IBM Business Process Manager
バージョン 7.5.0

IBM Business Process Manager の概要

IBM

PDF ブックおよびインフォメーション・センター

PDF ブックは、印刷およびオフラインでの参照用に提供されています。最新情報は、オンラインのインフォメーション・センターを参照してください。

セットとして、PDF ブックには、インフォメーション・センターと同一の内容が含まれます。PDF ブック内のリンクの中には、インフォメーション・センターで使用するように調整されていて、正常に機能しないものがあります。

PDF 資料は、バージョン 7.0 またはバージョン 7.5 など、インフォメーション・センターのメジャー・リリースの後の四半期以内にご利用いただけます。

PDF 資料の更新頻度は、インフォメーション・センターより低いですが、Redbooks® よりも頻繁に更新されます。通常、PDF ブックはブックに十分な変更が累積されたときに更新されます。

目次

第 1 章 IBM Business Process Manager V7.5 入門

IBM® Business Process Manager に用意されたビジネス・プロセス・マネジメント用の機能、およびビジネス・プロセス・マネジメントのさまざまなフェーズ (Process App の作成とデプロイなど) がどのように関連しているかについて説明します。

プロセス・アプリケーションは、IBM Business Process Manager 内のプロセスおよびそれらのコンポーネント用の基本的なコンテナーです。プロセス設計者は、Authoring Environment で Process App を作成しますが、実行をサポートするために必要となるサービス、タスク、および成果物を組み込むことができます。

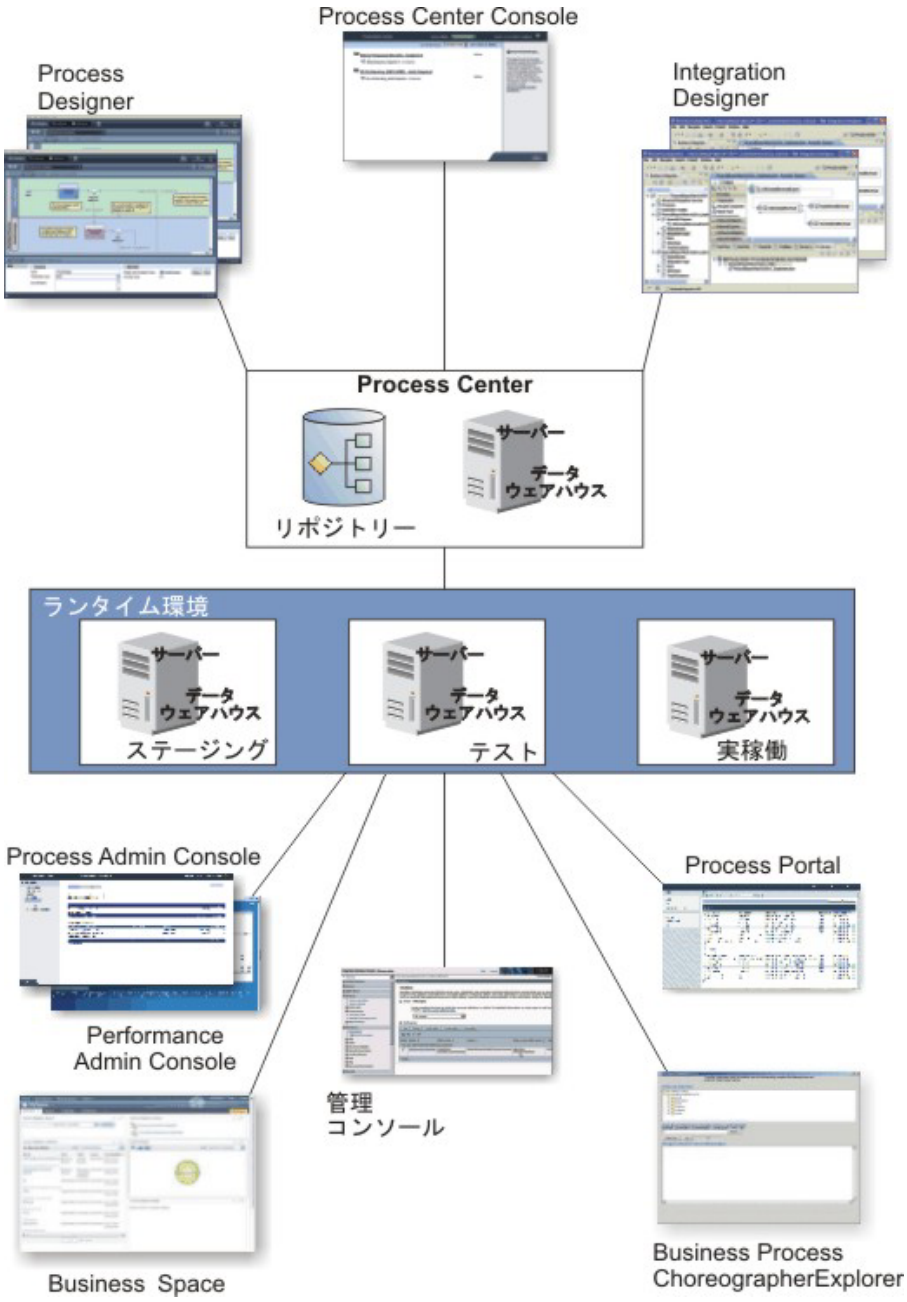
拡張統合サービスは Integration Designer で実装され、Process App に関連付けられます。 Process App は、Process Center から、IBM Business Process Manager のプロセス・ランタイム環境である Process Server にデプロイされます。

同様に、Integration Designer で作成された自動化プロセスは、IBM Process Designer で作成されたヒューマン・アクティビティー・フローを使用できます。

製品概要

IBM Business Process Manager のコンポーネントは、統合された BPM リポジトリー、作成者、管理者、およびユーザー向けのツール、そしてランタイム・プラットフォームが用意されています。製品のさまざまな構成により、ビジネス・プロセス・マネジメントにおける多様なレベルの複雑さや関連作業をサポートします。

以下の図は代表的な IBM Business Process Manager 構成を示しています。



- IBM Process Designer および IBM Integration Designer Authoring Environment から、複数のユーザーが Process Center に接続しています。
- Process Designer および Integration Designer Authoring Environment では、Process Designer および Service Designer がデプロイ可能な Process App および再利用可能な Toolkit を作成します。 Process App には、プロセス・モデルとサービスの実装が、必要なサポート・ファイルとともに含まれています。これらは Process Center リポジトリーに保管され、ここで共有できます。
- Process Center には Process Center Server と Business Performance Data Warehouse が含まれており、IBM Process Designer で作業するユーザーが Process App を実行し、開発作業中のテストおよびプレイバックの目的でパフォーマンス・データを保管することができます。
- 管理者は Process Center Console から、ステージング、テストまたは実動用に準備が整った Process App を、それらの環境の Process Server にインストールします。

- 管理者は Process Center Console から、すべての構成済み環境内で実行中の Process App インスタンスを管理します。
- エンド・ユーザーは IBM Process Portal から、割り当てられたタスクを実行します。構成済みランタイム環境の Process Center Server および Process Server は、割り当てられたタスクを作成する Process App を実行できます。
- 割り当てます。 を使用して、プロセス参加者は、プロセスが開発中かテスト中か、それとも実稼働環境へリリースされたかに応じて、いずれかの構成済みランタイム環境の Process Center Server または Process Server に接続することができます。
- Performance Data Warehouse は、Process Server または Process Center Server から、トラッキングされたデータを一定の間隔で取り出します。ユーザーは、Authoring Environment および IBM Process Portal で、このデータを活用したレポートを作成および表示できます。
- 管理者は、Process Admin Console および Performance Admin Console から、すべてのランタイム・サーバーを管理および保守できます。

IBM Business Process Manager V7.5 の構成

IBM Business Process Manager のさまざまな構成は、企業のビジネス・プロセス・マネジメント・プログラムにおける標準的なエントリー・ポイントまたはステージと関連しています。

表 1. IBM Business Process Manager 構成

構成	フェーズ
Advanced	**変換** ビジネス・プロセス・マネジメント機能の完全なセット • 大ボリューム・プロセスの自動化の拡張サポート • エンタープライズ規模の広範なサービス統合およびオーケストレーションのための、組み込み SOA コンポーネント
Standard	**プログラム** 標準的なビジネス・プロセス・マネジメント・プロジェクト用の構成 • さまざまなビジネスが関与する複数のプロジェクト改善プログラム • 基本システム統合サポート • TTV (Time-To-Value) およびユーザー生産性の向上
Express	**プロジェクト** 最初のビジネス・プロセス・マネジメント・プロジェクト用の構成 • TTV (Time-To-Value) およびユーザー生産性の向上 • 低価格 • インストールと構成の容易性

IBM Business Process Manager V7.5 の構成機能

IBM が提供するビジネス・プロセス・マネジメント用の製品および機能について理解し、対象となる各エンタープライズに適切なものを選択してください。

IBM Business Process Manager は、ヒューマンセントリックな機能とインテグレーションセントリックな機能を 1 つの製品に結合した、単一の BPM プラットフォームです。エンタープライズ内のさまざまなユ

ーザーのさまざまなニーズを満たすため、この製品では複数の構成が利用可能です。複数の製品構成を組み合わせて、コラボレーティブ・オーサリング環境およびネットワーク・デプロイされたランタイム環境を作成できます。

表 2. IBM Business Process Manager の構成機能

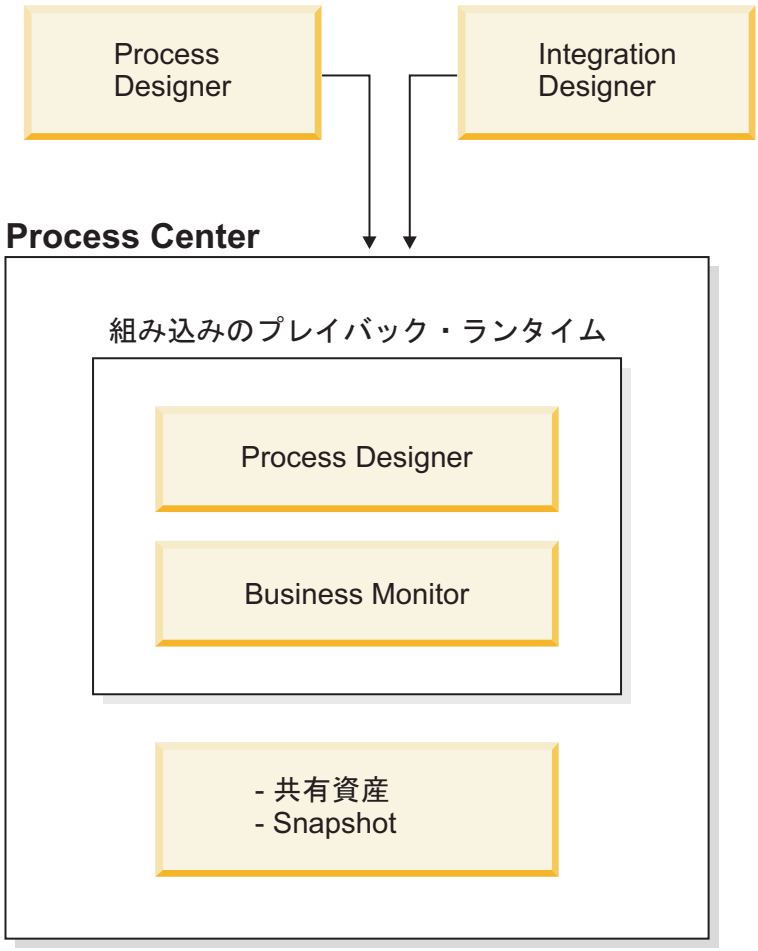
機能	拡張	Stan-dard	Express
WebSphere Lombardi Edition 互換実行	X	X	X
Process Designer (BPMN)	X	X	X
コラボレーション編集/即時プレイバック	X	X	X
対話式の「Process Coach」ユーザー・インターフェース	X	X	X
ILOG ベースのプロセス・ルール	X	X	X
リアルタイム・モニターおよびレポート作成	X	X	X
パフォーマンス分析および最適化プログラム	X	X	X
Performance Data Warehouse	X	X	X
Process Center/共有アセット・リポジトリー	X	X	X
無制限のプロセス作成者およびエンド・ユーザー	X	X	200 ユーザー/3 作成者
高可用性: クラスター化および無制限のコア	X	X	• 4 コア製品 • 2 コア開発 • クラスターなし
WebSphere Process Server 互換実行	X		
Integration Designer (BPEL/SOA)	X		
組み込みエンタープライズ・サービス・バス (ESB)	X		
トランザクション・サポート	X		
統合アダプター	X		
柔軟な Business Space ユーザー・インターフェース	X		

Process Center リポジトリー

Process Center には、IBM Business Process Manager Authoring Environment (Process Designer および Integration Designer) で作成されたすべてのプロセス、サービス、およびその他のアセット用のリポジトリーが含まれます。

Process Center は、Process Designer と Integration Designer が資産を共有するためのランタイムであり、実質的には、それらが極めて緊密に連携してビジネス・プロセスを開発するための場所を提供します。それらのビジネス・プロセスでは、Business Monitor Development Toolkit を使用して作成されたモニタリング・ポイントを使用することができます。そのようにすると、実際の作業状態におけるビジネス・プロセスの有効性をランタイムで検証することができるようになります。Business Monitor は、ゲージおよびスコアカードを含むダッシュボード・ビューを提供します。ビジネス・プロセスの状況を逐一知らせるアラートおよび通知を追加できます。実行されているビジネス・プロセスのリソース割り振りに関するボトルネック、非効率性、およびエラーを発見して訂正できるので、ビジネス・プロセスのパフォーマンスが改善されます。

以下の図は、連携して複雑なビジネス・プロセスを構築できるいくつかの関連コンポーネントを示しています。



Process Center Console には、リポジトリーの保守に必要なツールが用意されています。

- Process Center コンソールでは、プロセス・アプリケーションおよびツールキットを作成し、他のユーザーにそれらのプロセス・アプリケーションおよびツールキットへのアクセス権を付与できます。
- Authoring Environment では、Process App 内に、プロセス・モデル、サービス、および他のアセットを作成することができます。
- Process Center には Process Center Server および Performance Data Warehouse が含まれており、Authoring Environment で作業中のユーザーが、テストや再生用にプロセスを実行したり、パフォーマンス・データを保管したりできます。
- Process Center コンソールから、管理者はそれらの環境でテストや実動が可能なプロセス・アプリケーションをプロセス・サーバーにインストールします。
- Process Center コンソールから、管理者は構成済みの環境で、実行中のプロセス・アプリケーションのインスタンスを管理します。

Process Center Console は、Process App や Toolkit といった高位のコンテナーを作成および維持するのに便利な場所を提供します。現在「デザイナー (Designer)」ビューでの作業を行っていない管理者は、Process Center コンソールを使用して、BPM アナリストおよび開発者がプロセスや基となる実装環境を構築できるフレームワークを提供できます。管理者の基本タスクにはその他に、ユーザーおよびグループに適切な権限を設定して、Process Center リポジトリーへのアクセスを管理することがあります。

適切な権限を持ったユーザーは、Process Designer および Integration Designer で、いくつかの管理タスクを直接実行することができます。例えば、Process App への書き込みアクセス権を持つ開発者が、特定のマイルストーンですべてのプロジェクト・アセットの状態を取り込みたい場合、Designer ビューでの作業中に Snapshot を作成することができます。

Process Server およびランタイム環境

Process Server には、広範なビジネス・プロセス、サービス・オーケストレーション、および統合機能をサポートできる、1 つの BPM ランタイム環境が用意されています。

Authoring Environment では、Process Center 内の統合 Process Server により、プロセスのビルド直後にそれらのプロセスを実行できます。準備が整ったら、ランタイム環境の Process Server に、その同じプロセスをインストールして実行できます。Business Performance Data Warehouse コンポーネントは、Process Server 上で実行されているプロセスのプロセス・データを収集および集約します。このデータを使用することで、ビジネス・プロセスを向上させることができます。

Process Admin Console を使用すると、Process Center の一部である Process Server と同様に、ご使用のランタイム環境 (ステージング、テスト、実稼働などの環境) で Process Server を管理できます。

Authoring Environment

IBM Business Process Manager Advanced には 2 つの Authoring Environment が用意されています。ヒューマン・タスクを伴うビジネス・プロセスを効率的にモデル化するには、IBM Process Designer を使用します。内蔵タイプのサービスを作成する場合、または他の既存サービス (Web サービス、エンタープライズ・リソース・アプリケーション、CICS および IMS で実行されるアプリケーションなど) を呼び出すサービスを作成する場合は、IBM Integration Designer を使用します。

Process Designer は、製品のすべてのエディションで使用できます。さらに IBM Business Process Manager Advanced は、Integration Designer を、関連するエディターおよびアダプターとともに提供します。

Process Designer

プロセスは、IBM Business Process Manager におけるロジックの主要な単位です。プロセスは、プロセス定義のすべてのコンポーネントのコンテナーです。プロセス定義のコンポーネントには、サービス、アクティビティー、ゲートウェイ、タイマー・イベント、メッセージ・イベント、例外イベント、シーケンス・ライン、ルール、変数などがあります。プロセスをモデル化する場合は、再利用可能なビジネス・プロセス定義 (BPD) を作成します。IBM Process Designer を使用して、ヒューマン・タスクを格納できるプロセス・モデルを作成します。

Process Designer は、ビジネス・プロセスの開発に役立ちます。使いやすいグラフィック指向のツールで、ビジネス・プロセスを構成するアクションのシーケンスを作成したり、時間とともに状況が変化した場合に応じて、そのプロセスに変更を加えたりすることができます。ビジネス・プロセスのデータを提供する大規模なバックエンド・システムまたはサービスに (例えば、顧客に関する情報収集などを目的として) 1 つ以上のアクティビティーからアクセスする必要がある場合は、Integration Designer の使用が必要になることがあります。シンプルなインターフェースを使用して、Process Designer 内のアクティビティーから、Integration Designer で作成されたサービスを呼び出すことができます。そのようなサービスでは、メディエーション・フローを使用して、データおよびアダプターを変換、経路指定、ならびに拡張して、一般的な方法で多数のバックエンド・システムにアクセスすることができます。つまり、Process Designer はビジネス・プロセスに重点的に対応し、Integration Designer はビジネス・プロセスを補完するための自動化サービスに重点的に対応するということです。

すべての Process Designer プロジェクトは、Process App に格納されます。これらの Process App および関連付けられた成果物を、Process Center リポジトリーに保管します。Process App は、Toolkit に入れられた資産を共有できます。

IBM Business Process Manager には、いくつかのユーザー・インターフェースが用意されています。これらを使用することで、ビジネス・プロセスのモデル化、実装、シミュレーション、および検査を行うことができます。 Process App、Toolkit、トラック、および Snapshot は、Process Center Console で作成および管理します。プロセス・モデル、レポート、および単純なサービスは Process Designer で作成できます。Inspector では、プロセスを実行およびデバッグできます。また、 Optimizer でシミュレーションを実行できます。

Process Designer で作成された Process App は、いつでも Process Center Server で実行するか、Snapshot に保存して Process Server にデプロイすることができます。Integration Designer で作成され、Process App に関連付けられたサービスでも同じことが該当します。

Integration Designer

Integration Designer には、エディターおよび補助機能が用意されており、開発者は複雑な自動化プロセスおよびサービスを作成することができます。 IBM Business Process Manager Advanced でコンポーネントとして使用することも、他の用途でスタンドアロン・ツール・セットとして使用することもできます。

IBM Integration Designer は、統合アプリケーションを作成するための完全な統合開発環境として設計されています。統合アプリケーションは、単純なものではありません。部門間やエンタープライズ間のビジネス・プロセスにかかわる Enterprise Information Systems (EIS) のアプリケーションを呼び出せます。また、さまざまな言語で作成され、さまざまなオペレーティング・システムで稼働するアプリケーションを、ローカルとリモートで呼び出すことが可能です。コンポーネントは、ビジュアル・エディターを使用して作成され、他の統合アプリケーション (一連のコンポーネントで作成されているアプリケーション) にアセンブルされます。ビジュアル・エディターは、コンポーネントとその実装の間の抽象化層を提示します。開発者は、このツールを使用することで、基盤となっている各コンポーネントの実装内容を詳細に把握しなくても、統合アプリケーションを作成することができます。

Integration Designer ツールは、サービス指向アーキテクチャーに基づいています。コンポーネントはサービスであり、また、多数のコンポーネントが含まれる統合アプリケーションも 1 つのサービスです。作成されるサービスは、主要な業界標準に準拠します。 BPEL プロセス (これもコンポーネントになります) も同様に、業界標準の Business Process Execution Language (BPEL) に準拠した使いやすいビジュアル・ツールで作成されます。

Integration Designer パラダイムにおいて、コンポーネントはモジュールでアセンブルされます。インポートとエクスポートを使用して、モジュール間でデータを共有します。ライブラリーに格納された成果物は、モジュール間で共有できます。

モジュールおよびライブラリーは、Process App に関連付けて Process Center で使用することができます。また、Process Designer で作成されたプロセスによりサービスとして使用することができます。この場合、モジュールおよびライブラリーは、Process App を使用してデプロイすることもできます。

または、モジュールおよびライブラリーは、テスト環境または Process Server に直接デプロイすることもできます。メディエーション・モジュールを使用して、メディエーション・フローを作成できます。これは、WebSphere Enterprise Service Bus または Process Server にデプロイできます。

IBM Integration Designer には、WebSphere DataPower アプライアンスにデプロイできるデータ型と XML マップを作成する機能もあります。また、WebSphere DataPower との間でファイル転送を行うこともできます。

バージョン 7.5 Business Process Management 製品の新機能

各製品またはコンポーネントの個々の新機能ページに記載された IBM Business Process Management の新機能をお読みください。

新機能および拡張機能は、以下のようなハイレベルのビジネス・プロセス・マネジメントのニーズに焦点を合わせています。

- ビジネス・プロセスの可視化の向上
- 再利用の最大化
- 変更の管理
- プロジェクトのスコープとスケールの拡張可能化

以下のページでは、IBM Business Process Manager V7.5、およびその他のビジネス・プロセス・マネジメント製品およびコンポーネントの新機能について説明します。

- IBM Business Process Manager V7.5 の新機能
- IBM Integration Designer バージョン 7.5 の新機能
- Business Space powered by WebSphere バージョン 7.5 の新機能
- IBM Business Monitor バージョン 7.5 の新機能

IBM Business Process Manager V7.5 の新機能

IBM Business Process Manager は、WebSphere Process Server、WebSphere Lombardi Edition、および IBM Integration Designer (旧 WebSphere Integration Developer) の主要機能を取り込み、1 つの統合されたユーザー環境を作成します。また、この 3 つの製品の最新機能を組み込み、前の製品バージョンの機能を拡張しています。

IBM Business Process Manager は、ビジネス・プロセスを可視化し、それらのプロセスを管理するための、包括的かつ取り込み可能なビジネス・プロセス・マネジメント・プラットフォームです。ここには、プロセスの設計、実行、モニター、および最適化のためのツールおよびランタイムが含まれています。また、プロセス所有者とビジネス・ユーザーが、ビジネス・プロセスの改善に直接取り組めるように支援する設計になっています。

以下のリストにある新機能および拡張機能の中には、このいずれかの製品 (WebSphere Process Server、WebSphere Lombardi Edition、または WebSphere Integration Developer) の旧バージョンに存在していたものがあります。しかしそれらの機能は、以前に 3 つの製品のすべてはなく、1 つまたは 2 つだけを使用していたユーザーにとっては、新機能となります。

統合された製品の旧バージョンに存在していた機能の中には、IBM Business Process Manager で新機能に置き換えられたものもあります。非推奨となった機能のリストについては、使用すべきでないフィーチャーを参照してください。

IBM Business Process Manager V7.5 の新機能および拡張機能

IBM Business Process Manager V7.5 には、以下の新機能または拡張機能があります。

- ヒューマンセントリックな標準ベースの Business Process Modeling Notation (BPMN) プロセスを作成 (Process Designer を使用) し、それらのプロセスをシステムセントリックな Business Process Execution Language (BPEL) プロセスと統合 (Integration Designer を使用) するための、統合された Authoring Environment。
- BPMN プロセスおよび BPEL プロセスの両方のための共有リポジトリーと共通実行環境。統合された Authoring Environment、共有リポジトリー、および共通実行環境が連携して、以下の機能拡張を提供します。
  - Process App、サービス、ユーザー・インターフェース、およびルールのグラフィカルな実装およびテスト。
  - BPMN を使用した標準ベースのプロセス設計。
  - サービス、データ変換、BPEL オーケストレーション、およびアプリケーションとバックエンド・システムとの統合を構成するための IBM Integration Designer ツール。
  - アクセス可能な方法でビジネス・ロジックを表現し、WebSphere ILOG JRules と同じルール・オーサリング動作を可能にする、ビジネス・ルール・オーサリング。ルールの内容は簡単に WebSphere ILOG JRules にエクスポートできるため、ビジネス上の決定を行うための開始点となります。
  - IBM ILOG JViews Chart チャート・エンジン。IBM Business Process Manager V7.5 に追加された機能です。既存のカスタマイズ済みチャートについては、CDL フォーマットからカスケーディング・スタイル・シート (CSS) フォーマットに変換するためのツールが提供されます。
  - プロパティー・シート。これにより、実装の詳細をコーディングするのではなく、構成することが可能になるため、設計に参加する技術ユーザーの数を削減できます。
  - 完全なライフサイクル統合。デプロイメントによって、モデル設計からソリューションの同期を維持します。
  - 対話式プレイバック。これにより、プロセスの要件をいつでも検証できます。
  - すべてのプロセス資産を含む共有ライブラリー。ドラッグ・アンド・ドロップによる再利用とコラボレーティブ実装のために利用します。
- 以下の機能による、未完了プロセスのリアルタイム制御。
  - IBM Process Portal 内の受信箱が、すべての未処理タスクの統合ビューを提供します。
  - プロセス状況のグラフィカル・ビューにより、プロセス参加者がプロセスの残りのステップを把握しやすくなります。
  - 明示的なイベント・モデル化により、ワークフローと例外処理が定義されます。
  - 公開済みのプロセス・パラメーターおよびしきい値により、未完了処理の間のリアルタイム制御を行います。
  - 随時アクションがサポートされます。指定されたユーザーは自発的タスクを実行し、ワークフローを変更できます。
  - プロセス実行中、ユーザーは特定のタスクに関して通信とコラボレーションを行えます。
  - リアルタイム・スコアボードにより、処理中の作業を可視化できるとともに、必要に応じて修正アクションを実行できます。
  - Business Process Manager に、Business Space ユーザー・インターフェースのフレームワークが組み込まれています。
  - 統合タスク・ビューにより、タスクの実行、作業項目の管理、パフォーマンスのトラッキング、イベントへの応答をすべてリアルタイムで実施できます。
  - オプションで、Microsoft Office および Microsoft SharePoint から、ユーザーが直接プロセス・タスクを実行できます。Microsoft との統合機能が、オプションのアドオン・コンポーネントとして提供されます。

- 拡張モニター機能。ユーザーおよびシステムの対話を両方とも表示できる、プロセス可視化を実現します。 IBM Business Process Manager は、以下の機能により、洞察に富んだ、動的なプロセス可視化をサポートします。
  - パフォーマンス測定が、重要業績評価指標 (KPI) およびサービス・レベル・アグリーメント (SLA) の観点で表現されます。
  - Business Performance Data Warehouse は、パフォーマンス・イベントおよびビジネス・データを自動的に収集したうえで、モデル化されたプロセス・メトリックと相関させて、プロセス可視化が常に最新に保たれるようにします。
  - Process Optimizer を使用して、パフォーマンスのボトルネックやその他の問題を直接プロセス・モデル・ダイアグラムで可視化できます。
- WebSphere Process Server の機能が IBM Process Server に取り込まれるようになり、トランザクションの保全性を確保し、最初のプロジェクトからエンタープライズ全体のソリューションに至るまでの拡張スケーラビリティーを実現します。
  - 組み込みの WebSphere Application Server により、優れたスケーラビリティーとアベイラビリティーが実現します。
  - 単一の BPM Runtime Server (Process Server) が、広範なビジネス・プロセス、サービス・オーケストレーション、および統合をサポートします。
  - 組み込みのサービス指向アーキテクチャー (SOA) コンポーネントに、アプリケーションとサービスを統合するための柔軟性の高いエンタープライズ・サービス・バス (ESB) 接続インフラストラクチャーが含まれています。
  - サポートされる標準には、BPMN と BPEL の実行に関する標準、およびデータおよびサービスに関する多くの標準があります。
  - 迅速にデプロイできる、ベスト・プラクティスに基づくエンタープライズ対応接続を利用して、統合アダプターはアプリケーションと情報資産を ESB に接続し、ビジネス・インテグレーション・プロジェクトを推進します。資産 (パッケージ・アプリケーション、カスタム・アプリケーション、およびレガシー・アプリケーションや、テクノロジー・プロトコル、データベースなど) をサービス対応にするための、包括的な機能セットが含まれています。
  - 豊富な修復機能とリカバリー機能が提供されます。自動再試行、手動修復、補正、ストア・アンド・フォワードなどがあります。
- 以下の機能により、ハイレベルなプログラム管理容易性とガバナンスがサポートされます。
  - Process Center は、BPM プログラムの一部として作成されたすべてのプロセス成果物、アプリケーション、およびサービスを編成および管理するための、スケーラブルな中央リポジトリーおよびコントロール・センターとして機能します。
  - Toolkit は、複数の Process App 間で再利用するためのプロセス成果物を管理します。
  - Snapshot は、Process App または Toolkit 内にあるすべての成果物 (ビジネス・プロセス・ダイアグラム、ルール、データ、フォーム、サービス、シミュレーション・シナリオなど) の特定時点における状態を、マウスを 1 回クリックするだけで取り込みます。
  - バージョン管理により、Process App または Toolkit のすべての履歴 Snapshot を復元します。
  - Process Server レジストリーは、さまざまな Runtime Server 環境に対して複数のプロセスをインストールし、それらのプロセスのデプロイ済みバージョンをトラッキングする、集中型ツールとして機能します。
  - ライフサイクル・ガバナンスでは、集中制御によって、実動ランタイムへのプロセスとサービスのデプロイメントを管理します。

- IBM Business Process Manager には、各企業がどの段階からビジネス・プロセス・マネジメントに参入するかに応じて、3 つの製品構成 (Advanced、Standard、および Express) が用意されています。詳しくは、このインフォメーション・センターの『IBM Business Process Manager 7.5 の構成 (IBM Business Process Manager 7.5 configurations)』を参照してください。
- IBM Business Process Manager は、拡張オーサリング・ツール、および一般的なエンド・ユーザー生産性向上ツールと統合するためのオプションのアドオンを提供します。
  - IBM Process Designer Standard ベースのプロセス設計ツール。これは、IBM Business Process Manager の 3 つすべての構成 (Advanced、Standard、および Express) 用の基本 Authoring Environment に含まれています。このプロセス設計ツールにより、迅速な構成と継続的なプロセス変更が可能になります。
  - サービス、データ変換、BPEL プロセス、およびアプリケーションとバックエンド・システムとの統合を視覚的に構成するための、IBM Integration Designer Advanced の構成 Authoring Environment。この Authoring Environment をプロセス設計ツールとコラボレーションさせて使用することで、堅牢かつスケーラブルなプロセス・ソリューションが作成されます。ここには、資産 (パッケージ・アプリケーション、カスタム・アプリケーション、およびレガシー・アプリケーションや、テクノロジー・プロトコル、データベースなど) をサービス対応にするための、包括的なアダプター・セットが含まれています。この Authoring Environment は、最新バージョンの WebSphere Integration Developer と完全な互換性があります。
  - IBM Business Process Manager for Microsoft Office アドオンは、3 つすべての構成 (Advanced、Standard、および Express) へのオプションのアドオンです。Microsoft Office ユーザーは、IBM Business Process Manager を使用して、自身の Office デスクトップから直接 IBM BPM タスクを表示し、実行できます。この Office アドオンは、エンド・ユーザーの Office デスクトップ・クライアントに IBM プラグインをインストールします。
  - IBM Business Process Manager for Microsoft SharePoint アドオンは、3 つすべての構成 (Advanced、Standard、および Express) へのオプションのアドオンです。Microsoft SharePoint ユーザーは、IBM Business Process Manager を使用して、SharePoint ポータル・ページにドロップされている Web パーツから直接 IBM BPM タスクを表示し、実行できます。この SharePoint アドオンは任意の SharePoint サーバーにインストールされ、このサーバーが、IBM BPM Web パーツを介するプロセスとの対話をホストするために使用されます。
- IBM Business Process Manager V7.5 は、以下のマイグレーションおよびデータ・インポートをサポートします。
  - WebSphere Process Server V6.1、V6.1.2、V6.2、および V7.0 からの Process App のマイグレーション。これには、設計時プロセス定義資産および未完了プロセス・インスタンスが含まれます。
  - Lombardi Teamworks 6.1、Teamworks 6.2、Teamworks 7.0、および WebSphere Lombardi Edition V7.1 と V7.2 からの Process App のマイグレーション。これには、設計時プロセス定義資産および未完了プロセス・インスタンスが含まれます。
  - WebSphere Business Modeler V7.0、WebSphere Business Compass V7.0、およびその他の一般的なモデル化ツールからの、BPMN 2.0 フォーマットの設計時プロセス・モデルのインポート。
  - Business Space 用の IBM BPM Advanced ウィジェットを IBM WebSphere Portal で使用できます。

Business Space powered by WebSphere の新機能

Business Process Manager およびその他の IBM 製品に共通するコンポーネントである Business Space が以下のように拡張されました。

- Business Space powered by WebSphere バージョン 7.5 の新機能
- WebSphere Portal バージョン 7.0 用の Business Space バージョン 7.5 の新機能

IBM Business Process Manager におけるアクセシビリティー

アクセシビリティー・フィーチャーは、運動障害または視覚障害など身体に障害を持つユーザーが情報技術製品を快適に使用できるようにサポートします。

IBM は、年齢や能力を問わず、すべての人が便利に使用できる製品の提供に努めています。スクリーン・リーダー・ソフトウェア (読み上げソフトウェア) およびデジタル音声シンセサイザーなどの支援テクノロジーを使用して、画面に表示されている内容を聴きます。本製品に付属のテクノロジーを使用する方法について詳しくは、支援テクノロジーの製品資料を参照してください。

マウスの代わりにキーボードを使用して、機能を操作することができます。

色、コントラスト、フォント・サイズなどの表示属性をカスタマイズできます。

グラフィカル・ビュー内の情報を拡大表示して、詳細を確認することができます。

米国リハビリテーション法第 508 条の Voluntary Product Accessibility Template (VPAT) を入手する場合は、IBM Web サイト (http://www.ibm.com/able/product_accessibility/index.html) で請求できます。

インフォメーション・センターのドキュメンテーションには、ほかにもアクセシビリティーを支援する以下のフィーチャーが盛り込まれています。

- ユーザーがスクリーン・リーダー・ソフトウェア・テクノロジーを利用できるように HTML 形式でドキュメンテーションが提供されています。
- ドキュメンテーション内の画像には、視覚障害のあるユーザーがその内容を理解できるように、代替テキストが付加されています。

