28 ページの『メッセージ・スタイルの構成』
- 28 ページの『JNDI の構成』
- 29 ページの『メタオブジェクトの構成』
- 40 ページの『開始スクリプトの構成』
- 40 ページの『複数のコネクタ・インスタンスの作成』
- 42 ページの『コネクタの始動』
- 43 ページの『コネクタの停止』

この章では、コネクタのインストール方法および構成方法と、メッセージ・フローをコネクタとともに動作させるための構成方法について説明します。

インストール・タスク

Adapter for JMS をインストールするには、以下の作業を実行する必要があります。

- **統合ブローカーのインストール** この作業では、WebSphere Business Integration システムのインストールと統合ブローカーの始動を行います。作業の詳細については、使用するブローカーおよびオペレーティング・システムのインストール文書に説明があります。
- **アダプターおよび関連ファイルのインストール** この作業では、アダプターのファイルをソフトウェア・パッケージから使用システムにインストールします。『アダプターと関連ファイルのインストール』を参照してください。

アダプターと関連ファイルのインストール

WebSphere Business Integration Adapter 製品のインストールについては、「*WebSphere Business Integration Adapters* インストール・ガイド」を参照してください。この資料は、次の Web サイトの WebSphere Business Integration Adapters Infocenter にあります。

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

インストール済みファイルの構造

以下のセクションでは、インストール後の製品のパスとファイル名について説明します。

Windows のコネクタ・ファイル構造

インストーラーは、コネクタ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・エージェントを *ProductDir*¥connectors¥JMS 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・エージェントへのショートカットを追加します。*ProductDir* は、IBM WebSphere Business Integration Adapters 製品がインストールされているディレクトリーを表していることに注意してください。環境変数には、*ProductDir* ディレクトリーへのパス (デフォルトでは IBM¥WebSphereAdapters) が含まれていません。

表 7 に、コネクタが使用する Windows ファイル構造が記載されており、インストーラーを介したコネクタのインストールを選択した際に自動的にインストールされるファイルを示します。

表 7. コネクタ用としてインストールされた Windows ファイル構造

<i>ProductDir</i> のサブディレクトリー	説明
connectors¥JMS¥CWJMS.jar	JMS コネクタに使用されるクラスを含みます。
connectors¥JMS¥jms.jar	コネクタが必要とするサード・パーティーのライブラリー。
connectors¥JMS¥start_JMS.bat	コネクタ (NT/2000) の始動スクリプト。
connectors¥messages¥JMSConnector.txt	コネクタのメッセージ・ファイル。
repository¥JMS¥CN_JMS.txt	コネクタのリポジトリー定義。
connectors¥JMS¥Samples¥JMSConnector.cfg	コネクタ構成ファイルのサンプル
connectors¥JMS¥Samples¥PortConnector.cfg	ポート・コネクタ構成ファイルのサンプル
connectors¥JMS¥Samples¥Sample_JMS_Contact.xsd	メタオブジェクトのサンプル
connectors¥JMS¥Samples¥Sample_JMS_MO_Config.xsd	データ・ハンドラー・メタオブジェクトのサンプル
connectors¥JMS¥Samples¥Sample_JMS_MO_DataHandler.xsd	区切りデータ・ハンドラー・メタオブジェクトのサンプル
connectors¥JMS¥Samples¥Sample_JMS_MO_DataHandler_DelimitedConfig.xsd	動的なメタオブジェクトのサンプル
connectors¥JMS¥Samples¥Sample_JMS_DynMO.xsd	JMS プロパティーのサンプル
connectors¥JMS¥Samples¥JMSPropertyPairs.xsd	

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。

UNIX のコネクタ・ファイル構造

インストーラーは、コネクタ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・エージェントを *ProductDir/connectors/JMS* ディレクトリーにインストールします。

表 8 に、コネクタが使用する UNIX ファイル構造が記載されており、インストーラーを介したコネクタのインストールを選択した際に自動的にインストールされるファイルを示します。

表 8. コネクタ用としてインストールされた UNIX ファイル構造

ProductDir のサブディレクトリー	説明
connectors/JMS/CWJMS.jar	JMS コネクタに使用されるクラスを含みます。
connectors/JMS/jms.jar	コネクタが必要とするサード・パーティーのライブラリー。
connectors/JMS/start_JMS.sh	コネクタのシステム始動スクリプト。このスクリプトは、汎用のコネクタ・マネージャー・スクリプトから呼び出されます。 System Manager の「コネクタ構成」画面をクリックすると、インストーラーは、このコネクタ・マネージャー・スクリプト用にカスタマイズされたラッパーを作成します。コネクタの始動および停止には、このカスタマイズされたラッパーを使用してください。
connectors/messages/JMSConnector.txt	コネクタのメッセージ・ファイル。
repository/JMS/CN_JMS.txt	コネクタのリポジトリ定義。
connectors/JMS/Samples/JMSConnector.cfg	コネクタ構成ファイルのサンプル
connectors/JMS/Samples/PortConnector.cfg	ポート・コネクタ構成ファイルのサンプル
connectors/JMS/Samples/Sample_JMS_Contact.xsd	メタオブジェクトのサンプル
connectors/JMS/Samples/Sample_JMS_MO_Config.xsd	データ・ハンドラー・メタオブジェクトのサンプル
connectors/JMS/Samples/Sample_JMS_MO_DataHandler.xsd	区切りデータ・ハンドラー・メタオブジェクトのサンプル
connectors/JMS/Samples/Sample_JMS_MO_DataHandler_DelimitedConfig.xsd	動的なメタオブジェクトのサンプル
connectors/JMS/Samples/Sample_JMS_DynMO.xsd	JMS プロパティのサンプル
connectors/JMS/Samples/JMSPropertyPairs.xsd	

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。

コネクタ構成

インストールしたアダプターは、コネクタを構成する必要があります。構成するには、以下のセクションで説明されている作業を行う必要があります。

- 22 ページの『コネクタ・プロパティの構成』
- 28 ページの『メッセージ・スタイルの構成』
- 28 ページの『JNDI の構成』
- 29 ページの『メタオブジェクトの構成』
- 40 ページの『開始スクリプトの構成』

コネクタ・プロパティの構成

コネクタには、以下のセクションで説明されている 2 つのタイプの構成プロパティがあります。

- 『標準コネクタ・プロパティの構成』
- 『コネクタ固有プロパティの構成』

アダプターを実行する前に、これらのプロパティの値を設定する必要があります。

コネクタのプロパティを構成するには、Connector Configurator を使用します。

- Connector Configurator の説明と段階的な手順については、69 ページの『付録 B. Connector Configurator』を参照してください。
- 標準コネクタ・プロパティの説明については、『標準コネクタ・プロパティの構成』、および 51 ページの『付録 A. コネクタの標準構成プロパティ』を参照してください。
- コネクタ固有のプロパティの詳細については、『コネクタ固有プロパティの構成』を参照してください。

標準コネクタ・プロパティの構成

標準構成プロパティにより、すべてのコネクタによって使用される情報が提供されます。標準構成プロパティの資料については、51 ページの『付録 A. コネクタの標準構成プロパティ』を参照してください。これらのプロパティの設定方法を説明したステップバイステップ手順については、69 ページの『付録 B. Connector Configurator』を参照してください。

注: Connector Configurator で構成プロパティを設定するときは、BrokerType プロパティで使用するブローカーを指定します。このプロパティの値を設定すると、使用するブローカーに関連するプロパティが「Connector Configurator」ウィンドウに表示されます。

コネクタ固有プロパティの構成

コネクタ固有の構成プロパティは、コネクタ・エージェントが実行時に必要とする情報を提供します。また、コネクタ固有の構成プロパティを使用すると、コネクタ・エージェントのコード変更や再ビルドを行わなくても、エージェント内の静的情報またはロジックを変更できます。

表 9 に、コネクタのコネクタ固有の構成プロパティのリストを示します。プロパティの説明については、以下の各セクションを参照してください。

表9. コネクタ固有のプロパティ

名前	指定可能な値	デフォルト値	必須
ArchiveDestination	正常に処理されたメッセージのコピーが送信される宛先		いいえ
ConfigurationMetaObject	構成メタオブジェクト		プロパティの説明を参照
ConnectionFactoryName	JNDI ストアで定義された JMS キューまたはトピックの接続ファクトリー		はい
CTX_InitialContextFactory	初期 JNDI コンテキストの設定に使用されるファクトリー・クラスの名前		はい
CTX_ProviderURL	接続ファクトリーが存在する JNDI コンテキストを示す URL		はい
DataHandlerClassName	インスタンスを生成するデータ・ハンドラー・クラスの名前		プロパティの説明を参照
DataHandlerConfigMO	DataHandlerMimeType の構成情報を含むデータ・ハンドラー・メタオブジェクトの名前	MO_DataHandler_Default	プロパティの説明を参照
DataHandlerMimeType	デフォルト・データ・ハンドラーの選択で使用する MIME タイプ	text/delimited	プロパティの説明を参照
DefaultVerb	着信ビジネス・オブジェクト内に設定する動詞を指定します。	Create	いいえ
ErrorDestination	未処理メッセージの宛先		いいえ
InDoubtEvents	FailOnStartup Reprocess Ignore LogError	Reprocess	いいえ
InProgressDestination	一時記憶域宛先		いいえ
InputDestination	ポーリング宛先の名前		いいえ
LookupDestinationsUsingJNDI	true または false	false	いいえ
MessageFormatProperty	メッセージ・フォーマットを指定するプロパティ名	JMSType	いいえ
MessageResponseResultProperty	要求された操作の結果を示す応答メッセージのプロパティ	WBI_Result	はい (同期処理の場合)
PollQuantity	InputDestination プロパティで指定された各宛先で検索するメッセージの数	1	いいえ
ReplyToDestination	コネクタからの要求発行時に応答メッセージが配信される宛先		はい (同期処理の場合)
UnsubscribedDestination	メッセージが認識されないか、またはそのメッセージがマップする対象のビジネス・オブジェクトがサポートされない場合に、インバウンド・メッセージのコピーが出力される宛先です。		いいえ
UnsubscribeOnTerminate	InputDestination から削除するトピックを指定します。		いいえ
UseDefaults	true または false	false	いいえ
UseDurableSubscriptions	true または false	false	いいえ

ArchiveDestination

正常に処理されたメッセージのコピーが送信される宛先です。

デフォルト値は `CWLD_ARCHIVE` です。

ConfigurationMetaObject

コネクターの構成情報を含む静的なメタオブジェクトの名前です。

デフォルト値はありません。

ConnectionFactoryName

JMS プロバイダーとの接続を確立するときに、コネクターが検索し、使用する必要のある、JNDI ストアで定義された JMS キューまたはトピックの接続ファクトリーの名前です。この名前を検索する場合、コネクターは、`CTX_InitialContextFactory` および `CTX_ProviderURL` の各プロパティによって設定された初期 JNDI コンテキストを使用します。

デフォルト = なし。

CTX_InitialContextFactory

初期 JNDI コンテキストの設定に使用されるファクトリー・クラスの名称です。

デフォルト = なし。

CTX_ProviderURL

接続要因が存在する JNDI コンテキストを示す完全修飾 URL です。この値はコンテキスト要因に渡されます。

デフォルト = なし。

DataHandlerClassName

ビジネス・オブジェクトとの間でのメッセージ変換に使用するデータ・ハンドラー・クラスです。 `DataHandlerConfigMO` と `DataHandlerMimeType` の両方を指定するか、`DataHandlerClassName` のみを指定します。3 つのプロパティすべてを指定しないでください。

注: 静的メタオブジェクト、または動的メタオブジェクト内の

`DataHandlerClassName` 値は、このコネクターの構成プロパティで指定される値よりも優先されます。メタオブジェクト内で `DataHandlerClassName` 値を指定しない場合、コネクターはコネクター構成プロパティから値を取得します。