IBM Business Process Manager で使用可能な各国語

IBM Business Process Manager では、以下の言語がサポートされます。ドキュメンテーションは、完全には翻訳されていない場合があります。

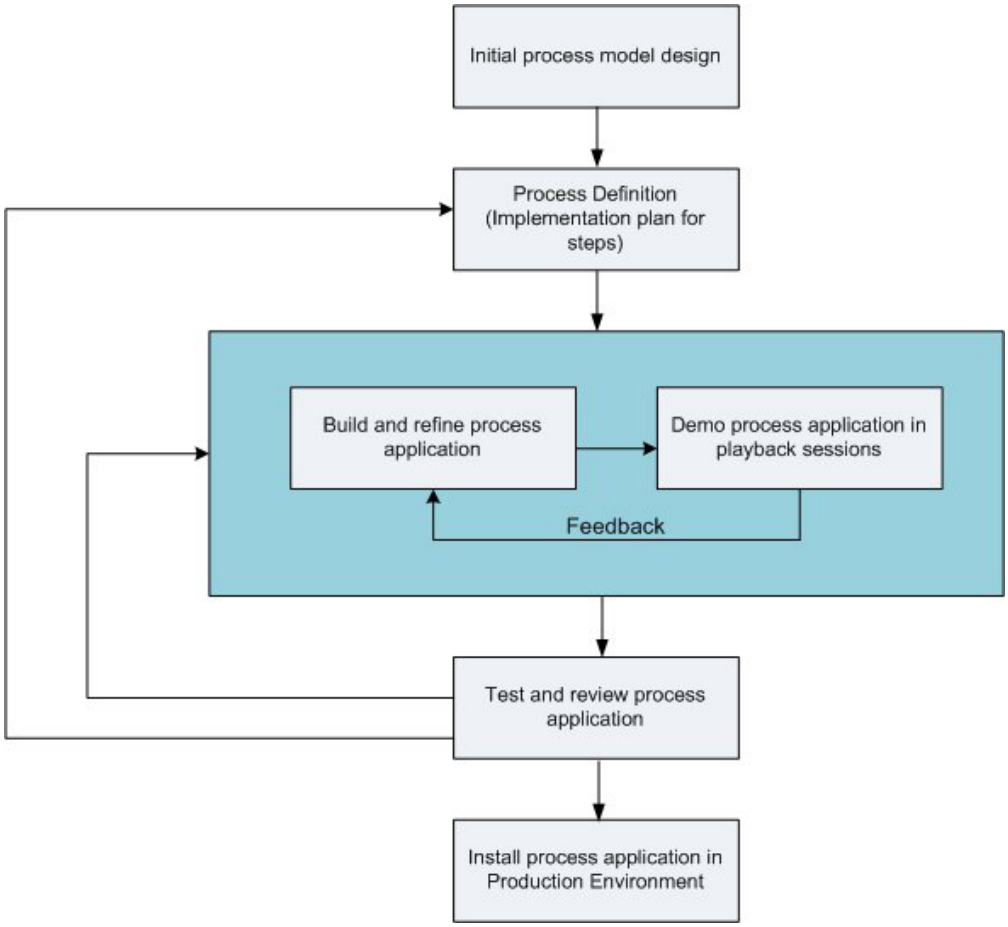
- 中国語 (簡体字)
- 中国語 (繁体字)
- チェコ語
- 英語 (米国)
- フランス語
- ドイツ語
- ハンガリー語
- イタリア語
- 日本語
- 韓国語
- ポーランド語
- ブラジル・ポルトガル語
- ルーマニア語
- ロシア語
- スペイン語

ビジネス・プロセス・マネジメントの概要

Process Designer でプロセスを開発する場合には、テスト環境および実稼働環境のサーバーへの Process App の最終的なインストールを計画する必要があります。

以下の図に、一般的なプロセス開発作業のライフサイクルを示します。ここには、実稼働環境に Process App をインストールできるように、インストール・サービスを構築および改善するためのステップが含まれています。

このダイアグラムで示されているように、開発環境で排他的に作業を行うことができます。ただし、Process Server をテスト環境と実動環境の両方で構成する必要があります。



プロセス・モデル化の概要

プロセスは、IBM Business Process Manager におけるロジックの主要な単位です。プロセスは、プロセス定義のすべてのコンポーネントのコンテナーです。プロセス定義のコンポーネントには、サービス、アクティビティー、ゲートウェイ、タイマー・イベント、メッセージ・イベント、例外イベント、シーケンス・ライン、ルール、変数などがあります。プロセスをモデル化する場合は、再利用可能なビジネス・プロセス定義 (BPD) を作成します。

プロセス・コンポーネントを使用して、エンド・ユーザーのためのプロセス・ワークフローを定義して、プロセス内のロジックを作成し、他のアプリケーションおよびデータ・ソースと統合することができます。実行時にプロセス内でどのようなことが発生するか理解するためには、設計時にプロセスを構成する要素について理解することが重要です。

IBM BPM によるプロセスの構築

通常、IBM BPM によるプロセス開発には、さまざまな組織のさまざまな人々が関与します。最優先すべき考慮事項は、プロジェクトについて規定されている目標に合う最善のソリューションが構築されるようにすることです。確実に成果を挙げられるようにするため、チームのメンバーが協力して、プロセス要件を取り込み、モデルとその実装環境を反復的に開発する必要があります。

Process Designer における項目の再利用

Process Designer では、プロセス開発者が既存の項目を Process App の内部または複数の Process App の間で再使用できます。例えば、自分および他の開発者が必要とする Coach ならびに他の共有項目を含むいくつかのサービスが既に存在することがわかっている場合には、それらの項目を Toolkit に含めることで、それらにアクセスして再利用できるようになります。次に、Process App から、共有項目が存在する Toolkit への依存関係を追加します。このようにすることで、アクティビティーの実装を選択するときに、既存のサービスのいずれかを選択できるようになります。Toolkit 内の項目は、別の Process App に対して作業を行っている他の開発者も使用することができます。

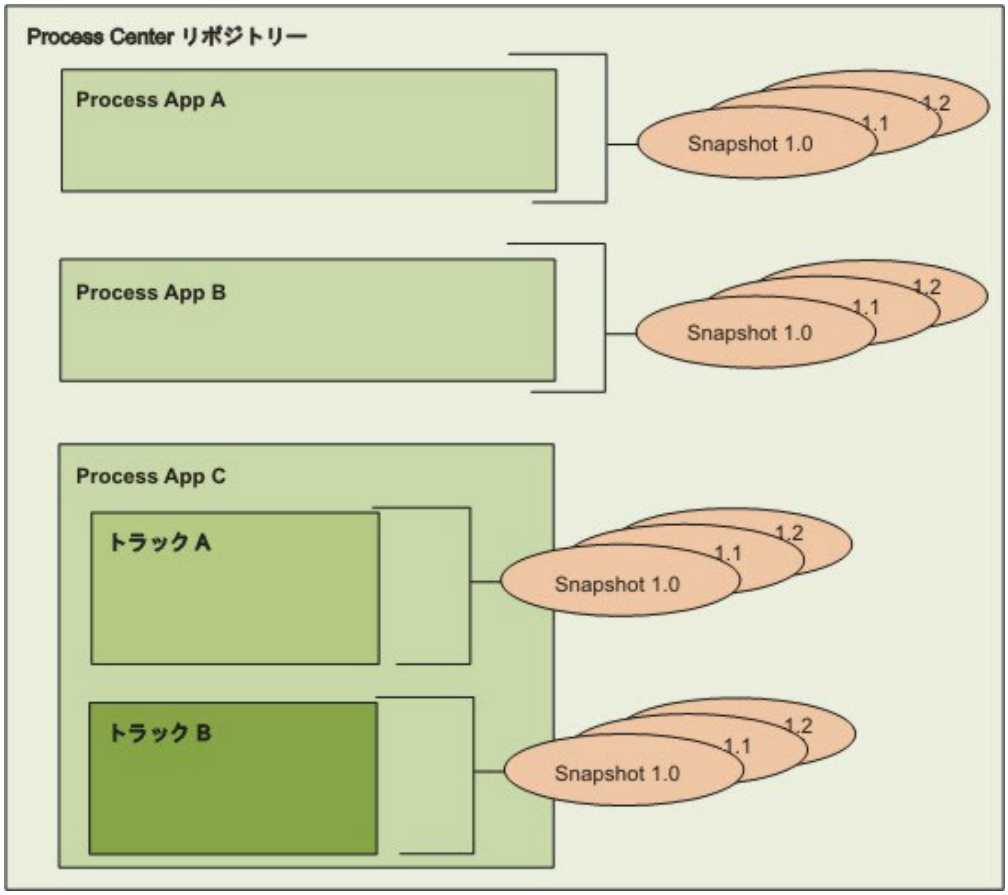
IBM Process Designer における Designer の使用

Designer インターフェースは、IBM BPM でプロセスをモデル化するために必要なツールを提供します。

Process Center によるプロセス開発

IBM Process Center は、Process Designer で作成されるすべてのプロジェクト資産のためのセントラル・リポジトリーとしての役割を果たします。複数の Process Designer クライアントを Process Center に接続すると、ユーザーはプロセスやサービスなどの項目を共有でき、他のユーザーによって変更が加えられるとそれを確認することもできます。Process Center は、IBM Integration Designer で作成される資産のリポジトリーとしても使用できます。

Process Designer でのプロセス開発時には、プロジェクトの管理に利用できるように設計されている階層を Process Center リポジトリーで使用することができます。以下の図に、リポジトリー階層の概念的な概要を示します。



上記の図からわかるように、Process Center リポジトリーには以下の成果物が含まれています。

Process App	BPM アナリストおよび開発者が IBM Process Designer の Designer で作成する、プロセス・モデルならびにサポートする実装環境のためのコンテナーです。
トラック	チーム作業またはアプリケーションのバージョンに基づく、Process App 内のオプションのサブディビジョンです。トラックを有効にすると、開発を並行して行えるようになります。管理者は、それぞれの Process App に追加のトラックが必要かどうか判断して、必要な場合は、トラックを有効にすることができます。
Snapshot	特定の時点における Process App 内またはトラック内の項目の状態を記録します。通常、Snapshot は、マイルストーンを表すか、プレイバックまたはインストールに使用されます。Snapshot 同士を比較したり、以前の Snapshot に戻したりすることができます。管理者は、Process App のトラックを有効にすると、Snapshot を新規トラックの基準として使用します。

Process App: 概説

Process App は、プロセス・モデルとそれらをサポートする実装用のコンテナーで、リポジトリーに保管されます。成果物がオーサリングまたは作成されると、これらは Process App にアセンブルされ、Snapshot が作成されます。 Process App の Snapshot のテスト、インストール、および管理を行うことができます。

Process App は以下の成果物の一部、またはすべてを含んでいます。

- 1 つ以上のプロセス・モデル (ビジネス・プロセス定義 (BPD) とも呼ばれる)
- アクティビティーの実装、または他システムとの統合に必要なサービス

- サービス・コンポーネント・アーキテクチャー (SCA) モジュール
- Toolkit
- IBM Integration Designer ライブラリー
- 1 つ以上のワークスペース
- ビジネス・パフォーマンスをモニターするための IBM Business Monitor モデル
- プロセス実行に必要な他の要素

Process App とビジネス・レベル・アプリケーション

各 Process App は、Process App およびそのアセット用のコンテナーとして機能する、ビジネス・レベル・アプリケーション (BLA) を持っています (アセットには、モニター・モデル、SCA モジュール、Toolkit、およびライブラリーなどが含まれます)。さらに、Process App の各 Snapshot も、独自の BLA を持っています。 Snapshot に関する多くの管理タスク (例えば、実動サーバー上での Snapshot の起動、停止など) が BLA のレベルで実行され、Snapshot とそのアセットすべてに関する管理が、より迅速で簡単になります。

Inspector によるプロセスの実行およびデバッグ

IBM Process Designer の Inspector は、反復的なプロセス開発手法の鍵となります。インスペクターを使用すると、個々の開発者は Process Center Server またはリモートのランタイム Process Server 上でプロセスとサービスを実行できます。また、開発チーム全体で Inspector を使用して、現行のプロセス設計と実装をプレイバック・セッションで実際に実行することができます。プレイバック・セッションは、プロセス内のさまざまな利害関係者 (管理者、エンド・ユーザー、およびビジネス分析者など) から重要な情報を収集するのに役立ちます。反復的なプロセス開発手法をとることにより、プロセス・アプリケーションは、目標と関与する全員の必要を確実に満たすようになります。

IBM Process Designer の Inspector には、いくつかのツールが組み込まれており、それらのツールを使用して、それぞれの構成済み環境で以下のようなタスクを実行できます。

タスク	説明
プロセスのインスタンスの管理	プロセスを実行したとき、環境内の IBM Business Process Manager サーバー上で以前に実行されたインスタンスと現在実行中のインスタンスをすべて表示できます。実行中インスタンスを管理するには、例えば、それらのインスタンスを一時停止してから再開します。また、特定のレコードをフィルターに掛けるか削除することにより、以前に実行されたインスタンスを管理できます。
プロセスのステップスルーとデバッグ	選択したインスタンスについて、現在実行中のステップを調べた後、プロセスの次のステップに進み、プロセスの実行をステップごとに評価します。プロセスのツリー表示とトークンと呼ばれるプロセス・ダイアグラム内のインディケーターを組み合わせると、現行プロセスのどこにいるかを簡単に理解できます。また、各ステップで使用された変数と、それらに対応する値を表示できるという利点もあります (該当する場合)。

Inspector インターフェースについて詳しくは、以下のトピックを参照してください。

処理	参照トピック
選択したサーバー上で現在および過去に実行されたプロセスのインスタンスを管理する。	プロセス・インスタンスの管理
プロセス実行をステップスルーして、BPD が期待どおりに機能していることを確認する。	プロセスのステップスルー

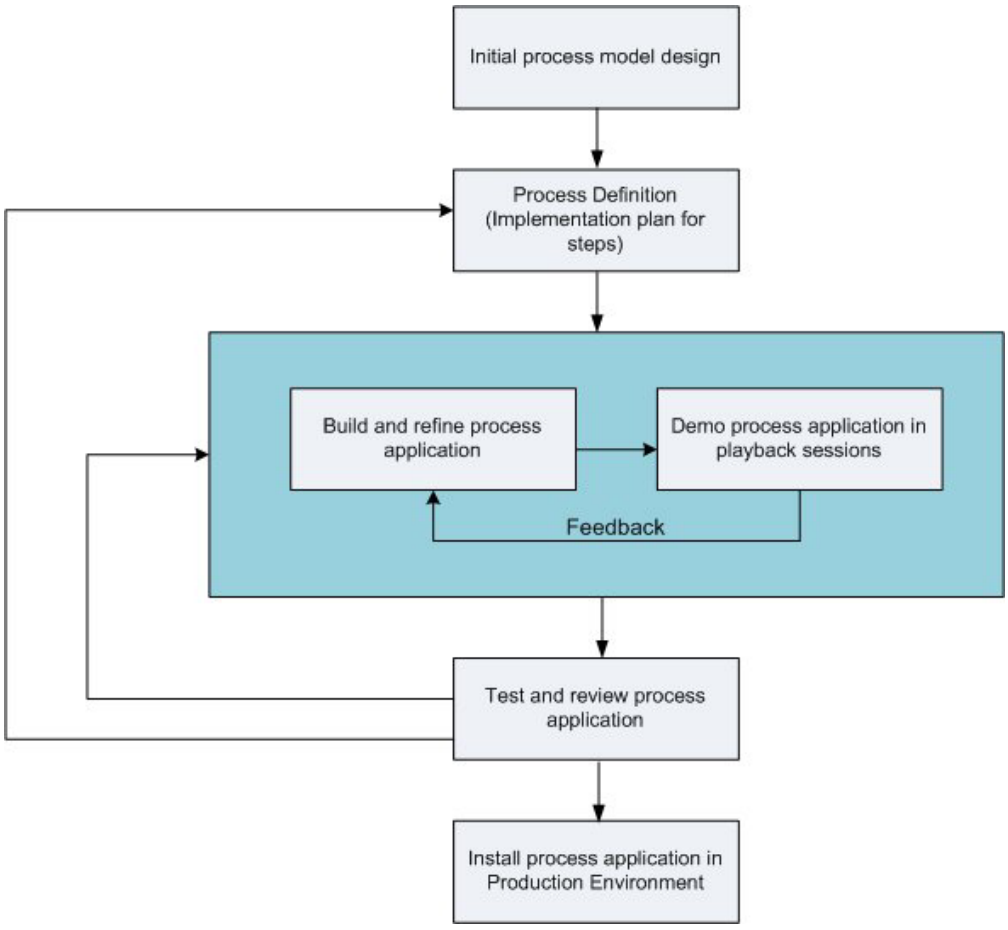
処理	参照トピック
基礎となっている各プロセスまたはサービスをプロセス実行のステップごとに検査して、単にプロセスをステップスルーする以上の詳しい調査を行う。	プロセスのデバッグ
BPD を実行したときに生成されたエラーの原因を簡単に見つけて、それらを解決する。	エラーの解決
Inspector が提供する各フィーチャーにアクセスし、使用する。	Inspector の解説

Process App のデプロイおよび管理

Process App のライフサイクルには、Snapshot のデプロイとアンデプロイ、および Snapshot の状態の管理が含まれます。バージョン管理に関する考慮事項も、ライフサイクルの一部です。

プロセスを開発する際には、Process Designer 内のツールでサポートされる反復アプローチを存分に活用することができます。プロセスは、初期の開発状態からテスト段階を経て実稼働段階へと、時間とともに進化していきます。実動段階でも、ニーズの変化によってプロセスが進化を続けることがあります。そのため、絶えず進化するプロセスのライフサイクルに対して準備することが重要です。こうした準備により、最初から効率的に設計を進めることができます。

以下の図は、プロセス開発の反復アプローチを示したものです。



標準的な Business Process Manager 構成には、プロセスの開発と最終デプロイメントをサポートする 3 つの環境が含まれています。

環境	説明
開発	IBM Process Designer で Process App をビルドして改善します。Designer を使用してプロセス・モデルを作成し、このモデル内のステップを実装します。プロトタイプを素早く評価して改善できるように、インスペクターを使用して、開発の進行状況のデモをプレイバック・セッションで行います。Process Center Console を使用して、Process App をテスト環境と実稼働環境でデプロイします。
テスト	Process Center Console を使用して Process App をテスト環境の Process Server にデプロイし、正式な品質保証テストを実行します。インスペクターを使用すると、問題の検証と解決に役立ちます。
実稼働	正式なテストで報告されたすべての問題が解決したら、Process Center Console を使用して、Process App を実稼働環境の Process Server にデプロイします。インスペクターを使用すると、実稼働環境で報告された問題の調査と解決に役立ちます。

リリースおよびインストールの方針

実装およびデプロイする Process App が組織の品質基準を満たすようにするため、リリースとインストールの方針を定義することを検討してください。新規および更新済みの Process App のリリースとインストールの目標と要件を定義したら、プログラムの承認と起動に必要なプロセスを自動化することができます。

例えば、プロセスによっては、組織内のさまざまな報告構造にまたがって、複数の異なる管理者を経由することが必要な場合があります。新規プロセスまたは更新プロセスは、各管理者がそのプロセスからサインオフした後でのみ、実稼働環境にデプロイして、エンド・ユーザーにロールアウトできます。IBM Business Process Manager Advanced を使用すると、このようなレビューに含まれるステップを作成および実施して、すべての企業ガイドラインを満たしながら、必要な署名を得ることができます。レビューの最終ステップでは、承認された Process App がデプロイメント可能になったことを IT チームに通知できます。

以下のセクションでは、プロセスのデプロイと、デプロイ後のプロセスの管理について説明します。

サービスの作成、アクセス、取り込み

アプリケーションの外部にあるサービスへのアクセス

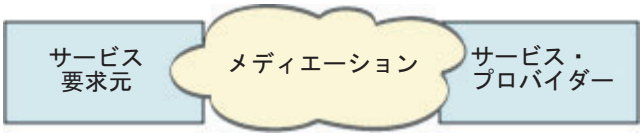
このシナリオでは、アプリケーション外部のサービスにアクセスするためのさまざまな方法について説明し、このような外部サービスへのアクセスのための上位タスクを記載します。

注: このシナリオは、 WebSphere® Enterprise Service Bus および IBM Business Process Manager に適用されます。 WebSphere Enterprise Service Bus および IBM Business Process Manager には、メディエーション・モジュールをデプロイすることができます。モジュールは、IBM Business Process Manager にデプロイ可能です。

統合ビジネス・アプリケーションでは、ビジネス・サービス 同士が、必要な機能を提供するために相互に対話します。ビジネス・サービスは、ビジネスの目標達成に役立つ反復可能な機能やタスクを実行します。しかし、あるサービスの場所を探し、そのサービスに接続する作業は、ビジネス機能には関連していません。サービスへの接続を管理するタスクからビジネス機能を分離することにより、ソリューションに柔軟性が与えられます。

サービスの対話は、サービス・リクエスター がビジネス機能を実行するために、サービス・プロバイダーに対して要求を送信したときに開始します。この要求は、実行すべき機能を定義するメッセージ という形

で送信されます。サービス・プロバイダーは、要求された機能を実行して、その結果をメッセージとしてサービス・リクエスターに送信します。一般に、サービスでデータを交換できるようにするために、また、ビジネス機能やデータに依存しない下位レベルのほかの IT 機能を実装するために、メッセージを処理する必要があります。例えば、ルーティング、プロトコル変換、データ変換、失敗した呼び出しの再試行、動的サービス呼び出しがこれに当たります。この処理をメディエーション と呼びます。



IBM Integration Designer には、2 種類のモジュールがあります。主としてビジネス・ロジック (ビジネス・プロセス、ビジネス・ルール、およびビジネス・ステート・マシンなど) から構成される設計になっているモジュール (つまり、ビジネス・インテグレーション・モジュール) と、メディエーション・フローを実装するメディエーション・モジュールです。この 2 種類のモジュールの間には、ある程度の機能の重複がありますが、一般的にはビジネス・ロジックはビジネス・モジュール内に分離し、メディエーション・ロジックはメディエーション・モジュールで実行することをお勧めします。

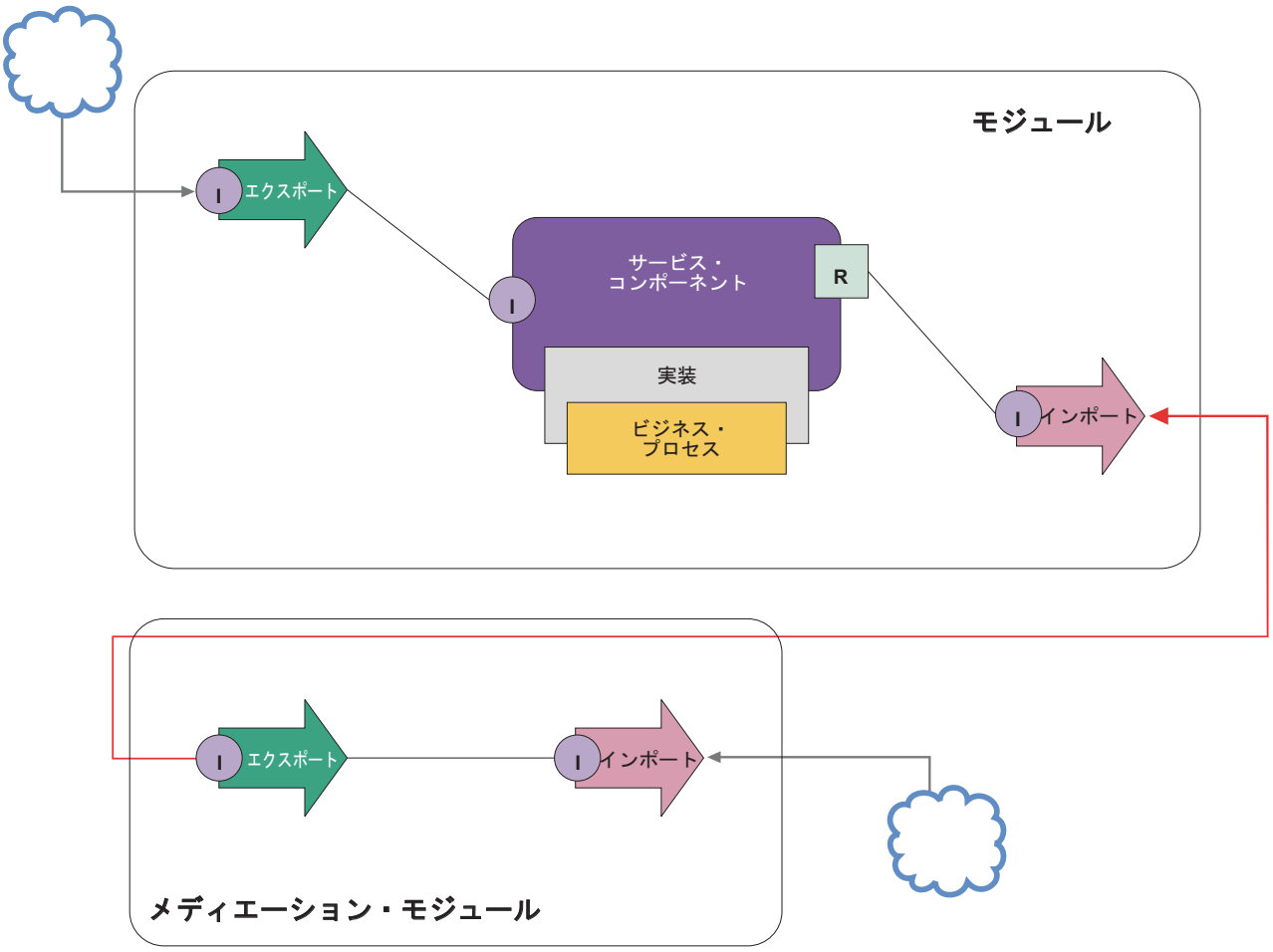
ただし、ビジネス・ロジックとメディエーション・ロジックの間に、常に明確な区別があるとは限りません。このような場合には、ステート、つまり、サービス呼び出し間で処理する必要のある変数内のデータの量を考慮してください。通常、ステート処理が不要であるか、またはほとんど必要でない場合は、メディエーション・フロー・コンポーネントを使用することを考えてください。サービス呼び出し間のステートを保管する必要がある場合、または変数内に保管して処理する必要があるデータが存在する場合は、ビジネス・プロセス・コンポーネントを使用することを考えます。例えば、複数のサービスを呼び出し、それぞれのサービスから返された情報を記録しておき、すべてのサービスを呼び出した後に、返されたデータを使用してさらに何らかの処理を実行する場合は、ビジネス・プロセスを使用すると、返された情報を簡単に変数に代入することができます。すなわち、ステートが多すぎる場合は、ビジネス・ロジックの領域に入っています。

統合のシナリオは 1 つだけではなく、厳密に言えば誤った答えはありません。ここで扱う指針は、柔軟性と再利用を可能にする適正な実施例であり、検討していただくために提示しています。ご使用のビジネス・インテグレーション・アプリケーションに対して、これらのパターンを実装することの利点と欠点を通常どおり入念に検討する必要があります。いくつかの状況を考えてみましょう。

SCA コンポーネントへのアクセス

サービスへのアクセスの基本的な例は、インポートで別の SCA コンポーネントを呼び出すときに、データ変換の必要がない場合です。この状況でも、ビジネス・モジュールから直接外部サービスにアクセスするのではなく、メディエーション・モジュールから外部サービスにアクセスすることが可能です。これにより、サービスを利用するビジネス・コンポーネントに影響を与えることなく、サービス・エンドポイントや、サービスまたはガバナンスの特性の変更 (ロギングの追加など) のための柔軟性が将来において得られます。このアーキテクチャー・パターンは、「関心の分離」と呼ばれます。

このパターンの実装を決める前に、このパターンの利点を、別のモジュールによって発生するオーバーヘッドの潜在的な影響と対照して慎重に検討してください。柔軟性が主な要件である場合に、アクセスされるサービスに頻繁に変更を行うことになるときは、ここで示すように別のモジュールを使用することを考えてください。パフォーマンスが最も重要である場合に、ビジネス・ロジックの更新や再デプロイを行う用意があれば、単一のモジュールを使用することを考えてください。



この例を実現するための上位タスクを以下に示します。

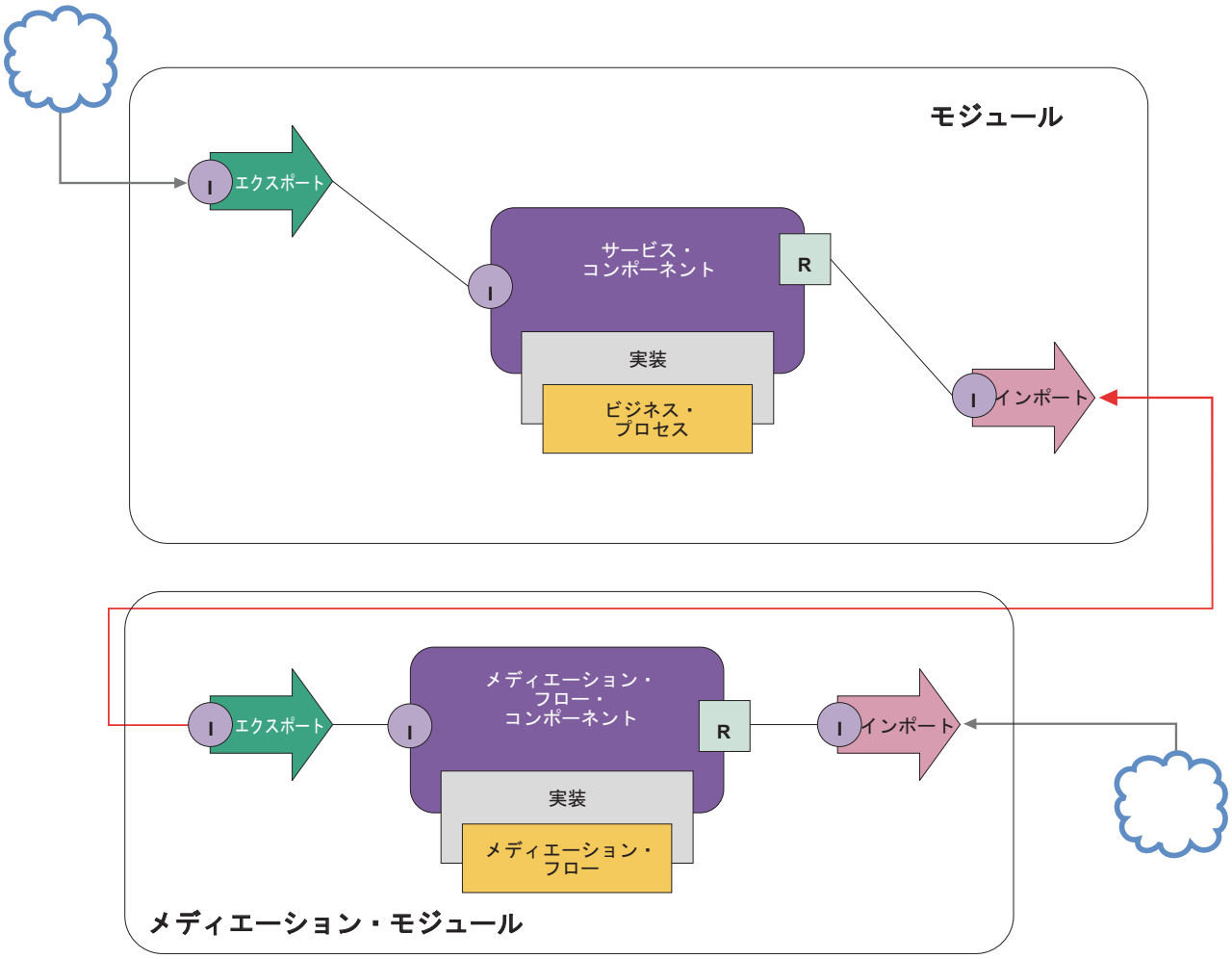
1. メディエーション・モジュールを作成します。段階的な説明については、『メディエーション・モジュールの作成』 を参照してください。
2. メディエーション・モジュール内で、アクセスする外部サービス用の適切なバインディングを持つインポートを作成します。段階的な説明については、 『インポートの作成』を参照してください。バインディングについて詳しくは、 『バインディング』を参照してください。
3. エクスポートを作成し、これにインポートと同じインターフェースを指定します。段階的な説明については、『エクスポートの作成』を参照してください。
4. エクスポート用の SCA バインディングを生成します。段階的な説明については、 『SCA バインディングの生成』を参照してください。
5. メディエーション・モジュールのアセンブリーで、エクスポートをインポートにワイヤリングします。メディエーション・モジュールを保管します。
6. モジュールを作成します。段階的な説明については、『ビジネス・サービスに関するモジュールの作成』を参照してください。
7. エクスポートを追加し、次にコンポーネントを追加します。
8. 「ビジネス・インテグレーション」ビューで、メディエーション・モジュールで作成したエクスポート (ステップ 4) をモジュール・アセンブリー内にドラッグします。エクスポートと同じバインディングを持つインポートが作成されます。

9. エクスポートをコンポーネントにワイヤリングし、コンポーネントをインポートにワイヤリングします。
10. コンポーネントの実装を追加します。実装タイプについては、『実装』を参照してください。

後で、ロギングやルーティングなどのメディエーション・ロジックを、ビジネス・モジュールに影響を与えずにメディエーション・モジュールに追加することができます。

メディエーションの追加

時として、単に外部サービスを呼び出すだけでは不十分なことがあります。場合によっては、メディエーション・モジュールをサービスのリクエスターとプロバイダーの間の中継として追加することにより、最初に処理を実行する必要があります。



以下に中継メディエーション・フローで実行する機能のいくつかを示します。

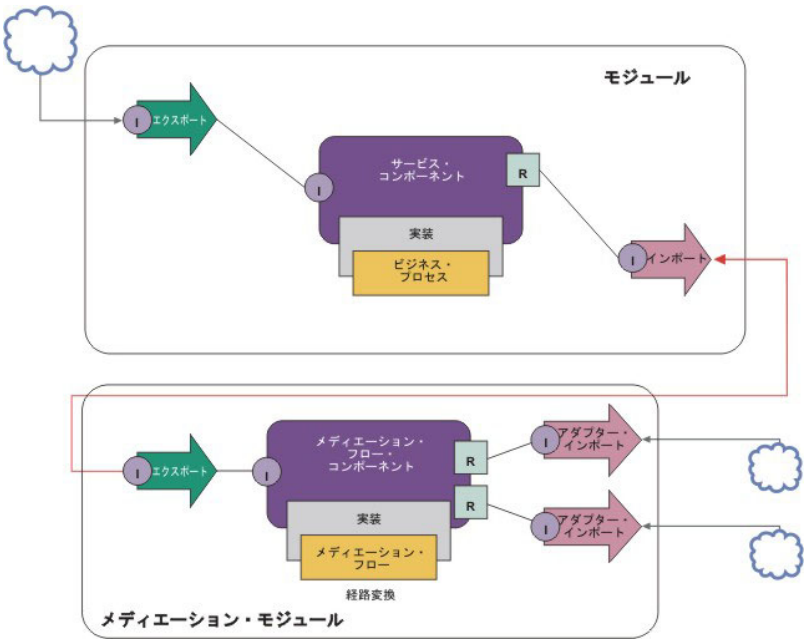
- プロトコル・ヘッダーを設定する。詳しくは、WebSphere Enterprise Service Bus インフォメーション・センターの『プロトコル変換』のトピックを参照してください。
- ビジネス・オブジェクト・マップまたは XSL 変換プリミティブを使用して、インターフェースまたはパラメーターを変換する。メッセージの変換
- メッセージ・フィルター・プリミティブを使用して、静的リストから特定のサービスを選択する。メッセージ・フィルター (Message filter)

- ファンアウトおよびファンイン・プリミティブを使用して、複数のサービスを呼び出し、結果を集約する。メッセージのブロードキャストと集約
- サービス呼び出しの失敗に対処するため、サービス呼び出しプリミティブを使用して、同じサービスを再試行するか、別のサービスを呼び出す。失敗したサービス呼び出しの再試行
- 使用するサービスを統合時ではなく実行時に選択することによる動的ルーティング。これにより、サービスはより疎結合となり、企業は変更に対しより迅速に対応できるようになります。ランタイム環境にデプロイされたモジュールに手を加えずに、新規サービスを追加できます。 動的ルーティングは、レジストリーと共に使用されたときに最も強力になります。この場合、エンドポイント・ルックアップ・メディエーション・プリミティブを使用する必要があります。エンドポイントの動的選択

エンタープライズ情報システムへのアクセス

外部システムのサービスおよび成果物を、Integration Designer にインポートすることができます。ウィザードにより、エンタープライズ情報システム (EIS) 上のアプリケーションやデータが検出され、検出されたアプリケーションやデータからサービスを生成することができます。生成された成果物はインターフェースやビジネス・オブジェクトです。これらは、モジュール内のコンポーネントが使用することができます。

モジュールとホスト・システム間で中継メディエーション・モジュールを使用すると、アプリケーションやデータの再利用性が高まります。以下に示す例では、メディエーション・フローを使用して、正しいホスト・システムへの経路指定を行い、データをそのホスト・システムが必要とする形式に変換します。



この例の上位タスクを以下に示します。

1. 外部サービス・ウィザードを使用して、ホスト・システムに接続します。外部サービス・ウィザードを使用して外部サービスにアクセスする場合は、使用するアダプターに関係なく、同様のパターンをたどります。外部サービス・ウィザードの使用法については、『アダプターを使用した外部サービスへのアクセス・パターン』を参照してください。
2. モジュールを作成します。段階的な説明については、『ビジネス・サービスに関するモジュールの作成』を参照してください。
3. SCA バインディングを持つエクスポート、コンポーネント、およびインポートを追加します。詳しくは、『サービスの呼び出し』を参照してください。

4. エクスポートにインターフェースを追加し、そのエクスポートをコンポーネントにワイヤリングします。
5. コンポーネントの実装を追加します。実装では、どのホスト・サービスにアクセスするかを示すプロパティーを設定します。実装タイプについては、『実装』を参照してください。
6. SCA バインディングと、ステップ 2 で作成したモジュールのインポートと同じインターフェースを持つエクスポートにより、メディエーション・モジュールを作成します。
7. エクスポートをメディエーション・フロー・コンポーネントにワイヤリングします。
8. アセンブリー・エディターのパレットから適切なアウトバウンド・アダプターを使用して、アクセスするホスト・システムごとにインポートを作成します。
9. メディエーション・フロー・コンポーネントを各インポートにワイヤリングします。
10. メディエーション・フロー・コンポーネントを実装します。メッセージ・フィルター・プリミティブを使用して、ビジネス・ロジックに設定されたプロパティーに基づいてインポートを選択し、アダプターのインポートごとに XSL 変換プリミティブを使用します。メッセージ・フィルター (Message filter)
11. モジュール内で、モジュールにインポートするサービスとして、メディエーション・モジュールのエクスポートを選択します。段階的な説明については、『別のモジュールからのサービスの呼び出し』を参照してください。

後で、ビジネス・ロジックに対する影響をできるだけ抑えながら、アダプターを追加したり、アダプターを別のホスト・システムを指すように変更したりするなどの変更を行うことができます。

メッセージング・システムへのアクセス

Service Component Architecture (SCA) モジュールが既存の JMS、MQ、または MQ JMS メッセージング・クライアントと通信するためには、インターフェース、ビジネス・オブジェクト、およびインポートとエクスポートのバインディングを作成する必要があります。『SCA インターフェースへのメッセージのマッピング』を参照してください。

メディエーション・フローでは、メッセージを使用します。メッセージは、ビジネス・オブジェクトに加えて、コンテキストやヘッダー情報にアクセスできるようにします。JMS ヘッダー情報やカスタム JMS プロパティーへのアクセスが必要な場合に、メディエーション・フローを使用します。また、MQ システムと統合する場合や、MQ ヘッダー情報にアクセスしたい場合も、メディエーション・フローを使用します。

Web サービスの作成および呼び出し

Web サービスは、単純な照会から複雑なビジネス・プロセスの連携まで、さまざまなビジネス機能を実行する内蔵タイプのアプリケーションです。既存の Web サービスを呼び出すことも、あるいはニーズに合わせて新規 Web サービスを作成することもできます。このシナリオでは、ステップについて説明し、追加情報を示します。

すべてのサービスを、IBM Integration Designer を使用して最初から作成するわけではないとしても、そのうちのいくつかはこの方法で作成することになります。アセンブリー・エディターおよびビジネス・プロセス・エディターを使用してサービスをビジネス・プロセスに組み立てるときに、いくつかのサービスが足りないと気付くことがあります。したがって、この不足しているサービスを、IBM Integration Designer のツールを使用して作成するとよいでしょう。この逆もまた真です。新しいプロセスを作成した後に、プロセス操作の全部または一部を、他のユーザーが利用できるように、サービスとして公開すると有用であると判断することもあります。

注: このシナリオは、IBM Integration Designer for IBM Process Server と WebSphere Enterprise Service Bus のユーザーに適用されます。

IBM Integration Designer を使用して Web サービスを開発することには、いくつかの理由があります。

- IBM Integration Designer でサービスを作成すると、ビジネス・ルールを使用してサービスを実装できます。
- IBM Integration Designer で開発すると、Java サービスを開発し、それを Web サービスとして、および SCA を通して公開することができます。
- コーディングの必要がないインターフェース・マッピングが利点です。Java コードからすべてのデータ・マッピングを取り出して、Java 開発者に、単純なブラック・ボックスの Java プログラムを委ねることができます。
- IBM Integration Designer では、サービスおよびリレーションシップのすべてを 1 カ所に示します。
- IBM Integration Designer を使用した Web サービスの開発には、リファクタリング機能の支援もあります。

Web サービスがすべての統合の問題の解決策だとみなしてはならないということに留意してください。しかし、他のすべてのテクノロジーまたはアーキテクチャー・アプローチと同じように、適切な場面とタイミングで Web サービスを使用することには、それ固有の利点があります。

エクスポート、インポート、およびバインディング

IBM Integration Designer によって、標準の Web サービスをインポートし、これらのサービスを複合アプリケーションで利用することができます。

IBM Integration Designer では、サービスの開発にアセンブリー・エディターを使用します。標準のプロセスに従って、モジュール、メディエーション・モジュール、ライブラリー、およびコンポーネントを作成します。その結果、エクスポート、インポート、およびバインディングを使用して、これらのサービスを共有し、これらのサービスにアクセスすることができます。このような基本的なタスクの手順と、各タスクについての詳しい情報へのリンクを以下に示します。

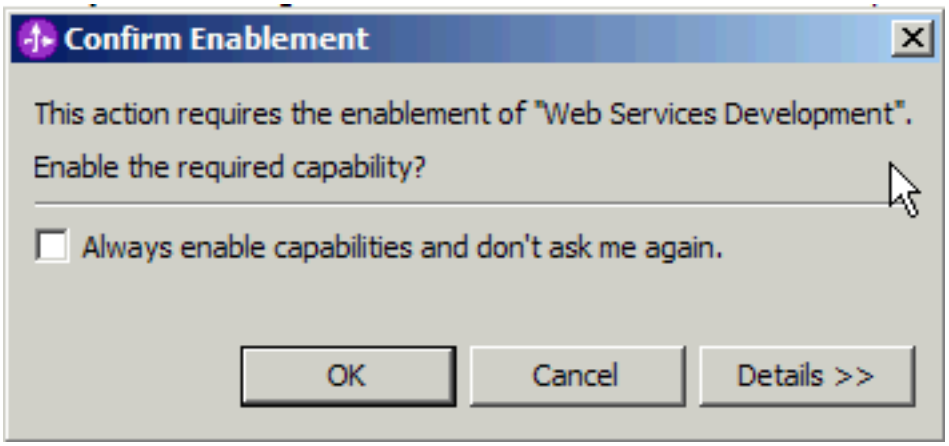
Web サービスでは、Web サービス・バインディングまたは HTTP バインディングという 2 つのバインディングのいずれかを使用できます。Web サービス・バインディングは、Web サービスとの間でメッセージの伝送を行うための仕様を規定します。ツールを使用すると、Web サービス・バインディングを自動的に生成することができます。HTTP バインディングは、World Wide Web Consortium (W3C) によって公開されている HTTP プロトコルで規定された、クライアントとサーバー間の標準の要求/応答プロトコルです。HTTP バインディングを使用する場合は、一定の初期バインディング構成情報を提供する必要があります。

1. モジュールのサービスをほかのモジュールで使用できるように公開するために、エクスポートを作成します。
2. エクスポートのバインディングを生成します。
   - エクスポートの Web サービス・バインディングを生成します。
   - HTTP エクスポート・バインディングを生成します。
3. 組み立てるモジュールには含まれていない、既存のサービスを呼び出すために、インポートを作成します。
   - インポートの Web サービス・バインディングを生成します。
   - HTTP インポート・バインディングを生成します。

JavaServer Pages から Web サービスを呼び出す場合は、このリンク先のトピックを参照してください。

Web サービス開発ケイパビリティー

Web サービス作成プロセスに関連付けられているエディターを開くと、以下のエラーが表示されることがあります。



IBM Integration Designer では、*ケイパビリティー* と呼ばれるフィルター機能を提供しています。「設定」では、各機能およびツールはケイパビリティーに分類されていて、ケイパビリティーの分類またはいずれかの分類のサブセット機能を使用可能または使用不可に設定できます。詳しくは、『ケイパビリティー』を参照してください。

第 2 章 IBM Business Process Manager の詳細の習得

このセクションは、IBM Business Process Manager 内で使用されているテクノロジー、およびこの製品によって使用されているテクノロジーを調査するための開始点として利用してください。

バージョン管理

Process App のライフサイクルは、Process App の作成から始まり、Process App の更新、デプロイ、混在デプロイ、アンデプロイ、アーカイブのサイクルにわたって継続します。*バージョン管理* とは、Process App の個々のバージョンを一意的に識別することで、Process App のライフサイクルを管理する際に使用されるメカニズムです。

IBM Business Process Manager でのバージョン管理の仕組みは、デプロイするアプリケーションのタイプ、つまり IBM Process Center のリポジトリーからデプロイされた Process App であるのか、IBM Integration Designer から直接デプロイされたエンタープライズ・アプリケーションであるのかに応じて異なります。

デフォルトでは、Process Center からランタイム環境にデプロイする Process App および Toolkit はバージョン管理されます。エンタープライズ・アプリケーションの場合、IBM Integration Designer でモジュールおよびライブラリーのバージョン管理を行うことができます。

また、ヒューマン・タスクまたはステート・マシンのバージョンを作成すると、複数のバージョンのタスクまたはステート・マシンがランタイム環境で共存できます。

Process App のバージョン管理

バージョン管理は、ランタイム環境で Process App のライフサイクル内の Snapshot を識別し、複数の Snapshot を同時に並行して実行できるようにする機能です。

Process App はコンテナーと考えてください。すべての Snapshot、デプロイメント、およびバージョン管理は、コンテナー内の成果物のレベルではなく、コンテナー・レベルで処理されます。Snapshot は、Process Center Console から管理されます。

変更内容は、Process Center リポジトリーの作業トラックのチップに動的に保存されます。Snapshot (sn1) の作成を決定するまで、Process App はそのチップ・レベルに残ります。Process App Snapshot は、テスト、ステージング、または実動用として Process Center サーバーまたは Process Server にデプロイできます。

変更を加えて新しいバージョンをデプロイする場合は、新規 Snapshot (sn2) を作成する必要があります。sn2 のデプロイ時には、sn1 を削除しても、sn1 をサーバー上で実行させておいても構いません。

バージョン・コンテキスト

バージョン・コンテキストは、バージョンを識別するメタデータです。これに ID を割り当てることもできますが、IBM では <major>.<minor>.<service> という形式の 3 桁の数字によるバージョン構造を使用することをお勧めします。このバージョン管理方式について詳しくは、命名規則に関するトピックを参照してください。

IBM Business Process Manager は、各 Process App にグローバル名前空間を割り当てます。グローバル名前空間は、具体的には Process App チップまたは特定の Process App Snapshot を表します。サーバーが使用するバージョン名は 7 文字までです。したがって、割り当て名は、割り当てられた Snapshot 名の文字を使用した頭字語になります。Snapshot 名が推奨の IBM VRM スタイルに従っていて、7 文字を超えていない場合、Snapshot の頭字語とその Snapshot 名は同じになります。例えば、 1.0.0 という Snapshot 名の頭字語は 1.0.0、10.3.0 という Snapshot 名の頭字語は 10.3.0 になります。 Snapshot の頭字語は、Process Center Server のスコープにある Process App のコンテキスト内で固有になることが保証されます。このため、Snapshot の頭字語を編集することはできません。

Process Designer Process App および Toolkit のバージョン管理

Process Center リポジトリーに保管されている Process App および Toolkit のバージョン管理のため、Snapshot の保存および名前指定を行うことができます。そのようにすることで、ある Snapshot を他の Snapshot と比較して違いを見つけることができるようになります。例えば、ある開発者がサービスに関する問題を修正して、その時点でそのサービスを含む Process App または Toolkit の Snapshot を作成し、他の開発者が同じサービスに対していくつかの追加変更を加えて新たに Snapshot を作成した場合、プロジェクト・マネージャーはそれら 2 つの Snapshot を比較して、どの変更が、いつ、どのユーザーによって行われたかを確認することができます。プロジェクト・マネージャーは、サービスに対するそれらの追加変更が不要であると判断した場合には、元の修正状態の Snapshot に戻すことができます。

通常、実動へデプロイする準備、または統合をテストする準備ができた、または準備ができた可能性があるときに、毎回 Process App の Snapshot を作成します。スタンドアロンの Process Server にデプロイする場合は、Process App の Snapshot を作成する必要があります。 Toolkit の場合の対応は多少異なります。Toolkit の Snapshot を作成するのは、その Toolkit を Process App で使用する準備ができた時点です。後で Toolkit を更新する場合は、準備ができた時点で「チップ」の別の Snapshot を作成する必要があります。その後で、Process App と Toolkit の所有者は、新しい Snapshot に移行するかどうかを決定できます。チップは特殊な Snapshot であり、内容を変更できる唯一の種類の Snapshot ですが、Process Center Server 上でしか実行できません。チップは Process Server にデプロイできません。

複数のクラスター内の Process App

同じバージョンの Process App を、同じセル内の複数のクラスターにデプロイすることができます。同じバージョンの Process App のこうした複数のデプロイメントを区別するには、各デプロイメントの Snapshot を作成し、その Snapshot 名にセル固有の ID を含めます (v1.0_cell1_1 や v1.0_cell1_2 など)。(純粋なライフサイクル管理の観点から見ると) 各 Snapshot は、厳密には新しいバージョンの Process App と言えますが、内容と機能は同じです。

Process App がクラスターにデプロイされる際に、ノードの自動同期が実行されます。

モジュールおよびライブラリーのバージョン管理

Process App または Toolkit に含まれているモジュールまたはライブラリーは、その Process App または Toolkit のライフサイクル (バージョン、Snapshot、トラックなど) を継承します。モジュールおよびライブラリーの名前は、Process App または Toolkit の範囲内で固有でなければなりません。

このトピックでは、Process App で使用するモジュールおよびライブラリーのバージョン管理について説明します。ただし、IBM Integration Designer から Process Server にモジュールを直接デプロイする場合は、『バージョン管理されたモジュールおよびライブラリーの作成』で説明されているように、デプロイメント中にモジュールに対してバージョン番号を割り当てる手順を使用することができます。

IBM Process Center と関連付けられたモジュールまたはライブラリーは、自身の従属ライブラリーを同じ Process App 内または従属 Toolkit 内に持つ必要があります。

次の表は、ライブラリーが Process App または Toolkit に関連付けられている場合に、IBM Integration Designer の依存関係エディターで選択できる内容についてリストしています。

表 3. モジュール、Process App または Toolkit、およびグローバル・ライブラリーの依存関係

ライブラリーのスコープ	説明	依存可能な対象
モジュール	このライブラリーのコピーは、それを使用する各モジュールのサーバー上に置かれます。	モジュール・スコープ・ライブラリーは、他のすべてのタイプのライブラリーに依存できます。
Process App または Toolkit	このライブラリーは、Process App または Toolkit の範囲内にあるすべてのモジュール間で共有されます。この設定は、IBM Process Center からデプロイメントが行われた場合に有効になります。 IBM Process Center の外部でデプロイメントが行われた場合は、このライブラリーが各モジュールにコピーされます。 **注:** IBM Integration Designer バージョン 7.5 で作成されたライブラリーのデフォルトの共有レベルは「**Process App または Toolkit**」になります。	このタイプのライブラリーは、グローバル・ライブラリーにのみ依存できます。
グローバル	このライブラリーは、実行中のすべてのモジュール間で共有されます。	グローバル・ライブラリーは、他のグローバル・ライブラリーにのみ従属することができます。 **注:** グローバル・ライブラリーをデプロイするには、WebSphere 共有ライブラリーを構成する必要があります。詳しくは、『モジュールおよびライブラリーの依存関係』を参照してください。

命名規則

Process App では、更新、デプロイ、混在デプロイ、アンデプロイ、およびアーカイブというライフサイクルを進むため、命名規則を使用して、Process App のさまざまなバージョンを区別します。

このセクションでは、Process App のバージョンを一意的に識別するための規則について説明します。

バージョン・コンテキストは、Process App または Toolkit を一意的に示す頭字語の組み合わせです。頭字語のタイプごとに、以下の命名規則があります。頭字語は、最大長は 7 文字で、使用できる文字セットは [A-Z0-9_] です (Snapshot の頭字語は例外で、ピリオドも使用できます)。

- Process App の頭字語は、Process App の作成時に作成されます。最大長は 7 文字です。
- Snapshot の頭字語は、Snapshot の作成時に自動的に作成されます。最大長は 7 文字です。

  Snapshot 名が有効な Snapshot の頭字語基準を満たしている場合、Snapshot 名と頭字語は同じになります。

注: メディエーション・フロー・コンポーネントのバージョン認識ルーティング機能を使用する場合は、<バージョン>.<リリース>.<変更> スキーム (例: **1.0.0**) に従うように Snapshot に名前を付けてください。 Snapshot の頭字語は 7 文字までに制限されているため、数字の値は合計で最大 5 桁 (5 桁 + 2 つのピリオド) までに制限されます。したがって、値が大きくなっていく数字フィールドの場合は、最初の 7 文字を超える部分は切り捨てられるので注意してください。

例えば、**11.22.33** という Snapshot 名は、Snapshot の頭字語では **11.22.3** となります。

- トラックの頭字語は、トラック名の各単語の先頭文字を使用して、自動的に生成されます。例えば、**My New Track** という名前で作成された新規トラックの頭字語値は **MNT** になります。

  デフォルトのトラック名および頭字語は **Main** です。トラックの頭字語が **Main** でない場合、IBM Process Center Server へのデプロイメントには、バージョン管理コンテキストのトラックの頭字語が組み込まれます。

Process App 内のビジネス・プロセス定義は一般に、Process App 名の頭字語、Snapshot の頭字語、およびビジネス・プロセス定義の名前で識別されます。ビジネス・プロセス定義には、できる限り固有の名前を選択してください。重複する名前があると、以下の問題が発生する可能性があります。

- 何らかの形式のメディエーションがないと、ビジネス・プロセス定義を Web サービスとして公開できない可能性があります。
- IBM Process Designer で作成されたビジネス・プロセス定義を、IBM Integration Designer で作成された BPEL プロセスから呼び出すことができない可能性があります。

バージョン・コンテキストは、Process App のデプロイ方法によって異なります。

Process Center サーバー・デプロイメントの命名規則

IBM Process Center Server では、Process App の Snapshot、および Toolkit の Snapshot をデプロイすることができます。さらに、Process App のチップ、または Toolkit のチップをデプロイすることもできます。チップ とは、Process App または Toolkit の現在の作業バージョンです。バージョン・コンテキストは、デプロイメントのタイプによって異なります。

Process App の場合は、バージョンを固有に識別するために、Process App チップまたは Process App の固有の Snapshot が使用されます。

Toolkit は 1 つ以上の Process App とともにデプロイできますが、各 Toolkit のライフサイクルは、Process App のライフサイクルにバインドされます。各 Process App には、サーバーにデプロイされた従属 Toolkit (複数の場合もある) の独自のコピーがあります。デプロイされた Toolkit は Process App 間で共有されません。

Process App と関連付けられたトラックに、デフォルトの **Main** 以外の名前が付いている場合は、トラックの頭字語もバージョン・コンテキストの一部になります。

Process App の Snapshot

Process App の Snapshot のデプロイメントでは、以下の項目の組み合わせがバージョン・コンテキストになります。

- Process App 名の頭字語
- Process App トラックの頭字語 (**Main** 以外のトラックが使用されている場合)
- Process App の Snapshot の頭字語

スタンドアロン Toolkit

Toolkit の Snapshot のデプロイメントでは、以下の項目の組み合わせがバージョン・コンテキストになります。

- Toolkit 名の頭字語
- Toolkit トラックの頭字語 (**Main** 以外のトラックが使用されている場合)
- Toolkit の Snapshot の頭字語

ヒント

Process App チップは、Process Designer での反復テスト中に使用されます。Process Center Server のみにデプロイできます。

Process App チップのデプロイメントでは、以下の項目の組み合わせがバージョン・コンテキストになります。

- Process App 名の頭字語
- Process App トラックの頭字語 (**Main** 以外のトラックが使用されている場合)
- 「チップ」

Toolkit チップも、Process Designer での反復テスト中に使用されます。これらは、実動サーバーにはデプロイされません。

Toolkit チップのデプロイメントでは、以下の項目の組み合わせがバージョン・コンテキストになります。

- Toolkit 名の頭字語
- Toolkit トラックの頭字語 (**Main** 以外のトラックが使用されている場合)
- 「チップ」

例

リソースには固有の名前を付け、バージョン・コンテキストを使用して外部で識別する必要があります。

- 以下の表は、一意的に識別される名前の例を示しています。この例では、Process App チップには、デフォルトのトラッキング名 (**Main**) が使用されています。

表 4. デフォルトのトラッキング名を持つ Process App チップ

Process App 名	**Process Application 1**
Process App 名の頭字語	**PA1**
Process App のトラック	**main**
Process App のトラックの頭字語	“” (トラックが **Main** の場合)
Process App の Snapshot	
Process App の Snapshot の頭字語	**Tip**

以下の表で示すように、この Process App チップに関連付けられた SCA モジュールには、いずれもバージョン・コンテキストが含まれます。

表 5. SCA モジュールおよびバージョン認識 EAR ファイル

SCA モジュール名	バージョン認識名	バージョン認識 EAR/アプリケーション名
M1	PA1-Tip-M1	PA1-Tip-M1.ear

表 5. SCA モジュールおよびバージョン認識 EAR ファイル (続き)

SCA モジュール名	バージョン認識名	バージョン認識 EAR/アプリケーション名
M2	PA1-Tip-M2	PA1-Tip-M2.ear

- 以下の表は、デフォルト以外のトラッキング名を使用する Process App チップの例を示しています。

表 6. デフォルト以外のトラッキング名を持つ Process App チップ

Process App 名	**Process Application 1**
Process App 名の頭字語	**PA1**
Process App のトラック	**Track1**
Process App のトラックの頭字語	**T1**
Process App の Snapshot	
Process App の Snapshot の頭字語	**Tip**

以下の表で示すように、この Process App チップに関連付けられた SCA モジュールには、いずれもバージョン・コンテキストが含まれます。

表 7. SCA モジュールおよびバージョン認識 EAR ファイル

SCA モジュール名	バージョン認識名	バージョン認識 EAR/アプリケーション名
M1	PA1-T1-Tip-M1	PA1-T1-Tip-M1.ear
M2	PA1-T1-Tip-M2	PA1-T1-Tip-M2.ear

Process Server デプロイメントの命名規則

Process Server では、Process App の Snapshot をデプロイすることができます。 Process App の Snapshot の頭字語は、バージョンを固有に識別する際に使用されます。

Process App の Snapshot のデプロイメントでは、以下の項目の組み合わせがバージョン・コンテキストになります。

- Process App 名の頭字語
- Process App の Snapshot の頭字語

リソースには固有の名前を付け、バージョン・コンテキストを使用して外部で識別する必要があります。以下の表は、一意的に識別される名前の例を示しています。

表 8. 名前および頭字語の例

Process App 名	**Process Application 1**
Process App 名の頭字語	**PA1**
Process App の Snapshot	**1.0.0**
Process App の Snapshot の頭字語	**1.0.0**

リソース (モジュールまたはライブラリーなど) の ID にはバージョン・コンテキストが含まれます。

以下の表は、2 つのモジュールと、関連付けられている EAR ファイルにバージョン・コンテキストがどのように組み込まれるかを示しています。

表 9. SCA モジュールおよびバージョン認識 EAR ファイル

SCA モジュール名	バージョン認識名	バージョン認識 EAR/アプリケーション名
M1	PA1-1.0.0-M1	PA1-1.0.0-M1.ear
M2	PA1-1.0.0-M2	PA1-1.0.0-M2.ear

以下の表は、2 つの Process App スコープ・ライブラリーと、関連付けられている JAR ファイルにバージョン・コンテキストがどのように組み込まれるかを示しています。

表 10. Process App スコープ・ライブラリーおよびバージョン認識 JAR ファイル

SCA Process App スコープ・ライブラリー名	バージョン認識名	バージョン認識 JAR 名
Lib1	PA1-1.0.0-Lib1	PA1-1.0.0-Lib1.jar
Lib2	PA1-1.0.0-Lib2	PA1-1.0.0-Lib2.jar

バージョン認識バインディング

Process App には、インポート・バインディングおよびエクスポート・バインディングを含む SCA モジュールが含まれている場合があります。複数のアプリケーションを一緒にデプロイする場合、アプリケーションの各バージョンに対するバインディングを固有にする必要があります。一部のバインディングは、バージョン間の固有性を確保するために、デプロイメント時に自動的に更新されます。それ以外の場合は、デプロイメント後にユーザーがバインディングを更新して、固有性を確保する必要があります。

バージョン認識バインディングは特定バージョンの Process App に対してスコープが設定され、これによって Process App 間での固有性が確保されます。以下のセクションでは、バージョン認識できるように自動的に更新されるバインディングと、バインディングがバージョン認識できない場合に実行時に行う必要があるアクションについて説明します。モジュールの作成時の考慮事項については、『バインディング使用時の考慮事項』を参照してください。

SCA

モジュールのインポート・バインディングとエクスポート・バインディングが同じ Process App スコープ内に定義されている場合、SCA バインディングのターゲットはデプロイメント時に自動的に名前変更され、バージョンを認識するようになります。

両方のバインディングが同じ Process App スコープ内に定義されていない場合は、情報メッセージがログに記録されます。デプロイメント後に、インポート・バインディングを変更して、エンドポイント・ターゲット・アドレスを変更する必要があります。管理コンソールを使用してエンドポイント・ターゲット・アドレスを変更できます。

Web サービス (JAX-WS または JAX-RPC)

以下のすべての条件を満たしている場合、Web サービス・バインディングのエンドポイント・ターゲット・アドレスはデプロイメント時に自動的に名前変更され、バージョンを認識するようになります。

- デフォルトのアドレス命名規則に従っている。

  http://*ip*:*port*/*ModuleName*Web/sca/*ExportName*
- エンドポイント・アドレスが SOAP/HTTP である。

- モジュールのインポート・バインディングおよびエクスポート・バインディングが同じ Process App スコープ内に定義されている。

これらの条件を満たしていない場合、情報メッセージがログに記録されます。必要なアクションは、Process App のデプロイ方法によって異なります。

- Process App を一緒にデプロイする場合は、SOAP/HTTP エンドポイント URL または SOAP/JMS 宛先キューを手動で名前変更して、Process App のバージョン間での固有性を確保する必要があります。デプロイメント後に、管理コンソールを使用してエンドポイント・ターゲット・アドレスを変更できます。
- 単一バージョンの Process App のみをデプロイする場合は、このメッセージを無視できます。

SOAP/JMS Web サービス・バインディング Snapshot の混在デプロイメントの場合、必要なアクションは、Process App のデプロイ方法に応じて異なります。

- インポートとターゲット・エクスポートが同じ Process App 内である場合、Process App を Process Center に公開して Snapshot を作成する前に、以下の手順を実行します。
  1. エクスポートのエンドポイント URL を変更します。宛先と接続ファクトリーが固有であることを確認してください。
  2. インポートのエンドポイント URL を変更して、前のステップでエクスポートに対して指定したものと同じになるようにします。
- インポートとターゲット・エクスポートが異なる Process App 内にある場合、以下の手順を実行します。
  1. エクスポートのエンドポイント URL を変更します。宛先と接続ファクトリーが固有であることを確認してください。
  2. Process App を Process Center に公開します。
  3. Snapshot を作成します。
  4. Process App を Process Server にデプロイします。
  5. WebSphere 管理コンソールを使用して、対応するインポートのエンドポイント URL を変更し、エクスポートに対して指定したものと同じになるようにします。

HTTP

以下のすべての条件を満たしている場合、HTTP バインディングのエンドポイント URL アドレスはデプロイメント時に自動的に名前変更され、バージョンを認識するようになります。

- デフォルトのアドレス命名規則に従っている。

  http(s)://*ip*:*port*/*ModuleName*Web/*contextPathinExport*
- モジュールのインポート・バインディングおよびエクスポート・バインディングが同じ Process App スコープ内に定義されている。

これらの条件を満たしていない場合、情報メッセージがログに記録されます。必要なアクションは、Process App のデプロイ方法によって異なります。

- Process App を一緒にデプロイする場合は、手動でエンドポイント URL を名前変更して、Process App のバージョン間での固有性を確保する必要があります。デプロイメント後に、管理コンソールを使用してエンドポイント・ターゲット・アドレスを変更できます。
- 単一バージョンの Process App のみをデプロイする場合は、このメッセージを無視できます。

JMS および汎用 JMS

システムによって生成された JMS バインディングと汎用 JMS バインディングは、自動的にバージョンが認識されます。

注: ユーザー定義の JMS バインディングと汎用 JMS バインディングの場合、デプロイメント時にバインディングのバージョン認識を可能にするための自動名前変更は行われません。バインディングがユーザー定義の場合、Process App のバージョン間での固有性を確保するには、以下の属性を名前変更する必要があります。

- エンドポイント構成
- 受信宛先キュー
- リスナー・ポート名 (定義されている場合)

ターゲット・モジュール・エンドポイントを変更する場合は、対応する送信宛先を設定してください。

MQ/JMS および MQ

デプロイメント時には、タイプが MQ/JMS または MQ のバインディングをバージョン認識対応にするための自動名前変更は行われません。

Process App のバージョン間での固有性を確保するには、以下の属性を名前変更する必要があります。

- エンドポイント構成
- 受信宛先キュー

ターゲット・モジュール・エンドポイントを変更する場合は、対応する送信宛先を設定してください。

EJB

デプロイメント時には、タイプが EJB のバインディングをバージョン認識対応にするための自動名前変更は行われません。

Process App のバージョン間での固有性を確保するには、JNDI 名前属性を名前変更する必要があります。

新しい JNDI 名を使用するように、クライアント・アプリケーションも更新する必要があることに注意してください。

EIS

デフォルトのリソース名 (***ModuleName*App:Adapter Description**) が変更されない限り、リソース・アダプターは、デプロイメント中にバージョン認識対応となるように自動的に名前変更されます。

デフォルトのリソース名が変更された場合、リソース・アダプター名は、Process App の複数のバージョン間で固有である必要があります。

リソース・アダプター名が固有でない場合、デプロイメント中に情報メッセージが記録されて警告されます。管理コンソールを使用することで、デプロイメント後にリソース・アダプターを手動で名前変更できます。

バージョン認識での動的起動

実行時に動的に決定されたエンドポイントにメッセージが送信されるように、メディエーション・フロー・コンポーネントを構成することができます。メディエーション・モジュールを作成する際に、バージョン認識での送信に使用するエンドポイント・ルックアップを構成します。

Snapshot に IBM_VRM スタイル (<*version*>.<*release*>.<*modification*) を使用する場合、Process App の EAR ファイルを WebSphere Service Registry and Repository (WSRR) にエクスポートできます。メディエーション・モジュールを作成する際に、バージョン認識での送信に使用するエンドポイント・ルックアップを構成します。例えば、「**一致ポリシー**」フィールドで「**SCA モジュール・ベース・サービスの最新互換バージョンに一致するエンドポイントを返す (Return endpoint matching latest compatible version of SCA module-based services)**」を選択し、「**バインディング・タイプ**」に「**SCA**」を選択します。

最新バージョンの Process App がサーバーにデプロイされ、WSRR に対して公開されて、メディエーション・モジュール・エンドポイント・ルックアップが当該サービス・エンドポイントの最新互換バージョンを動的に起動します。

また、SMOHeader でターゲットを設定し、要求メッセージにより値を伝搬することもできます。

Java モジュールおよびプロジェクトを使用した Process App のデプロイ

Process App には、カスタム Java EE モジュールと Java プロジェクトを含めることができます。複数のアプリケーションを一緒にデプロイする場合、アプリケーションの各バージョンに対するカスタム Java EE モジュールを固有にする必要があります。

カスタム Java EE モジュールおよび Java プロジェクトが、宣言されている依存関係を持つ SCA モジュールとともにデプロイされている場合、これらのモジュールおよびプロジェクトはサーバーにデプロイされます。依存関係の宣言時に「**モジュールと共にデプロイ**」(デフォルト) を選択していない場合、モジュールまたはプロジェクトを手動でデプロイする必要があります。

ビジネス・ルールおよびセレクターを使用した Process App のデプロイ

ビジネス・ルールまたはセレクター・コンポーネントが組み込まれた Process App の複数のバージョンをデプロイする場合は、各バージョンで関連メタデータを使用する方法に注意してください。

ビジネス・ルールまたはセレクター・コンポーネントの動的メタデータは、コンポーネント名、コンポーネントのターゲット名前空間、およびコンポーネント・タイプによって実行時に定義されます。ビジネス・ルールまたはセレクターが含まれた Process App の複数のバージョンが同一のランタイム環境にデプロイされている場合、それらのバージョンは同じルール論理 (ビジネス・ルール) またはルーティング (セレクター) メタデータを共有します。

Process App のビジネス・ルールまたはセレクター・コンポーネントの各バージョンがそれぞれ専用の動的メタデータ (ルール論理またはルーティング) を使用できるようにするには、Process App のバージョンごとに固有になるようにターゲット名前空間をリファクタリングします。

バインディング

サービス指向アーキテクチャーの中核をなすのは「サービス」の概念で、これはコンピューター・デバイス間での対話によって実行される機能単位のことです。「エクスポート」はモジュールの外部インターフェース (またはアクセス・ポイント) を定義し、これによってモジュール内の Service Component Architecture (SCA) コンポーネントは外部クライアントにサービスを提供できます。「インポート」はモジュール外部のサービスへのインターフェースを定義し、これによってサービスはモジュール内部から呼び出すことがで

きます。インポートおよびエクスポートと共にプロトコル固有の「バインディング」を使用して、データをモジュールの内部または外部に移送する方法を指定します。

エクスポート

外部クライアントがインテグレーション・モジュール内の SCA コンポーネントを呼び出すとき、さまざまなプロトコル (HTTP、JMS、MQ、および RMI/IIOP など) を経由して、さまざまなデータ・フォーマット (XML、CSV、COBOL、および JavaBean など) が使用されます。エクスポートはこれらの要求を外部ソースから受信し、SCA プログラミング・モデルを使用して IBM Business Process Manager コンポーネントを呼び出すコンポーネントです。

例えば以下の図で、エクスポートはクライアント・アプリケーションから HTTP プロトコルを経由して要求を受信します。データは SCA コンポーネントによって使用されるフォーマットであるビジネス・オブジェクトに変換されます。そのコンポーネントはそのデータ・オブジェクトによって呼び出されます。