デフォルト = なし。

DataHandlerConfigMO

`DataHandlerMimeType` プロパティで指定された MIME タイプの構成情報を含むメタオブジェクトの名前です。データ・ハンドラーの構成情報を提供します。

`DataHandlerConfigMO` と `DataHandlerMimeType` を指定するか、

`DataHandlerClassName` のみを指定します。3 つのプロパティすべてを指定しないでください。

注: 静的メタオブジェクト、または動的メタオブジェクト内の `DataHandlerConfigMO` 値は、このコネクターの構成プロパティーで指定される値よりも優先されます。メタオブジェクト内で `DataHandlerConfigMO` 値を提供しない場合、コネクターはコネクター構成プロパティーから値を取得します。

デフォルト値は `MO_DataHandler_Default` です。

DataHandlerMimeType

使用すると、特定の MIME タイプに基づいたデータ・ハンドラーを要求できます。`DataHandlerConfigMO` と `DataHandlerMimeType` を指定するか、`DataHandlerClassName` のみを指定します。3 つのプロパティーすべてを指定しないでください。

注: 静的メタオブジェクト、または動的メタオブジェクト内の `DataHandlerMimeType` 値は、このコネクターの構成プロパティーで指定される値よりも優先されます。メタオブジェクト内で `DataHandlerMimeType` 値を指定しない場合、コネクターはコネクター構成プロパティーから値を取得します。

デフォルト = `text/delimited`

DefaultVerb

着信ビジネス・オブジェクト内に設定する動詞を指定します。ただし、この動詞がポーリング中にデータ・ハンドラーにより設定されていないことが前提です。

デフォルト = `Create`

ErrorDestination

処理中にコネクターがエラーを検出したときに、インバウンド・メッセージのコピーが送信される宛先です。

デフォルト値は `CWLD_ERROR` です。

InDoubtEvents

コネクターの予期しないシャットダウンのために、処理が完了していない進行中イベントの処理方法を指定します。初期化中に進行中のキューにイベントが見つかった場合に実行するアクションを、以下の 4 つから選択してください。

- `FailOnStartup`: エラーをログに記録し、ただちにシャットダウンします。
- `Reprocess`: 残りのイベントを先に処理してから、入力キューのメッセージを処理します。
- `Ignore`: 進行中のキューのメッセージをすべて無視します。
- `LogError`: エラーをログに記録しますが、シャットダウンしません。

デフォルト値は `Reprocess` です。

注: `InProgressDestination` プロパティーを構成する場合は、このプロパティーの値を指定する必要があります。

InProgressDestination

処理中にメッセージが保留される一時的宛先です。

デフォルト = なし。

InputDestination

コネクタが新規のメッセージの有無を確認するためにポーリングする宛先です。コネクタは、セミコロンで区切られた複数の名前を受け入れます。例えば、キューに基づく構成で、MyQueueA、MyQueueB、および MyQueueC の 3 つのキューにポーリングするには、コネクタ構成プロパティ *InputQueue* の値を MyQueueA;MyQueueB;MyQueueC とします。

InputDestination プロパティが指定されていない場合、コネクタはポーリングしません。

デフォルト = なし。

LookupDestinationsUsingJNDI

このプロパティが *true* の場合、コネクタは、JNDI ストアのすべての JMS 宛先名を検索します。この場合、指定された宛先がすべて JNDI ストアで定義されている必要があります。

デフォルトで、コネクタはこのステップをスキップし、JMS プロバイダーが実行時に名前を適切な宛先へ変換することを許可します。

デフォルト = *false*

MessageFormatProperty

メッセージの入出力フォーマットを含む JMS メッセージのフィールドです。デフォルトで、コネクタは、インバウンド・メッセージの *JMSType* フィールドでメッセージ・フォーマットを調べ、そのメッセージ・フォーマットを、アウトバウンド・メッセージの *JMSType* フィールドに書き込みます。

デフォルト = *JMSType*

MessageResponseResultProperty

同期要求処理で必要とされます。このプロパティは、コネクタが要求結果を確認するために検査する、応答 JMS メッセージのフィールドを指定します。このプロパティは、非同期処理では使用されません。

デフォルト値は *WBI_Result* です。

PollQuantity

pollForEvents サイクル中に、*InputDestination* プロパティで指定された各宛先で検索するメッセージの最大数です。

デフォルト値は 1 です。

ReplyToDestination

コネクタからの要求発行時に応答メッセージが配信される宛先です。ターゲット・アプリケーションとの間での要求メッセージの交換を調整するために、コネクタが使用するデフォルトの宛先です。同期処理の場合にのみ、このプロパティを指定します。

デフォルト = なし。

UnsubscribedDestination

メッセージが認識されないか、またはそのメッセージがマップする対象のビジネス・オブジェクトがサポートされない場合に、インバウンド・メッセージのコピーが出力される宛先です。このプロパティが定義されており、有効な場合、コネクタは、アンサブスクライブされたメッセージのコピーをこの宛先に置きます。そうでない場合、メッセージは廃棄されます。

デフォルト = なし。

UnsubscribeOnTerminate

UserDurableSubscriptions が true に設定されている場合にのみ、適用できます。永続サブスクリプションを使用すると、コネクタ構成からトピックを削除すると問題が発生します。コネクタが永続サブスクリプションを調べなくなっても、JMS プロバイダーはそのサブスクリプションのメッセージを保管し続けようとします。

InputDestination で指定されたリストからトピックを削除するときは常に、このプロパティ値で、削除するそれらのトピック (セミコロンで区切る) を指定します。既存の永続サブスクリプションを破棄するには、以下の手順を実行します。

1. サブスクリプションを終了する該当のトピック名を、InputDestination から UnsubscribeOnTerminate へと移動します。
2. コネクタを開始および停止します (これにより永続サブスクリプションが破棄されます)。
3. UnsubscribeOnTerminate に指定されたすべてのトピックを消去します。

この操作は InputDestination の値には影響ありません。

上記ステップを実行しなくてもコネクタには影響しませんが、JMS プロバイダーが、不必要なメッセージを保管するようになります。

デフォルト = なし。

UseDefaults

UseDefaults が true に設定されている場合、コネクタは、isRequired とマークされている各ビジネス・オブジェクト属性に、有効値またはデフォルト値が指定されているかどうかを検査します。

デフォルト = false

UseDurableSubscriptions

パブリッシュ/サブスクライブのトピック・スタイル・メッセージングにのみ、これを使用します。このプロパティが true に設定されている場合、コネクタは、該当する宛先に対して永続サブスクライバーとして働きます。コネクタは、オフ

ラインの状態であっても、JMS プロバイダーに対し、サブスクライブするトピックのメッセージをすべて保管するように指示します。これには大きなオーバーヘッドが伴います。コネクタは、オンラインの状態に戻ると、失ったすべての公表されたメッセージを再処理します。

デフォルト = false

メッセージ・スタイルの構成

アダプターは、JMS 規格で定義された point-to-point (PTP) メッセージングおよびパブリッシュ/サブスクライブ (Pub/Sub) メッセージングの両方のインターフェースをサポートします。アダプターが使用するメッセージング・スタイルは、コネクタ固有プロパティ `ConnectionFactoryName` にユーザーが指定する管理オブジェクトのタイプによって決まります。以下の手順に進む前に、24 ページの『`ConnectionFactoryName`』を参照してください。

- 『PTP メッセージ・スタイルの構成』
- 『Pub/Sub スタイルの構成』

PTP メッセージ・スタイルの構成

PTP メッセージ・スタイルでアダプターのインスタンスを構成するには、次のようにします。

1. 「Connector Configurator」を開きます。
2. 「コネクタ固有プロパティ」タブをクリックします。
3. JNDI ストアの `JMS QueueConnectionFactory` のインスタンスにマップする `ConnectionFactoryName` の名前を指定します。アダプターは PTP スタイルで動作し、宛先を示すすべてのコネクタおよびメタオブジェクトのプロパティ (`OutputDestination` プロパティなど) が、キューを表すものと見なします。

Pub/Sub スタイルの構成

Pub/Sub メッセージ・スタイルでアダプターのインスタンスを構成するには、次のようにします。

1. 「Connector Configurator」を開きます。
2. 「コネクタ固有プロパティ」タブをクリックします。
3. JNDI ストアの `JMS TopicConnectionFactory` のインスタンスにマップする `ConnectionFactoryName` の名前を指定します。アダプターはパブリッシュ/サブスクライブ・スタイルで動作し、宛先を示すすべてのコネクタおよびメタオブジェクトのプロパティ (`OutputDestination` プロパティなど) が、トピックを表すものと見なします。

JNDI の構成

JMS プロバイダーへの接続を確立するには、コネクタは、JMS 接続ファクトリーへアクセスする必要があります。JMS はファクトリーのインターフェースを定義しています。ただし、各 JMS プロバイダーは、独自のインプリメンテーションを提供する必要があります。コネクタにこのファクトリー・インプリメンテーションへの参照が作成されると、コネクタは、プロバイダーの専有プロトコルまたは ID の知識がなくても、JMS プロバイダーとの接続を確立し、通信できるようになります。

コネクタを移植可能にするには、接続ファクトリーを JNDI ストアに置く必要があります。ユーザーまたはシステム管理者は、インプリメンテーションのときに、接続ファクトリーの作成と構成を行い、それをユーザー定義名で JNDI ストアに置く必要があります。実行時に、コネクタは、JNDI ストアとの接続を確立し、接続ファクトリーを検索し、それを使用して JMS プロバイダーへの接続を確立します。

接続ファクトリーまたはユーザーが作成するその他の管理 JMS オブジェクトを含む、独自の JNDI インプリメンテーションを提供する JMS プロバイダーもあります。この方法を使用すると、ユーザーは JMS アダプターを非常に簡単に構成することができます。その他の JMS プロバイダーでは、ユーザーは、外部 JNDI プロバイダーのインストールと構成を行い、接続ファクトリーを作成し、それをアダプターが使用できるようにしなければならない場合があります。詳細については JNDI プロバイダーの資料を参照してください。

JNDI の環境変数および構成の詳細については、www.java.com を参照してください。JNDI (MA88 パッチ適用済み) の構成の詳細については、『WebSphere MQ Java クライアント・ライブラリーを使用した JNDI の構成』を参照してください。

WebSphere MQ Java クライアント・ライブラリーを使用した JNDI の構成

WebSphere MQ Java クライアント・ライブラリーを使用した JNDI の構成方法の解説については、93 ページの『キュー・ベース・メッセージングの構成』および 94 ページの『トピック・ベース・メッセージングの構成』を参照してください。

メタオブジェクトの構成

Connector for JMS は、2 種類のメタオブジェクトを認識および読み取ることができます。

- 静的なコネクタ・メタオブジェクト
- 動的な子メタオブジェクト

動的な子メタオブジェクトの属性値は、静的なメタオブジェクトの属性値と重複し、それらをオーバーライドします。メタデータ、および静的メタオブジェクトと動的メタオブジェクトとの比較の概要については、8 ページの『メタデータおよびメタオブジェクト』を参照してください。

どのメタオブジェクトが使用しているインプリメンテーションに最適かを判別するには、以下のことを検討してください。

- 静的メタオブジェクト
 - 各メッセージに対するメタデータがすべて決まっていて、構成時に指定できる場合に有効です。
 - ビジネス・オブジェクト・タイプによって値を指定するよう制限されます。例えば、Customer タイプ・オブジェクトはすべて同じ宛先に送信する必要があります。
- 動的メタオブジェクト