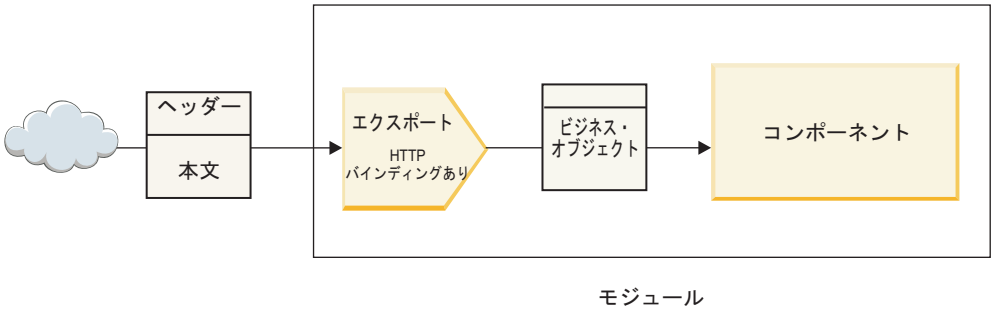


図 1. HTTP バインディングを持つエクスポート

インポート

SCA コンポーネントから、SCA とは異なるフォーマットのデータを要求する SCA 以外の外部サービスを呼び出すことが必要な場合があります。インポートは SCA プログラミング・モデルを使用して外部サービスを呼び出すために、SCA コンポーネントによって使用されます。インポートは次に、ターゲット・サービスが要求する方法でサービスを呼び出します。

例えば以下の図で、SCA コンポーネントからの要求がインポートによって外部サービスに送信されます。SCA コンポーネントによって使用されるフォーマットであるビジネス・オブジェクトは、サービスが要求するフォーマットに変換され、サービスが呼び出されます。

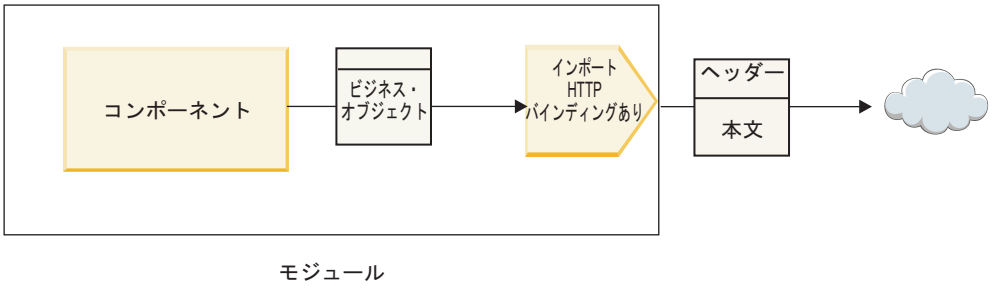


図 2. HTTP バインディングを持つインポート

バインディングのリスト

インポートまたはエクスポートのバインディングを生成して、バインディングを構成するには、WebSphere Integration Developer を使用します。使用可能なバインディングのタイプを以下の一覧に記載します。

- SCA

  SCA バインディングはデフォルトのバインディングで、使用するサービスを他の SCA モジュールのサービスと通信させることができます。インポートと SCA バインディングを使用して、別の SCA モジュールのサービスにアクセスします。エクスポートと SCA バインディングを使用して、別の SCA モジュールにサービスを提供します。

- Web サービス

  Web サービス・バインディングを使用すれば、相互運用可能な SOAP メッセージおよびサービス品質を使用して外部サービスにアクセスすることができます。Web サービス・バインディングを使用すると、添付ファイルを SOAP メッセージの一部として組み込むこともできます。

  Web サービス・バインディングでは SOAP/HTTP (SOAP over HTTP) または SOAP/JMS (SOAP over JMS) のいずれかのトランスポート・プロトコルを使用できます。SOAP メッセージの伝達に使うトランスポート (HTTP または JMS) に関係なく、Web サービス・バインディングは常に要求/応答の対話を同期的に処理します。

- HTTP

  HTTP バインディングを使用すれば、SOAP 以外のメッセージが使用されている場合、または直接 HTTP アクセスが必要な場合に、HTTP プロトコルを使用して外部サービスにアクセスすることができます。このバインディングは、HTTP モデルに基づく Web サービス (GET、PUT、DELETE などのよく知られた HTTP インターフェース操作を使用するサービス) を処理するときに使用されます。

- Enterprise JavaBeans (EJB)

  EJB バインディングにより、SCA コンポーネントは、Java EE サーバー上で実行される Java EE ビジネス・ロジックで提供されるサービスと対話できるようになります。

- EIS

  EIS (エンタープライズ情報システム) バインディングを JCA リソース・アダプターと一緒に使用すると、エンタープライズ情報システム上のサービスにアクセスしたり、またはサービスを EIS で使用可能にすることができます。

- JMS バインディング

  Java Message Service (JMS)、汎用 JMS、および WebSphere MQ JMS (MQ JMS) バインディングは、メッセージ・キューを経由した非同期通信が信頼性の維持に欠かせない場合に、メッセージング・システムとの対話に使用されます。

  これらの JMS バインディングのいずれかを使用するエクスポートは、キューにメッセージが到着するのを監視し、応答 (該当する場合) を応答キューに非同期に送信します。これらの JMS バインディングのいずれかを使用するインポートは、メッセージを構築して JMS キューに送信し、キューに応答 (該当する場合) が到着するのを監視します。

  – JMS

    JMS バインディングを使用すれば、WebSphere 組み込み JMS プロバイダーにアクセスすることができます。

– 汎用 JMS

汎用 JMS バインディングを使用すれば、IBM 以外のベンダーのメッセージング・システムにアクセスできます。

– MQ JMS

MQ JMS バインディングを使用すれば、WebSphere MQ メッセージング・システムの JMS サブセットにアクセスできます。ご使用のアプリケーションにとって JMS サブセットの機能で十分な場合、このバインディングを使用します。

- MQ

WebSphere MQ バインディングを使用すれば、MQ ネイティブ・アプリケーションと通信できるようになるため、これらのアプリケーションがサービス指向アーキテクチャー・フレームワークに組み込まれ、MQ 固有のヘッダー情報にアクセスできるようになります。MQ のネイティブ機能を使用する必要がある場合に、このバインディングを使用します。

エクスポートおよびインポート・バインディングの概要

エクスポートによってインテグレーション・モジュール内のサービスが外部クライアントに使用可能となり、インポートによってインテグレーション・モジュール内の SCA コンポーネントが外部サービスを呼び出すことができるようになります。エクスポートまたはインポートに関連付けられたバインディングは、プロトコル・メッセージとビジネス・オブジェクトとの関係を指定します。また、操作と障害を選択する方法も指定します。

エクスポートによる情報のフロー

エクスポートは、エクスポートがワイヤリングされたコンポーネントを対象とする要求を受信します。要求の受信に使用される特定のトランスポート (例えば、HTTP) は、エクスポートに関連付けられたバインディングによって決まります。

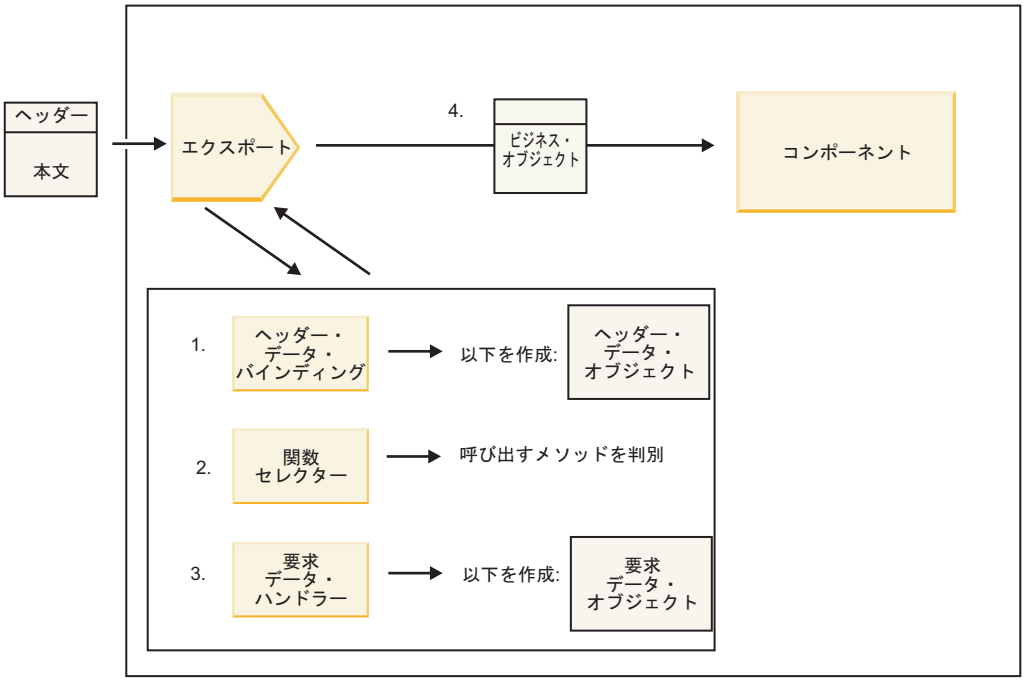


図 3. エクスポートを経由したコンポーネントへの要求のフロー

エクスポートが要求を受信するとき、以下の一連のイベントが発生します。

1. WebSphere MQ バインディングの場合に限り、ヘッダー・データ・バインディングがプロトコル・ヘッダーをヘッダー・データ・オブジェクトに変換します。
2. 関数セレクターがプロトコル・メッセージからネイティブ・メソッド名を判別します。ネイティブ・メソッド名は、エクスポート構成により、エクスポートのインターフェース上の操作名にマップされています。
3. メソッドの要求データ・ハンドラーまたは要求データ・バインディングは、要求を要求ビジネス・オブジェクトに変換します。
4. エクスポートは要求ビジネス・オブジェクトでコンポーネント・メソッドを呼び出します。
   - HTTP エクスポート・バインディング、Web サービス・エクスポート・バインディング、および EJB エクスポート・バインディングは、SCA コンポーネントに対して同期呼び出しを行います。
   - JMS、汎用 JMS、MQ JMS、および WebSphere MQ エクスポート・バインディングは、SCA コンポーネントに対して非同期呼び出しを行います。

エクスポートは、コンテキスト伝搬が有効になっている場合、プロトコル経由で受信したヘッダーおよびユーザー・プロパティーを伝搬できます。その後、エクスポートにワイヤリングされたコンポーネントは、これらのヘッダーおよびユーザー・プロパティーにアクセスできます。詳しくは、WebSphere Integration Developer インフォメーション・センターのトピック『伝搬』を参照してください。

これが両方向操作の場合、コンポーネントは応答を返します。

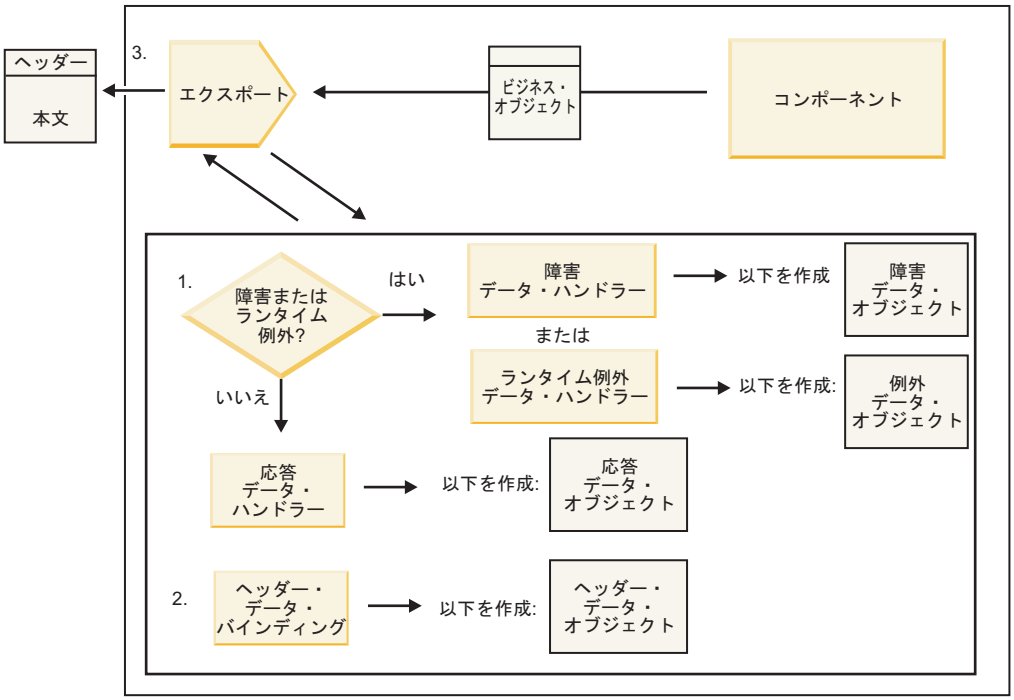


図 4. エクスポートを経由して戻る応答のフロー

以下の一連のステップが発生します。

1. 通常の応答メッセージがエクスポート・バインディングによって受信されると、メソッドの応答データ・ハンドラーまたは応答データ・バインディングは、ビジネス・オブジェクトを応答に変換します。

   応答が障害の場合、メソッドの障害データ・ハンドラーまたは障害データ・バインディングは障害を障害応答に変換します。

   HTTP エクスポート・バインディングにのみ適用されることですが、応答がランタイム例外の場合、ランタイム例外データ・ハンドラー (構成済みの場合) が呼び出されます。
2. WebSphere MQ バインディングの場合に限り、ヘッダー・データ・バインディングがヘッダー・データ・オブジェクトをプロトコル・ヘッダーに変換します。
3. エクスポートはサービス応答をトランスポート経由で送信します。

インポートによる情報のフロー

コンポーネントはインポートを使用して、モジュール外部のサービスに要求を送信します。要求は、関連付けられたバインディングによって決まる特定のトランスポートを使用して送信されます。

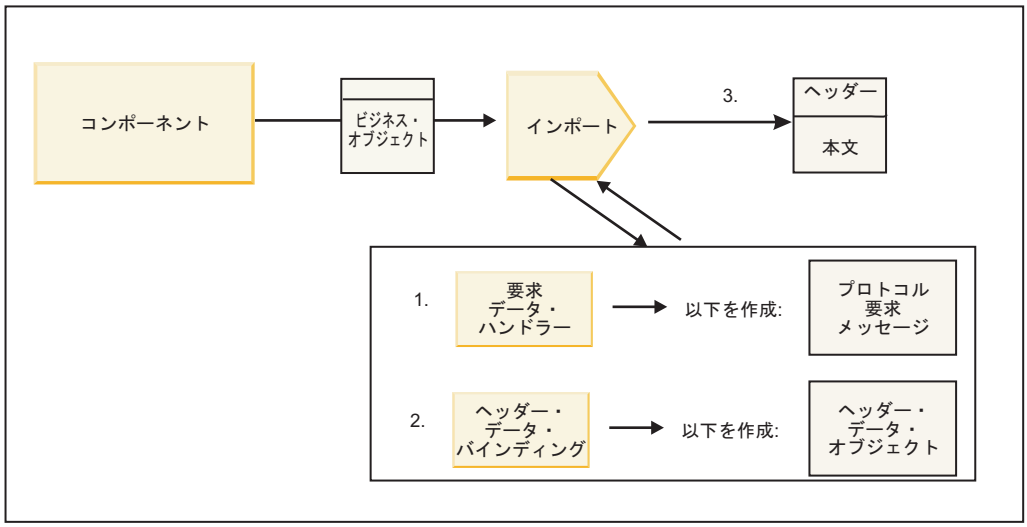


図 5. インポートを経由したコンポーネントからサービスへのフロー

コンポーネントは、要求ビジネス・オブジェクトでインポートを呼び出します。

注:

- HTTP インポート・バインディング、Web サービス・インポート・バインディング、および EJB インポート・バインディングは、呼び出し側コンポーネントが同期方式で呼び出す必要があります。
- JMS、汎用 JMS、MQ JMS、および WebSphere MQ インポート・バインディングは、非同期方式で呼び出す必要があります。

コンポーネントがインポートを呼び出した後、以下の一連のイベントが発生します。

1. メソッドの要求データ・ハンドラーまたは要求データ・バインディングは、要求ビジネス・オブジェクトをプロトコル要求メッセージに変換します。
2. WebSphere MQ バインディングの場合に限り、メソッドのヘッダー・データ・バインディングがヘッダー・ビジネス・オブジェクトをプロトコル・ヘッダーに設定します。
3. インポートはトランスポートを経由して、サービス要求でサービスを呼び出します。

これが両方向操作の場合、サービスは応答を戻し、以下の一連のステップが発生します。

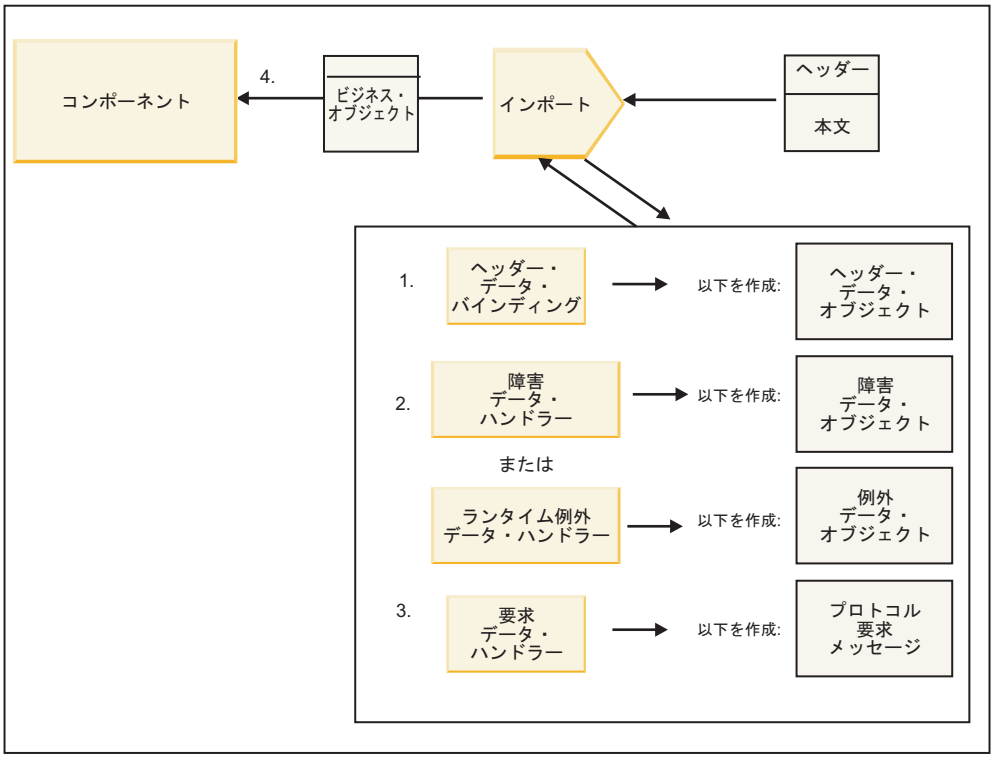


図 6. インポートを経由して戻る応答のフロー

1. WebSphere MQ バインディングの場合に限り、ヘッダー・データ・バインディングがプロトコル・ヘッダーをヘッダー・データ・オブジェクトに変換します。
2. 応答が障害かどうかを判別します。
   - 応答が障害の場合、障害セレクターは障害を検査して、どの WSDL 障害にマップするかを決定します。次にメソッドの障害データ・ハンドラーは、障害を障害応答に変換します。
   - 応答がランタイム例外の場合、ランタイム例外データ・ハンドラー (構成済みの場合) が呼び出されます。
3. メソッドの応答データ・ハンドラーまたは応答データ・バインディングは、応答を応答ビジネス・オブジェクトに変換します。
4. インポートは応答ビジネス・オブジェクトをコンポーネントに返します。

エクスポートおよびインポート・バインディング構成

エクスポートおよびインポート・バインディングの重要な側面の 1 つはデータ・フォーマットの変換です。これによって、データをネイティブ・ワイヤー・フォーマットからビジネス・オブジェクトにマップ (非直列化) する方法、またはビジネス・オブジェクトからネイティブ・ワイヤー・フォーマットにマップ (直列化) する方法が指定されます。 エクスポートに関連したバインディングでは、データに対して実行すべき操作を指示する関数セレクターも指定できます。エクスポートまたはインポートに関連したバインディングでは、処理中に発生する障害の処理方法を指示することができます。

さらに、トランスポート固有情報をバインディングに指定できます。例えば HTTP バインディングでは、エンドポイント URL を指定します。HTTP バインディングの場合、トランスポート固有の情報は『HTTP インポート・バインディングの生成』トピックと『HTTP エクスポート・バインディングの生成』に記載されています。インフォメーション・センターで、その他のバインディングに関する情報を見つけることもできます。

インポートおよびエクスポートでのデータ・フォーマット変換

WebSphere Integration Developer でエクスポートまたはインポート・バインディングを構成するときには、構成プロパティーの 1 つとしてバインディングで使用するデータ・フォーマットを指定します。

- エクスポート・バインディングでは、クライアント・アプリケーションが要求を SCA コンポーネントに送信し、応答を SCA コンポーネントから受信しますが、この場合、ネイティブ・データのフォーマットを指示します。システムはこのフォーマットに基づいて、ネイティブ・データからビジネス・オブジェクト (SCA コンポーネントによって使用される) に変換し、逆にビジネス・オブジェクトをネイティブ・データ (クライアント・アプリケーションへの応答) に変換するための適切なデータ・ハンドラーまたはデータ・バインディングを選択します。
- インポート・バインディングでは、SCA コンポーネントが要求をモジュール外部のサービスに送信し、応答をモジュール外部のサービスから受信しますが、この場合、ネイティブ・データのフォーマットを指示します。システムはこのフォーマットに基づいて、ビジネス・オブジェクトからネイティブ・データへ、およびその逆に変換するための適切なデータ・ハンドラーまたはデータ・バインディングを選択します。

IBM Business Process Manager には事前定義された一連のデータ・フォーマットと、そのフォーマットをサポートする対応データ・ハンドラーまたはデータ・バインディングが提供されています。また、固有のカスタム・データ・ハンドラーを作成して、そのデータ・ハンドラーのデータ・フォーマットを登録することもできます。詳しくは、WebSphere Integration Developer インフォメーション・センターのトピック『データ・ハンドラーの開発』を参照してください。

- データ・ハンドラーはプロトコルに中立的で、1 つのフォーマットから別のフォーマットにデータを変換します。IBM Business Process Manager では、データ・ハンドラーはネイティブ・データ (XML、CSV、および COBOL) からビジネス・オブジェクトへ、およびビジネス・オブジェクトからネイティブ・データへの変換を行うのが標準的です。これらはプロトコルに中立的なため、いろいろなエクスポートおよびインポート・バインディングで同じデータ・ハンドラーを再利用することができます。例えば、同一の XML データ・ハンドラーを HTTP エクスポートまたはインポート・バインディング、あるいは JMS エクスポートまたはインポート・バインディングで使用できます。
- データ・バインディングもネイティブ・データからビジネス・オブジェクトへの変換 (およびその逆) を行いますが、これらはプロトコルに固有です。例えば、HTTP データ・バインディングは HTTP エクスポートまたは HTTP インポート・バインディングのみで使用できます。データ・ハンドラーとは異なり、HTTP データ・バインディングは MQ エクスポートまたは MQ インポート・バインディングでは再利用できません。

  注: 3 つの HTTP データ・バインディング (HTTPStreamDataBindingSOAP、HTTPStreamDataBindingXML、および HTTPServiceGatewayDataBinding) は、IBM Business Process Manager バージョン 7.0 では推奨されていません。可能な場合はデータ・ハンドラーを使用してください。

前に述べたように、必要に応じてカスタム・データ・ハンドラーを作成することができます。また、カスタム・データ・バインディングを作成することもできますが、カスタム・データ・ハンドラーは複数のバインディングで使用できるため、カスタム・データ・ハンドラーを作成することをお勧めします。

データ・ハンドラー:

データ・ハンドラーは、プロトコルに中立的な方法でデータ・フォーマットを変換するために、エクスポートおよびインポート・バインディングに対して構成されます。いくつかのデータ・ハンドラーが製品の一部として提供されていますが、必要に応じて固有のデータ・ハンドラーを作成することもできます。データ・ハンドラーをエクスポート・バインディングまたはインポート・バインディングに関連付けるレベルは 2

つあります。一方のレベルでは、エクスポートまたはインポートのインターフェースにあるすべての操作に関連付けることが可能です。もう一方のレベルでは、要求または応答に対する特定の操作に関連付けることができます。

事前定義データ・ハンドラー

使用するデータ・ハンドラーの指定は、IBM Integration Designer を使用して行います。

使用できるように事前定義されたデータ・ハンドラーを以下の表で示します。表には、各データ・ハンドラーがインバウンドおよびアウトバウンドのデータを変換する方法についても記載されています。

注: 注記されていない限り、これらのデータ・ハンドラーは、JMS、汎用 JMS、MQ JMS、WebSphere MQ、および HTTP バインディングで使用できます。
詳しくは、Integration Designer インフォメーション・センターのトピック『データ・ハンドラー』を参照してください。

表 11. 事前定義データ・ハンドラー

データ・ハンドラー	ネイティブ・データからビジネス・オブジェクトへ	ビジネス・オブジェクトからネイティブ・データへ
ATOM	ATOM フィードを ATOM フィード・ビジネス・オブジェクトに解析します。	ATOM フィード・ビジネス・オブジェクトを ATOM フィードに直列化します。
Delimited	区切り文字で区切られているデータをビジネス・オブジェクトに解析します。	ビジネス・オブジェクトを区切り文字で区切られているデータ (CSV など) に直列化します。
Fixed Width	固定長データをビジネス・オブジェクトに解析します。	ビジネス・オブジェクトを固定長データに直列化します。
WTX により処理	データ・フォーマット変換を WebSphere Transformation Extender (WTX) に委任します。WTX マップ名は、データ・ハンドラーから派生します。	データ・フォーマット変換を WebSphere Transformation Extender (WTX) に委任します。WTX マップ名は、データ・ハンドラーから派生します。
WTX 呼び出し側により処理	データ・フォーマット変換を WebSphere Transformation Extender (WTX) に委任します。WTX マップ名は、ユーザーが提供します。	データ・フォーマット変換を WebSphere Transformation Extender (WTX) に委任します。WTX マップ名は、ユーザーが提供します。
JAXB	Java Architecture for XML Binding (JAXB) 仕様で定義されているマッピング・ルールを使用して、Java Bean をビジネス・オブジェクトに直列化します。	JAXB 仕様で定義されているマッピング・ルールを使用して、ビジネス・オブジェクトを Java Bean に非直列化します。
JAXWS **注:** JAXWS データ・ハンドラーは、EJB バインディングでのみ使用できます。	Java API for XML Web Services (JAX-WS) 仕様で定義されているマッピング・ルールを使用して、応答 Java オブジェクトまたは例外 Java オブジェクトを応答ビジネス・オブジェクトに変換するために EJB バインディングによって使用されます。	JAX-WS 仕様で定義されているマッピング・ルールを使用して、ビジネス・オブジェクトを発信 Java メソッド・パラメーターに変換するために EJB バインディングによって使用されます。
JSON	JSON データをビジネス・オブジェクトに解析します。	ビジネス・オブジェクトを JSON データに直列化します。

表 11. 事前定義データ・ハンドラー (続き)

データ・ハンドラー	ネイティブ・データからビジネス・オブジェクトへ	ビジネス・オブジェクトからネイティブ・データへ
ネイティブ本体	ネイティブのバイト、テキスト、マップ、ストリーム、またはオブジェクトを 5 つの基本ビジネス・オブジェクト (テキスト、バイト、マップ、ストリーム、またはオブジェクト) のいずれかに解析します。	5 つの基本ビジネス・オブジェクトをバイト、テキスト、マップ、ストリーム、またはオブジェクトに変換します。
SOAP	SOAP メッセージ (およびヘッダー) をビジネス・オブジェクトに解析します。	ビジネス・オブジェクトを SOAP メッセージに直列化します。
XML	XML データをビジネス・オブジェクトに解析します。	ビジネス・オブジェクトを XML データに直列化します。
UTF8XMLDataHandler	UTF-8 にエンコードされた XML データをビジネス・オブジェクトに解析します。	メッセージの送信時に、ビジネス・オブジェクトを UTF-8 にエンコードされた XML データに直列化します。

データ・ハンドラーの作成

データ・ハンドラーの作成に関する詳細情報については、Integration Designer インフォメーション・センターのトピック『データ・ハンドラーの開発』を参照してください。

データ・バインディング:

データ・バインディングは、データ・フォーマットを変換するために、エクスポートおよびインポート・バインディングに対して構成されます。データ・バインディングは、それぞれのプロトコルに固有です。いくつかのデータ・バインディングが製品の一部として提供されていますが、必要に応じて固有のデータ・バインディングを作成することもできます。データ・バインディングをエクスポートまたはインポート・バインディングに関連付けるレベルは 2 つあります。一方のレベルでは、エクスポートまたはインポートのインターフェースにあるすべての操作に関連付けることが可能です。もう一方のレベルでは、要求または応答に対する特定の操作に関連付けることができます。

使用するデータ・バインディングの指定または固有のデータ・バインディングの作成は、WebSphere Integration Developer を使用して行います。データ・バインディングの作成については、WebSphere Integration Developer インフォメーション・センターの『JMS、MQ JMS、および汎用 JMS データ・バインディングの概要』セクションを参照してください。

JMS バインディング

次の表は、以下のバインディングで使用できるデータ・バインディングの一覧です。

- JMS バインディング
- 汎用 JMS バインディング
- WebSphere MQ JMS バインディング

また表には、データ・バインディングが実行するタスクの説明も記載されています。

表 12. JMS バインディング用の事前定義データ・バインディング

データ・バインディング	ネイティブ・データからビジネス・オブジェクトへ	ビジネス・オブジェクトからネイティブ・データへ
直列化 Java オブジェクト	Java 直列化オブジェクトをビジネス・オブジェクト (WSDL で入力または出力タイプとしてマップされているビジネス・オブジェクト) に変換します。	ビジネス・オブジェクトを JMS オブジェクト・メッセージの Java 直列化オブジェクトに直列化します。
ラップされたバイト	着信 JMS バイト・メッセージからバイトを抽出して、これらを JMSBytesBody ビジネス・オブジェクトにラップします。	JMSBytesBody ビジネス・オブジェクトからバイトを抽出して、これらを発信 JMS バイト・メッセージにラップします。
ラップされたマップ項目	着信 JMS マップ・メッセージの各項目の名前、値、およびタイプ情報を抽出して、MapEntry ビジネス・オブジェクトのリストを作成します。次に、そのリストを JMSMapBody ビジネス・オブジェクトにラップします。	JMSMapBody ビジネス・オブジェクトの MapEntry リストから名前、値、およびタイプ情報を抽出して、発信 JMS マップ・メッセージ内に対応する項目を作成します。
ラップされたオブジェクト	着信 JMS オブジェクト・メッセージからオブジェクトを抽出して、これを JMSObjectBody ビジネス・オブジェクトにラップします。	JMSObjectBody ビジネス・オブジェクトからオブジェクトを抽出して、これを発信 JMS オブジェクト・メッセージにラップします。
ラップされたテキスト	着信 JMS テキスト・メッセージからテキストを抽出して、これを JMSTextBody ビジネス・オブジェクトにラップします。	JMSTextBody ビジネス・オブジェクトからテキストを抽出して、これを発信 JMS テキスト・メッセージにラップします。

WebSphere MQ バインディング

次の表は、WebSphere MQ で使用できるデータ・バインディングの一覧と、データ・バインディングが実行するタスクを説明したものです。

表 13. WebSphere MQ バインディング用の事前定義データ・バインディング

データ・バインディング	ネイティブ・データからビジネス・オブジェクトへ	ビジネス・オブジェクトからネイティブ・データへ
直列化 Java オブジェクト	着信メッセージからの Java 直列化オブジェクトをビジネス・オブジェクト (WSDL で入力または出力タイプとしてマップされているビジネス・オブジェクト) に変換します。	ビジネス・オブジェクトを出力メッセージの Java 直列化オブジェクトに変換します。
ラップされたバイト	構造化されていない MQ バイト・メッセージからバイトを抽出して、これらを JMSBytesBody ビジネス・オブジェクトにラップします。	JMSBytesBody ビジネス・オブジェクトからバイトを抽出して、バイトを構造化されていない発信 MQ バイト・メッセージにラップします。
ラップされたテキスト	構造化されていない MQ テキスト・メッセージからテキストを抽出して、これを JMSTextBody ビジネス・オブジェクトにラップします。	JMSTextBody ビジネス・オブジェクトからテキストを抽出して、これを構造化されていない MQ テキスト・メッセージにラップします。

表 13. WebSphere MQ バインディング用の事前定義データ・バインディング (続き)

データ・バインディング	ネイティブ・データからビジネス・オブジェクトへ	ビジネス・オブジェクトからネイティブ・データへ
ラップされたストリーム項目	着信 JMS ストリーム・メッセージの各項目の名前およびタイプ情報を抽出して、StreamEntry ビジネス・オブジェクトのリストを作成します。次に、そのリストを JMSStreamBody ビジネス・オブジェクトにラップします。	JMSStreamBody ビジネス・オブジェクトの StreamEntry リストから名前およびタイプ情報を抽出して、発信 JMSStreamMessage 内に対応する項目を作成します。

47 ページの表 13 にリストされたデータ・バインディングの他、WebSphere MQ はヘッダー・データ・バインディングも使用します。詳しくは、WebSphere Integration Developer インフォメーション・センターを参照してください。

HTTP バインディング

次の表は、HTTP で使用できるデータ・バインディングの一覧と、データ・バインディングが実行するタスクを説明したものです。

表 14. HTTP バインディング用の事前定義データ・バインディング

データ・バインディング	ネイティブ・データからビジネス・オブジェクトへ	ビジネス・オブジェクトからネイティブ・データへ
ラップされたバイト	着信 HTTP メッセージの本体からバイトを抽出して、これらを HTTPBytes ビジネス・オブジェクトにラップします。	HTTPBytes ビジネス・オブジェクトからバイトを抽出して、これらを発信 HTTP メッセージの本体に追加します。
ラップされたテキスト	着信 HTTP メッセージの本体からテキストを抽出して、これを HTTPText ビジネス・オブジェクトにラップします。	HTTPText ビジネス・オブジェクトからテキストを抽出して、これを発信 HTTP メッセージの本体に追加します。

エクスポート・バインディングでの関数セレクター

関数セレクターは、要求メッセージのデータに対して実行すべき操作を指示するために使用します。関数セレクターは、エクスポート・バインディングの一部として構成されます。

インターフェースを公開する SCA エクスポートを例として考えてみます。このインターフェースには、作成および更新の 2 つの操作が組み込まれています。エクスポートの JMS バインディングは、キューから読み取られます。

メッセージがキューに到着すると、エクスポートには関連するデータが渡されますが、ワイヤリングされたコンポーネントで、エクスポートのインターフェースからどの操作を呼び出すべきかの情報は渡されません。その操作は、関数セレクターとエクスポート・バインディング構成によって決まります。

関数セレクターはネイティブの関数名 (メッセージを送信したクライアント・システムの関数名) を返します。ネイティブの関数名は、エクスポートが関連付けられたインターフェースの操作名または関数名にマップされます。例えば以下の図で、関数セレクターは着信メッセージからネイティブの関数名 (CRT) を返し、ネイティブの関数名は作成操作にマップされ、ビジネス・オブジェクトは作成操作を持つ SCA コンポーネントに送信されます。

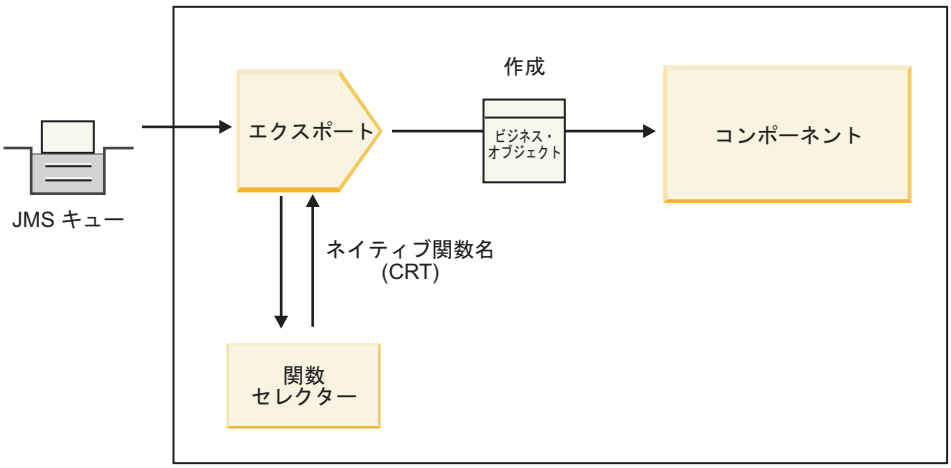


図 7. 関数セレクター

インターフェースに 1 つの操作しかない場合には、関数セレクターを指定する必要はありません。

複数の関数セレクターがプリパッケージされています。以降のセクションで、これらの関数セレクターをリストします。

JMS バインディング

次の表は、以下のバインディングで使用できる関数セレクターの一覧です。

- JMS バインディング
- 汎用 JMS バインディング
- WebSphere MQ JMS バインディング

表 15. JMS バインディング用の事前定義関数セレクター

関数セレクター	説明
単純な JMS データ・バインディング用 JMS 関数セレクター	メッセージの JMSType プロパティーを使用して操作名を選択します。
JMS ヘッダー・プロパティー関数セレクター	ヘッダーの JMS ストリング・プロパティー、TargetFunctionName の値を返します。
JMS サービス・ゲートウェイ関数セレクター	クライアントによって設定された JMSReplyTo プロパティーを調べ、要求が片方向操作と両方向操作のどちらであるかを判別します。

WebSphere MQ バインディング

次の表は、WebSphere MQ バインディングで使用できる関数セレクターの一覧です。

表 16. WebSphere MQ バインディング用の事前定義関数セレクター

関数セレクター	説明
MQ handleMessage 関数セレクター	handleMessage を値として返します。この値は、エクスポート・メソッド・バインディングを使用してインターフェースの操作名にマップされています。
MQ が JMS デフォルト関数セレクターを使用	MQRFH2 ヘッダーのフォルダーの TargetFunctionName プロパティーからネイティブ操作を読み取ります。

表 16. WebSphere MQ バインディング用の事前定義関数セレクター (続き)

関数セレクター	説明
MQ がメッセージ本体のフォーマットをネイティブ関数として使用	最後のヘッダーの Format フィールドを検索し、そのフィールドをストリングとして返します。
MQ タイプ関数セレクター	MQRFH2 ヘッダーで検出した Msd、Set、Type、および Format プロパティーを含む URL を取得して、エクスポート・バインディング内にメソッドを作成します。
MQ サービス・ゲートウェイ関数セレクター	MQMD ヘッダー内の MsgType プロパティーを使用して操作名を判断します。

HTTP バインディング

次の表は、HTTP バインディングで使用できる関数セレクターの一覧です。

表 17. HTTP バインディング用の事前定義関数セレクター

関数セレクター	説明
TargetFunctionName ヘッダーに基づく HTTP 関数セレクター	クライアントからの TargetFunctionName HTTP ヘッダー・プロパティーを使用して、実行時にエクスポートから呼び出す操作を判断します。
URL および HTTP メソッドに基づく HTTP 関数セレクター	クライアントからの HTTP メソッドに付加された URL の相対パスを使用して、エクスポートに定義されたネイティブ操作を判断します。
操作名が設定された URL に基づく HTTP サービス・ゲートウェイ関数セレクター	要求 URL に「operationMode = oneWay」が付加されている場合、その URL に基づいて呼び出すメソッドを判断します。

注: IBM Integration Designer を使用して固有の関数セレクターを作成することもできます。関数セレクターの作成に関する詳細情報については、IBM Integration Designer インフォメーション・センターを参照してください。例えば、WebSphere MQ バインディング用の関数セレクターの作成に関する説明は、『MQ 関数セレクターの概要』に記載されています。

障害の処理

ご使用のインポート・バインディングおよびエクスポート・バインディングについて、障害データ・ハンドラーを指定することによって、処理中に発生する障害 (ビジネス例外など) を処理するように構成できます。障害データ・ハンドラーは、3 つのレベルでセットアップできます。具体的には、障害データ・ハンドラーを障害に関連付けるか、操作に関連付けるか、またはバインディングを使用するすべての操作を対象に関連付けます。

障害データ・ハンドラーは障害データを処理し、エクスポート・バインディングまたはインポート・バインディングによって送信される正しいフォーマットに変換します。

- エクスポート・バインディングでは、障害データ・ハンドラーはコンポーネントから送信された例外ビジネス・オブジェクトを、クライアント・アプリケーションによって使用できる応答メッセージに変換します。
- インポート・バインディングでは、障害データ・ハンドラーはサービスから送信された障害データまたは応答メッセージを、SCA コンポーネントによって使用できる例外ビジネス・オブジェクトに変換します。

インポート・バインディングの場合、バインディングは障害セレクターを呼び出します。障害セレクターは、応答メッセージが通常応答、ビジネス障害、またはランタイム例外のいずれであるかを判別します。

障害データ・ハンドラーは、特定の障害、操作、およびバインディングを使用するすべての操作について指定することができます。

- 障害データ・ハンドラーが 3 つすべてのレベルで設定されている場合、特定の障害に関連付けられたデータ・ハンドラーが呼び出されます。
- 障害データ・ハンドラーが操作およびバインディングのレベルで設定されている場合、操作に関連付けられたデータ・ハンドラーが呼び出されます。

IBM Integration Designer での障害処理の指定には、2 つのエディターが使用されます。インターフェース・エディターは、操作に障害があるかどうかを示すのに使用されます。このインターフェースでバインディングが生成された後、プロパティー・ビューのエディターによって、障害をどのように処理するかを構成できます。詳しくは、IBM Integration Designer インフォメーション・センターのトピック『障害セレクター』を参照してください。

エクスポート・バインディングでの障害の処理方法:

クライアント・アプリケーションからの要求の処理中に障害が発生したとき、エクスポート・バインディングは障害情報をクライアントに返すことができます。障害を処理してクライアントに返す方法を指定するようにエクスポート・バインディングを構成します。

エクスポート・バインディングの構成は、IBM Integration Designer を使用して行います。

要求を処理するとき、クライアントは要求でエクスポートを呼び出し、エクスポートは SCA コンポーネントを呼び出します。要求の処理中に、SCA コンポーネントはビジネス応答を返すか、またはサービス・ビジネス例外あるいはサービス・ランタイム例外を throw することができます。これが発生すると、エクスポート・バインディングは例外を障害メッセージに変換して、クライアントに送信します。これについて以下の図で示し、その次のセクションで説明します。

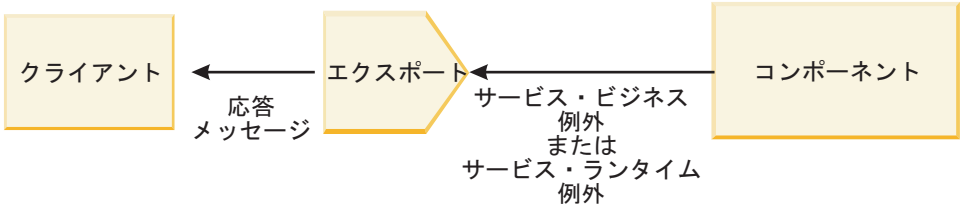


図 8. 障害情報をエクスポート・バインディングを経由してコンポーネントからクライアントに送信する方法

障害を処理するカスタムのデータ・ハンドラーまたはデータ・バインディングを作成することができます。

ビジネス障害

ビジネス障害とは、処理中に発生するビジネス・エラーまたはビジネス例外です。

createCustomer 操作を持つ以下のインターフェースについて考えてみます。この操作には、CustomerAlreadyExists および MissingCustomerId の 2 つのビジネス障害が定義されています。

操作

操作およびパラメーター

	名前	タイプ
createCustomer		
入力	input	CustomerInfo
出力	output	CustomerInfo
障害	Customer Already Exists	Customer Already ExistsBO
障害	MissingCustomerID	MissingCustomerIDBO

図 9. 2 つの障害を持つインターフェース

この例で、クライアントが顧客を作成する要求を (この SCA コンポーネントに) 送信したときにその顧客が既に存在する場合、コンポーネントは CustomerAlreadyExists 障害をエクスポートに throw します。エクスポートはこのビジネス障害を、呼び出し側のクライアントに伝搬して返す必要があります。エクスポートはこれを行うために、エクスポート・バインディングに設定された障害データ・ハンドラーを使用します。

ビジネス障害がエクスポート・バインディングによって受信されると、以下の処理が行われます。

1. バインディングは、どの障害を障害データ・ハンドラーを呼び出して処理させるかを決定します。サービス・ビジネス例外に障害名が含まれている場合、その障害について設定されているデータ・ハンドラーが呼び出されます。サービス・ビジネス例外が障害名を含まない場合、障害名は障害タイプをマッチングすることによって導き出されます。
2. バインディングは、サービス・ビジネス例外からのデータ・オブジェクトを使用して、障害データ・ハンドラーを呼び出します。
3. 障害データ・ハンドラーは障害データ・オブジェクトを応答メッセージに変換し、これをエクスポート・バインディングに返します。
4. エクスポートは応答メッセージをクライアントに返します。

サービス・ビジネス例外に障害名が含まれている場合、その障害について設定されているデータ・ハンドラーが呼び出されます。サービス・ビジネス例外が障害名を含まない場合、障害名は障害タイプをマッチングすることによって導き出されます。

ランタイム例外

ランタイム例外とは、要求の処理中に SCA アプリケーション内で発生する、ビジネス障害に対応しない例外のことです。ビジネス障害とは異なり、ランタイム例外はインターフェースで定義されません。

シナリオによっては、これらのランタイム例外をクライアント・アプリケーションに伝搬して、クライアント・アプリケーションが適切なアクションを実行できるようにしたい場合もあります。

例えば、クライアントが顧客を作成する要求を (SCA コンポーネントに) 送信したとき、要求の処理中に権限エラーが発生した場合、コンポーネントはランタイム例外を throw します。このランタイム例外は呼び出し側のクライアントに戻すよう伝搬させて、クライアントが権限に関して適切なアクションを実行できるようにする必要があります。これはランタイム例外データ・ハンドラーをエクスポート・バインディングに構成することによって実現できます。

注: ランタイム例外データ・ハンドラーは、HTTP バインディングでのみ構成できます。

ランタイム例外の処理は、ビジネス障害の処理と似ています。ランタイム例外データ・ハンドラーが設定されている場合、以下の処理が実行されます。

1. エクスポート・バインディングはサービス・ランタイム例外について適切なデータ・ハンドラーを呼び出します。
2. データ・ハンドラーは障害データ・オブジェクトを応答メッセージに変換し、これをエクスポート・バインディングに返します。
3. エクスポートは応答メッセージをクライアントに返します。

障害処理およびランタイム例外処理はオプションです。障害またはランタイム例外を呼び出し側のクライアントに伝搬させたくない場合、障害データ・ハンドラーまたはランタイム例外データ・ハンドラーを構成しないでください。

インポート・バインディングでの障害の処理方法:

コンポーネントはインポートを使用して、要求をモジュール外部のサービスに送信します。 要求の処理中に障害が発生すると、サービスはインポート・バインディングに障害を返します。障害を処理してコンポーネントに返す方法は、インポート・バインディングを構成して指定することができます。

インポート・バインディングの構成は、IBM Integration Designer を使用して行います。障害データ・ハンドラー (または障害データ・バインディング) を指定できますが、障害セレクターを指定することもできます。

障害データ・ハンドラー

要求を処理するサービスがインポート・バインディングに障害情報を送信するときには、例外の形式または障害データが含まれる応答メッセージが使用されます。

インポート・バインディングはサービス例外または応答メッセージをサービス・ビジネス例外またはサービス・ランタイム例外に変換します。これについて以下の図で示し、その次のセクションで説明します。

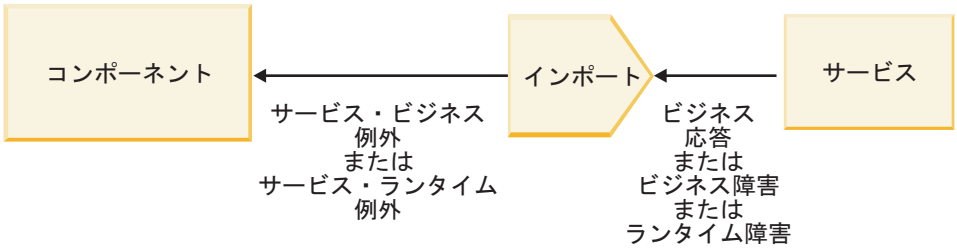


図 10. 障害情報をインポートを経由してサービスからコンポーネントに送信する方法

障害を処理するカスタムのデータ・ハンドラーまたはデータ・バインディングを作成することができます。

障害セレクター

インポート・バインディングを構成するとき、障害セレクターを指定することができます。障害セレクターは、インポートの応答が実際の応答か、ビジネス例外か、またはランタイム障害であるかを判別します。またこれは、応答の本体またはヘッダーからネイティブの障害名を判別します。ネイティブの障害名は、バインディング構成によって、関連するインターフェースの障害名にマップされます。

JMS、MQ JMS、汎用 JMS、WebSphere MQ、および HTTP インポートでは、プリパッケージされた 2 つのタイプの障害セレクターを使用できます。

表 18. プリパッケージされている障害セレクター

障害セレクター・タイプ	説明
ヘッダー・ベース	着信応答メッセージのヘッダーに基づいて、応答メッセージがビジネス障害、ランタイム例外、または通常のメッセージのいずれであるかを判別します。
SOAP	応答 SOAP メッセージが通常応答、ビジネス障害、またはランタイム例外のいずれであるかを判別します。

以下は、ヘッダー・ベースの障害セレクターおよび SOAP 障害セレクターの例です。

- ヘッダー・ベースの障害セレクター

  アプリケーションで、着信メッセージがビジネス障害であることを示す場合、着信メッセージにはビジネス障害に対応する以下の 2 つのヘッダーが含まれます。

  ```
  Header name = FaultType, Header value = Business
  Header name = FaultName, Header value = <user defined native fault name>
  ```

  アプリケーションで、着信応答メッセージがランタイム例外であることを示す場合、着信メッセージには以下に示す 1 つのヘッダーが含まれます。

  ```
  Header name = FaultType, Header value = Runtime
  ```

- SOAP 障害セレクター

  ビジネス障害は、SOAP メッセージの一部として送信できます。それには、SOAP メッセージに以下のカスタム SOAP ヘッダーを設定します。この場合の障害名は、「CustomerAlreadyExists」です。

  ```
  <ibmSoap:BusinessFaultName
  xmlns:ibmSoap="http://www.ibm.com/soap">CustomerAlreadyExists
  <ibmSoap:BusinessFaultName>
  ```

障害セレクターはオプションです。障害セレクターを指定しないと、インポート・バインディングは応答のタイプを判別できません。そのため、バインディングは応答をビジネス応答として扱い、応答データ・ハンドラーまたはデータ・バインディングを呼び出します。

カスタム障害セレクターを作成することができます。カスタム障害セレクターを作成するステップは、IBM Integration Designer インフォメーション・センターのトピック『カスタム障害セレクターの開発』にあります。

ビジネス障害

要求の処理にエラーがあると、ビジネス障害が発生することがあります。例えば、顧客を作成する要求を送信したときにその顧客が既に存在する場合、サービスはインポート・バインディングにビジネス例外を送信します。

ビジネス例外がバインディングによって受信された場合、処理ステップは障害セレクターがバインディングにセットアップされているかどうかに依存します。

- 障害セレクターが設定されていない場合、バインディングは応答データ・ハンドラーまたは応答データ・バインディングを呼び出します。
- 障害セレクターが設定されている場合、以下の処理が実行されます。
  1. インポート・バインディングは障害セレクターを呼び出して、応答がビジネス障害、ビジネス応答、またはランタイム障害のいずれであるかを判別します。

2. 応答がビジネス障害である場合、インポート・バインディングは、ネイティブ障害名を提供するために障害セレクターを呼び出します。
3. インポート・バインディングは、障害セレクターによって返されたネイティブ障害名に対応する WSDL 障害を判別します。
4. インポート・バインディングは、この WSDL 障害用に構成された障害データ・ハンドラーを判別します。
5. インポート・バインディングは障害データについて、この障害データ・ハンドラーを呼び出します。
6. 障害データ・ハンドラーは障害データをデータ・オブジェクトに変換し、これをインポート・バインディングに返します。
7. インポート・バインディングはデータ・オブジェクトおよび障害名を使用して、サービス・ビジネス例外オブジェクトを構成します。
8. インポートはサービス・ビジネス例外オブジェクトをコンポーネントに返します。

ランタイム例外

ランタイム例外は、サービスとの通信に問題があるときに発生することがあります。ランタイム例外の処理は、ビジネス例外の処理と似ています。障害セレクターが設定されている場合、以下の処理が実行されます。

1. インポート・バインディングは例外データについて適切なランタイム例外データ・ハンドラーを呼び出します。
2. ランタイム例外データ・ハンドラーは例外データをサービス・ランタイム例外オブジェクトに変換し、これをインポート・バインディングに返します。
3. インポートはサービス・ランタイム例外オブジェクトをコンポーネントに返します。

SCA モジュールとオープン SCA サービスの間のインターオペラビリティー

IBM WebSphere Application Server V7.0 Feature Pack for Service Component Architecture (SCA) は、オープン SCA 仕様に基づいてアプリケーションを構築するための、単純であるが強力なプログラミング・モデルを提供します。IBM Business Process Managerの SCA モジュールは、インポートおよびエクスポート・バインディングを使用して、Rational® Application Developer 環境で開発され、WebSphere Application Server Feature Pack for Service Component Architecture によりホストされるオープン SCA サービスと相互運用します。

SCA アプリケーションは、インポート・バインディングを使ってオープン SCA アプリケーションを呼び出します。SCA アプリケーションは、エクスポート・バインディングを使ってオープン SCA アプリケーションからの呼び出しを受け取ります。サポートされるバインディングのリストを、57 ページの『相互運用可能なバインディングを利用したサービスの呼び出し』に示します。

SCA モジュールからのオープン SCA サービスの呼び出し

IBM Integration Designer を使って開発した SCA アプリケーションは、Rational Application Developer 環境で開発されたオープン SCA アプリケーションを呼び出すことができます。このセクションでは、SCA インポート・バインディングを使用して SCA モジュールからオープン SCA サービスを呼び出す例を示します。

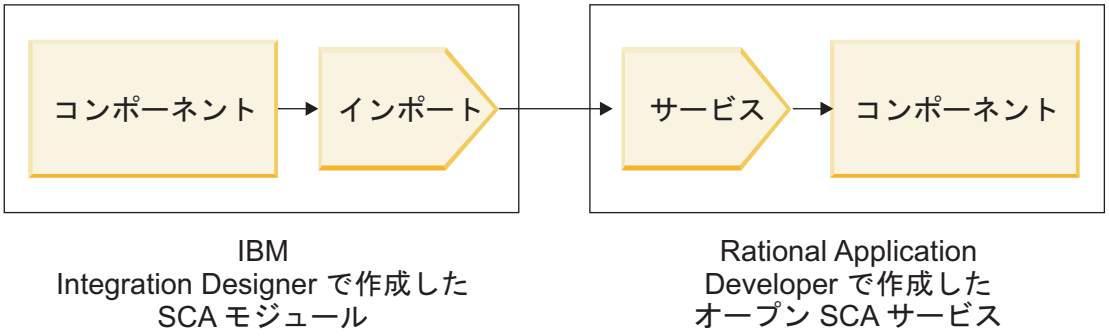


図 11. オープン SCA サービスを呼び出す SCA モジュールのコンポーネント

オープン SCA サービスの呼び出しには特別の構成は必要ありません。

SCA インポート・バインディングを使ってオープン SCA サービスに接続するには、オープン SCA サービスのコンポーネント名とサービス名をインポート・バインディングに指定します。

1. オープン SCA コンポジットからターゲットのコンポーネントおよびサービスの名前を取得するには、以下の手順を実行します。
   a. 「**ウィンドウ**」 > 「**ビューの表示 (Show View)**」 > 「**プロパティー**」をクリックして、「**プロパティー**」タブを開きます。
   b. コンポーネントとサービスが含まれるコンポジット・ダイアグラムをダブルクリックして、コンポジット・エディターを開きます。例えば、**customer** という名前のコンポーネントの場合、コンポジット・ダイアグラムは **customer.composite_diagram** となります。
   c. ターゲット・コンポーネントをクリックします。
   d. 「**プロパティー**」タブの「**名前**」フィールドに表示されるターゲット・コンポーネントの名前をメモします。
   e. このコンポーネントに関連付けられたサービス・アイコンをクリックします。
   f. 「**プロパティー**」タブの「**名前**」フィールドに表示されるサービスの名前をメモします。
2. オープン SCA サービスに接続するように IBM Business Process Manager インポートを構成するには、以下の手順を実行します。
   a. IBM Integration Designer で、オープン SCA サービスに接続する SCA インポートの「**プロパティー**」タブにナビゲートします。
   b. 「**モジュール名**」フィールドに、ステップ 1d でメモしたコンポーネント名を入力します。
   c. 「**エクスポート名**」フィールドに、ステップ 1f でメモしたサービス名を入力します。
   d. Ctrl+S キーを押して、作業内容を保存します。

オープン SCA サービスからの SCA モジュールの呼び出し

Rational Application Developer 環境で開発されたオープン SCA アプリケーションは、IBM Integration Designer を使って開発された SCA アプリケーションを呼び出すことができます。このセクションでは、オープン SCA サービスから (SCA エクスポート・バインディングを使用して) SCA モジュールを呼び出す例を示します。

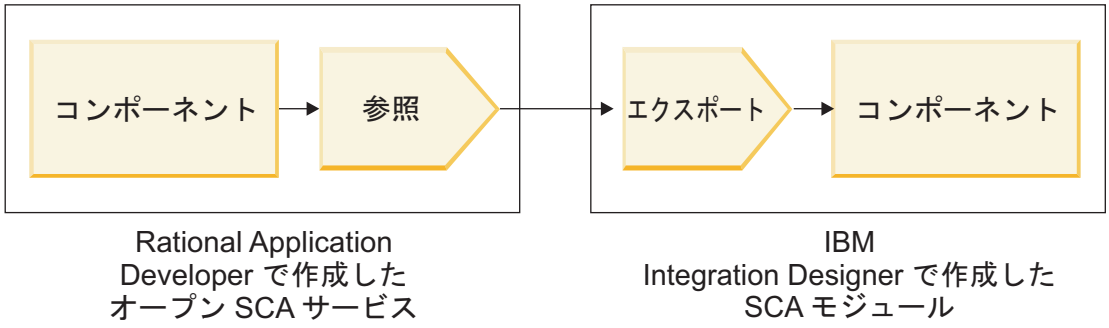


図 12. SCA モジュールのコンポーネントを呼び出すオープン SCA サービス

オープン SCA 参照バインディングにより SCA コンポーネントに接続するには、モジュール名およびエクスポート名を指定します。

1. ターゲットのモジュールおよびエクスポートの名前を取得するには、以下の手順を実行します。
   a. IBM Integration Designer でモジュールをダブルクリックして、アセンブリー・エディターでそのモジュールを開きます。
   b. エクスポートをクリックします。
   c. 「**プロパティー**」タブの「**名前**」フィールドに表示されるエクスポートの名前をメモします。
2. IBM Business Process Manager モジュールおよびエクスポートに接続するオープン SCA 参照を構成します。
   a. Rational Application Developer で、コンポーネントとサービスが含まれるコンポジット・ダイアグラムをダブルクリックして、コンポジット・エディターを開きます。
   b. コンポーネント参照の参照アイコンをクリックして、「**プロパティー**」タブに参照プロパティーを表示します。
   c. ページの左側にある「**バインディング**」タブをクリックします。
   d. 「**バインディング**」をクリックし、「**追加**」をクリックします。
   e. 「**SCA**」バインディングを選択します。
   f. 「**URI**」フィールドに IBM Business Process Manager モジュール名を入力し、それに続いてスラッシュ (『/』) とエクスポート名 (ステップ 1c で確認した名前) を入力します。
   g. 「**OK**」をクリックします。
   h. Ctrl+S キーを押して、作業内容を保存します。

相互運用可能なバインディングを利用したサービスの呼び出し

オープン SCA サービスとの相互運用のために次のバインディングがサポートされています。

- SCA バインディング

  IBM Business Process Manager で、SCA モジュールが SCA インポート・バインディングを使ってオープン SCA サービスを呼び出す場合は、以下の呼び出しスタイルがサポートされます。

  – 非同期 (片方向)
  – 同期 (要求/応答)

  SCA インポート・インターフェースおよびオープン SCA サービス・インターフェースは、Web Services Interoperability (WS-I) に準拠した WSDL インターフェースを使用する必要があります。

SCA バインディングは、トランザクションおよびセキュリティー・コンテキストの伝搬をサポートしている点に留意してください。

- SOAP1.1/HTTP または SOAP1.2/HTTP プロトコルを用いた Web サービス (JAX-WS) バインディング

  SCA インポート・インターフェースおよびオープン SCA サービス・インターフェースは、Web Services Interoperability (WS-I) に準拠した WSDL インターフェースを使用する必要があります。

  また、次のサービス品質がサポートされています。

  – Web Services Atomic Transaction
  – Web サービスのセキュリティー
- EJB バインディング

  EJB バインディングが使用されているときは、Java インターフェースを使って SCA モジュールとオープン SCA サービス間の相互作用を定義します。

  EJB バインディングは、トランザクションおよびセキュリティー・コンテキストの伝搬をサポートしている点に留意してください。
- JMS バインディング

  SCA インポート・インターフェースおよびオープン SCA サービス・インターフェースは、Web Services Interoperability (WS-I) に準拠した WSDL インターフェースを使用する必要があります。

  次の JMS プロバイダーがサポートされています。

  – WebSphere Platform Messaging (JMS バインディング)
  – WebSphere MQ (MQ JMS バインディング)

注: ビジネス・グラフは SCA バインディングの中で相互運用できません。このため、WebSphere Application Server Feature Pack for Service Component Architecture との相互運用で使用されているインターフェースではサポートされません。

バインディング・タイプ

インポートおよびエクスポートと共にプロトコル固有の「バインディング」を使用して、データをモジュールの内部または外部に移送する方法を指定します。

適切なバインディングの選択

アプリケーションのニーズに合わせて使用できるさまざまなバインディングがあります。

WebSphere Integration Developer で使用できるバインディングには、さまざまな選択肢があります。このリストは、アプリケーションのニーズにより適しているバインディングのタイプを識別する助けになります。

以下の要因が当てはまる場合は、*SCA* バインディングを検討してください。

- すべてのサービスが WebSphere Integration Developer モジュールに組み込まれている (外部サービスがない)。
- 互いに直接対話する複数の SCA モジュールに機能を分けたい。
- モジュールが密結合されている場合

以下の要因が当てはまる場合は、*Web* サービス・バインディングを検討してください。

- インターネットを介して外部サービスにアクセスする必要があるか、インターネットを介してサービスを提供する必要がある。
- サービスが疎結合になっている。
- 同期通信が推奨される場合 (特定のサービスからの要求が、別のサービスからの応答を待機することがある場合)
- アクセスする外部サービスまたは提供するサービスのプロトコルが SOAP/HTTP または SOAP/JMS である。

以下の要因が当てはまる場合は、*HTTP* バインディングを検討してください。
- インターネットを介して外部サービスにアクセスする必要があるか、インターネットを介してサービスを提供する必要があり、HTTP モデルに基づく他の Web サービスを処理している (GET、PUT、DELETE などのよく知られた HTTP インターフェース操作を使用している)。
- サービスが疎結合になっている。
- 同期通信が推奨される場合 (特定のサービスからの要求が、別のサービスからの応答を待機することがある場合)

以下の要因が当てはまる場合は、*EJB* バインディングを検討してください。
- バインディングの対象が、それ自体が EJB であるインポート済みサービスか、EJB クライアントによるアクセスが必要なインポート済みサービスである。
- インポートされたサービスが疎結合されている場合
- ステートフル EJB 対話が不要な場合
- 同期通信が推奨される場合 (特定のサービスからの要求が、別のサービスからの応答を待機することがある場合)

以下の要因が当てはまる場合は、*EIS* バインディングを検討してください。
- リソース・アダプターを使用して EIS システム上のサービスにアクセスする必要がある。
- 非同期データ伝送よりも同期データ伝送が望ましい。

以下の要因が当てはまる場合は、*JMS* バインディングを検討してください。

注: JMS バインディングには、いくつかのタイプがあります。JMS を使用して SOAP メッセージを交換する必要がある場合は、SOAP/JMS プロトコルを使用した Web サービス・バインディングを検討してください。 60 ページの『Web サービス・バインディング』を参照してください。
- メッセージング・システムにアクセスする必要がある。
- サービスが疎結合になっている。
- 同期データ伝送よりも非同期データ伝送が望ましい。

次の点が当てはまる場合は、*汎用 JMS* バインディングを考慮してください。
- IBM 以外のベンダーのメッセージング・システムにアクセスする必要がある。
- サービスが疎結合になっている。
- パフォーマンスよりも信頼性を重視している場合、つまり、同期データ伝送よりも非同期データ伝送が好まれる場合

以下の要因が当てはまる場合は、*MQ* バインディングを検討してください。
- WebSphere MQ メッセージング・システムにアクセスする必要があり、MQ ネイティブ機能を使用する必要がある。

- サービスが疎結合になっている。
- パフォーマンスよりも信頼性を重視している場合、つまり、同期データ伝送よりも非同期データ伝送が好まれる場合

以下の要因が当てはまる場合は、*MQ JMS* バインディングを検討してください。

- WebSphere MQ メッセージング・システムにアクセスする必要があるが、JMS コンテキスト内でアクセス可能である (アプリケーションで、機能の JMS サブセットを利用すれば十分である)。
- サービスが疎結合になっている。
- パフォーマンスよりも信頼性を重視している場合、つまり、同期データ伝送よりも非同期データ伝送が好まれる場合

SCA バインディング

Service Component Architecture (SCA) バインディングを使用すると、サービスは他のモジュール内の他のサービスと通信できるようになります。SCA バインディングを持つインポートを使用すると、別の SCA モジュール内のサービスにアクセスできるようになります。SCA バインディングを持つエクスポートを使用すると、サービスを他のモジュールに提供することができます。

SCA モジュールでのインポートおよびエクスポートに対して SCA バインディングを生成および構成するには、WebSphere Integration Developer を使用します。

モジュールが同じサーバー上で実行されている場合、または同じクラスターにデプロイされている場合、最も簡単に使用できる最も速いバインディングは SCA バインディングです。

SCA バインディングが組み込まれているモジュールをサーバーにデプロイすると、管理コンソールを使用して、バインディングに関する情報を表示したり、バインディングの選択済みプロパティーを変更したり (インポート・バインディングの場合) することができます。

Web サービス・バインディング

Web サービス・バインディングとは、Service Component Architecture (SCA) コンポーネントと Web サービスとの間でメッセージを送信する手段です。

Web サービス・バインディングの概要:

Web サービス・インポート・バインディングを使用すると、Service Component Architecture (SCA) コンポーネントから外部の Web サービスを呼び出すことができます。 Web サービス・エクスポート・バインディングを使用すると、SCA コンポーネントを Web サービスとしてクライアントに公開できます。

Web サービス・バインディングを使用することにより、相互運用可能な SOAP メッセージおよびサービス品質 (Qos) を使用して外部サービスにアクセスできます。

SCA モジュールでのインポートおよびエクスポートに関して Web サービス・バインディングを生成および構成するには、Integration Designer を使用します。以下のタイプの Web サービス・バインディングが使用可能です。

- SOAP1.2/HTTP および SOAP1.1/HTTP

  これらのバインディングは、Java API for XML Web Services (JAX-WS) (Web サービスを作成するための Java プログラミング API) に基づいています。

  – Web サービスが SOAP 1.2 仕様に準拠している場合は、SOAP1.2/HTTP を使用します。
  – Web サービスが SOAP 1.1 仕様に準拠している場合は、SOAP1.1/HTTP を使用します。

重要: Web サービス (JAX-WS) バインディングでアプリケーションをデプロイする場合は、ターゲット・サーバーに対して「**必要に応じてコンポーネントを始動 (Start components as needed)**」オプションを選択しないでください。詳しくは、 70 ページの『サーバー構成の確認』を参照してください。

これらのいずれかのバインディングを選択すると、SOAP メッセージで添付ファイルを送信できます。

Web サービス・バインディングは、標準 SOAP メッセージと連動します。ただし、いずれかの Web サービス JAX-WS バインディングを使用すると、SOAP メッセージを解析または作成する方法をカスタマイズすることができます。例えば、SOAP メッセージで非標準エレメントを処理したり、SOAP メッセージに追加の処理を適用したりすることができます。バインディングの構成時に、このような処理を SOAP メッセージに対して実行するカスタム・データ・ハンドラーを指定します。

Web サービス (JAX-WS) バインディングでは、ポリシー・セットを使用できます。ポリシー・セットとは、サービス品質 (QoS) を規定するポリシー・タイプのコレクションのことです。例えば、WSAddressing ポリシー・セットを使用すると、トランスポートに中立的な方法で、Web サービスとメッセージを一律にアドレス指定することができます。バインディングのポリシー・セットを選択するには、 Integration Designer を使用します。

注: Security Assertion Markup Language (SAML) ポリシー・セットを使用する場合は、 67 ページの『SAML ポリシー・セットのインポート』の説明に従って、いくつかの追加構成を行う必要があります。

- SOAP1.1/HTTP

  Java API for XML-based RPC (JAX-RPC) に基づく SOAP エンコード・メッセージを使用する Web サービスを作成する場合は、このバインディングを使用します。

- SOAP1.1/JMS

  Java Message Service (JMS) 宛先を使用して SOAP メッセージを送信または受信するには、このバインディングを使用します。

SOAP メッセージの伝達に使うトランスポート (HTTP または JMS) に関係なく、Web サービス・バインディングは常に要求/応答の対話を同期的に処理します。サービス・プロバイダー上で呼び出しを実行するスレッドは、プロバイダーから応答を受け取るまでブロックされます。この呼び出しスタイルについて詳しくは、『同期呼び出し』を参照してください。

重要: 以下の Web サービス・バインディングの組み合わせは、同じモジュール内のエクスポートに対しては使用できません。これらのエクスポート・バインディングを複数使用してコンポーネントを公開する必要がある場合は、それぞれを別個のモジュールに分けてから、SCA バインディングを使用してそれらのモジュールをコンポーネントに接続する必要があります。

- JAX-RPC を使用する SOAP 1.1/JMS と、JAX-RPC を使用する SOAP 1.1/HTTP
- JAX-RPC を使用する SOAP 1.1/HTTP と、JAX-WS を使用する SOAP 1.1/HTTP
- JAX-RPC を使用する SOAP 1.1/HTTP と、JAX-WS を使用する SOAP 1.2/HTTP

Web サービス・バインディングが組み込まれている SCA モジュールをサーバーにデプロイすると、管理コンソールを使用してバインディングに関する情報を表示するか、バインディングの選択済みプロパティーを変更することができます。

注: Web サービスを使用すると、複数のアプリケーションが、標準のサービス記述を使用したり、交換するメッセージに標準形式を使用したりすることによって、相互運用できるようになります。例えば、Web サービスのインポート・バインディングおよびエクスポート・バインディングは、Web Services

Enhancements (WSE) Version 3.5 および Windows Communication Foundation (WCF) Version 3.5 for Microsoft .NET を使用して実装されたサービスと相互運用できます。このようなサービスと相互協調処理する場合、必ず以下のようにする必要があります。

- Web サービス・エクスポートへのアクセスに使用される Web サービス記述言語 (WSDL) ファイルに、インターフェースの各操作に対する空ではない SOAP アクション値が含まれていること。
- メッセージを Web サービス・エクスポートに送信するときに、Web サービス・クライアントが SOAPAction ヘッダーまたは wsa:Action ヘッダーのいずれかを設定する。