- メッセージ・ヘッダーの情報に対し、ビジネス・プロセス・アクセスを提供します。
- ビジネス・タイプに関係なく、ビジネス・プロセスが、実行時にメッセージの処理を変更できるようにします。例えば、動的メタオブジェクトを使用すると、アダプターに送られる各 Customer タイプ・オブジェクトに別々の宛先を指定できます。
- サポートされるビジネス・オブジェクトの構造を変更する必要があります。— そのような変更では、マップおよびビジネス・プロセスを変更しなければならない場合があります。
- カスタム・データ・ハンドラーを変更する必要があります。

メタオブジェクト・プロパティ

表 10 は、メタオブジェクトでサポートされるプロパティの完全なリストです。メタオブジェクトをインプリメントする場合は、これらのプロパティを参照してください。

両方のオブジェクトですべてのプロパティを使用できるわけではありません。メッセージ・ヘッダーとの間で、すべてのプロパティが読み取り可能または書き込み可能であるわけでもありません。コネクターが特定のプロパティをどのように解釈および使用するかを確認するには、1 ページの『第 1 章 Adapter for JMS の概要』の、イベントおよび要求の処理に関する該当セクションを参照してください。

表 10. JMS メタオブジェクト・プロパティ

プロパティ名	静的メタオブジェクトで定義可能	動的メタオブジェクトで定義可能	説明
DataHandlerConfigMO	はい	はい	構成情報を提供するために、データ・ハンドラーに渡されるメタオブジェクト。静的なメタオブジェクトに指定された場合、この値は DataHandlerConfigMO コネクター・プロパティに指定された値をオーバーライドします。さまざまなビジネス・オブジェクト・タイプを処理するために各種のデータ・ハンドラーが必要な場合は、この静的なメタオブジェクトのプロパティを使用します。データ形式が実際のビジネス・データに依存する可能性がある場合は、要求処理には動的な子メタオブジェクトを使用します。指定されたビジネス・オブジェクトはコネクター・エージェントによりサポートされていることが必要です。22 ページの『コネクター固有プロパティの構成』の説明を参照してください。

表 10. JMS メタオブジェクト・プロパティ (続き)

プロパティ名	静的メタオブジェクトで定義可能	動的メタオブジェクトで定義可能	説明
DataHandlerMimeType	はい	はい	使用すると、特定の MIME タイプに基づいたデータ・ハンドラーを要求できます。静的なメタオブジェクトに指定された場合、この値は DataHandlerMimeType コネクター・プロパティに指定された値をオーバーライドします。さまざまなビジネス・オブジェクト・タイプを処理するために各種のデータ・ハンドラーが必要な場合は、この静的なメタオブジェクトのプロパティを使用します。データ形式が実際のビジネス・データに依存する可能性がある場合は、要求処理には動的な子メタオブジェクトを使用します。DataHandlerConfigMO に指定されたビジネス・オブジェクトは、このプロパティの値に対応する属性を含める必要があります。22 ページの『コネクター固有プロパティの構成』の説明を参照してください。
DataHandlerClassName	はい	はい	22 ページの『コネクター固有プロパティの構成』の説明を参照してください。
InputFormat	はい	はい	インバウンド (イベント) メッセージのフォーマットまたはタイプ。この値は、メッセージ内容の識別を支援します。メッセージを生成したアプリケーションによって指定されます。メッセージ・フォーマットの定義でコネクターが考慮するフィールドは、コネクター固有プロパティ MessageFormatProperty によって、ユーザーが定義できます。
OutputFormat	はい	はい	アウトバウンド・メッセージで読み込まれるフォーマット。OutputFormat が指定されていない場合、使用可能であれば入力フォーマットが使用されます。
InputDestination	はい	はい	このプロパティは、着信メッセージをビジネス・オブジェクトとマッチングする目的のみで使用されます。対照的に、InputDestination のコネクター固有のプロパティは、アダプターがポーリングする宛先を定義します。これはポーリングする宛先を判別するためにアダプターが使用する唯一のプロパティです。MO 内では、InputDestination プロパティおよび InputFormat プロパティは、アダプターが任意のメッセージを特定のビジネス・オブジェクトにマップする基準として機能します。この機能をインプリメントするためには、コネクター固有のプロパティを使用して複数の入力宛先を構成し、オプションで、着信メッセージの入力フォーマットに基づき異なるデータ・ハンドラーをそれぞれにマップします。
OutputDestination	はい	はい	このプロパティは、デフォルトの型変換プロパティを使用して設定しないでください。値は以下で使用されます。アウトバウンド・メッセージが書き込まれる宛先。

表 10. JMS メタオブジェクト・プロパティ (続き)

プロパティ名	静的メタオブジェクトで定義可能	動的メタオブジェクトで定義可能	説明
ResponseTimeout	はい	はい	同期要求処理の応答を待機した状態で、タイムアウトになるまでの時間をミリ秒で表します。このプロパティが定義されていないかまたはゼロより小さい値が設定されている場合、コネクタは応答を待機せずにすぐに SUCCESS を戻します。
TimeoutFatal	はい	はい	同期要求処理において、応答が受信されない場合にコネクタにエラー・メッセージを戻させるために使用されます。このプロパティが True の場合、応答が ResponseTimeout で指定した時間内に受信されなければ、コネクタはブローカーに APPRESPONSETIMEOUT を戻します。このプロパティが未定義か、または False にセットされている場合、応答がタイムアウトになるとコネクタは要求をエラーにしますが、終了はしません。デフォルト = False。
<p>JMS メッセージ・ヘッダーに限定してマッピングされるフィールドを以下に示します。説明、値の解釈などについては、JMS API 仕様を参照してください。JMS プロバイダーは、一部のフィールドについて、異なった解釈をする場合があります。それらの違いについて、JMS プロバイダーの資料も確認してください。</p>			
ReplyToDestination		はい	要求の応答メッセージが送られる宛先。
Type		はい	メッセージのタイプ。JMS プロバイダーによって異なりますが、一般には、ユーザーによる定義が可能。
MessageID		はい	メッセージの固有 ID (JMS プロバイダーに固有)。
CorrelationID	はい	はい	この応答を開始した要求メッセージの ID を示すために、応答メッセージで使用されます。
Delivery Mode	はい	はい	MOM システムでメッセージを永続させるかどうかを指定します。許容値は以下のとおりです。 1=非永続 2=永続 JMS プロバイダーによっては、その他の値を使用できます。
Priority		はい	メッセージの優先順位 (数値)。許容値は、0 から 9 です (数値が小さいほど優先順位は高くなる)。
Destination		はい	MOM システムでの、メッセージの現在または最後の (削除された場合) 場所。
Expiration		はい	メッセージの存続時間。ゼロが指定されると、有効期限がゼロに設定されます。ゼロを指定すると JMS プロバイダーは、メッセージを有効期限切れにしません。
Redelivered		はい	JMS プロバイダーが以前にクライアントへのメッセージ配信を試みた可能性は高いが、受取が確認されなかったことを示します。
Timestamp		はい	時間メッセージが JMS プロバイダーに渡されました。
UserID		はい	メッセージを送信するユーザーの ID。
AppID		はい	メッセージを送信するアプリケーションの ID。
DeliveryCount		はい	配信を試みる回数。
GroupID		はい	メッセージ・グループの ID。
GroupSeq		はい	グループ ID で指定されたメッセージ・グループにおける、このメッセージの順序。

表 10. JMS メタオブジェクト・プロパティ (続き)

プロパティ名	静的メタオブジェクトで定義可能	動的メタオブジェクトで定義可能	説明
JMSProperties		はい	38 ページの『JMS プロパティ』を参照してください。

静的メタオブジェクトの構成

JMS 構成の静的なメタオブジェクトには、さまざまなビジネス・オブジェクトに定義された変換プロパティのリストが含まれます。静的メタオブジェクトのサンプルを参照するには、Business Object Designer を起動し、アダプターと一緒に出荷される、次のサンプルを開きます。

```
connectors¥JMS¥Samples¥Sample_JMS_MO_Config.xsd
```

コネクタは、いつでも、最大で 1 つの静的メタオブジェクトをサポートします。静的メタオブジェクトは、コネクタ・プロパティ ConfigurationMetaObject で名前を指定してインプリメントします。

静的メタオブジェクトは、ビジネス・オブジェクトと動詞の単一の組み合わせ、およびそのオブジェクトの処理に関連したすべてのメタデータを、各属性が表す構造になっています。各属性の名前は、Customer_Create のように、ビジネス・オブジェクト・タイプと動詞の名前を下線で区切ったものです。属性のアプリケーション固有情報は、セミコロンで区切られた 1 つ以上の名前と値のペアで構成され、この固有なオブジェクトと動詞の組み合わせに指定するメタデータ・プロパティを表します。

表 11. 静的メタオブジェクトの構造

属性名	アプリケーション固有のテキスト
<business object type>_<verb>	property=value;property=value;...
<business object type>_<verb>	property=value;property=value;...

以下のメタオブジェクトを例にとります。

表 12. 静的メタオブジェクト構造のサンプル

属性名	アプリケーション固有情報
Customer_Create	OutputFormat=CUST;OutputDestination=QueueA
Customer_Update	OutputFormat=CUST;OutputDestination=QueueB
Order_Create	OutputFormat=ORDER;OutputDestination=QueueC

このサンプルのメタオブジェクトは、コネクタが、タイプ Customer に動詞 Create が付いた要求ビジネス・オブジェクトを受け取った場合、それをフォーマット CUST のメッセージに変換し、宛先 QueueA に置くことを知らせます。カスタマー・オブジェクトが動詞 Update を持つ場合、メッセージは QueueB に置かれます。オブジェクト・タイプが Order であり、動詞 Create を持つ場合、コネクタは、そのオブジェクトを、フォーマット ORDER で変換し、QueueC に配信します。コネクタに渡されるその他のビジネス・オブジェクトは、アンサブスクライブされているものとして処理されます。

オプションで、1つの属性 Default を指定し、それに ASI の1つ以上のプロパティを割り当てることができます。メタオブジェクトに含まれるすべての属性で、デフォルト属性のプロパティは、特定のオブジェクトと動詞属性のプロパティに結合されます。これは、全体に適用する1つ以上のプロパティがある（オブジェクトと動詞の組み合わせに関係なく）場合に便利です。以下の例の場合、コネクタは、Customer_Create および Order_Create のオブジェクトと動詞の組み合わせが、それらの個別のメタデータ・プロパティ以外に、OutputDestination=QueueA を持つとみなします。

表 13. 静的メタオブジェクト構造のサンプル

属性名	アプリケーション固有情報
Default	OutputDestination=QueueA
Customer_Update	OutputFormat=CUST
Order_Create	OutputFormat=ORDER

静的メタオブジェクトで、アプリケーション固有情報として指定できるプロパティの説明については、30ページの『メタオブジェクト・プロパティ』の30ページの表10を参照してください。

静的メタオブジェクトをインプリメントするには、以下のようになります。

1. Business Object Designer を起動します。詳細については、「ビジネス・オブジェクト開発ガイド」を参照してください。
2. サンプルのメタオブジェクト
connectors¥JMS¥Samples¥Sample_JMS_MO_Config.xsd を開きます。図3に Business Object Designer 内のサンプルの静的メタオブジェクトを示します。

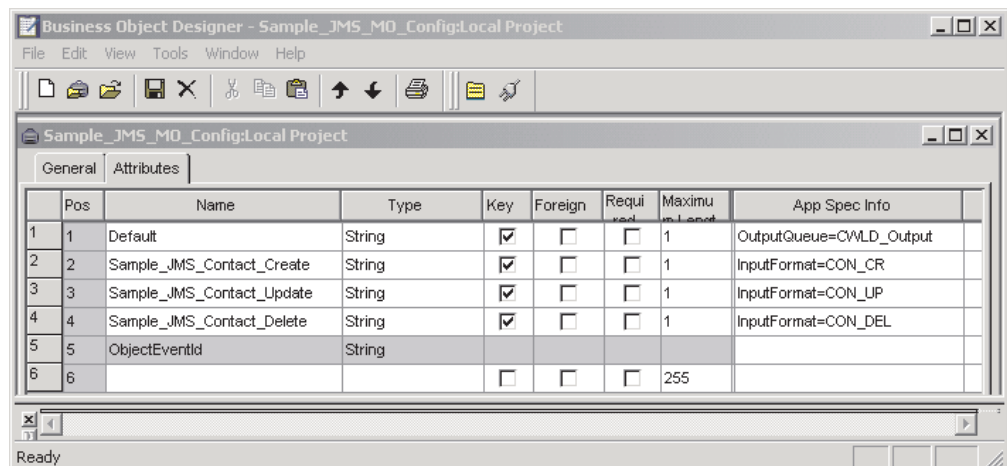


図 3. 静的メタオブジェクトのサンプル

3. 30ページの表10を参照して、要件に適合するように属性およびASIを編集してから、メタオブジェクト・ファイルを保管します。
4. このメタオブジェクト・ファイルの名前を、コネクタ・プロパティ ConfigurationMetaObject の値として指定します。

入力宛先へのデータ・ハンドラーのマッピング

静的メタオブジェクトのアプリケーション固有情報で `InputDestination` プロパティを使用することにより、データ・ハンドラーと入力宛先を関連付けることができます。この機能は、異なる書式や変換要件を持つ複数の取引先と取り引きする場合に役立ちます。

データ・ハンドラーを入力宛先にマップするには、以下のようになります。

1. `Connector Configurator` を起動します。詳細については、69 ページの『付録 B. `Connector Configurator`』を参照してください。
2. コネクタ固有プロパティ（26 ページの『`InputDestination`』を参照）を使用して、1 つ以上の入力宛先を構成します。複数の宛先名はセミコロンで区切る必要があります。
3. それぞれの入力宛先ごとに、宛先 (PTP メッセージング・スタイルをインプリメントしている場合はキュー・マネージャー) および入力宛先名を指定し、またアプリケーション固有情報にデータ・ハンドラーのクラス名および `MIME` タイプを指定します。

例えば、次に示す静的メタオブジェクトの属性は、データ・ハンドラーと、`CompReceipts` という名前の `InputDestination` を関連付けています。

```
[Attribute]
Name = Customer_Create
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo =
InputDestination=//queue.manager/CompReceipts;DataHandlerClassName=com.crossworlds.
DataHandlers.MQ.disposition_notification;DataHandlerMimeType=message/
disposition_notification
IsRequiredServerBound = false
[End]
```

動的子メタオブジェクトの構成

静的なメタオブジェクトに必要なメタデータを指定することが困難または実行不可能な場合、コネクタは、ビジネス・オブジェクト・インスタンスごとに実行時に配信されたメタデータをオプションで受け入れることができます。

動的メタオブジェクトを使用すると、要求処理のときに、要求ごとに、コネクタが使用するメタデータを変更してビジネス・オブジェクトを処理できます。また、イベント処理のときに、イベント・メッセージの情報を検索することができます。

動的メタオブジェクトは、各属性が、次のような単一のメタデータのプロパティと値を表す構造になっています。`meta-object property name =meta-object property value`

動的メタオブジェクトをインプリメントするには、それを子としてトップレベル・オブジェクトに追加し、トップレベル・オブジェクト ASI に名前と値のペア `cw_mo_conn=<MO attribute>` を組み込みます。`<MO attribute>` は、動的メタオブジェクトを表すトップレベル・オブジェクトの属性名です。以下に例を示します。

```

Customer (ASI = cw_mo_conn=MetaData)
|  -- Id
|  -- FirstName
|  -- LastName
|  -- ContactInfo
|  -- MetaData
|      -- OutputFormat = CUST
|      -- OutputDestination = QueueA

```

上記のように指定された要求を受け取ると、コネクタは、Customer オブジェクトを CUST というフォーマットを持ったメッセージに変換してから、メッセージをキュー QueueA に置きます。

ビジネス・オブジェクトは、同じ動的メタオブジェクトでも、異なった動的メタオブジェクトでも使用できます。また、まったく使用しないことも可能です。

注: すべての標準 IBM WebSphere データ・ハンドラーは、cw_mo_ タグを認識することによって、この動的メタオブジェクト属性を無視するように設計されています。アダプターに使用するカスタム・データ・ハンドラーを開発する場合、同じようにする必要があります。

コネクタは、コネクタに渡されるトップレベル・ビジネス・オブジェクトに子として追加される動的なメタオブジェクトから、変換プロパティを認識し、読み取ります。この動的な子メタオブジェクトの属性値は、コネクタの構成に使用される静的なメタオブジェクトに指定可能であった変換プロパティと重複します。

動的な子メタオブジェクトのプロパティは静的なメタオブジェクトから検出されるプロパティをオーバーライドするため、動的な子メタオブジェクトを指定する場合は、静的なメタオブジェクトを指定するコネクタ・プロパティを組み込む必要はありません。したがって、動的な子メタオブジェクトは、静的なメタオブジェクトとは無関係に使用することができ、その逆もまた同様です。

動的メタオブジェクトで、アプリケーション固有情報として指定できるプロパティの説明については、30 ページの『メタオブジェクト・プロパティ』の 30 ページの表 10 を参照してください。

動的メタオブジェクトを構成するには、以下のようにします。

1. Business Object Designer を起動します。詳細については、「ビジネス・オブジェクト開発ガイド」を参照してください。
2. サンプルのメタオブジェクト connectors¥JMS¥Samples¥Sample_JMS_DynMO.xsd を開きます。図 4 に Business Object Designer 内のサンプルの動的メタオブジェクトを示します。

Pos	Name	Type	Key	Foreign	Required	Card	Maximum Length	Default
1	OutputQueue	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	SCONN.IN
2	DataHandlerConfigMO	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		1	
3	DataHandlerMimeType	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		1	
4	OutputFormat	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		1	
5	InputQueue	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
6	InputFormat	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
7	ResponseTimeout	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		3	-1
8	TimeoutFatal	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		6	false
9	DeliveryMode	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
10	Priority	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
11	Destination	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
12	Expiration	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
13	MessageID	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
14	Redelivered	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
15	TimeStamp	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
16	Type	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
17	UserID	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
18	AppID	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
19	DeliveryCount	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
20	GroupID	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
21	GroupSeq	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
22	CorrelationID	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	
23	田 JMSProperties	JMSPropertyPairs	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		
24	ObjectEventId	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
25			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	

図4. 動的メタオブジェクトのサンプル

- このビジネス・オブジェクトの要件に適合するように属性およびプロパティを編集し、保管します。
- 動的メタデータ・オブジェクトを子としてトップレベル・オブジェクトに追加し、トップレベル・オブジェクト ASI に名前と値のペア `cw_mo_conn=<MO attribute>` を組み込みます。`<MO attribute>` は、動的メタオブジェクトを表すトップレベル・オブジェクトの属性名です。

ポーリング中の動的な子メタオブジェクトの含まれるデータ

ポーリング中に検索されたメッセージについてさらに詳しい情報をコラボレーションに提供するため、コネクタは、作成されたビジネス・オブジェクトに動的なメタオブジェクトが定義済みである場合、その特定の属性に値を取り込みます。

38 ページの表 14 に、動的な子メタオブジェクトがポーリング用に構造化される方法を示します。

表 14. ポーリング用の JMS 動的子メタオブジェクト構造

属性名	サンプル値
InputFormat	CUST_IN
InputQueue	MYInputQueue
OutputFormat	CxIgnore
OutputQueue	CxIgnore
ResponseTimeout	CxIgnore
TimeoutFatal	CxIgnore

表 14 に示すように、動的子メタオブジェクトで追加の属性 `Input_Format` および `Inputdestination` を定義できます。 `Input_Format` は検索したメッセージのフォーマットで読み込まれ、 `InputDestination` 属性には特定のメッセージの検索先となる宛先の名前が含まれます。子メタオブジェクト内にこれらのプロパティが定義されていない場合、これらには値が取り込まれません。

シナリオ例:

- コネクターは、キュー `MyInputQueue` からフォーマット `CUST_IN` でメッセージを取得します。
- コネクターはこのメッセージを `Customer` ビジネス・オブジェクトに変換し、アプリケーション固有のテキストを調べてメタオブジェクトが定義されているかどうかを判断します。
- メタオブジェクトが定義されている場合、コネクターはこのメタオブジェクトのインスタンスを作成し、定義に基づいて `InputDestination` および `InputFormat` 属性に値を取り込んで、ビジネス・オブジェクトを使用可能なコラボレーションにパブリッシュします。

JMS ヘッダーおよび動的子メタオブジェクト属性

動的メタオブジェクトに属性を追加すると、メッセージ・トランスポートの詳細情報を取得したりメッセージ・トランスポートを詳細に制御したりすることができます。このセクションでは、これらの属性、およびそれらがイベント通知と要求処理にどのような影響を与えるかについて説明します。

JMS プロパティ: 動的メタオブジェクトの他の属性と異なり、 `JMSProperties` は単一カーディナリティー子オブジェクトを定義する必要があります。この子オブジェクトの各属性は、以下のように JMS メッセージ・ヘッダーの可変部分で読み取り/書き込みを行う単一プロパティを定義する必要があります。

1. 属性の名前はセマンティック値を持ちません。
2. 属性のタイプは、JMS プロパティ・タイプに無関係に必ず `String` でなければなりません。
3. 属性のアプリケーション固有情報は、属性をマップする JMS メッセージ・プロパティの名前と形式を定義する 2 つの名前と値の組を含まなければなりません。名前はユーザー定義可能です。値の型は、以下のいずれかでなければなりません。
 - `Boolean`
 - `String`

- Int
- Float
- Double
- Long
- Short
- Byte

以下の表に、JMSProperties オブジェクトの属性に対して定義する必要があるアプリケーション固有情報プロパティを示します。

表 15. JMS プロパティ属性のアプリケーション固有情報

属性	指定可能な値	ASI	コメント
名前	有効な JMS プロパティ名 (有効 = ASI で定義されたタイプと互換性がある)	name=<JMS プロパティ名>;type=<JMS プロパティ・タイプ>	ベンダーによっては、拡張機能を提供するために特定のプロパティを予約している場合があります。一般に、ユーザーはベンダー固有の機能にアクセスする場合以外は、JMS で開始するカスタム・プロパティを定義してはなりません。
Type	String	type=<コメントを参照>	これは JMS プロパティのタイプです。JMS API は、JMS メッセージに値を設定するための多くのメソッドを提供します (例: setIntProperty、setLongProperty、setStringProperty)。ここで指定する JMS プロパティのタイプによって、どのメソッドを使用してメッセージのプロパティ値を設定するかが決まります。

下の例では、メッセージ・ヘッダーのユーザー定義フィールドにアクセスできるように、Customer オブジェクトに JMSProperties 子オブジェクトが定義されています。

```
Customer (ASI = cw_mo_conn=MetaData)
  -- Id
  -- FirstName
  -- LastName
  -- ContactInfo
  -- MetaData
    |-- OutputFormat = CUST
```

```

|-- OutputDestination = QueueA
|-- JMSProperties
|-- RoutingCode = 123 (ASI= name=RoutingCode;type=Int)
|-- Dept = FD (ASI= name=RoutingDept;type=String)

```

別の例を示すために、図 5 に、動的メタオブジェクトの属性 JMSProperties および JMS メッセージ・ヘッダーの 4 つのプロパティ (ID、GID、RESPONSE、および RESPONSE_PERSIST) の定義を示します。属性のアプリケーション固有情報はそれぞれの名前およびタイプを定義します。例えば、属性 ID はタイプ String の JMS プロパティ ID にマップされます。