SOAP ヘッダーの伝搬:

SOAP メッセージを処理するときは、受信したメッセージの特定の SOAP ヘッダー情報にアクセスすること、 SOAP ヘッダーを持つメッセージが特定の値とともに送信されていることを確認すること、または SOAP ヘッダーがモジュールを通過できるようにすることが必要になる場合があります。

Integration Designer で Web サービス・バインディングを構成する場合、 SOAP ヘッダーを伝搬するかどうかを指定できます。

- エクスポートで要求を受信する場合、またはインポートで応答を受信する場合は、 SOAP ヘッダー情報にアクセスして、モジュール内のロジックをヘッダー値に基づくようにし、それらのヘッダーを変更できます。
- エクスポートから要求を送信する場合、またはインポートから応答を送信する場合は、 SOAP ヘッダーをこれらのメッセージに含めることができます。

伝搬される SOAP ヘッダーの形式、およびそれが存在するかどうかは、インポートまたはエクスポートで構成されたポリシー・セットの影響を受ける場合があります (63 ページの表 19 を参照)。

インポートまたはエクスポートの SOAP ヘッダーの伝搬を構成するには、 (Integration Designer の「プロパティー」ビューから) 「**プロトコル・ヘッダーの伝搬**」タブを選択し、必要なオプションを選択します。

WS-Addressing ヘッダー

WS-Addressing ヘッダーは、Web サービス (JAX-WS) バインディングで伝搬できます。

WS-Addressing ヘッダーを伝搬する場合は、以下の情報に注意してください。

- WS-Addressing ヘッダーの伝搬を有効にした場合、以下の状況では、ヘッダーはモジュール内に伝搬されます。
  - 要求がエクスポートで受信される場合
  - 応答がインポートで受信される場合
- WS-Addressing ヘッダーは、IBM Business Process Manager からのアウトバウンド・メッセージ内には伝搬されません (つまり、要求がインポートから送信される場合、または応答がエクスポートから送信される場合、ヘッダーは伝搬されません)。

WS-Security ヘッダー

WS-Security ヘッダーは、Web サービス (JAX-WS) バインディング、および Web サービス (JAX-RPC) バインディングの両方で伝搬できます。

Web サービスの WS-Security 仕様には、メッセージ保全性、メッセージ機密性、および単一メッセージ認証を通じて保護品質を提供するための SOAP メッセージングの拡張が記述されています。これらのメカニズムを使用することにより、さまざまなセキュリティー・モデルと暗号化テクノロジーに対応できます。

WS-Security ヘッダーを伝搬する場合は、以下の情報に注意してください。

- WS-Security ヘッダーの伝搬を有効にした場合、以下の状況では、ヘッダーはモジュールを通過して伝搬されます。
  - 要求がエクスポートで受信される場合
  - 要求がインポートから送信される場合
  - 応答がインポートで受信される場合
- デフォルトでは、応答がエクスポートから送信される場合は、ヘッダーは伝搬されません。ただし、JVM プロパティー **WSSECURITY.ECHO.ENABLED** に **true** を設定すると、応答がエクスポートから送信される場合にヘッダーは伝搬されます。この場合、要求パスの WS-Security ヘッダーが変更されないときは、要求から応答に WS-Security ヘッダーが自動的にエコーされることがあります。
- 要求に対してインポートから送信される SOAP メッセージ、または応答に対してエクスポートから送信される SOAP メッセージの厳格な形式は、最初に受信した SOAP メッセージとは正確に一致しない場合があります。このため、すべてのデジタル署名は無効になると想定する必要があります。送信メッセージでデジタル署名が必要な場合は、適切なセキュリティー・ポリシー・セットを使用してデジタル署名を設定し、受信メッセージのデジタル署名に関連する WS-Security ヘッダーをモジュール内で削除する必要があります。

WS-Security ヘッダーを伝搬するには、WS-Security スキーマをアプリケーション・モジュールに組み込む必要があります。スキーマを組み込む手順については、64 ページの『WS-Security スキーマのアプリケーション・モジュールへの組み込み』を参照してください。

ヘッダーの伝搬方法

ヘッダーの伝搬方法は、インポート・バインディングまたはエクスポート・バインディングのセキュリティー・ポリシー設定に依存します。表 19 を参照してください。

表 19. セキュリティー・ヘッダーを渡す方法

	セキュリティー・ポリシーを使用しないエクスポート・バインディング	**セキュリティー・ポリシーを使用したエクスポート・バインディング**
セキュリティー・ポリシーを使用しないインポート・バインディング	セキュリティー・ヘッダーは、そのままモジュールを通過します。暗号化解除はされません。 ヘッダーは、受信時と同じ形式でアウトバウンドに送信されます。 デジタル署名は、無効になる場合があります。	セキュリティー・ヘッダーは、暗号化解除され、モジュールを通過します。このとき、署名の検証と認証が実行されます。 暗号化解除されたヘッダーは、アウトバウンドに送信されます。 デジタル署名は、無効になる場合があります。
セキュリティー・ポリシーを使用したインポート・バインディング	セキュリティー・ヘッダーは、そのままモジュールを通過します。暗号化解除はされません。 ヘッダーは、インポートに伝搬してはなりません。そうしないと、重複のためにエラーが発生します。	セキュリティー・ヘッダーは、暗号化解除され、モジュールを通過します。このとき、署名の検証と認証が実行されます。 ヘッダーは、インポートに伝搬してはなりません。そうしないと、重複のためにエラーが発生します。

適切なポリシー・セットをエクスポート・バインディングとインポート・バインディングで構成してください。なぜなら、これにより、サービス・プロバイダーの構成または QoS 要件に対する変更と、サービス要求元が分離されるからです。標準 SOAP ヘッダーをモジュールで可視にすると、モジュールの処理 (ロギングやトレースなど) に影響を与えることができます。モジュールを通過して受信メッセージから送信メッセージに SOAP ヘッダーを伝搬すると、モジュールを分離する利点が減少します。

標準ヘッダー (WS-Security ヘッダーなど) が通常生成されるポリシー・セットがインポートまたはエクスポートに関連付けられているときは、標準ヘッダーをインポート (要求時) またはエクスポート (応答時) に伝搬してはなりません。そうしないと、ヘッダーの重複によりエラーが発生します。代わりに、ヘッダーを明示的に削除するか、もしくはプロトコル・ヘッダーが伝搬されないようにインポート・バインディングまたはエクスポート・バインディングを構成する必要があります。

SOAP ヘッダーへのアクセス

SOAP ヘッダーを含むメッセージが Web サービスのインポートまたはエクスポートから受信されると、それらのヘッダーは、サービス・メッセージ・オブジェクト (SMO) のヘッダー・セクションに配置されます。ヘッダー情報にアクセスするには、『SMO の SOAP ヘッダー情報へのアクセス』を参照してください。

WS-Security スキーマのアプリケーション・モジュールへの組み込み

以下の手順では、スキーマをアプリケーション・モジュールに組み込むステップを説明します。

- Integration Designer が実行中のコンピューターがインターネットにアクセスできる場合は、以下のステップを実行します。
  1. ビジネス・インテグレーション・パースペクティブで、プロジェクトの「**依存関係**」を選択します。
  2. 「**事前定義リソース**」を展開し、「**WS セキュリティー 1.0 スキーマ・ファイル**」または「**WS セキュリティー 1.1 スキーマ・ファイル**」を選択して、スキーマをモジュールにインポートします。
  3. プロジェクトをクリーンにし再ビルドします。
- Integration Designer を実行中のコンピューターがインターネットにアクセスできない場合は、インターネットにアクセスできる 2 番目のコンピューターにスキーマをダウンロードします。その後、Integration Designer を実行中のコンピューターにスキーマをコピーします。
  1. インターネットにアクセスできるコンピューターから、以下の手順でリモート・スキーマをダウンロードします。
     a. 「**ファイル**」 > 「**インポート**」 > 「**ビジネス・インテグレーション**」 > 「**WSDL および XSD**」をクリックします。
     b. 「**リモート WSDL または XSD ファイル**」を選択します。
     c. 以下のスキーマをインポートします。

        http://www.w3.org/2003/05/soap-envelope/

        http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd

        http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd
  2. インターネットにアクセスできないコンピューターにスキーマをコピーします。
  3. インターネットにアクセスできないコンピューターから、以下の手順でスキーマをインポートします。
     a. 「**ファイル**」 > 「**インポート**」 > 「**ビジネス・インテグレーション**」 > 「**WSDL および XSD**」をクリックします。
     b. 「**ローカル WSDL (Local WSDL)**」または「**XSD ファイル (XSD file)**」を選択します。

4. oasis-wss-wssecurity_secext-1.1.xsd のスキーマの場所を以下の手順で変更します。
   a. *workplace_location*/*module_name*/StandardImportFilesGen/oasis-wss-wssecurity-secext-1.1.xsd でスキーマを開きます。
   b. 変更前:

      ```
      <xs:import namespace='http://www.w3.org/2003/05/soap-envelope'
      schemaLocation='http://www.w3.org/2003/05/soap-envelope/'/>
      ```

      次のように変更:

      ```
      <xs:import namespace='http://www.w3.org/2003/05/soap-envelope'
      schemaLocation='../w3/_2003/_05/soap_envelope.xsd'/>
      ```

   c. 変更前:

      ```
      <xs:import namespace='http://www.w3.org/2001/04/xmlenc#'
      schemaLocation='http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd'/>
      ```

      次のように変更:

      ```
      <xs:import namespace='http://www.w3.org/2001/04/xmlenc#'
      schemaLocation='../w3/tr/_2002/rec_xmlenc_core_20021210/xenc-schema.xsd'/>
      ```

5. oasis-200401-wss-wssecurity-secext-1.0.xsd のスキーマの場所を以下の手順で変更します。
   a. *workplace_location*/*module_name*/StandardImportFilesGen/oasis-200401-wss-wssecurity-secext-1.0.xsd でスキーマを開きます。
   b. 変更前:

      ```
      <xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
      schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>
      ```

      次のように変更:

      ```
      <xsd:import namespace="http://www.w3.org/2000/09/xmldsig#"
      schemaLocation="../w3/tr/_2002/rec_xmldsig_core_20020212/xmldsig-core-schema.xsd"/>
      ```

6. プロジェクトをクリーンにし再ビルドします。

トランスポート・ヘッダーの伝搬:

SOAP メッセージを処理する場合は、受信メッセージの特定のトランスポート・ヘッダー情報にアクセスしたり、トランスポート・ヘッダーを持つメッセージが特定の値とともに送信されているかどうかを確認したり、トランスポート・ヘッダーがモジュールを通過できるように設定したりすることが必要になる場合があります。

Integration Designer で Web サービス・バインディングを構成する場合、トランスポート・ヘッダーが伝搬されるように指定することができます。

- エクスポートで要求を受信する場合、またはインポートで応答を受信する場合、トランスポート・ヘッダー情報にアクセスすることができます。これにより、モジュール内のロジックをヘッダー値に基づくようにし、対象のヘッダーを変更することができます。
- エクスポートから応答を送信する場合、またはインポートから要求を送信する場合は、トランスポート・ヘッダーをこれらのメッセージに含めることができます。

ヘッダーの伝搬の指定

インポートまたはエクスポートに対してトランスポート・ヘッダーの伝搬を構成するには、以下の手順を実行します。

1. Integration Designer の「プロパティー」ビューから、「**バインディング**」 > 「**伝搬**」を選択します。

2. トランスポート・ヘッダーの伝搬について必要なオプションを設定します。

注: トランスポート・ヘッダーの伝搬は、デフォルトでは無効になっています。トランスポート・ヘッダーの伝搬をデプロイできるのは、バージョン 7.0.0.3 以降のランタイム環境の場合だけです。また、バージョン 7.0.0.3 の場合、トランスポート・ヘッダーの伝搬は HTTP トランスポート・ヘッダーだけに制限されることにも注意してください。

トランスポート・ヘッダーの伝搬を有効にすると、受信メッセージからモジュール全体にヘッダーが伝搬されます。この場合、明示的にヘッダーを削除しない限り、ヘッダーは同じスレッドの後続の呼び出しにも使用されます。

注: Web サービス (JAX-RPC) バインディングを使用している場合は、トランスポート・ヘッダーを伝搬できません。

ヘッダー情報へのアクセス

受信メッセージに対してトランスポート・ヘッダーの伝搬が有効になっている場合は、(ユーザー定義ヘッダーを含む) すべてのトランスポート・ヘッダーがサービス・メッセージ・オブジェクト (SMO) で可視になります。各ヘッダーを異なる値に設定したり、ヘッダーを新規作成したりすることができます。ただし、設定した値のチェックや妥当性検査は実行されないため、適切ではないヘッダーや間違ったヘッダーを使用すると、Web サービス・ランタイムの問題が発生する可能性があります。

HTTP ヘッダーの設定については、以下の情報を考慮してください。

- Web サービス・エンジン用に予約されたヘッダーに対する変更は、アウトバウンド・メッセージ内では反映されません。例えば、HTTP バージョンまたは HTTP メソッドと、Content-Type、Content-Length、SOAPAction の各ヘッダーは、Web サービス・エンジン用に予約されています。
- ヘッダー値が数値の場合は、(ストリングではなく) 数値を直接設定する必要があります。例えば、**Max-Forwards = Max-Forwards: 5** ではなく **Max-Forwards = 5**、**Age = Age: 300** ではなく **Age = 300** を使用します。
- 要求メッセージのサイズが 32 KB 未満の場合は、Web サービス・エンジンによって Transfer-Encoding ヘッダーが削除され、代わりに Content-Length ヘッダーがメッセージの固定サイズに設定されます。
- Content-language は、応答パス上の WAS.channel.http によってリセットされます。
- Upgrade に対して無効な設定を行うと、500 エラーが発生します。
- 以下のヘッダーは、Web サービス・エンジンによって予約済みの値をカスタマー設定に付加します。
  - User-Agent
  - Cache-Control
  - Pragma
  - Accept
  - 接続

ヘッダー情報には、以下のいずれかの方法でアクセスできます。

- メディエーション・プリミティブを使用して SMO 構造にアクセスする

  メディエーション・プリミティブの使用法については、『関連情報』にあるリンクを参照してください。
- コンテキスト・サービス SPI を使用する

以下のサンプル・コードでは、コンテキスト・サービスから HTTP トランスポート・ヘッダーが読み込まれます。

```
HeadersType headerType = ContextService.INSTANCE.getHeaders();
HTTPHeaderType httpHeaderType = headerType.getHTTPHeader();
List HTTPHeader  httpHeaders = httpHeaderType.getHeader();
if(httpHeaders!=null){
 for(HTTPHeader httpHeader: httpHeaders){
   String httpHeadername = httpHeader.getName();
   String httpHeaderValue = httpHeader.getValue();
     }
}
List PropertyType  properties = headerType.getProperties();
if(properties!=null){
  for(PropertyType property: properties){
   String propertyName = property.getName();
   String propertyValue = property.getValue().toString();
     }
}
```

トラブルシューティング

変更されたヘッダーの送信時に問題が発生した場合は、 Integration Designer の TCP/IP モニターなどのツールを使用して、TCP/IP メッセージを代行受信できます。TCP/IP モニターにアクセスするには、「設定」ページから「**実行/デバッグ (Run/Debug)**」 > 「**TCP/IP モニター (TCP/IP Monitor)**」を選択してください。

次の JAX-WS エンジン・トレースを使用してヘッダー値を表示することもできます:
org.apache.axis2.*=all: com.ibm.ws.websvcs.*=all:

Web サービス (JAX-WS) バインディングの操作:

アプリケーションで Web サービス (JAX-WS) バインディングを使用する場合は、Security Assertion Markup Language (SAML) の Quality of Service (QoS) をバインディングに追加できます。最初に管理コンソールを使用して、ポリシー・セットをインポートする必要があります。また、管理コンソールを使用して、Web サービス (JAX-WS) バインディングで使用するためにサーバーが正しく構成されていることを確認することもできます。

SAML ポリシー・セットのインポート:

Security Assertion Markup Language (SAML) は、ユーザー ID とセキュリティー属性情報を交換するための XML ベースの OASIS 標準です。Web サービス (JAX-WS) バインディングを Integration Designer で構成する際には、SAML ポリシー・セットを指定できます。最初に、IBM Business Process Manager 管理コンソールを使用して SAML ポリシー・セットを使用可能にします。この処理により、SAML ポリシー・セットを Integration Designer にインポートできるようになります。

通常の場合、SAML ポリシー・セットは次のプロファイル構成ディレクトリーに格納されています。

profile_root/config/templates/PolicySets

この手順を実行する前に、ポリシー・セットが格納されている以下のディレクトリーがプロファイル構成ディレクトリー内に存在することを確認します。

- SAML11 Bearer WSHTTPS default
- SAML20 Bearer WSHTTPS default
- SAML11 Bearer WSSecurity default

- SAML20 Bearer WSSecurity default
- SAML11 HoK Public WSSecurity default
- SAML20 HoK Public WSSecurity default
- SAML11 HoK Symmetric WSSecurity default
- SAML20 HoK Symmetric WSSecurity default
- Username WSHTTPS default

これらのディレクトリーがプロファイル構成ディレクトリーにない場合は、以下の場所からプロファイル構成ディレクトリーにコピーしてください。

app_server_root/profileTemplates/default/documents/config/templates/PolicySets

ポリシー・セットを管理コンソールにインポートし、Integration Designer で使用可能にするポリシー・セットを選択します。次に、選択した各ポリシー・セットの zip ファイルを、Integration Designer からアクセス可能な場所に保存します。

1. 以下の手順に従って、ポリシー・セットをインポートします。
   a. 管理コンソールで、「**サービス**」 > 「**ポリシー・セット**」 > 「**アプリケーション・ポリシー・セット (Application policy sets)**」をクリックします。
   b. 「**インポート**」 > 「**デフォルトのリポジトリーから (From Default Repository)**」をクリックします。
   c. SAML デフォルト・ポリシー・セットを選択し、「**OK**」をクリックします。
2. Integration Designer が使用できるように、ポリシー・セットをエクスポートします。
   a. 「アプリケーション・ポリシー・セット (Application policy sets)」ページで、エクスポートする SAML ポリシー・セットを選択し、「**エクスポート**」をクリックします。

      注: 「アプリケーション・ポリシー・セット (Application policy sets)」ページが表示されていない場合は、管理コンソールで「**サービス**」 > 「**ポリシー・セット**」 > 「**アプリケーション・ポリシー・セット (Application Policy Sets**をクリックします。
   b. 次のページで、ポリシー・セットの .zip ファイル・リンクをクリックします。
   c. 「ファイルのダウンロード」ウィンドウで、「**保存**」をクリックし、 Integration Designer からアクセス可能な場所を指定します。
   d. 「**戻る**」をクリックします。
   e. エクスポートするポリシー・セットごとに、ステップ 2a から 2d を実行します。

SAML ポリシー・セットが .zip ファイルに保存され、 Integration Designer にインポートできる状態になります。

『ポリシー・セット』トピックの説明に従って、ポリシー・セットを Integration Designer にインポートします。

HTTP 基本認証が必要な Web サービスの呼び出し:

HTTP 基本認証ではユーザー名とパスワードを使用して、セキュア・エンドポイントに対してサービス・クライアントを認証します。HTTP 基本認証のセットアップは、Web サービス要求の送信時または受信時に行うことができます。

『Web サービス・エクスポートに対するセキュリティー・ロールの作成および割り当て』トピックで説明したように、Web サービス要求を受信できるように HTTP 基本認証をセットアップするには、Java API for XML Web Service (JAX-WS) エクスポート・バインディングを構成します。

JAX-WS インポート・バインディングによって送信された Web サービス要求に対して HTTP 基本認証を有効にするには、以下の 2 つのいずれかの方法を用います。

- SCA モジュールでインポート・バインディングを構成する際に、用意されている BPMHTTPBasicAuthentication という名前の HTTP 認証ポリシー・セット (Web サービス (JAX-WS) インポート・バインディングに付属) か、HTTPTransport ポリシーが含まれた別のポリシー・セットを選択することができます。
- SCA モジュールを構成する際に、メディエーション・フロー機能を使用して新規 HTTP 認証ヘッダーを動的に作成し、ヘッダー内にユーザー名とパスワードの情報を指定することができます。

注: ポリシー・セットは、ヘッダーで指定された値よりも優先します。 HTTP 認証ヘッダーに設定された値を実行時に使用したい場合は、HTTPTransport ポリシーが含まれたポリシー・セットを添付しないでください。特に、デフォルトの BPMHTTPBasicAuthentication ポリシー・セットを使用しないでください。また、ポリシー・セットを定義している場合は、HTTPTransport ポリシーが除外されていることを確認してください。

Web サービス・ポリシー・セットとポリシー・バインディング、およびそれらの使用方法について詳しくは、WebSphere Application Server インフォメーション・センターの『Web サービス・ポリシー・セット』トピックを参照してください。このトピックのリンクは、『関連情報』の下にリストされています。

- 用意されているポリシー・セットを使用するには、以下の手順を実行します。
  1. オプション: 管理コンソールで、クライアント一般ポリシー・バインディングを作成するか、または必要なユーザー ID とパスワードの値を使用して、HTTPTransport ポリシーが含まれた既存のポリシー・バインディングを編集します。
  2. IBM Integration Designerで、Web サービス (JAX-WS) インポート・バインディングを生成して、BPMHTTPBasicAuthentication ポリシー・セットを関連付けます。
  3. 以下のいずれかのステップを実行します。
     - IBM Integration Designer の Web サービス (JAX-WS) インポート・バインディング・プロパティーで、HTTPTransport ポリシーを含む、既存のクライアントの汎用ポリシー・バインディングの名前を指定します。
     - SCA モジュールをデプロイしたら、Process Server の管理コンソールから、既存のクライアント・ポリシー・バインディングを選択するか、新規のクライアント・ポリシー・バインディングを作成し、それをインポート・バインディングと関連付けます。
  4. オプション: Process Server の管理コンソールで、選択したポリシー・セット・バインディングを編集して、必要な ID とパスワードを指定します。
- HTTP 認証ヘッダーでユーザー名とパスワードを指定するには、以下のいずれかのステップ一式を実行します。
  - IBM Integration Designer の HTTP ヘッダー・セッター・メディエーション・プリミティブを使用して HTTP 認証ヘッダーを作成し、ユーザー名およびパスワードを指定します。
  - 追加のロジックが必要な場合は、カスタム・メディエーション・プリミティブに Java コードを使用して、以下のことを実行します (以下の例を参照)。
    1. HTTP 認証ヘッダーを作成します。
    2. ユーザー名およびパスワードの情報を指定します。
    3. 新規 HTTP 認証ヘッダーを HTTPControl に追加します。

4. 更新した HTTPControl をコンテキスト・サービスで設定します。

```
//Get the HeaderInfoType from contextService
ContextService contextService = (ContextService) ServiceManager.INSTANCE
.locateService("com/ibm/bpm/context/ContextService");
HeaderInfoType headers = contextService.getHeaderInfo();
if(headers == null){
    headers = ContextObjectFactory.eINSTANCE.createHeaderInfoType();
}
//Get the HTTP header and HTTP Control from HeaderInfoType
HTTPHeaderType httpHeaderType = headers.getHTTPHeader();
HTTPControl cp = httpHeaderType.getControl();
HeadersFactory factory = HeadersFactory.eINSTANCE;
if(cp == null){
    cp = factory.createHTTPControl();
}
//Create new HTTPAuthentication and set the HTTPCredentials
HTTPAuthentication authorization = factory.createHTTPAuthentication();
HTTPCredentials credentials = factory.createHTTPCredentials();
authorization.setAuthenticationType(HTTPAuthenticationType.BASIC_LITERAL);
credentials.setUserId("USERNAME");
credentials.setPassword("PASSWORD");
authorization.setCredentials(credentials);
cp.setAuthentication(authorization);
httpHeaderType.setControl(cp);
// Set header info back to the current execution context.
contextService.setHeaderInfo(headers);
```

サーバー構成の確認:

Web サービス (JAX-WS) バインディングでアプリケーションをデプロイする際には、アプリケーションがデプロイされるサーバーで「**必要に応じてコンポーネントを始動 (Start components as needed)**」オプションが選択されていないことを確認する必要があります。

このオプションが選択されているかどうかを確認するには、管理コンソールで以下の手順を実行します。

1. 「**サーバー**」 > 「**サーバー・タイプ**」 > 「**WebSphere Application Server**」をクリックします。
2. サーバー名をクリックします。
3. 「構成」タブで、「**必要に応じてコンポーネントを始動 (Start components as needed)**」が選択されているかどうかを確認します。
4. 以下のいずれかのステップを実行します。
   - 「**必要に応じてコンポーネントを始動 (Start components as needed)**」が選択されている場合は、チェック・マークを外して「**適用**」をクリックします。
   - 「**必要に応じてコンポーネントを始動 (Start components as needed)**」が選択されていない場合は、「**キャンセル**」をクリックします。

SOAP メッセージの添付ファイル:

添付ファイルとしてバイナリー・データ (PDF ファイルや JPEG イメージ) が含まれている SOAP メッセージを送信および受信することができます。添付ファイルには、*参照されている* 添付ファイル (サービス・インターフェースでメッセージ・パーツとして明示的に表わされる) と、*参照されていない* 添付ファイル (任意の数とタイプの添付ファイルを組み込むことができる) があります。

参照されている添付ファイルは、以下のいずれかの方法で表わされます。

- メッセージ・スキーマの wsi:swaRef タイプのエレメントとして

wsi:swaRef タイプを使用して定義される添付ファイルは、メッセージ・エレメントがどのように MIME パーツに関連しているかを定義する Web Services Interoperability Organization (WS-I) の「*Attachments Profile Version 1.0*」(http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html) に準拠しています。

- バイナリー・スキーマ・タイプを使用した最上位メッセージ・パーツとして

  最上位メッセージ・パーツとして表わされる添付ファイルは、「*SOAP Messages with Attachments*」(http://www.w3.org/TR/SOAP-attachments) 仕様に準拠しています。

  最上位メッセージ・パーツとして表された添付ファイルは、バインディングによって生成された WSDL 文書とメッセージが WS-I *Attachments Profile Version 1.0* および WS-I *Basic Profile Version 1.1* (http://www.ws-i.org/Profiles/BasicProfile-1.1.html) に準拠するように構成することもできます。

参照されていない添付ファイルは、メッセージ・スキーマ内で表わされることなく、SOAP メッセージに組み込まれます。

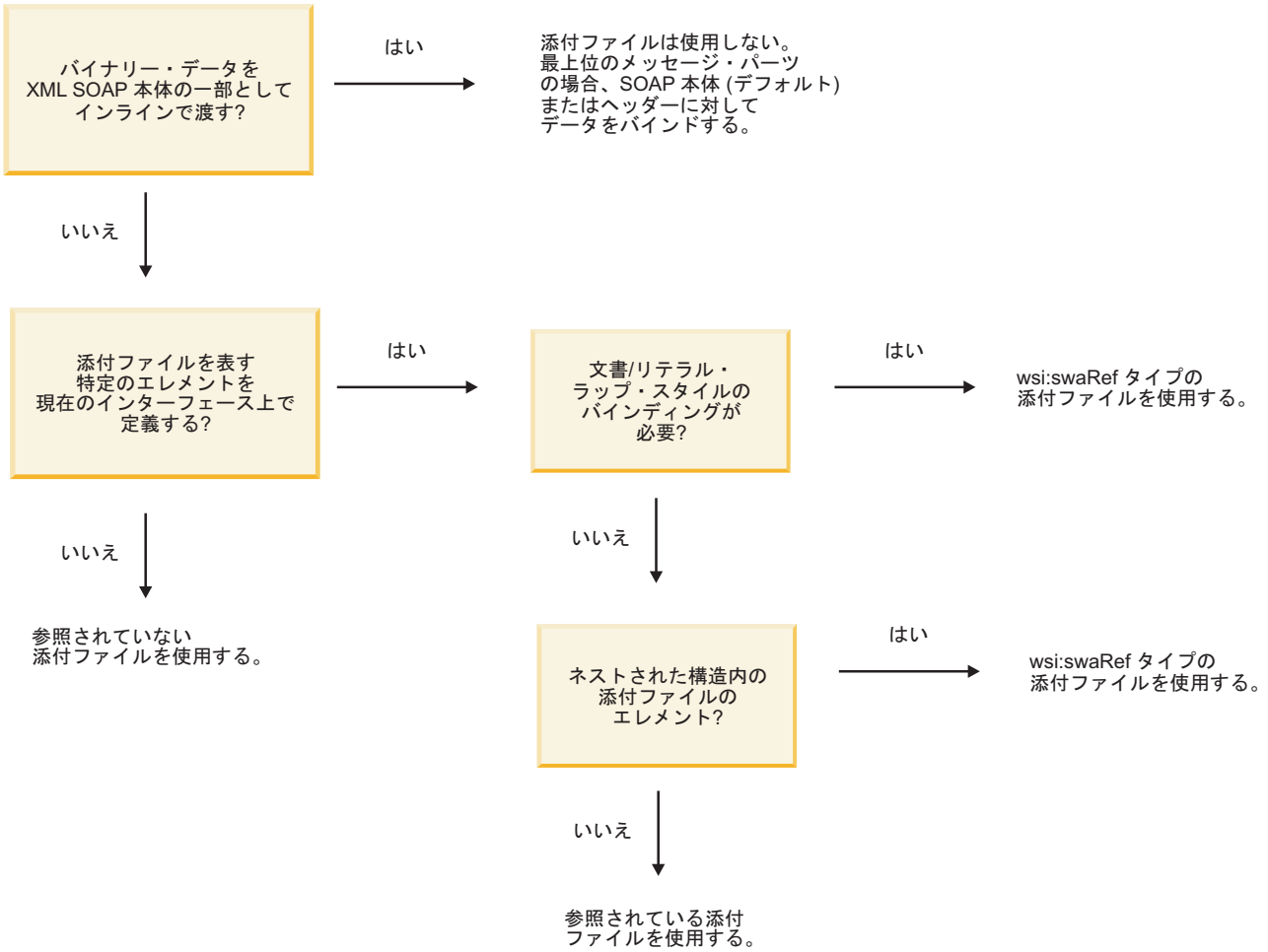
いずれの場合にも、WSDL SOAP バインディングには、使用する予定の添付ファイルの MIME バインディングを組み込み、添付ファイルの最大サイズは 20 MB を超えないようにしてください。

注: 添付ファイル付きの SOAP メッセージを送信または受信するには、Java API for XML Web Services (JAX-WS) に基づく、いずれかの Web サービス・バインディングを使用する必要があります。

適切な添付ファイル・スタイルの選択方法:

バイナリー・データを使用する新しいサービス・インターフェースを設計する場合は、サービスによって送受信される SOAP メッセージ内におけるバイナリー・データの伝送方法を検討してください。

以下の一連の質問に答えて、適切な添付ファイル・スタイルを判断してください。



参照されている添付ファイル: swaRef タイプのエレメント:

サービス・インターフェースで swaRef タイプのエレメントとして表わされている添付ファイルが組み込まれた SOAP メッセージを送信および受信することができます。

swaRef タイプのエレメントは、メッセージ・エレメントがどのように MIME パーツに関連しているかを定義する Web Services Interoperability Organization (WS-I) の「*Attachments Profile Version 1.0*」(http://www.ws-i.org/Profiles/AttachmentsProfile-1.0.html) で定義されています。

SOAP メッセージ内の SOAP body には、添付ファイルのコンテンツ ID を識別する swaRef タイプのエレメントが含まれています。

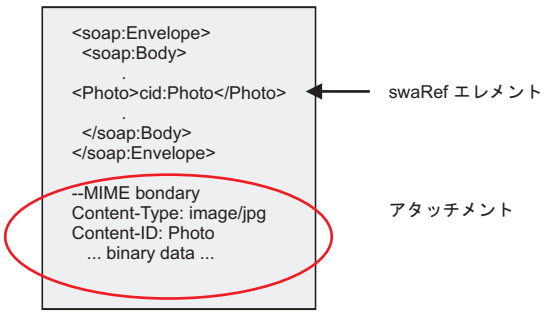


図 13. swaRef エレメントが含まれている SOAP メッセージ

この SOAP メッセージの WSDL では、添付ファイルを識別するメッセージ・パーツ内に swaRef タイプのエレメントが含まれています。

```
<element name="sendPhoto">
  <complexType>
     <sequence>
       <element name="Photo" type="wsi:swaRef"/>
     </sequence>
  </complexType>
 </element>
```

WSDL には、MIME multipart メッセージを使用することを示す MIME バインディングも含まれている必要があります。

注: MIME バインディングは最上位メッセージ・パーツにのみ適用されるため、WSDL には特定の swaRef タイプのメッセージ・エレメント用の MIME バインディングは含まれていません。

swaRef タイプのエレメントとして表わされている添付ファイルは、メディエーション・フロー・コンポーネントを介した場合にのみ伝搬できます。添付ファイルに別のコンポーネント・タイプでアクセスする必要がある場合や、別のコンポーネント・タイプに伝搬する必要がある場合には、メディエーション・フロー・コンポーネントを使用して、このコンポーネントがアクセスできる場所へ添付ファイルを移動してください。

添付ファイルのインバウンド処理

Integration Designer を使用して、添付ファイルを受信するようエクスポート・バインディングを構成します。モジュールと、モジュールに関連付けられたインターフェースや操作 (タイプが swaRef のエレメント) を作成します。次に、Web サービス (JAX-WS) バインディングを作成します。

注: 詳しくは、Integration Designer インフォメーション・センターのトピック『添付ファイルの操作 (Working with attachments)』を参照してください。

クライアントが swaRef 添付ファイル付きの SOAP メッセージを Service Component Architecture (SCA) コンポーネントに渡すと、Web サービス (JAX-WS) エクスポート・バインディングは、最初に添付ファイルを除去します。次に、メッセージの SOAP パーツを解析して、ビジネス・オブジェクトを作成します。最後に、バインディングは、ビジネス・オブジェクト内で添付ファイルのコンテンツ ID を設定します。

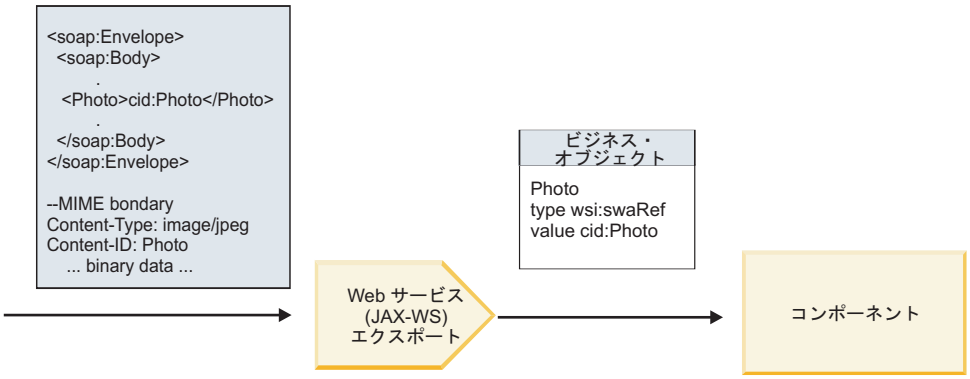


図 14. Web サービス (JAX-WS) エクスポート・バインディングが、swaRef 添付ファイル付きの SOAP メッセージを処理する方法

メディエーション・フロー・コンポーネント内の添付ファイル・メタデータへのアクセス

図 15 に示されているとおり、swaRef 添付ファイルがコンポーネントによってアクセスされると、添付ファイル・コンテンツ ID はタイプが swaRef のエレメントとして表現されます。

SOAP メッセージのそれぞれの添付ファイルの SMO には、対応する **attachments** エレメントがあります。WS-I swaRef タイプを使用する場合、**attachments** エレメントには、添付ファイルの実際のバイナリー・データだけでなく、添付ファイル・コンテンツ・タイプとコンテンツ ID も組み込まれます。