	Pos	Name	Type	Key	Reqd	Card	App Spec Info
1	1	JMSProperties	TeamCenter_JMS_Properties	<input type="checkbox"/>	<input type="checkbox"/>	1	
1.1	1.1	ID	String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		name=ID,type=String
1.2	1.2	GID	String	<input type="checkbox"/>	<input checked="" type="checkbox"/>		name=GID,type=String
1.3	1.3	RESPONSE	String	<input type="checkbox"/>	<input checked="" type="checkbox"/>		name=RESPONSE,type=Boolean
1.4	1.4	RESP_PERSIST	String	<input type="checkbox"/>	<input checked="" type="checkbox"/>		name=RESPONSE_PERSIST,type=Boolean
1.5	1.5	ObjectEventId	String	<input type="checkbox"/>	<input type="checkbox"/>		
2	2	OutputFormat	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>		

図 5. 動的メタオブジェクトの JMS プロパティ属性

開始スクリプトの構成

コネクタには、開始スクリプトが付いてきます。ご使用のオペレーティング・システムによって、JMS.bat または JMS.sh のいずれかになります。コネクタの始動、コネクタの停止、およびコネクタの一時始動ログ・ファイルについては、ご使用のプラットフォームの「システム・インストール・ガイド」の始動に関する章を参照してください。

複数のコネクタ・インスタンスの作成

コネクタ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・インスタンス用に新規ディレクトリを作成します。
- 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。

新規ディレクトリの作成

それぞれのコネクタ・インスタンスごとにコネクタ・ディレクトリを作成する必要があります。このコネクタ・ディレクトリには、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・インスタンスを一意的に示します。

コネクタに、コネクタ固有のメタオブジェクトがある場合、コネクタ・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・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. 初期コネクタのファイルは、次のディレクトリに入っていない限りません。

```
ProductDir¥repository¥initialConnectorInstance
```

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

コネクタ定義の作成

Connector Configurator 内で、コネクタ・インスタンスの構成ファイル (コネクタ定義) 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. 初期コネクタの構成ファイル (コネクタ定義) をコピーし、名前変更します。
2. 各コネクタ・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・プロパティ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. 初期コネクタの始動スクリプトをコピーし、コネクタ・ディレクトリ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、40 ページの『新規ディレクトリの作成』で作成したコネクタ・ディレクトリに格納します。
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. 初期コネクタのショートカット・テキストをコピーし、新規コネクタ・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・コネクタ作成の詳細については、「コネクタ開発ガイド (C++ 用)」または「コネクタ開発ガイド (Java 用)」を参照してください。

コネクタの始動

コネクタは、**コネクタ始動スクリプト**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のランタイム・ディレクトリに存在していなければなりません。

```
ProductDir%\connectors%\connName
```

ここで、*connName* はコネクタを示します。始動スクリプトの名前は、表 16 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 16. コネクタの始動スクリプト

オペレーティング・システム	始動スクリプト
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」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* はコネクタの名前であり、*brokerName* は以下の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。

第 3 章 ビジネス・オブジェクトの作成または変更

- 『コネクタのビジネス・オブジェクトの構造』

コネクタとともに提供されるのは、サンプルのビジネス・オブジェクトのみです。システム・インテグレーター、コンサルタント、またはカスタマーは、ビジネス・オブジェクトをビルドする必要があります。

この章では、ビジネス・オブジェクトのコネクタ要件について説明します。これらの情報は、新規のビジネス・オブジェクトをインプリメントするための手引きとして役立ちます。

コネクタのビジネス・オブジェクトの構造

コネクタのインストールが完了したら、ビジネス・オブジェクトを作成する必要があります。ビジネス・オブジェクトの構造については、構成済みのデータ・ハンドラーによって定められている以外の要件はありません。コネクタ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、統合ブローカーの命名規則を参照してください。

コネクタは宛先からメッセージを検索し、ビジネス・オブジェクト (メタオブジェクトによって定義されたもの) にメッセージの内容を読み込もうとします。厳密に言うと、コネクタはビジネス・オブジェクト構造を制御したり、それに影響を及ぼしたりはしません。そのような働きは、メタオブジェクト定義とコネクタのデータ・ハンドラー要件の機能です。実際に、ビジネス・オブジェクト・レベルのアプリケーション・テキストは存在しません。むしろ、ビジネス・オブジェクトの検索や受け渡しの際のコネクタの主な役割は、ビジネス・オブジェクトへのメッセージ (逆もまた同様) の処理をモニターしてエラーの有無を確認することです。

ビジネス・オブジェクトの作成

1. 統合ブローカーを JMS アダプターのインスタンスで構成するときに、ブローカーがビジネス・オブジェクトを送信するアプリケーションを指定し、それを構成します。
2. JMS 宛先のメッセージをターゲット・アプリケーションによる処理に適したビジネス・オブジェクトに変換できるデータ・ハンドラーで、コネクタを構成します。これを行うには、`DataHandlerConfigMO` および `DataHandlerMimeType` のコネクタ・プロパティを指定するか、`DataHandlerClassName` プロパティを指定します。詳細については、22 ページの『コネクタ・プロパティの構成』を参照してください。オプションとして、静的および動的メタオブジェクトで、特別なデータ・ハンドラー処理規則を指定できます。詳細については、30 ページの『メタオブジェクト・プロパティ』を参照してください。
3. *Business Object Designer* を使用して、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。詳細については、*Business Object Designer* のガイドを参照してください。

4. 作成したビジネス・オブジェクトをサポートされているビジネス・オブジェクトに追加します。Connector Configurator を使用して、JMS アダプターの「サポートされているビジネス・オブジェクト」タブをクリックして、作成したビジネス・オブジェクトを追加し、**Message Set ID** をサポートされている各ビジネス・オブジェクトの固有な値に設定します。Connector Configurator を使用してサポートされているビジネス・オブジェクトを追加する方法の詳細については、79 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』を参照してください。

第 4 章 トラブルシューティング

- 『エラー処理』
- 48 ページの『トレース』
- 49 ページの『開始に関する問題の修正』

この章では、コネクターがエラーおよびトレースを処理する方法、およびコネクターの始動時および実行時に発生する可能性がある問題について説明します。

エラー処理

コネクターが生成するすべてのエラー・メッセージは、`JMSConnector.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によって決定されます)。それぞれのエラー・メッセージの前にはエラー番号が付けられています。

Message number
Message text

コネクターは、以下の各セクションで説明するような特定のエラーを処理します。

アプリケーションのタイムアウト

以下のような場合には、エラー・メッセージ `ABON_APPRESPONSETIMEOUT` が戻されます。

- メッセージ検索中に、コネクターが JMS サービス・プロバイダーとの接続を確立できない。
- コネクターはビジネス・オブジェクトを正常にメッセージに変換したが、接続が切断されたためにメッセージを出力キューに配信できない。
- コネクターはメッセージを発行したが、変換プロパティ `TimeoutFatal` の値が `True` であるビジネス・オブジェクトの応答待ちがタイムアウトになった。
- コネクターが戻りコード `APP_RESPONSE_TIMEOUT` または `UNABLE_TO_LOGIN` を含む応答メッセージを受信した。

アンサブスクライブされたビジネス・オブジェクト

アンサブスクライブされたビジネス・オブジェクトに関連するメッセージを検索する場合、あるいは `gotAppEvent()` メソッドから `NO_SUBSCRIPTION_FOUND` が戻された場合、コネクターは、`UnsubscribedDestination` プロパティに指定されたキューにメッセージを配信します。

注: `UnsubscribedDestination` が定義されていない場合、アンサブスクライブされたメッセージは廃棄されます。

アクティブでないコネクタ

`gotAppEvent()` メソッドが `CONNECTOR_NOT_ACTIVE` コードを戻すと、
`pollForEvents()` メソッドは `APP_RESPONSE_TIMEOUT` コードを戻し、イベント
は `InProgress Destination` に置かれたままになります (指定されている場合)。

データ・ハンドラーの変換

データ・ハンドラーがメッセージをビジネス・オブジェクトに変換できなかった場合や (JMS プロバイダーではなく) ビジネス・オブジェクトに固有の処理エラーが発生した場合、メッセージは、`ErrorDestination` で指定されたキューに送信されま
す。`ErrorDestination` が定義されていない場合、エラーが原因で処理できないメッ
セージは廃棄されます。

データ・ハンドラーがビジネス・オブジェクトのメッセージへの変換に失敗した場
合、`BON_FAIL` が戻されます。

トレース

トレース・メッセージはアダプターにハードコーディングされています。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・メッセージの構成の詳細については、コネクタ構成プ
ロパティを参照してください。トレースの有効化および設定の方法など、詳細に
ついては、「コネクタ開発ガイド (Java 用)」を参照してください。

以下に、コネクターのトレース・メッセージの推奨レベルを示します。

- | | |
|-------|--|
| レベル 0 | このレベルは、コネクターのバージョンを示すトレース・メッセージに使用します。 |
| レベル 1 | このレベルは、処理される各ビジネス・オブジェクトに関するキー情報を提供したり、ポーリング・スレッドが入力キューに新規のメッセージを検出するたびに記録したりするトレース・メッセージに使用します。 |
| レベル 2 | ビジネス・オブジェクトが <code>gotAppEvent()</code> または <code>executeCollaboration()</code> からブローカーに送付されるたびに記録されるトレース・メッセージに使用します。 |
| レベル 3 | このレベルは、メッセージからビジネス・オブジェクトおよびその反対の変換についての情報を提供したり、メッセージの出力キューへのデリバリーについての情報を提供したりするトレース・メッセージに使用します。 |
| レベル 4 | このレベルは、コネクタがある関数を入力または出力する場合を示すトレース・メッセージに使用します。 |
| レベル 5 | このレベルは、コネクターの初期化の通知、アプリケーション内で実行されるステートメントの表現、メッセージがキューから出し入れされる際の通知、またはビジネス・オブジェクト・ダンプの記録をするトレース・メッセージに使用します。 |

開始に関する問題の修正

問題	可能性のある解決方法/説明
コネクターが初期設定中に不意にシャットダウンして、 「Exception in thread "main" java.lang.NoClassDefFoundError: javax/jms/JMSEException...」というメッセージがレポートされた。	コネクターがファイル <code>jms.jar</code> を検出できません。
コネクターが初期設定中に不意にシャットダウンして、 「Exception in thread "main" java.lang.NoClassDefFoundError: javax/naming/Referenceable...」というメッセージがレポートされた。	コネクターがファイル <code>jndi.jar</code> を検出できません。

付録 A. コネクターの標準構成プロパティ

この付録では、WebSphere Business Integration Adapter のコネクタ・コンポーネントの標準構成プロパティについて説明します。この付録の内容は、以下の統合ブローカーで実行されるコネクタ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)

コネクタによっては、一部の標準プロパティ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きには、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のリストが表示されます。

コネクタ固有のプロパティの詳細については、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・パスに円記号 (¥) 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。

新規プロパティと削除されたプロパティ

以下の標準プロパティは、本リリースで追加されました。

新規プロパティ

- XMLNamespaceFormat

削除されたプロパティ

- RestartCount
- RHF2MessageDomain

標準コネクタ・プロパティの構成

アダプター・コネクタには 2 つのタイプの構成プロパティがあります。

- 標準構成プロパティ
- コネクタ固有のプロパティ

このセクションでは、標準構成プロパティについて説明します。コネクタ固有の構成プロパティについては、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・プロパティを構成します。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、付録の『Connector Configurator』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。コネクタを UNIX システム上で稼動している場合でも、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のコネクタ・プロパティを設定する場合は、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、コネクタ用の Connector Configurator を開く必要があります。

プロパティ値の設定と更新

プロパティ・フィールドのデフォルトの長さは 255 文字です。

コネクタは、以下の順序に従ってプロパティの値を決定します (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 (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・プロパティの値を変更する場合は、プロパティの**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・プロパティには、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を変更できます。この場合、動的更新は実行できません。

• コンポーネント再始動

System Manager でコネクタ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。

• 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。

特定のプロパティ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、次に示すプロパティの要約の表の「更新メソッド」列を参照してください。

標準プロパティの要約

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

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

表 17. 標準構成プロパティの要約

プロパティ名	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>
AgentTraceLevel	0 から 5	0	動的	
ApplicationName	アプリケーション名	コネクタ・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			
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>
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
ControllerStoreAndForwardMode	true または false	True	動的	Repository Directory は <REMOTE>
ControllerTraceLevel	0 から 5	0	動的	Repository Directory は <REMOTE>
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

表 17. 標準構成プロパティの要約 (続き)

プロパティ名	指定可能な値	デフォルト値	更新メソッド	注
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
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>
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>
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>
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

表 17. 標準構成プロパティの要約 (続き)

プロパティ名	指定可能な値	デフォルト値	更新メソッド	注
MaxEventCapacity	1 から 2147483647	2147483647	動的	Repository Directory は <REMOTE> でなければならぬ
MessageFileName	パスまたはファイル名	InterchangeSystem.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 でなければならぬ
OADAutoRestartAgent	True または False	False	動的	Repository Directory は <REMOTE> でなければならぬ
OADMaxNumRetry	正数	1000	動的	Repository Directory は <REMOTE> でなければならぬ
OADRetryTimeInterval	正数 (単位: 分)	10	動的	Repository Directory は <REMOTE> でなければならぬ
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の要約 (続き)

プロパティ名	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	
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
WsifSynchronousRequest Timeout	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

このセクションでは、各標準コネクタ構成プロパティの定義を示します。

AdminInQueue

統合ブローカーからコネクタ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` プロパティは、`orb.init[]` により開かれる ORB 接続の数を制御します。

デフォルトでは、このプロパティの値は 1 に設定されます。このデフォルト値を変更する必要はありません。

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。コネクタを実行する前に、このプロパティ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 ベースのコネクタでは、このプロパティは使用しません。C++ ベースのコネクタでは、現在、このプロパティに `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・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・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、`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 構成プロパティに、1 より大きい値を設定します。

ConcurrentEventTriggeredFlows プロパティは、順次に行わ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 No value です。

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も構成する必要があります。

- PollQuantity = 1 から 500
- SourceQueue = CONNECTORNAME/SOURCEQUEUE

また、MimeType、DHClass、および DataHandlerConfigMOName (オプション) プロパティを設定したデータ・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。「データ・ハンドラー」タブ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。

このプロパティは、DeliveryTransport プロパティが値 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・コントローラーが検出した場合に、コネクタ・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・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・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・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・コントローラーはその要求を失敗させます。

このプロパティを false に設定した場合、コネクタ・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・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。

- ICS がブローカー・タイプの場合は、DeliveryTransport プロパティの指定可能な値は 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` プロパティに指定されている値が、MQ または IDL である場合、コネクタは、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・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・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・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、コネクタとクライアント・コネクタ・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です。

重要: 以下の環境では、コネクタに JMS トランスポート機構を使用すると、メモリ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が使用されるため、(サーバー側の) コネクタ・コントローラーと (クライアント側の) コネクタの両方を

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 プロパティの値を 11 または 12 に設定する。このプロパティの詳細については、「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

このプロパティを true に設定すると、JMS 対応コネクタによるデリバリー・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。これはコネクタ開発時に設定されます。

このプロパティは、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

コネクタでメッセージを処理中にエラーが発生すると、コネクタは、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・プロパティを必ず 設定してください。

デフォルト値は crossworlds.queue.manager です。

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。

<code>ll</code>	2 文字の言語コード (普通は小文字)
<code>TT</code>	2 文字の国または地域コード (普通は大文字)
<code>codeset</code>	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・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・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、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。コネクタがグローバル化に対応していない場合、このプロパティの有効な値は `en_US` のみです。特定のコネクタ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/wicsserver/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・バッファ内のイベントの最大数。このプロパティはフロー制御が使用し、`RepositoryDirectory` プロパティの値が `<REMOTE>` の場合のみ適用されます。

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・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 `¥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・メッセージ・ファイルが存在しない場合は、コネクタは `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: 特定のコネクタについて、コネクタ独自のメッセージ・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。このプロパティは、`DeliveryTransport` プロパティ値が `JMS` であり、かつ `DuplicateEventElimination` が `TRUE` に設定されている場合のみ使用されます。

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を再始動したり、System Monitor からリモート・コネクタ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、このプロパティを `true` に設定する必要があります。MQ により起動される 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 により起動される OAD がコネクタ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。このプロパティを有効にするためには、`OADAutoRestartAgent` プロパティを `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 により起動される OAD の再試行時間間隔の分数を指定します。コネクタ・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・コント

ローラーはコネクタ・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 プロパティで指定された回数だけ繰り返します。このプロパティを有効にするためには、OADAutoRestartAgent プロパティを 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

ポーリング・アクション間の時間の長さです。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。
- ワード key。コネクタは、コネクタのコマンド・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: 一部のコネクタでは、このプロパティの使用が制限されています。このプロパティが使用されるかどうかを特定のコネクタについて判別するには、該当するアダプター・ガイドのインストールと構成についての章を参照してください。

PollQuantity

コネクタ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固有のポーリング数設定プロパティがある場合、標準プロパティの値は、このコネクタ固有のプロパティの設定値によりオーバーライドされます。

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

コネクタが XML スキーマ文書を読み取るリポジトリの場所です。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、コネクタが InterChange Server リポジトリ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・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

コネクタによるコネクタ自体の再始動の試行回数を指定します。このプロパティを並列コネクタに対して使用する場合、コネクタ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

コネクタによるコネクタ自体の再始動の試行間隔を分単位で指定します。このプロパティを並列コネクタに対して使用する場合、コネクタ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。

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・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、58 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・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・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・フレームワークに配信します。このキューは、コネクタが同期実行を使用する場合にのみ必要です。

デフォルトは CONNECTORNAME/SYNCHRONOUSRESPONSEQUEUE です。

SynchronousRequestTimeout

DeliveryTransport が JMS の場合のみ適用されます。

コネクタ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。

WsifSynchronousRequest Timeout

WAS 統合ブローカーでのみ使用されます。

コネクタ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。

- コネクタを構成するためのコネクタ固有のプロパティ・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を設定する

注:

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。

- 69 ページの『Connector Configurator の概要』
- 70 ページの『Connector Configurator の始動』
- 71 ページの『コネクタ固有のプロパティ・テンプレートの作成』
- 74 ページの『新しい構成ファイルを作成』
- 77 ページの『構成ファイル・プロパティの設定』
- 85 ページの『グローバル化環境における Connector Configurator の使用』

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・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。

- コネクタを構成するためのコネクタ固有のプロパティ・テンプレートを作成する。
- **コネクタ構成ファイル**を作成します。インストールするコネクタ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を設定する。
場合によっては、コネクタ・テンプレートでプロパティ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 (70 ページの『スタンドアロン・モードでの Configurator の実行』を参照)。

コネクタ構成プロパティには、標準の構成プロパティ (すべてのコネクタにもつプロパティ) と、コネクタ固有のプロパティ (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に必要なプロパティ) とが含まれます。

標準プロパティはすべてのコネクタにより使用されるので、標準プロパティ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の値を設定する必要があります。

標準プロパティの範囲は、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があります。Connector Configurator の「標準のプロパティ」ウィンドウには、特定の構成で設定可能なプロパティが表示されます。

ただし**コネクタ固有プロパティ**の場合は、最初にプロパティを定義し、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固有プロパティ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、71 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 Toolset**」>「**開発**」>「**Connector Configurator**」をクリックします。
- 「ファイル」>「新規」>「構成ファイル」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 (76 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」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」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。

1. 「System Manager」ウィンドウの「コネクタ」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。
2. 「標準のプロパティ」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を確認します。

コネクタ固有のプロパティ・テンプレートの作成

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

コネクタ固有プロパティのテンプレートを新規に作成するか、または既存のファイル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、71 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、新しい名前でのこのテンプレートを保管します。

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を作成し、プロパティの一般特性および値を定義し、プロパティ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。

テンプレートは以下のように作成します。

1. 「ファイル」>「新規」>「コネクタ固有プロパティ・テンプレート」をクリックします。
2. 以下のフィールドを含む「コネクタ固有プロパティ・テンプレート」ダイアログ・ボックスが表示されます。
 - 「テンプレート」、「名前」
このテンプレートが使用されるコネクタ（またはコネクタのタイプ）を表す固有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。
 - 「旧テンプレート」、「変更する既存のテンプレートを選択してください」
「テンプレート名」表示に、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。
 - テンプレートに含まれているコネクタ固有のプロパティ定義を調べるには、「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。テンプレートを作成するときには、ご使用のコネクタに必要なプロパティ定義に類似したプロパティ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。
3. 「テンプレート名」表示からテンプレートを選択し、その名前を「名前の検索」フィールドに入力し（または「テンプレート名」で自分の選択項目を強調表示し）、「次へ」をクリックします。

ご使用のコネクタで使用するコネクタ固有のプロパティ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。

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: コネクタ固有プロパティ・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・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. 「プロパティを編集」表示でプロパティの名前を選択します。
3. 「最大長」および「最大複数値」のフィールドで、変更を行います。次のステップで説明するように、プロパティの「プロパティ値」ダイアログ・ボックスを開かない限り、そのプロパティの変更内容は受け入れられませんので、注意してください。
4. 値テーブルの左上の隅にあるボックスを右マウス・ボタンでクリックしてから、「追加」をクリックします。「プロパティ値」ダイアログ・ボックスが表示されます。このダイアログ・ボックスではプロパティのタイプに応じて、値だけを入力できる場合と、値と範囲の両方を入力できる場合があります。適切な値または範囲を入力し、「OK」をクリックします。
5. 「値」パネルが最新表示され、「最大長」および「最大複数値」で行った変更が表示されます。以下のような 3 つの列があるテーブルが表示されます。

「値」の列には、「プロパティ値」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・テンプレート」ダイアログ・ボックスが表示されます。

依存プロパティは、別のプロパティ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。プロパティを依存プロパティとして指定し、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」表示で、依存プロパティとして指定するプロパティを選択します。
2. 「プロパティを選択」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」表示で依存プロパティを強調表示させて矢印をクリックし、「依存プロパティ」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。

新しい構成ファイルを作成

構成ファイルを新規に作成するには、最初に統合ブローカーを選択します。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。

- Connector Configurator のホーム・メニューで、「ファイル」>「新規」>「コネクター構成」をクリックします。「新規コネクター」ダイアログ・ボックスが表示されます。
- 「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。

また、以下の作業も実行できます。

- 「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。

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・ボックス表示されず。

- **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。

- 「コネクタ固有プロパティ・テンプレート」を選択します。

ご使用のコネクタ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・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固有のプロパティが表示されます。

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」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、コネクタの始動ファイルで指定するコネクタ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。

- コネクタ定義ファイル。
コネクタ定義ファイルは、特定のコネクタのプロパティと、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 コネクタの場合は `CN_XML.txt` です)。
- ICS リポジトリー・ファイル。
コネクタの以前の ICS インプリメンテーションで使用した定義は、そのコネクタの構成で使用されたりポジトリー・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、コネクタのコネクタ固有プロパティのほとんど、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、コネクタ構成ファイルは、ファイルを開いて、プロパティ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 (*.in、*.out)
ICS 環境でのコネクタの構成にリポジトリ・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・ファイルに複数のコネクタ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」タブで、BrokerType プロパティ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が「標準のプロパティ」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 79 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。

- 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すべてに値が設定されているかどうかを確認されます。必須の標準プロパティ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・プロパティの設定

新規のコネクタ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で、以下のカテゴリのプロパティ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
- コネクタ固有のプロパティ
- 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の場合に該当する)