したがって、swaRef 添付ファイルの値を取得するには、swaRef タイプのエレメントの値を取得する必要があり、対応する **contentID** 値が含まれている **attachments** エレメントを見つける必要があります。通常、**contentID** 値では、**cid:** プレフィックスが swaRef 値から除去されています。

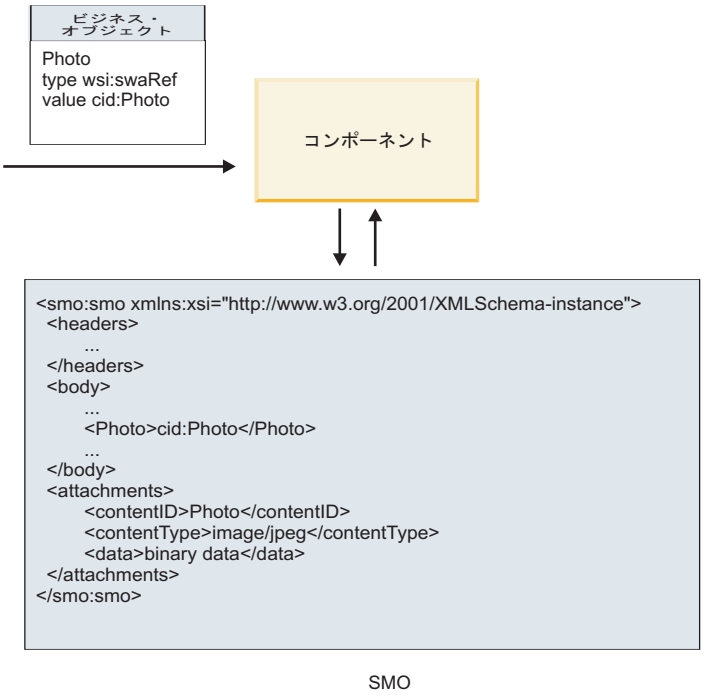


図 15. swaRef 添付ファイルが SMO 内で表わされる方法

アウトバウンド処理

外部 Web サービスを呼び出すように Web サービス (JAX-WS) インポート・バインディングを構成するには、Integration Designer を使用します。インポート・バインディングは、WSDL 文書を使用して構成されます。この文書は、呼び出される Web サービスを記述し、Web サービスに渡される添付ファイルを定義しています。

SCA メッセージが Web サービス (JAX-WS) インポート・バインディングによって受信されると、インポートがメディエーション・フロー・コンポーネントにワイヤリングされている場合で、対応する **attachments** エレメントが swaRef タイプのエレメントにある場合は、 swaRef タイプのエレメントが添付ファイルとして送信されます。

アウトバウンド処理の場合、swaRef タイプのエレメントは常にコンテンツ ID 値と一緒に送信されます。ただし、メディエーション・モジュールは、**contentID** 値が一致する、対応する **attachments** エレメントが存在することを確認する必要があります。

注: WS-I Attachments Profile に準拠するためには、**content ID** 値は、WS-I *Attachments Profile* 1.0 のセクション 3.8 に記述されている「content-id への part 名エンコーディング (content-id part encoding)」に従う必要があります。

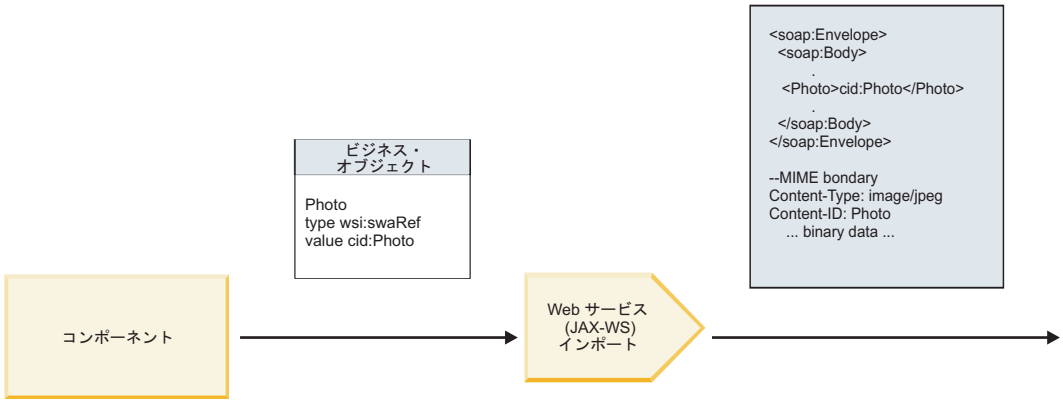


図 16. Web サービス (JAX-WS) インポート・バインディングが、swaRef 添付ファイル付きの SOAP メッセージを生成する方法

メディエーション・フロー・コンポーネント内の添付ファイル・メタデータの設定

SMO 内に、swaRef タイプのエレメント値と **attachments** エレメントがある場合、バインディングは SOAP メッセージ (添付ファイル付き) を作成し、受信側に送信します。

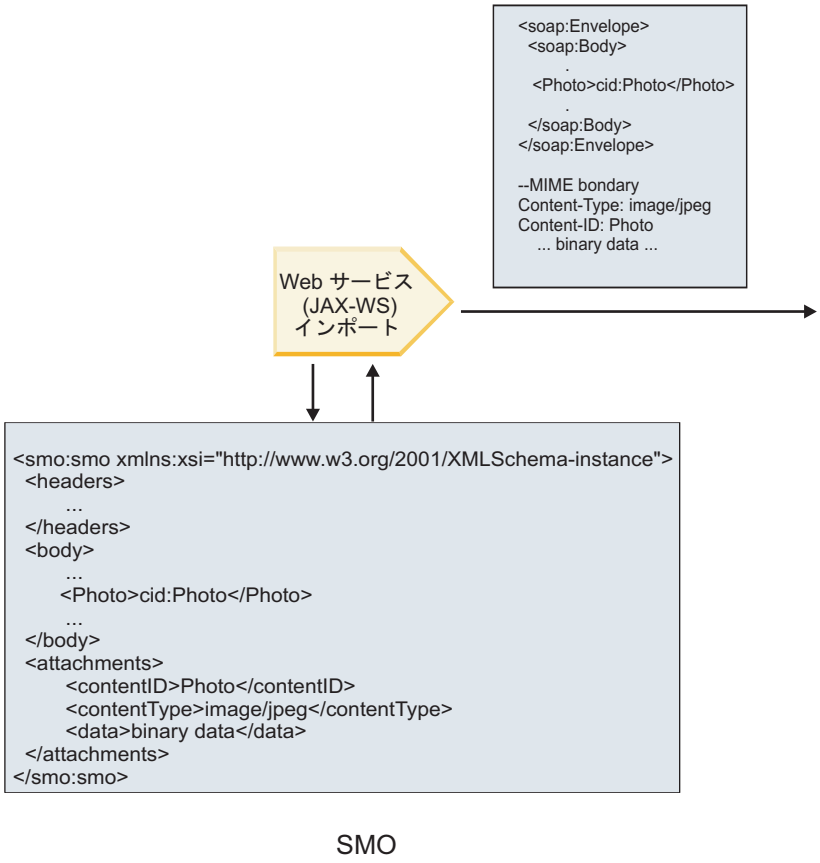


図 17. SOAP メッセージを作成するために SMO 内の swaRef 添付ファイルがアクセスされる方法

attachments エレメントは、メディエーション・フロー・コンポーネントがインポートまたはエクスポートに直接接続されている場合にのみ、SMO 内に存在します。このエレメントは、他のコンポーネント・タイプによって渡されることはありません。他のコンポーネント・タイプが含まれているモジュールで値が必要になる場合は、メディエーション・フロー・コンポーネントを使用して、モジュール内でアクセス可能な場所に値をコピーする必要があり、別のメディエーション・フロー・コンポーネントを使用して、Web サービス・インポートによるアウトバウンド呼び出しの前に、正しい値を設定する必要があります。

重要: 『SMO の XML 表記』で説明しているように、XSL 変換メディエーション・プリミティブは、XSLT 1.0 変換を使用してメッセージを変換します。変換は、SMO の XML 直列化で作動します。XSL 変換メディエーション・プリミティブより、直列化のルートを指定することができます。また、XML 文書のルート・エレメントはこのルートを反映します。

SOAP メッセージを添付ファイルと共に送信する場合、選択したルート・エレメントによって、添付ファイルの伝搬方法が決まります。

- 「/body」を XML マップのルートとして使用した場合は、デフォルトですべての添付ファイルがマップ全体に伝搬します。
- 「/」をマップのルートとして使用した場合は、添付ファイルの伝搬を制御できます。

参照されている添付ファイル: 最上位メッセージ・パーツ:

サービス・インターフェースでパーツとして宣言されているバイナリー添付ファイルが組み込まれた SOAP メッセージを送信および受信することができます。

MIME multipart SOAP メッセージの場合、SOAP 本体が最初のメッセージ・パーツとなり、添付ファイル (複数可) は後続のパーツに含まれます。

参照されている添付ファイルを SOAP メッセージで送信または受信することの利点は何でしょうか。 添付ファイルを構成するバイナリー・データ (多くの場合、大容量のデータ) は、SOAP メッセージ本体とは別に保持されているため、XML として解析する必要はありません。このため、バイナリー・データが XML エレメントに保持される場合よりも効率的に処理することができます。

参照されている添付ファイル付き SOAP メッセージのタイプ

IBM Business Process Manager バージョン 7.0.0.3 から、SOAP メッセージの生成方法を選択できるようになりました。

- **WS-I 準拠メッセージ (WS-I-compliant messages)**

  WS-I *Attachments Profile Version 1.0* および WS-I *Basic Profile Version 1.1* に準拠した SOAP メッセージをランタイムで生成することができます。これらのプロファイルに準拠する SOAP メッセージの場合、SOAP 本体にバインドされるのは 1 つのメッセージ・パーツだけです。添付ファイルとしてバインドされる SOAP メッセージの場合は、content-id パーツのエンコード (WS-I *Attachments Profile Version 1.0* で記述) を使用して、添付ファイルがメッセージ・パーツに関連付けられます。

- **WS-I 非準拠メッセージ (Non-WS-I-compliant messages)**

  WS-I プロファイルには準拠しないが、IBM Business Process Manager バージョン 7.0 または 7.0.0.2 で生成されたメッセージと互換性のある SOAP メッセージを、ランタイムで生成することができます。これらの SOAP メッセージは、添付ファイル **content-id** を指定する **href** 属性を持つメッセージ・パーツに基づいて名前が付けられた最上位エレメントを使用しますが、content-id パーツのエンコード (WS-I *Attachments Profile Version 1.0* で記述) は使用されません。

Web サービス・エクスポートに対する WS-I 準拠の選択

Integration Designer を使用して、エクスポート・バインディングを構成します。モジュールと、モジュールに関連付けられたインターフェースや操作を作成します。次に、Web サービス (JAX-WS) バインディングを作成します。「参照されている添付ファイル (Referenced attachments)」ページに、作成された操作のすべてのバイナリー・パーツが表示されるため、添付ファイルとして使用するパーツを選択します。次に、Integration Designer の「WS-I AP 1.0 準拠の指定 (Specify the WS-I AP 1.0 compliance)」ページで、以下のいずれかの選択項目を指定します。

- **WS-I AP 1.0 準拠の SOAP メッセージを使用する (Use WS-I AP 1.0 compliant SOAP message)**

  このオプションを選択した場合は、SOAP 本体にバインドすべきメッセージ・パーツも指定します。

  注: このオプションを使用できるのは、対応する WSDL ファイルも WS-I に準拠している場合だけです。

  Integration Designer バージョン 7.0.0.3 によって生成される WSDL ファイルは、WS-I に準拠しています。ただし、WS-I 準拠でない WSDL ファイルをインポートする場合は、このオプションを選択できません。

- **WS-I AP 1.0 非準拠の SOAP メッセージを使用する (Use non WS-I AP 1.0 compliant SOAP message)**

  このオプションを選択した場合 (デフォルト) は、最初のメッセージ・パーツが SOAP 本体にバインドされます。

注: 参照されている添付ファイルとして送受信できるのは、バイナリー・タイプ (base64Binary または hexBinary) を持つ最上位メッセージ・パーツ (入力メッセージまたは出力メッセージ内のパーツとして WSDL portType で定義されているエレメント) だけです。
詳しくは、Integration Designer インフォメーション・センターのトピック『添付ファイルの操作 (Working with attachments)』を参照してください。

WS-I 準拠のメッセージの場合、SOAP メッセージ内に生成された content-ID が以下のエレメントと連結されます。

- **mime:content** によって参照される **wsdl:part** エレメントの **name** 属性の値
- 等号 (=)
- グローバルな固有値 (UUID など)
- アットマーク (@)
- 有効なドメイン名

参照されている添付ファイルのインバウンド処理

クライアントが添付ファイル付きの SOAP メッセージを Service Component Architecture (SCA) コンポーネントに渡すと、Web サービス (JAX-WS) エクスポート・バインディングは、最初に添付ファイルを除去します。次に、メッセージの SOAP パーツを解析して、ビジネス・オブジェクトを作成します。最後に、バインディングは、ビジネス・オブジェクト内で添付ファイル・バイナリーを設定します。

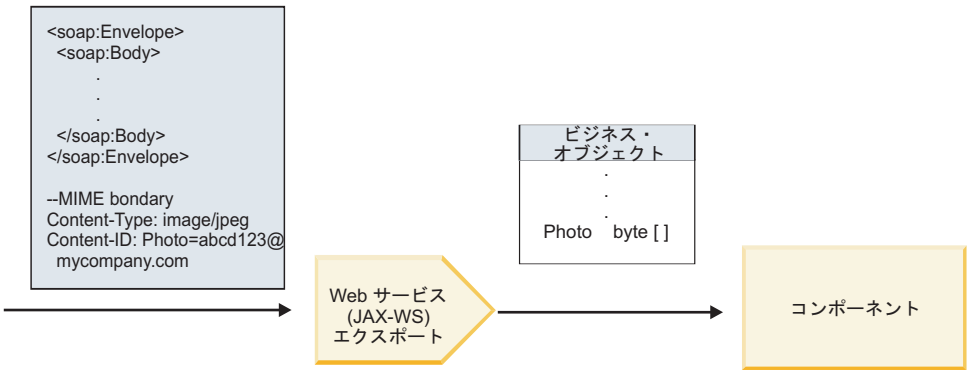


図 18. Web サービス (JAX-WS) エクスポート・バインディングが、参照されている添付ファイル付きの WS-I 準拠 SOAP メッセージを処理する方法

メディエーション・フロー・コンポーネント内の添付ファイル・メタデータへのアクセス

図 18 に示されているとおり、参照されている添付ファイルがコンポーネントによってアクセスされると、添付ファイル・データはバイト配列として表現されます。

SOAP メッセージのそれぞれの参照されている添付ファイルの SMO には、対応する **attachments** エレメントがあります。**attachments** エレメントには、添付ファイルのコンテンツ・タイプと、添付ファイルが保持されているメッセージの body エレメントへのパスが組み込まれています。

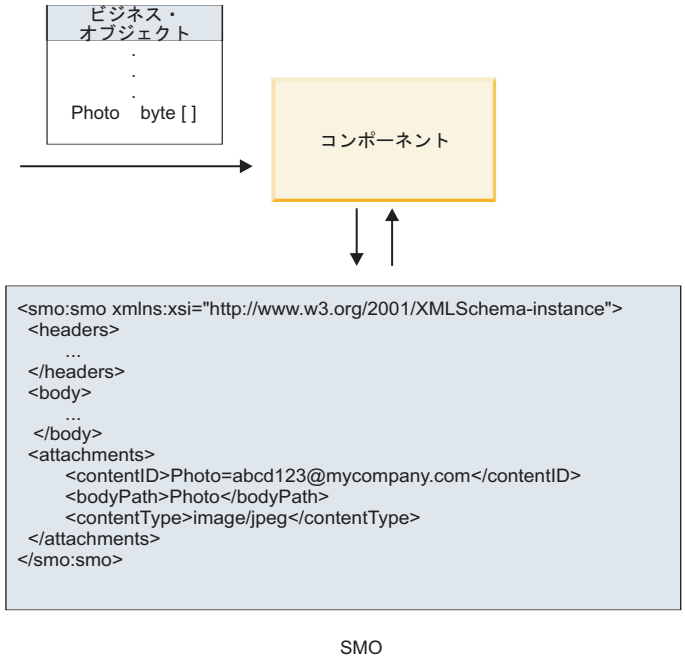


図 19. 参照されている添付ファイルが SMO 内で表わされる方法

重要: メッセージが変換され、添付ファイルが移動されると、メッセージの body エレメントへのパスは、自動的に更新されません。メディエーション・フローを使用することにより、(変換の一環として、または別のメッセージ・エレメント・セッターを使用するなどして) **attachments** エレメントを新しいパスで更新することができます。

アウトバウンド SOAP メッセージの構成方法

外部 Web サービスを呼び出すように Web サービス (JAX-WS) インポート・バインディングを構成するには、Integration Designer を使用します。インポート・バインディングは、WSDL 文書を使用して構成されます。この文書は、呼び出される Web サービスを記述し、添付ファイルとして渡すメッセージ・パーツを定義しています。Integration Designer の「WS-I AP 1.0 準拠の指定 (Specify the WS-I AP 1.0 compliance)」ページで、以下のいずれかの選択項目を指定することもできます。

- **WS-I AP 1.0 準拠の SOAP メッセージを使用する (Use WS-I AP 1.0 compliant SOAP message)**

  このオプションを選択した場合は、SOAP 本体にバインドすべきメッセージ・パーツも指定します。それ以外はすべて、添付ファイルまたはヘッダーにバインドされます。バインディングによって送信されるメッセージの SOAP 本体には、添付ファイルを参照するエレメントは含まれません。この関係は、メッセージ・パーツ名を含む添付ファイル・コンテンツ ID によって表されます。
- **WS-I AP 1.0 非準拠の SOAP メッセージを使用する (Use non WS-I AP 1.0 compliant SOAP message)**

  このオプションを選択した場合 (デフォルト) は、最初のメッセージ・パーツが SOAP 本体にバインドされます。それ以外はすべて、添付ファイルまたはヘッダーにバインドされます。バインディングによって送信されるメッセージの SOAP 本体には、**href** 属性を使用して添付ファイルを参照する 1 つ以上のエレメントが含まれます。

注: 添付ファイルを表すパーツ (WSDL 内で定義されている) は、単純タイプ (base64Binary または hexBinary) でなければなりません。complexType によって定義されているパーツは、添付ファイルとしてバインドすることはできません。

参照されている添付ファイルのアウトバウンド処理

インポート・バインディングでは、SMO の情報を使用して、バイナリーの最上位メッセージ・パーツを添付ファイルとして送信する方法が決定されます。

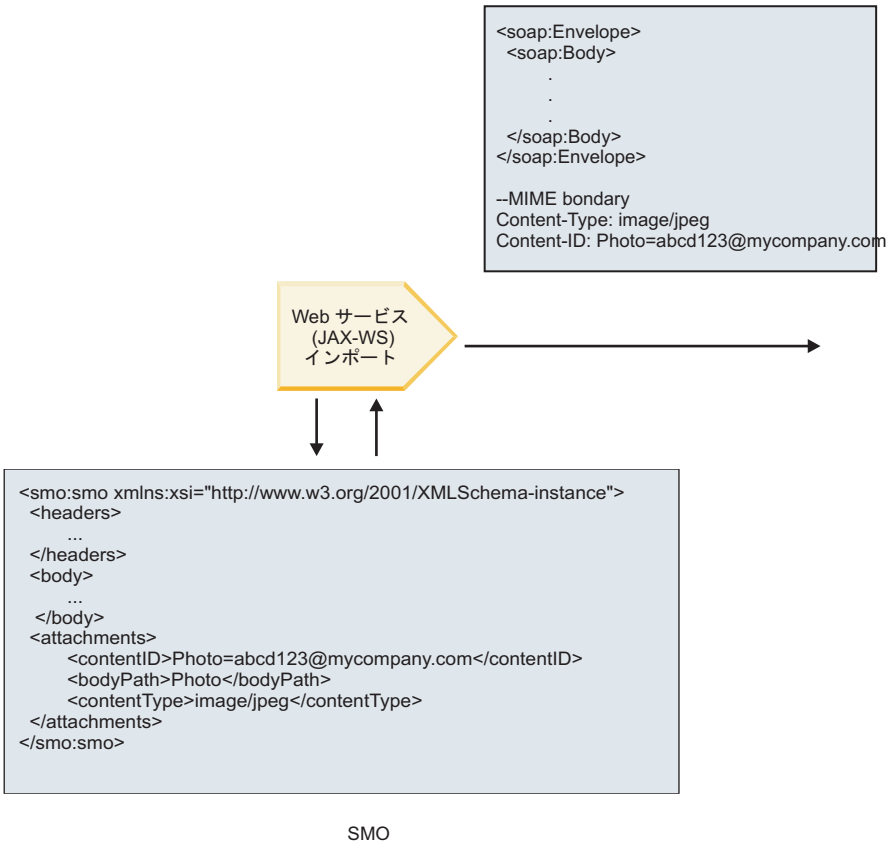


図 20. SOAP メッセージを作成するために SMO 内の参照されている添付ファイルがアクセスされる方法

attachments エレメントは、メディエーション・フロー・コンポーネントがインポートまたはエクスポートに直接接続されている場合にのみ、SMO 内に存在します。このエレメントは、他のコンポーネント・タイプによって渡されることはありません。他のコンポーネント・タイプが含まれているモジュールで値が必要になる場合は、メディエーション・フロー・コンポーネントを使用して、モジュール内でアクセス可能な場所に値をコピーする必要があり、別のメディエーション・フロー・コンポーネントを使用して、Web サービス・インポートによるアウトバウンド呼び出しの前に、正しい値を設定する必要があります。

バインディングでは、以下の条件の組み合わせを使用して、メッセージの送信方法 (または、メッセージを送信するかどうか) が決定されます。

- 最上位バイナリー・メッセージ・パーツの WSDL MIME バインディングが存在するかどうか。存在する場合は、コンテンツ・タイプの定義方法。
- **bodyPath** 値が最上位バイナリー・パーツを参照している **attachments** エレメントが SMO に存在するかどうか。

attachment エレメントが SMO に存在するときの添付ファイルの作成方法

bodyPath がメッセージ名パーツに一致する **attachment** エレメントが SMO に含まれている場合、添付ファイルを作成して送信する方法を以下の表に示します。

表 20. 添付ファイルの生成方法

最上位バイナリー・メッセージ・パーツの WSDL MIME バインディングの状況	**メッセージを作成して送信する方法**
以下のいずれかの場合に存在します。 • メッセージ・パーツのコンテンツ・タイプが定義されていない • 複数のコンテンツ・タイプが定義されている • ワイルドカード・コンテンツ・タイプが定義されている	メッセージ・パーツは、添付ファイルとして送信されます。 **Content-Id** には、添付ファイル・エレメントの値が設定されます (存在する場合)。それ以外の場合は、Content-Id 値が生成されます。 **Content-Type** には、添付ファイル・エレメントの値が設定されます (存在する場合)。それ以外の場合は、**application/octet-stream** が設定されます。
メッセージ・パーツのコンテンツが単一で非ワイルドカードの場合に存在します	メッセージ・パーツは、添付ファイルとして送信されます。 **Content-Id** には、添付ファイル・エレメントの値が設定されます (存在する場合)。それ以外の場合は、Content-Id 値が生成されます。 **Content-Type** には、添付ファイル・エレメントの値が設定されます (存在する場合)。それ以外の場合は、WSDL MIME コンテンツ・エレメントで定義されたタイプが設定されます。
存在しません	メッセージ・パーツは、添付ファイルとして送信されます。 **Content-Id** には、添付ファイル・エレメントの値が設定されます (存在する場合)。それ以外の場合は、Content-Id 値が生成されます。 **Content-Type** には、添付ファイル・エレメントの値が設定されます (存在する場合)。それ以外の場合は、**application/octet-stream** が設定されます。 **注:** メッセージ・パーツを添付ファイルとして送信することが WSDL で定義されていないのに、メッセージ・パーツを添付ファイルとして送信すると、WS-I Attachments Profile 1.0 に準拠しなくなる可能性があります。そのため、このようなことは、できる限り避ける必要があります。

attachment エレメントが SMO に存在しないときの添付ファイルの作成方法

bodyPath がメッセージ名パーツに一致する **attachment** エレメントが SMO に含まれていない場合、添付ファイルを作成して送信する方法を以下の表に示します。

表 21. 添付ファイルの生成方法

最上位バイナリー・メッセージ・パーツの WSDL MIME バインディングの状況	**メッセージを作成して送信する方法**
以下のいずれかの場合に存在します。 • メッセージ・パーツのコンテンツ・タイプが定義されていない • 複数のコンテンツ・タイプが定義されている • ワイルドカード・コンテンツ・タイプが定義されている	メッセージ・パーツは、添付ファイルとして送信されます。 **Content-Id** が生成されます。 **Content-Type** には、**application/octet-stream** が設定されます。
メッセージ・パーツのコンテンツが単一で非ワイルドカードの場合に存在します	メッセージ・パーツは、添付ファイルとして送信されます。 **Content-Id** が生成されます。 **Content-Type** には、WSDL MIME コンテンツ・エレメントで定義されたタイプが設定されます。
存在しません	メッセージ・パーツは、添付ファイルとして送信されません。

重要: 『SMO の XML 表記』で説明しているように、XSL 変換メディエーション・プリミティブは、XSLT 1.0 変換を使用してメッセージを変換します。変換は、SMO の XML 直列化で作動します。XSL 変換メディエーション・プリミティブより、直列化のルートを指定することができます。また、XML 文書のルート・エレメントはこのルートを反映します。

SOAP メッセージを添付ファイルと共に送信する場合、選択したルート・エレメントによって、添付ファイルの伝搬方法が決まります。

- 「/body」を XML マップのルートとして使用した場合は、デフォルトですべての添付ファイルがマップ全体に伝搬します。
- 「/」をマップのルートとして使用した場合は、添付ファイルの伝搬を制御できます。

参照されていない添付ファイル:

サービス・インターフェースで宣言されていない、*参照*されていない添付ファイルを送信および受信できます。

MIME multipart SOAP メッセージでは、SOAP 本体がメッセージの最初の部分であり、添付ファイルが後続の部分です。SOAP 本体には、添付ファイルへの参照情報は組み込まれていません。

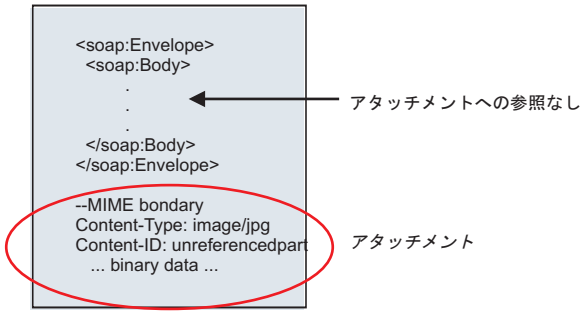


図 21. 参照されていない添付ファイル付きの SOAP メッセージ

参照されていない添付ファイル付きの SOAP メッセージは、Web サービス・エクスポートを介して Web サービス・インポートへ送信できます。ターゲットの Web サービスへ送信される出力メッセージには、添付ファイルが含まれます。

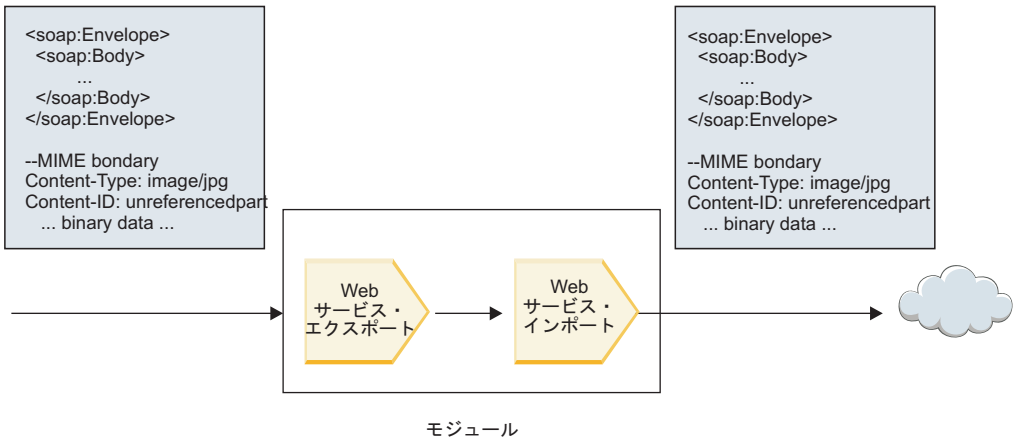


図 22. SCA モジュールを通過する添付ファイル

図 22 は、添付ファイル付きの SOAP メッセージが変更なしで通過する様子を示しています。

SOAP メッセージは、メディエーション・フロー・コンポーネントを使用して変更することもできます。例えば、メディエーション・フロー・コンポーネントを使用して SOAP メッセージからデータ (この場合は、メッセージの本体内部のバイナリー・データ) を抽出し、添付ファイル付きの SOAP メッセージを作成できます。データは、サービス・メッセージ・オブジェクト (SMO) の添付ファイル・エレメントの部分として処理されます。

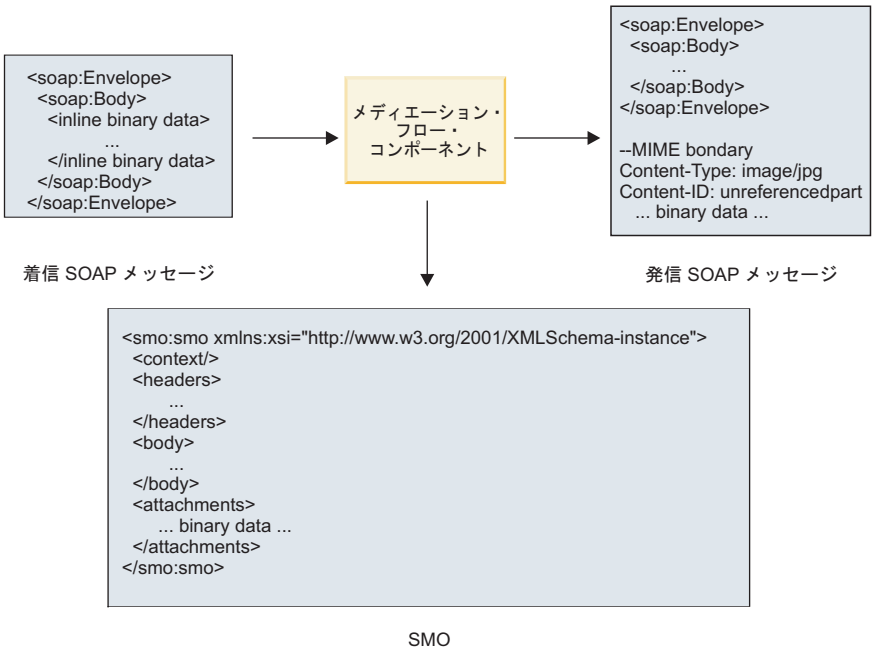


図 23. メディエーション・フロー・コンポーネントによって処理されるメッセージ

反対に、メディエーション・フロー・コンポーネントは、添付ファイルを抽出してエンコードすることによって着信メッセージを変換し、その後添付ファイルなしでメッセージを送信できます。

着信 SOAP メッセージからデータを抽出して添付ファイル付きの SOAP メッセージを作成する代わりに、データベースなどの外部ソースから添付ファイル・データを取得できます。

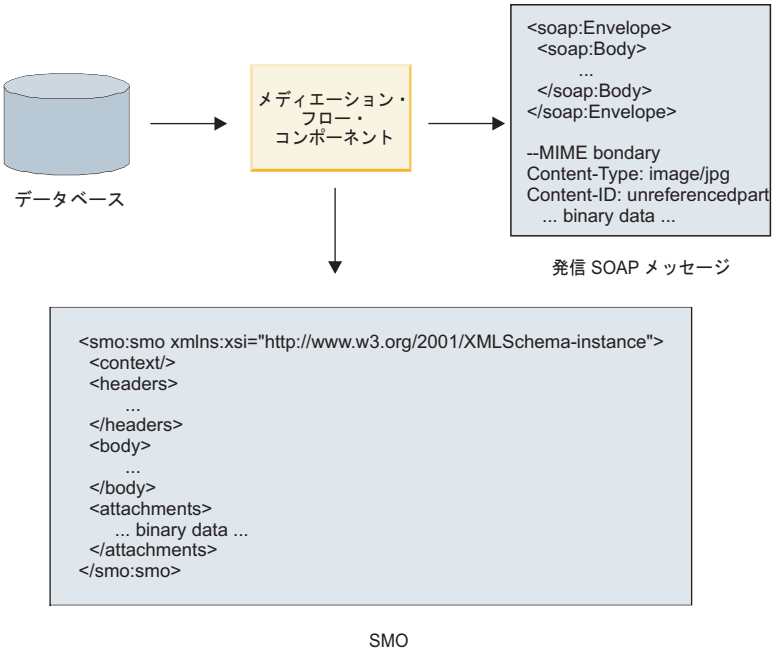


図 24. データベースから取得して SOAP メッセージに追加した添付ファイル

反対に、メディエーション・フロー・コンポーネントは着信 SOAP メッセージから添付ファイルを抽出して、メッセージを処理 (例えば、添付ファイルをデータベースに保管) できます。

参照されていない添付ファイルは、メディエーション・フロー・コンポーネントを介した場合にのみ伝搬できます。添付ファイルに別のコンポーネント・タイプでアクセスする必要がある場合や、別のコンポーネント・タイプに伝搬する必要がある場合には、メディエーション・フロー・コンポーネントを使用して、このコンポーネントがアクセスできる場所へ添付ファイルを移動してください。

重要: 『SMO の XML 表記』で説明しているように、XSL 変換メディエーション・プリミティブは、XSLT 1.0 変換を使用してメッセージを変換します。変換は、SMO の XML 直列化で作動します。XSL 変換メディエーション・プリミティブより、直列化のルートを指定することができます。また、XML 文書のルート・エレメントはこのルートを反映します。

SOAP メッセージを添付ファイルと共に送信する場合、選択したルート・エレメントによって、添付ファイルの伝搬方法が決まります。

- 「/body」を XML マップのルートとして使用した場合は、デフォルトですべての添付ファイルがマップ全体に伝搬します。
- 「/」をマップのルートとして使用した場合は、添付ファイルの伝搬を制御できます。

複数パーツ・メッセージでの WSDL 文書スタイルのバインディングの使用:

Web Services Interoperability Organization (WS-I) 組織により、相互運用性の実現を目的として、WSDL を使用した Web サービスの記述方法と、対応する SOAP メッセージの形成方法に関する一連の規則が定義されました。

これらの規則は、WS-I *Basic Profile バージョン 1.1* (http://www.ws-i.org/Profiles/BasicProfile-1.1.html) に明記されています。具体的には、「文書リテラル・バインディングは、soap:Body と共にエンベロープとしてシリアライズする必要があり、その soap:Body の子エレメントは、対応する wsdl:message パーツによって参照されるグローバル・エレメント宣言のインスタンスである」と WS-I Basic Profile 1.1 R2712 に明記されています。

つまり、複数のパーツを使用してメッセージ (入力、出力、または障害) が定義されている操作に文書スタイルの SOAP バインディングを使用する場合、WS-I Basic Profile 1.1 に準拠するには、これらのパーツのうち 1 つしか SOAP 本体とバインドできないことになります。

さらに、WS-I Attachments Profile 1.0 R2941 には、「記述内の wsdl:binding は、soapbind:body、soapbind:header、soapbind:fault、soapbind:headerfault、mime:content のいずれかを参照する wsdl:portType 内の wsdl:message のすべての wsdl:part をバインドする必要がある」と明記されています。

つまり、複数のパーツによってメッセージ (入力、出力、または障害) が定義されている操作に対して文書スタイルの SOAP バインディングを使用する場合、SOAP 本体とのバインド対象として選択された以外のパーツは、すべて添付ファイルまたはヘッダーとしてバインドする必要があるということです。