注: JMS メッセージングを使用するコネクタの場合は、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の場合、以下のプロパティの値も設定されている必要があります。

- 関連マップ
- 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値も設定可能です。ただし、標準のプロパティおよびコネクタ固有プロパティ、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。

標準プロパティとコネクタ固有プロパティの違いは、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有されます。すべてのコネクターが同じ標準プロパティのセットを使用します。これらのプロパティの説明は、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があります。これらのプロパティには、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および推奨値が記述されています。

「標準プロパティ」と「コネクター固有プロパティ」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を表します。これらのプロパティは、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の除去も可能です。
- 「値」フィールドは構成できます。
- 「更新メソッド」フィールドは通知用であり、構成できません。このフィールドは、値が変更されたプロパティをアクティブにするために必要なアクションを示します。

標準コネクター・プロパティの設定

標準のプロパティ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」(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の設定

アプリケーション固有の構成プロパティの場合、プロパティ名の追加または変更、値の構成、プロパティの削除、およびプロパティの暗号化が可能です。プロパティ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。プロパティを追加するときは「追加」をクリックします。子プロパティを追加するには、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. プロパティまたは子プロパティの値を入力します。
3. プロパティを暗号化するには、「暗号化」ボックスを選択します。
4. 78 ページの『標準コネクタ・プロパティの設定』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・プロパティ名を変更すると、コネクタに障害が発生する可能性があります。コネクタ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、特定のプロパティ名が必要である場合があります。

コネクタ・プロパティの暗号化

「プロパティを編集」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。

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

プロパティに複数の値がある場合には、プロパティの最初の値に「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、そのプロパティのすべての値が暗号化されます。プロパティの複数の値を暗号化解除するには、そのプロパティの最初の値の「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、すべての複数值が暗号化解除されます。

更新メソッド

付録『コネクタの標準構成プロパティ』の 52 ページの『プロパティ値の設定と更新』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、「コネクタ開発ガイド (C++ 用)」または「コネクタ開発ガイド (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定義が、System Manager の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、「プロジェクトに保管」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・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。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。

関係付けられたマップ (ICS のみ)

各コネクタは、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リプト・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、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 つのマップを見つけて、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・エージェントが、コネクタ・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・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・プロパティは、DeliveryTransport 標準プロパティの値として 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は、コネクタ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

コネクタ構成ファイルまたはコネクタ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、指定する格納場所へ移動し、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 の『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。
- System Manager から、指定したディレクトリーに `*.con` 拡張子付きファイルとして保管します。
- 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。

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* 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」タブを選択します。
- 「標準のプロパティ」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更新され、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、コネクタの始動時にコネクタ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、コネクタが使用する始動ファイルを開き、コネクタ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>
```

付録 C. チュートリアル

- 『チュートリアルの概要』
- 88 ページの『環境のセットアップ』
- 90 ページの『シナリオの実行』
- 90 ページの『静的メタオブジェクト・シナリオの実行』
- 91 ページの『動的メタオブジェクト・シナリオの実行』

この付録では、JMS メッセージ・キューを介して通信するアプリケーションとの間で、アダプターを使用してビジネス・オブジェクトを送受信する方法について説明します。このチュートリアルのシナリオは、アダプターの基本的な機能について説明することを目的としています。

表記規則のガイドについては、この文書のまえがきを参照してください。

チュートリアルの概要

このチュートリアルは 2 つのシナリオから構成されています。1 つは静的メタオブジェクトを使用したシナリオで、もう 1 つは動的メタオブジェクトを使用したシナリオです。いずれのシナリオでも ApplicationX を使用します。ApplicationX を使用すると、会社連絡先の作成、更新、削除時に情報を交換できます。シナリオで作成するビジネス・オブジェクト Sample_JMS_Contact は、ApplicationX からのメッセージに定義されたフィールドと一致します。ApplicationX が送受信するメッセージのフォーマットは、IBM WebSphere Business Integration 開発キットに付属している区切りデータ・ハンドラーに準拠しています。

また、このチュートリアルではポート・コネクタ・リポジトリを使用します。ポート・コネクタ・リポジトリは WebSphere アダプターをインストールすればそのコンポーネントとしてインストールされます。ポート・コネクタはコネクタの定義のみから構成され、基本となるコードは存在しないため、シミュレーション・シナリオに適しています。

始動した JMS アダプターは、ApplicationX が入力キューに書き込んだ連絡メッセージを検索します。アダプターは区切りデータ・ハンドラーを使用することにより、これらのメッセージを Sample_JMS_Contact ビジネス・オブジェクトに変換し、統合ブローカーに送達します。Test Connector (やはり、WBI をインストールすると組み込まれているコンポーネント) を使用することにより、ポート・コネクタをシミュレートし、JMS アダプターが発行したビジネス・オブジェクトを検索し、属性を確認することができます。データを変更してから、メッセージを統合ブローカーに再送達します。ここからメッセージは JMS アダプターに送信され、メッセージに変換され、アダプターの出力キュー (ApplicationX の入力キュー) に送達されます。このチュートリアルでは、アダプターは WebSphere MQ Integrator Broker 用に構成されていますが、チュートリアルを実行するためにこのブローカーを実際にインストールする必要はありません。

チュートリアルを開始する前に、以下のことを確認してください。

- IBM WebSphere 製品がインストールされ、その運用経験をもっていること。
- JMS サービス・プロバイダーがインストールされていること。
- WBI Adapter for JMS がインストールされていること。
- WebSphere MQ 5.3 がインストールされ、その運用経験をもっていること。

環境のセットアップ

このセクションでは、チュートリアルを使用して作業できる環境の準備の仕方について説明します。後出の *sample_folder* は、サンプルがあるフォルダーを指します。

1. **キューの作成** このチュートリアルでは、JMS サービス・プロバイダーに 6 つのキューが定義されていることが必要です。これらのキューを定義する前に、JMS プロバイダーの資料を参照してください。次のキューを定義してください (あるいは JNDI ルックアップを介して使用可能にしてください)。

- CWLD_Input
- CWLD_InProgress
- CWLD_Error
- CWLD_Archive
- CWLD_Unsubscribed
- CWLD_Output

2. **WebSphere MQ キュー・マネージャーの作成および開始** チャネル・イニシエーターとリスナーも実行します。

3. **キューの定義** WMQI ブローカーを構成するために WebSphere MQ アダプターおよびポート・コネクターが必要とするキューを以下のように定義します。

- DEFINE QL('JMSConnector/ADMININQUEUE')
- DEFINE QL('JMSConnector/ADMINOUTQUEUE')
- DEFINE QL('JMSConnector/DELIVERYQUEUE')
- DEFINE QL('JMSConnector/FAULTQUEUE')
- DEFINE QL('JMSConnector/REQUESTQUEUE')
- DEFINE QL('JMSConnector/RESPONSEQUEUE')
- DEFINE QL('JMSConnector/SYNCHRONOUSREQUESTQUEUE')
- DEFINE QL('JMSConnector/SYNCHRONOUSRESPONSEQUEUE')
- DEFINE QL('PortConnector/ADMININQUEUE')
- DEFINE QL('PortConnector/ADMINOUTQUEUE')
- DEFINE QL('PortConnector/DELIVERYQUEUE')
- DEFINE QL('PortConnector/FAULTQUEUE')
- DEFINE QL('PortConnector/REQUESTQUEUE')
- DEFINE QL('PortConnector/RESPONSEQUEUE')
- DEFINE QL('PortConnector/SYNCHRONOUSREQUESTQUEUE')
- DEFINE QL('PortConnector/SYNCHRONOUSRESPONSEQUEUE')

4. **アダプターの構成** Connector Configurator を使用して、*sample_folder*¥JMSConnector.cfg を開きます。Connector Configurator の使用法に関する詳細情報は、69 ページの『付録 B. Connector Configurator』を参照してく

ください。コネクタ固有プロパティの詳細については、22 ページの『コネクタ固有プロパティの構成』を参照してください。

以下の標準プロパティを設定します。

- **Broker Type** このプロパティを `WMQI` に設定します。
- **Repository Directory** このプロパティを `sample_folder` ディレクトリに設定します。

以下のコネクタ固有プロパティを設定します。

- **ConfigurationMetaObject** このプロパティを `Sample_JMS_MO_Config` に設定します。
 - **DataHandlerConfigMO** このプロパティを `Sample_JMS_MO_DataHandler` に設定します。
 - **DataHandlerMimeType** このプロパティを `text/delimited` に設定します。
 - **ErrorQueue** このプロパティを `CWLD_Error` に設定します。
 - **InProgressQueue** このプロパティを `CWLD_InProgress` に設定します。
 - **InputQueue** このプロパティを `CWLD_Input` に設定します。
 - **UnsubscribedQueue** このプロパティを `CWLD_Unsubscribed` に設定します。
5. **ポート・コネクタの構成** Connector Configurator を使用して、以下の標準プロパティを設定します。
- **Broker Type** このプロパティを `WMQI` に設定します。
 - **Repository Directory** このプロパティを `sample_folder` ディレクトリに設定します。
 - **RequestQueue** このプロパティを `JMSConnector/DELIVERYQUEUE` に設定します (JMS アダプターの `DeliveryQueue` プロパティ値)。
 - **DeliveryQueue** このプロパティを `JMSConnector/REQUESTQUEUE` に設定します (JMS アダプターの `RequestQueue` プロパティ値)。
6. **ビジネス・オブジェクトのサポート** ビジネス・オブジェクトを使用するには、まずアダプターがビジネス・オブジェクトをサポートする必要があります。Connector Configurator を使用して、JMS アダプターの「サポートされているビジネス・オブジェクト」タブをクリックして、表 18 に記載されているビジネス・オブジェクトを追加します。「メッセージ・セット ID」をサポートされているそれぞれのビジネス・オブジェクトごとに固有の値に設定します。

表 18. JMS アダプターについてサポートされるサンプル・ビジネス・オブジェクト

ビジネス・オブジェクト名	メッセージ・セット ID
Sample_JMS_MO_Config	1
Sample_JMS_MO_DataHandler	2
Sample_JMS_Contact	3

Connector Configurator を使用して、`sample_folder` 内にあるポート・コネクタ定義 `PortConnector.cfg` を開きます。次に、表 19 に記載されたサポートされるビジネス・オブジェクトとメッセージ・セット ID を追加します。

表 19. ポート・コネクタについてサポートされるサンプル・ビジネス・オブジェクト

ビジネス・オブジェクト名	メッセージ・セット ID
Sample_JMS_Contact	1

7. コネクター開始スクリプトの作成または更新

Windows の場合

- a. JMS 用アダプターのショートカットのプロパティーを開きます。
- b. ターゲットの最後の引き数として、`-c` の後ろに `<JMSConnector.cfg` ファイルの絶対パスおよびファイル名> を続けたものを追加します。例:

```
-cProduct_Dir¥connectors¥JMS¥samples¥JMSConnector.cfg
```

UNIX の場合

- a. ファイル `Product_Dir/bin/connector_manager_JMS` を開きます。
- b. `AGENTCONFIG_FILE` プロパティーを `-c` の後ろに `<JMSConnector.cfg` ファイルの絶対パスおよびファイル名> を続けた値に設定します。例:

```
AGENTCONFIG_FILE=-cProduct_Dir/connectors/JMS/samples/JMSConnector.cfg
```

シナリオの実行

シナリオを実行する前に、以下の手順を実行します。

1. **JMS** 用アダプターがまだ稼動していない場合は始動します。
2. **Visual Test Connector** がまだ稼動していない場合は始動します。

静的メタオブジェクト・シナリオの実行

チュートリアルはこのセクションでは、静的メタオブジェクトを使用したシナリオについて説明します。静的メタオブジェクトの詳細については、33 ページの『静的メタオブジェクトの構成』を参照してください。

1. **ポート・コネクターのシミュレート** Visual Test Connector を使用して、PortConnector のプロファイルを定義します。
 - a. 「Visual テスト・コネクター」メニューから「ファイル」->「プロファイルを作成/選択」を選択し、次に、「コネクター・プロファイル」メニューから「ファイル」->「新規プロファイル」を選択します。
 - b. Samples ディレクトリー内にあるポート・コネクター構成ファイル PortConnector.cfg を選択して、Connector Name および Broker Type を構成してから「OK」をクリックします。
 - c. 作成したプロファイルを選択し、「OK」をクリックします。
 - d. 「Visual テスト・コネクター」メニューから、「ファイル」->「接続」を選択してシミュレートを開始します。
2. **要求処理のテスト**
 - a. Test Connector を使用して、ビジネス・オブジェクト Sample_JMS_Contact の新規インスタンスを作成します。これを実行するには、**BoType** ドロップダウン・ボックスでビジネス・オブジェクトを選択してから、BOInstance の「作成」を選択します。
 - b. 必要に応じてデフォルト値を変更し、動詞を **Create** に設定して、「ビジネス・オブジェクトを送信」をクリックしてメッセージを送信します。
3. **メッセージ送達の検査** キュー CWLD_Output を開き、フォーマットが CON_CR の新規連絡メッセージが JMS アダプターから届いているか確認します。

4. **イベント処理のテスト** メッセージを JMS アダプターの入力キューに送信します。注: このステップでは、キューにメッセージを送ることができるユーティリティが必要で、このようなユーティリティが使用できない場合は、JMS アダプターの `InputQueue` プロパティを `CWLD_Output` に設定します。これにより、アダプターは自身のメッセージをポーリングできます (これが最も容易な方法です)。入力キューにメッセージが入ると、JMS アダプターはこのメッセージに対するポーリングを実行し、これを `Sample_JMS_Contact` ビジネス・オブジェクトに変換しようとします。アダプターにメッセージのポーリングを実行させるために重要なことは、メッセージ・フォーマットが、メタオブジェクト `Sample_JMS_MO_Config` 内の `Sample_JMS_Contact` ビジネス・オブジェクトに関連付けられた値と等しいことです。このシナリオの場合、フォーマットは `CON_CR` です。アダプターは、着信メッセージ・フォーマットを `CON_CR` であると認識すると、データ・ハンドラーを使用して、動詞 `create` 付きビジネス・オブジェクト `Sample_JMS_Contact` にメッセージを変換します。その後、この新しく作成されたビジネス・オブジェクトは `Test Connector` に送達されます。
5. **メッセージ送達の確認** 上記のステップがすべて正常に実行された場合には、適切なサンプル・シナリオが得られ、このシナリオにより JMS アダプターがメッセージを検索し、これらのメッセージを `Sample_JMS_Contact` ビジネス・オブジェクトに変換し、さらに、逆に `Sample_JMS_Contact` ビジネス・オブジェクトを連絡メッセージに変換することが可能になります。

動的メタオブジェクト・シナリオの実行

このシナリオでは、動的メタオブジェクトを使用して、JMS サービス・プロバイダーに定義された各種のキューにビジネス・オブジェクトを転送する方法について説明します。動的メタオブジェクトの詳細については、35 ページの『動的子メタオブジェクトの構成』を参照してください。以下のステップでは、`Sample_JMS_Contact` の子メタオブジェクトの属性を作成します。特に、この子メタオブジェクトの出力キュー値を変更することにより、`Sample_JMS_Contact` ビジネス・オブジェクトを各種のキューに転送します。

子メタオブジェクト・リポジトリ `Sample_JMS_DynMO.xsd` は `sample_folder` にあります。

1. **動的メタオブジェクト属性の識別** まず、動的メタオブジェクトが設定された属性を識別するために、アプリケーション固有情報を追加する必要があります。`Sample_JMS_Contact` で、`cw_mo_conn=DynMO` をアプリケーション固有情報に追加します。これにより属性が識別されます。
2. **属性の追加** `Business Object Designer` を使用して、以下の手順を実行します。
 - a. `sample_folder` から `Sample_JMS_DynMO.xsd` および `Sample_JMS_Contact.xsd` を開きます。
 - b. 「`Sample_JMS_Contact Object`」ウィンドウで、名前が `DynMO` でタイプが `Sample_JMS_DynMO` の属性を追加します。
 - c. 「`Sample_JMS_Contact Object`」をダブルクリックします。
 - d. 属性フォルダーを選択して、名前が `DynMO` でタイプが `Sample_JMS_DynMO` の属性を追加します。

3. **新しいターゲット・キューの定義** JMS サービス・プロバイダーで一時キュー REROUTE.IN を定義します。これは、動的メタオブジェクトによる Sample_JMS_Contact ビジネス・オブジェクトの転送先です。
4. **JMS 用アダプターがまだ稼動していない場合は始動します。**
5. **Visual Test Connector がまだ稼動していない場合は始動します。**
6. **ポート・コネクターのシミュレート** Visual Test Connector を使用して、PortConnector のプロファイルを定義します。
 - a. 「Visual テスト・コネクター」メニューから「ファイル」->「プロファイルを作成/選択」を選択し、次に、「コネクター・プロファイル」メニューから「ファイル」->「新規プロファイル」を選択します。
 - b. Samples ディレクトリー内にあるポート・コネクター構成ファイル PortConnector.cfg を選択して、Connector Name および Broker Type を構成してから「OK」をクリックします。
 - c. 作成したプロファイルを選択し、「OK」をクリックします。
 - d. 「Visual テスト・コネクター」メニューから、「ファイル」->「接続」を選択してシミュレートを開始します。
7. **親ビジネス・オブジェクトおよび子メタオブジェクトのインスタンスを作成** Visual Test Connector を使用して、以下の手順を実行します。
 - a. ビジネス・オブジェクト Sample_JMS_Contact の新規インスタンスを作成し、必要に応じてデフォルト値を変更します。
 - b. DynMO 属性を右マウス・ボタン・クリックして、そのインスタンス Sample_JMS_DynMO を作成します。
8. **新しいターゲット・キューの設定**
 - a. DynMO 属性の横にある + 符号をクリックして、この属性を展開します。
 - b. outputQueue という名前の属性に、ターゲット・キューの名前 (REROUTE.IN) を入力します。
9. **ビジネス・オブジェクトの送信** 「ビジネス・オブジェクトを送信」をクリックします。
10. **メッセージ送達の確認** キュー REROUTE.IN を開き、新規連絡メッセージが JMS アダプターから届いているか確認します。新しいメッセージが JMS アダプターから REROUTE.IN というキューに届いていれば、転送が成功したことを示しています。

付録 D. トピック・ベースおよびキュー・ベースのメッセージングの構成

- 『キュー・ベース・メッセージングの構成』
- 94 ページの『トピック・ベース・メッセージングの構成』

この付録では、共通の JMS プロバイダーとして WebSphere MQ を使用して、JMS 用のアダプターを構成する方法について説明します。詳細については、「*WebSphere MQ Using Java*」ガイドを参照してください。

注: WebSphere MQ を JMS プロバイダーとして使用する場合、統合用として、WebSphere MQ に WebSphere Business Integration Adapter を使用することを強くお勧めします。以下のステップは、共通の JMS プロバイダーを使用して JMS アダプターを構成する方法を示す参照用としてのみ記載されています。

表記規則のガイドについては、この文書のまえがきを参照してください。

キュー・ベース・メッセージングの構成

1. WebSphere MQ および WebSphere MQ クライアント・ライブラリー (JMS サポートを含む) をインストールする。
2. fscontext.jar および providerutil.jar を含め、すべての MQ クライアント・ライブラリーが使用しているシステム・クラスパスにあることを確認する。あるいは、jmsAdmin.bat ファイルを変更し、`-Djava.ext.dirs="<MQ のホーム・ディレクトリー>/Java/lib` を Java コマンド行スクリプトに追加して、ツールがすべてのクライアント・ライブラリー・ファイルを使用できるようにする。ツールによって報告される `ClassDefNotFoundExceptions` は、ライブラリーが欠落したことが原因であることに注意します。クラスパスを再チェックしてください。
3. `<MQ のホーム・ディレクトリー>Java/bin/jmsAdmin.config` を開き、次のプロパティーを設定する。
 - `INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.ReffSContextFactory`
 - `PROVIDER_URL=file://c:/temp`
 - `SECURITY_AUTHENTICATION=none`
4. 以下を含む `MyJNDI.txt` という名前のファイルを作成する。 `DEFINE QCF(MyQCF) HOST(<ホスト名>) +PORT(<1414 などの MQ リスナー・ポート名>) + CHANNEL(<CHANNEL1 などの MQ サーバー接続チャンネル名>) + QMGR(<MQ キュー・マネージャー名>) + TRAN(client) END`
5. `<MQ のホーム・ディレクトリー>/java/bin/jmsAdmin.bat < MyJNDI.txt` を実行して、オブジェクトを JNDI 名にバインドする。
6. 次の JMS コネクター固有プロパティーを、以下に示すように構成する。

```
CTX_InitialContextFactory = com.sun.jndi.fscontext.RefFSContextFactory
CTX_ProviderURL = file://c:/temp
ConnectionFactoryName = MyQCF
```

トピック・ベース・メッセージングの構成

1. WebSphere MQ および WebSphere MQ クライアント・ライブラリー (JMS サポートを含む) をインストールする。
2. fscontext.jar および providerutil.jar を含め、すべての MQ クライアント・ライブラリーが使用しているシステム・クラスパスにあることを確認する。あるいは、jmsAdmin.bat ファイルを変更し、-Djava.ext.dirs="<MQ のホーム・ディレクトリー>/Java/lib" を Java コマンド行スクリプトに追加して、ツールがすべてのクライアント・ライブラリー・ファイルを使用できるようにする。ツールによって報告される ClassDefNotFounds は、ライブラリーが欠落したことが原因であることに注意します。クラスパスを再チェックしてください。
3. 適切な WebSphere MQ MA0C SupportPac を IBM からダウンロードし、それをインストールして、トピック・ベース (パブリッシュ/サブスクライブ) のメッセージング・サポートを MQ で使用可能にする。例えば、ma0c_ntmq52 を検索すると、Windows では MQ 5.2 用のトピック・ベース・メッセージング・パッチが見つかります。
4. ディレクトリーを <MQ のホーム・ディレクトリー>/Java/bin に変更し、runmqsc <MQJMS_PSQ.mqsc を実行する。
5. IVTSetup.bat を実行する。プロセスは、エラーを報告せずに、「Done!」と表示されるはずだ。
6. <MQ のホーム・ディレクトリー>Java/bin/jmsAdmin.config を開き、次のプロパティーを設定する。
 - INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
 - PROVIDER_URL=file://c:/temp
 - SECURITY_AUTHENTICATION=none
7. 以下を含む MyJNDI.txt という名前のファイルを作成する。

```
DEFINE QCF(MyQCF) HOST(<ホスト名>) +PORT(<1414 などの MQ リスナー・ポート名>) +
CHANNEL(<CHANNEL1 などの MQ サーバー接続チャンネル名>) +
QMGR(<MQ キュー・マネージャー名>) +
TRAN(client)
END
```
8. <MQ のホーム・ディレクトリー>/java/bin/jmsAdmin.bat < MyJNDI.txt を実行して、オブジェクトを JNDI 名にバインドする。
9. 次の JMS コネクター固有プロパティーを、以下に示すように構成する。

```
CTX_InitialContextFactory = com.sun.jndi.fscontext.RefFSContextFactory
CTX_ProviderURL = file://c:/temp
ConnectionFactoryName = MyQCF
```

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を提供し、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せんが、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認できません。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をお願いします。

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ます、単に目標を示しているものです。

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
Domino
Lotus
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または登録商標です。

WebSphere Business Integration Adapter Framework V2.4.0





Printed in Japan