この場合、Web サービス (JAX-WS および JAX-RPC) のバインディングを使用したエクスポート用に WSDL 記述を生成する際には、以下のアプローチが使用されます。

- 非バイナリー・タイプのエレメントが複数ある場合は、SOAP 本体にバインドされるメッセージ・パーツを選択することができます。非バイナリー・タイプのエレメントが 1 つだけの場合は、そのエレメントが自動的に SOAP 本体にバインドされます。
- JAX-WS バインディングの場合、タイプ「hexBinary」または「base64Binary」のその他のメッセージ・パーツはすべて、参照されている添付ファイルとしてバインドされます。 76 ページの『参照されている添付ファイル: 最上位メッセージ・パーツ』を参照してください。
- それ以外のメッセージ・パーツはすべて SOAP ヘッダーとしてバインドされます。

JAX-RPC および JAX-WS インポート・バインディングでは、複数パーツの文書スタイル・メッセージを持つ既存の WSDL 文書において、複数のパーツが SOAP 本体にバインドされる場合でも、SOAP バインディングが尊重されます。ただし、Rational Application Developer でこのような WSDL 文書用の Web サービス・クライアントを生成することはできません。

注: JAX-RPC バインディングでは、添付ファイルはサポートされていません。

したがって、文書スタイルの SOAP バインディングを使用する操作で複数パーツ・メッセージを使用する場合は、以下のパターンが推奨されます。

1. 文書/リテラル折り返しスタイルを使用します。この場合、メッセージに含まれるパーツは必ず 1 つだけになります。ただし、この場合の添付ファイルは、参照されていない添付ファイル (82 ページの『参照されていない添付ファイル』を参照) または swaRef タイプの添付ファイル (72 ページの『参照されている添付ファイル: swaRef タイプのエレメント』を参照) にする必要があります。
2. RPC/リテラル・スタイルを使用します。この場合、 SOAP 本体にバインドされるパーツの数について WSDL バインディングに課せられる制限はありません。結果的に SOAP メッセージには必ず、呼び出される操作を表す 1 つの子と、そのエレメントの子となる複数のメッセージ・パーツが含まれます。
3. JAX-WS バインディングの場合、他の非バイナリー・パーツを SOAP ヘッダーにバインドすることを許可する場合を除き、「hexBinary」タイプまたは「base64Binary」タイプ以外のメッセージ・パーツは最大でも 1 つだけにする必要があります。
4. それ以外の場合は、既述の動作に従います。

注: WS-I *Basic Profile Version 1.1* に準拠しない SOAP メッセージを使用する場合は、追加の制約事項があります。

- 最初のメッセージ・パーツは非バイナリーにする必要があります。
- 添付ファイルが参照されている、複数パーツ文書スタイルの SOAP メッセージを受信する場合、JAX-WS バインディングでは、href 属性値で添付ファイルのコンテンツ ID を使用して添付ファイルが識別されている、SOAP 本体内の子エレメントで、参照されている各添付ファイルが表されている必要があります。JAX-WS バインディングは、そのようなメッセージの参照されている添付ファイルを同じ方法で送信します。この動作は、WS-I Basic Profile には準拠していません。

  メッセージを Basic Profile に準拠させるには、上記のパターン 1 または 2 に従ってください。あるいは、こうしたメッセージでは、参照されている添付ファイルを使用せずに、参照されていない添付ファイルか swaRef タイプの添付ファイルを使用してください。

HTTP バインディング

HTTP バインディングは、Service Component Architecture (SCA) と HTTP の接続を提供する目的で設計されています。これにより、既存のまたは新規作成された HTTP アプリケーションをサービス指向アーキテクチャー (SOA) 環境内に組み込むことができます。

Hypertext Transfer Protocol (HTTP) は、Web 上での情報転送プロトコルとして広く使用されています。HTTP プロトコルを使用する外部アプリケーションを扱う場合には、HTTP バインディングが必要です。HTTP バインディングは、メッセージとして渡されたデータのネイティブ・フォーマットから SCA アプリケーションのビジネス・オブジェクトへの変換を処理します。着信メッセージの場合、HTTP バインディングは、ビジネス・オブジェクトとして渡されたデータを、外部アプリケーションが期待するネイティブ・フォーマットに変換することもできます。

注: Web サービスの SOAP/HTTP プロトコルを使用するクライアントおよびサーバーと対話する場合は、Web サービス・バインディングのいずれかを使用することを検討してください。これにより、Web サービスの標準サービス品質の処理に関する追加機能が提供されます。

以下に、HTTP バインディングを使用する一般的なシナリオをリストして説明します。

- SCA によってホストされるサービスは、HTTP インポートを使用して HTTP アプリケーションを呼び出すことができます。
- SCA によってホストされるサービスは、HTTP 対応アプリケーションとして自己公開することができます。その場合、HTTP クライアントは HTTP エクスポートを使用して、これらのサービスを使用できるようになります。
- IBM Business Process Manager および Process Server は、HTTP インフラストラクチャーを介して互いに通信できるため、ユーザーは企業の標準に従って通信を管理できます。
- IBM Business Process Manager および Process Server は、HTTP 通信のメディエーターとしてメッセージを変換およびルーティングし、HTTP ネットワークを使用するアプリケーションの統合を強化することができます。
- IBM Business Process Manager および Process Server は、HTTP とその他のプロトコル (SOAP/HTTP Web サービス、Java Connector Architecture (JCA) ベースのリソース・アダプター、JMS など) との間のブリッジとして使用できます。

HTTP インポートおよびエクスポート・バインディングの作成に関する詳細情報については、Integration Designerインフォメーション・センターを参照してください。**『統合アプリケーションの開発』 > 『HTTP を使用した外部サービスへのアクセス』**>トピックを参照してください。

HTTP バインディングの概要:

HTTP バインディングは、HTTP がホストするアプリケーションに接続を提供します。 HTTP アプリケーション間の通信をメディエーションし、既存の HTTP ベースのアプリケーションをモジュールから呼び出せるようにします。

HTTP インポート・バインディング

HTTP インポート・バインディングは、Service Component Architecture (SCA) アプリケーションから HTTP サーバーまたはアプリケーションへのアウトバウンド接続を提供します。

インポートは、HTTP エンドポイントの URL を呼び出します。この URL は、以下の 3 つの方法のいずれかで指定できます。

- 動的オーバーライド URL を使用して、HTTP ヘッダー内に URL を動的に設定します。
- SMO ターゲット・アドレス・エレメント内に URL を動的に設定します。
- インポートの構成プロパティーとして URL を指定します。

本来、この呼び出しは常に同期呼び出しとなります。

HTTP 呼び出しは常に要求/応答となりますが、HTTP インポートは片方向操作と両方向操作の両方をサポートし、操作が片方向の場合には応答を無視します。

HTTP エクスポート・バインディング

HTTP エクスポート・バインディングは、HTTP アプリケーションから SCA アプリケーションへのインバウンド接続を提供します。

URL は、HTTP エクスポートで定義されます。要求メッセージをエクスポートに送信する必要がある HTTP アプリケーションは、この URL を使用してエクスポートを呼び出します。

HTTP エクスポートは ping もサポートします。

実行時の HTTP バインディング

実行時に HTTP バインディングを使用するインポートは、メッセージ本体 (データの有無に関わらず) を使用して、SCA アプリケーションから外部 Web サービスに要求を送信します。要求は、図 25 に示すように、SCA アプリケーションから外部 Web サービスに対して行われます。

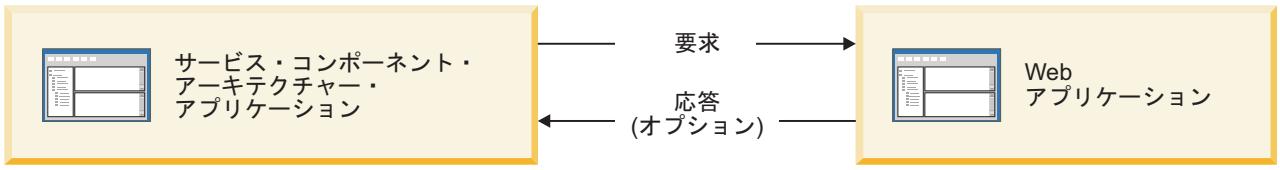


図 25. SCA アプリケーションから Web アプリケーションへの要求フロー

オプションで、HTTP バインディングを使用するインポートは要求に対する応答として Web アプリケーションからのデータを受信することができます。

エクスポートの場合は、図 26 に示すように、クライアント・アプリケーションが Web サービスに対して要求を行います。

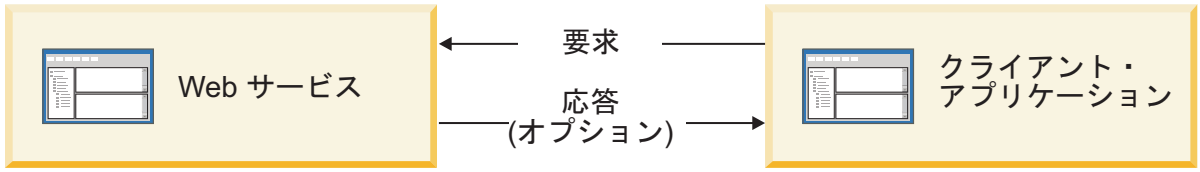


図 26. クライアント・アプリケーションから Web サービスへの要求フロー

Web サービスは、サーバーで稼働する Web アプリケーションです。この Web アプリケーションでは、エクスポートはサーブレットとして実装されるため、クライアントはその要求を URL アドレスに送信します。サーブレットは、実行時に要求を SCA アプリケーションに渡します。

オプションで、エクスポートは要求に対する応答として、クライアント・アプリケーションにデータを送信することができます。

HTTP ヘッダー:

HTTP インポートおよびエクスポート・バインディングを使用すると、HTTP ヘッダーを構成し、その値をアウトバウンド・メッセージに使用できます。HTTP インポートはそれらのヘッダーを要求に使用し、HTTP エクスポートは応答に使用します。

静的に構成されたヘッダーと制御情報は、実行時に動的に設定される値よりも優先されます。しかし、動的オーバーライド URL、バージョン、およびメソッド制御の値は、静的な値をオーバーライドします。それ以外の場合、静的な値がデフォルト値と見なされます。

バインディングは、実行時に HTTP ターゲットの URL、バージョン、およびメソッド値を判断することにより、HTTP インポート URL の動的性質をサポートします。これらの値は、エンドポイント参照、動的オーバーライド URL、バージョン、メソッドの値を抽出して判別されます。

- エンドポイント参照については、com.ibm.websphere.sca.addressing.EndpointReference API を使用するか、または SMO ヘッダーの /headers/SMOHeader/Target/address フィールドを設定します。
- 動的オーバーライド URL、バージョン、およびメソッドについては、Service Component Architecture (SCA) メッセージの HTTP 制御パラメーター・セクションを使用します。動的オーバーライド URL は、ターゲットのエンドポイント参照よりも優先されることに注意してください。ただし、エンドポイント参照はバインディング全体に適用されるため、望ましいアプローチです。このため、可能であればエンドポイント参照を使用してください。

HTTP エクスポートおよびインポート・バインディングのアウトバウンド・メッセージに関する制御およびヘッダー情報は、次の順序で処理されます。

1. SCA メッセージからの HTTP 動的オーバーライド URL、バージョン、およびメソッドを除くヘッダーおよび制御情報 (最低の優先順位)
2. エクスポート/インポート・レベルでの管理コンソールからの変更
3. エクスポートまたはインポートのメソッド・レベルでの管理コンソールからの変更
4. エンドポイント参照または SMO ヘッダーによって指定されたターゲット・アドレス
5. SCA メッセージからの動的オーバーライド URL、バージョン、およびメソッド
6. データ・ハンドラーまたはデータ・バインディングからのヘッダーおよび制御情報 (最高の優先順位)

HTTP エクスポートおよびインポートは、プロトコル・ヘッダーの伝搬が **True** に設定されている場合にのみ、インバウンド方向のヘッダーと制御パラメーターに、着信メッセージからのデータ (HTTPExportRequest および HTTPImportResponse) の取り込みを行います。逆に、アウトバウンドのヘッダーと制御パラメーター (HTTPExportResponse および HTTPImportRequest) については、プロトコル・ヘッダーの伝搬が **True** に設定されている場合にのみ、HTTP エクスポートおよびインポートが読み取りと処理を行います。

注: インポート応答またはエクスポート要求のなかでデータ・ハンドラーまたはデータ・バインディングがヘッダーまたは制御パラメーターに変更されても、インポートまたはエクスポート・バインディング内でのメッセージの処理命令が変更されることはありません。そのため、データ・ハンドラーまたはデータ・バインディングは、変更された値をダウンストリームの SCA コンポーネントに伝搬する目的のみで使用する必要があります。

コンテキスト・サービスは、コンテキスト (HTTP ヘッダーなどのプロトコル・ヘッダーや、アカウント ID などのユーザー・コンテキストを含む) を SCA 呼び出しパスに沿って伝搬します。 IBM Integration Designer での開発時には、インポート・プロパティーとエクスポート・プロパティーを使用してコンテキストの伝搬を制御できます。詳しくは、IBM Integration Designer インフォメーション・センターのインポートおよびエクスポート・バインディングに関する情報を参照してください。

提供される HTTP ヘッダー構造とサポート

90 ページの表 22 に、HTTP インポートと HTTP エクスポートの要求および応答の要求/応答パラメーターを項目別に示します。

表 22. 提供される HTTP ヘッダー情報

制御名	HTTP インポート要求	HTTP インポート応答	HTTP エクスポート要求	HTTP エクスポート応答
URL	無視	設定されない	要求メッセージから読み取られる。 **注:** 照会ストリングも URL 制御パラメーターの一部です。	無視
バージョン (可能な値: 1.0、1.1。デフォルトは 1.1)	無視	設定されない	要求メッセージから読み取られる	無視
メソッド	無視	設定されない	要求メッセージから読み取られる	無視
動的オーバーライド URL	データ・ハンドラーまたはデータ・バインディングに設定された場合、HTTP Import URL をオーバーライドする。要求行のメッセージに書き込まれる。 **注:** 照会ストリングも URL 制御パラメーターの一部です。	設定されない	設定されない	無視
動的オーバーライド・バージョン	設定された場合、HTTP Import バージョンをオーバーライドする。要求行のメッセージに書き込まれる。	設定されない	設定されない	無視
動的オーバーライド・メソッド	設定された場合、HTTP Import メソッドをオーバーライドする。要求行のメッセージに書き込まれる。	設定されない	設定されない	無視
メディア・タイプ (この制御パラメーターは、Content-Type HTTP ヘッダーの値の一部を搬送します。)	存在する場合、メッセージに Content-Type ヘッダーの一部として書き込まれる。 **注:** この制御エレメント値は、データ・ハンドラーまたはデータ・バインディングによって提供してください。	応答メッセージ、Content-Type ヘッダーから読み取られる	要求メッセージ、Content-Type ヘッダーから読み取られる	存在する場合、メッセージに Content-Type ヘッダーの一部として書き込まれる。 **注:** この制御エレメント値は、データ・ハンドラーまたはデータ・バインディングによって提供してください。

表 22. 提供される HTTP ヘッダー情報 (続き)

制御名	HTTP インポート要求	HTTP インポート応答	HTTP エクスポート要求	HTTP エクスポート応答
文字セット (デフォルト: UTF-8)	存在する場合、メッセージに Content-Type ヘッダーの一部として書き込まれる。 **注:** この制御エレメント値は、データ・バインディングによって提供してください。	応答メッセージ、Content-Type ヘッダーから読み取られる	要求メッセージ、Content-Type ヘッダーから読み取られる	サポート対象。メッセージに Content-Type ヘッダーの一部として書き込まれる。 **注:** この制御エレメント値は、データ・バインディングによって提供してください。
転送エンコード (可能な値: chunked、identity。デフォルトは identity)	存在する場合、メッセージにヘッダーとして書き込まれ、メッセージ変換のエンコードの方法を制御する。	応答メッセージから読み取られる	要求メッセージから読み取られる	存在する場合、メッセージにヘッダーとして書き込まれ、メッセージ変換のエンコードの方法を制御する。
コンテンツ・エンコード (可能な値: gzip、x-gzip、 deflate、identity。デフォルトは identity)	存在する場合、メッセージにヘッダーとして書き込まれ、ペイロードのエンコードの方法を制御する。	応答メッセージから読み取られる	要求メッセージから読み取られる	存在する場合、メッセージにヘッダーとして書き込まれ、ペイロードのエンコードの方法を制御する。
Content-Length	無視される	応答メッセージから読み取られる	要求メッセージから読み取られる	無視される
StatusCode (デフォルト: 200)	サポート対象外	応答メッセージから読み取られる	サポート対象外	存在する場合、応答行のメッセージに書き込まれる。
ReasonPhrase (デフォルト: OK)	サポート対象外	応答メッセージから読み取られる	サポート対象外	制御値は無視される。メッセージ応答行の値は StatusCode から生成される。
認証 (複数のプロパティーを含む)	存在する場合、基本認証ヘッダーの作成に使用される。 **注:** このヘッダーの値は、HTTP プロトコル上でのみエンコードされます。SCA では、これはデコードされ、平文として渡されます。	適用しません	要求メッセージの基本認証ヘッダーから読み取られる。このヘッダーの存在は、ユーザーが認証済みであることを示すものではありません。認証は、サーブレット構成で制御する必要があります。 **注:** このヘッダーの値は、HTTP プロトコル上でのみエンコードされます。SCA では、これはデコードされ、平文として渡されます。	適用しません

表 22. 提供される HTTP ヘッダー情報 (続き)

制御名	HTTP インポート要求	HTTP インポート応答	HTTP エクスポート要求	HTTP エクスポート応答
プロキシー (次のような複数のプロパティーを含む: Host、 Port、 Authentication)	存在する場合、プロキシーによる接続を確立するために使用される。	適用しません	適用しません	適用しません
SSL (次のような複数のプロパティーを含む: Keystore、 Keystore Password、 Trustore、 Trustore Password、 ClientAuth)	入力され、宛先 URL が HTTPS の場合、これは SSL による接続を確立するために使用される。	適用しません	適用しません	適用しません

HTTP データ・バインディング:

Service Component Architecture (SCA) メッセージと HTTP プロトコル・メッセージ間のデータの異なるマッピングごとに、データ・ハンドラーまたは HTTP データ・バインディングを構成する必要があります。データ・ハンドラーは、バインディングに関して中立的なインターフェースを提供します。このインターフェースは複数のトランスポート・バインディングに渡って再利用が可能であり、推奨される方法を表します。データ・バインディングは特定のトランスポート・バインディングに固有です。HTTP 固有のデータ・バインディング・クラスが提供されるほか、カスタム・データ・ハンドラーまたはデータ・バインディングを作成することもできます。

注: このトピックで説明した 3 つの HTTP データ・バインディング・クラス (HTTPStreamDataBindingSOAP、HTTPStreamDataBindingXML、および HTTPServiceGatewayDataBinding) は、IBM Business Process Manager バージョン 7.0 では推奨されていません。このトピックで説明したデータ・バインディングを使用する代わりに、次のデータ・ハンドラーの使用を検討してください。

- HTTPStreamDataBindingSOAP の代わりに SOAPDataHandler を使用する
- HTTPStreamDataBindingXML の代わりに UTF8XMLDataHandler を使用する
- HTTPServiceGatewayDataBinding の代わりに GatewayTextDataHandler を使用する

HTTP インポートおよび HTTP エクスポートに使用するデータ・バインディングが提供されます。それらは、バイナリー・データ・バインディング、XML データ・バインディング、および SOAP データ・バインディングです。応答データ・バインディングは、片方向の操作には必要ありません。データ・バインディングは、インスタンスが HTTP と ServiceDataObject の間で両方向に変換可能な Java クラスの名前によって表されます。エクスポートでは関数セレクターを使用する必要があります。関数セレクターは、使用されるデータ・バインディングと呼び出される操作を、メソッド・バインディングと連動して判別できます。提供されるデータ・バインディングは、次のとおりです。

- バイナリー・データ・バインディング。本文を非構造化バイナリー・データとして取り扱います。バイナリー・データ・バインディングの XSD スキーマの実装は、次のとおりです。

```
<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://com.ibm.websphere.http.data.bindings/schema"
  xmlns:tns="http://com.ibm.websphere.http.data.bindings/schema"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="HTTPBaseBody">
    <xsd:sequence/>
  </xsd:complexType>
```

```
<xsd:complexType name="HTTPBytesBody">
  <xsd:complexContent>
    <xsd:extension base="tns:HTTPBaseBody">
      <xsd:sequence>
        <xsd:element name="value" type="xsd:hexBinary"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

- XML データ・バインディング。本文を XML データとしてサポートします。XML データ・バインディングの実装は JMS XML データ・バインディングに類似しており、インターフェース・スキーマについての制限はありません。
- SOAP データ・バインディング。本文を SOAP データとしてサポートします。SOAP データ・バインディングの実装には、インターフェース・スキーマについての制限はありません。

カスタム HTTP データ・バインディングの実装

このセクションでは、カスタム HTTP データ・バインディングを実装する方法を説明します。

注: 推奨される方法は、カスタム・データ・ハンドラーを実装する方法です。カスタム・データ・ハンドラーは複数のトランスポート・バインディング間で再利用できるためです。

HTTPStreamDataBinding は、カスタム HTTP メッセージを処理するための主要なインターフェースです。このインターフェースは大きなペイロードを処理できるように設計されています。しかし、そのような実装を機能させるために、このデータ・バインディングは、ストリームにメッセージを書き込む前に制御情報とヘッダーを返す必要があります。

メソッドとその実行順序 (以下を参照) は、カスタム・データ・バインディングによって実装する必要があります。

データ・バインディングをカスタマイズするには、HTTPStreamDataBinding を実装するクラスを作成します。データ・バインディングには、以下の 4 つの private プロパティーが必要です。

- private DataObject pDataObject
- private HTTPControl pCtrl
- private HTTPHeaders pHeaders
- private yourNativeDataType nativeData

HTTP バインディングは、カスタマイズされたデータ・バインディングを以下の順序で呼び出します。

- アウトバウンド処理 (DataObject からネイティブ・フォーマットへ):
  1. setDataObject(...)
  2. setHeaders(...)
  3. setControlParameters(...)
  4. setBusinessException(...)
  5. convertToNativeData()
  6. getControlParameters()
  7. getHeaders()
  8. write(...)
- インバウンド処理 (ネイティブ・フォーマットから DataObject へ):
  1. setControlParameters(...)

2. setHeaders(...)
3. convertFromNativeData(...)
4. isBusinessException()
5. getDataObject()
6. getControlParameters()
7. getHeaders()

convertFromNativeData(...) で setDataObject(...) を呼び出して、dataObject の値を設定する必要があります。dataObject は、ネイティブ・データから private プロパティー「pDataObject」に変換されます。

```
public void setDataObject(DataObject dataObject)
   throws DataBindingException {
  pDataObject = dataObject;

}
public void setControlParameters(HTTPControl arg0) {
  this.pCtrl = arg0;
}

public void setHeaders(HTTPHeaders arg0) {
  this.pHeaders = arg0;
}
/*
* pHeaders に HTTP ヘッダー「IsBusinessException」を追加します。
* 以下の 2 つのステップを実行します。
* 1. まず IsBusinessException という名前 (大/小文字を区別しない) を持つ
*    ヘッダーをすべて削除します。
*    これは、1 つのヘッダーのみが存在するようにするためです。
* 2. 新しいヘッダー「IsBusinessException」を追加します。
*/
public void setBusinessException(boolean isBusinessException) {
  // まず IsBusinessException という名前 (大/小文字を区別しない) を持つ
  // ヘッダーをすべて削除します。
  // これは、1 つのヘッダーのみが存在するようにするためです。
  // 新しいヘッダー「IsBusinessException」を追加します。
  // コード例は以下のとおりです。
  HTTPHeader header=HeadersFactory.eINSTANCE.createHTTPHeader();
  header.setName("IsBusinessException");
  header.setValue(Boolean.toString(isBusinessException));
  this.pHeaders.getHeader().add(header);

}
public HTTPControl getControlParameters() {
  return pCtrl;
}
public HTTPHeaders getHeaders() {
  return pHeaders;
}
public DataObject getDataObject() throws DataBindingException {
  return pDataObject;
}
/*
* pHeaders からヘッダー「IsBusinessException」を取得し、そのブール値を返す
*/
public boolean isBusinessException() {
  String headerValue = getHeaderValue(pHeaders,"IsBusinessException");
  boolean result=Boolean.parseBoolean(headerValue);
  return result;
}
public void convertToNativeData() throws DataBindingException {
  DataObject dataObject = getDataObject();
  this.nativeData=realConvertWorkFromSDOToNativeData(dataObject);
}
```

```
public void convertFromNativeData(HTTPInputStream arg0){
  //HTTPInputStream からデータを読み取る
  //ユーザー開発のメソッド
  // DataObject に変換する
  DataObject dataobject=realConvertWorkFromNativeDataToSDO(arg0);
  setDataObject(dataobject);
}
public void write(HTTPOutputStream output) throws IOException {
  if (nativeData != null)
  output.write(nativeData);
}
```

EJB バインディング

Enterprise JavaBeans (EJB) インポート・バインディングにより、Service Component Architecture (SCA) コンポーネントは、Java EE サーバー上で実行される Java EE ビジネス・ロジックで提供されるサービスを起動することができます。EJB エクスポート・バインディングにより、 SCA コンポーネントを Enterprise JavaBeans として公開することができます。これにより、Java EE ビジネス・ロジックは、これ以外の方法では使用できない SCA コンポーネントを起動できるようになります。

EJB インポート・バインディング:

EJB インポート・バインディングでは、コンシュームする側のモジュールと外部 EJB とをバインドする方法を指定することによって、SCA モジュールが EJB 実装を呼び出すことができます。外部 EJB 実装からサービスをインポートすることによって、ユーザーはビジネス・ロジックを IBM Business Process Manager 環境に接続でき、ビジネス・プロセスに参加することができます。

Integration Designer を使用して EJB インポート・バインディングを作成します。以下のいずれかの手順により、バインディングを生成することができます。

- 外部サービス・ウィザードを使用して EJB インポート・バインディングを作成する

  Integration Designer の外部サービス・ウィザードを使用し、既存の実装に基づいて EJB インポートを作成できます。外部サービス・ウィザードは、指定された基準に基づいてサービスを作成します。次に、ディスカバーされたサービスに基づいて、ビジネス・オブジェクト、インターフェース、インポート・ファイルを生成します。
- アセンブリー・エディターを使用して EJB インポートを作成する

  Integration Designer アセンブリー・エディターを使用して、アセンブリー・ダイアグラム内に EJB インポートを作成できます。パレットで、インポートを使用するか、Java クラスを使用して、EJB バインディングを作成できます。

生成されたインポートには、Java ブリッジ・コンポーネントを必要とする代わりに Java-WSDL 接続を行うデータ・バインディングがあります。Web Services Description Language (WSDL) 参照を持つコンポーネントを、Java インターフェースを使って EJB ベースのサービスと通信する EJB インポートに直接接続できます。

EJB インポートは、EJB 2.1 プログラミング・モデルまたは EJB 3.0 プログラミング・モデルを使用して、Java EE ビジネス・ロジックと対話することができます。

Java EE ビジネス・ロジックの呼び出しは、ローカル (EJB 3.0 の場合のみ) でもリモートでもかまいません。

- ローカル呼び出しは、インポートと同じサーバーに存在する Java EE ビジネス・ロジックを呼び出す場合に使用します。

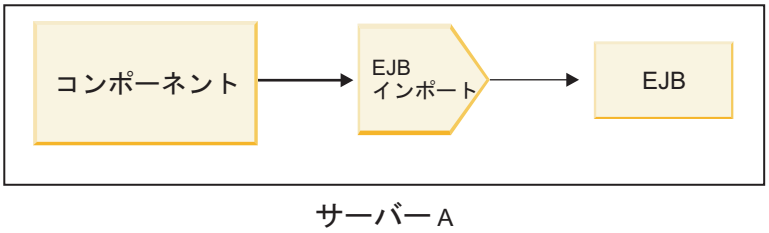


図 27. EJB (EJB 3.0 のみ) のローカル呼び出し

- リモート呼び出しは、インポートと同じサーバーに存在しない Java EE ビジネス・ロジックを呼び出す場合に使用します。

  例えば、次の図で、EJB インポートは、Remote Method Invocation over Internet InterORB Protocol (RMI/IIOP) を使用して、別のサーバーの EJB メソッドを呼び出しています。

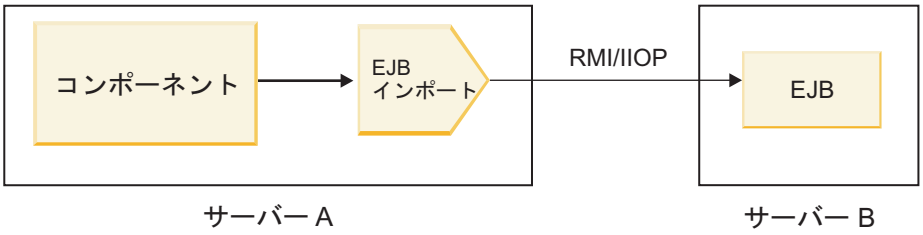


図 28. EJB のリモート呼び出し

EJB バインディングの構成時に、Integration Designer は JNDI 名を使用して、EJB プログラミング・モデル・レベルと呼び出しのタイプ (ローカルまたはリモート) を判別します。

EJB インポート・バインディングには、以下の主なコンポーネントが含まれています。

- JAX-WS データ・ハンドラー
- EJB 障害セレクター
- EJB インポート関数セレクター

ユーザー・シナリオが JAX-WS マッピングに基づいていない場合は、タスクを実行するために、カスタム・データ・ハンドラー、関数セレクター、および障害セレクターが必要になることがあります。このタスクは、ユーザー・シナリオがこのマッピングに基づいている場合には、EJB インポート・バインディングを構成するコンポーネントによって実行されるタスクです。これにはカスタム・マッピング・アルゴリズムによって通常実行されるマッピングが含まれます。

EJB エクスポート・バインディング:

外部 Java EE アプリケーションは、EJB エクスポート・バインディングによって SCA コンポーネントを呼び出すことができます。 EJB エクスポートを使用することにより、SCA コンポーネントを公開することができます。これにより、外部 Java EE アプリケーションは、 EJB プログラミング・モデルを使用してこれらのコンポーネントを呼び出せるようになります。

注: EJB エクスポートは、ステートレス Bean です。

Integration Designer を使用して EJB バインディングを作成します。以下のいずれかの手順により、バインディングを生成することができます。

- 外部サービス・ウィザードを使用して EJB エクスポート・バインディングを作成する

Integration Designer の外部サービス・ウィザードを使用し、既存の実装に基づいて EJB エクスポート・サービスを作成できます。外部サービス・ウィザードは、指定された基準に基づいてサービスを作成します。次に、ディスカバーされたサービスに基づいて、ビジネス・オブジェクト、インターフェース、エクスポート・ファイルを生成します。

- アセンブリー・エディターを使用して EJB エクスポート・バインディングを作成する

  Integration Designer アセンブリー・エディターを使用して、EJB エクスポートを作成できます。

重要: Java 2 Platform, Standard Edition (J2SE) クライアントは、Integration Designer で生成された EJB エクスポート・クライアントを呼び出すことができません。

既存の SCA コンポーネントからバインディングを生成できます。または、Java インターフェース用の EJB バインディングを使ってエクスポートを生成できます。

- 既存の WSDL インターフェースを持つ既存の SCA コンポーネント用のエクスポートを生成するときは、エクスポートは Java インターフェースに割り当てられます。
- Java インターフェース用のエクスポートを生成するときは、そのエクスポート用に WSDL または Java インターフェースを選択できます。

  注: EJB エクスポートの作成で使用する Java インターフェースには、リモート呼び出しでパラメーターとして渡されるオブジェクト (入出力パラメーターと例外) に関して次のような制約があります。
  - 具象タイプである必要がある (インターフェースや抽象タイプではない)。
  - これらは、Enterprise JavaBean の仕様に準拠している必要があります。これらは、直列化可能で、デフォルトの引数のないコンストラクターを持ち、すべてのプロパティーは getter メソッドおよび setter メソッドを使ってアクセスできる必要があります。

    Enterprise JavaBean の仕様については、Sun Microsystems, Inc. の Web サイト (http://java.sun.com) を参照してください。

  また、例外はチェック例外である必要があり、java.lang.Exception から継承され、単数形である必要があります (つまり、複数のチェック例外タイプのスローをサポートしていません)。

  Java EnterpriseBean のビジネス・インターフェースは簡潔な Java インターフェースであり、javax.ejb.EJBObject または javax.ejb.EJBLocalObject を拡張してはいけないという点に留意してください。ビジネス・インターフェースのメソッドは、java.rmi.Remote.Exception をスローしてはいけません。

EJB エクスポート・バインディングは、EJB 2.1 プログラミング・モデルまたは EJB 3.0 プログラミング・モデルを使用して、 Java EE ビジネス・ロジックと対話することができます。

呼び出しは、ローカル (EJB 3.0 の場合のみ) でもリモートでもかまいません。

- ローカルの呼び出しは、Java EE ビジネス・ロジックがエクスポートと同じサーバー上にある SCA コンポーネントを呼び出したときに使われます。
- リモートの呼び出しは、Java EE ビジネス・ロジックがエクスポートと同じサーバー上にないときに使われます。

  例えば、次の図で、EJB は RMI/IIOP を使用して、別のサーバー上の SCA コンポーネントを呼び出しています。

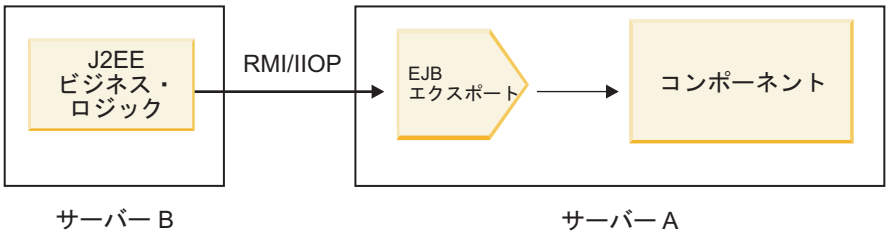


図 29. EJB エクスポートによるクライアントから SCA コンポーネントに対するリモート呼び出し

EJB バインディングの構成時に、Integration Designer は JNDI 名を使用して、EJB プログラミング・モデル・レベルと呼び出しのタイプ (ローカルまたはリモート) を判別します。

EJB エクスポート・バインディングには、以下の主なコンポーネントが含まれています。

- JAX-WS データ・ハンドラー
- EJB エクスポート関数セレクター

お使いのユーザー・シナリオが JAX-WS マッピングに基づいていない場合は、タスクを実行するためにカスタム・データ・ハンドラーおよび関数セレクターが必要になることがあります。そうでない場合は、EJB エクスポート・バインディングの一部であるコンポーネントにより処理が完了します。これにはカスタム・マッピング・アルゴリズムによって通常実行されるマッピングが含まれます。

EJB バインディング・プロパティー:

EJB インポート・バインディングは、構成済みの JNDI 名を使用して、EJB プログラミング・モデル・レベルと呼び出しのタイプ (ローカルまたはリモート) を判別します。EJB インポートおよびエクスポート・バインディングは、データ変換に JAX-WS データ・ハンドラーを使用します。EJB インポート・バインディングは EJB インポート関数セレクターと EJB 障害セレクターを使用し、EJB エクスポート・バインディングは EJB エクスポート関数セレクターを使用します。

JNDI 名および EJB インポート・バインディング:

インポートに対する EJB バインディングの構成時に、Integration Designer は JNDI 名を使用して、EJB プログラミング・モデル・レベルと呼び出しのタイプ (ローカルまたはリモート) を判別します。

JNDI 名が指定されていない場合、デフォルトの EJB インターフェース・バインディングが使用されます。作成されるデフォルトの名前は、EJB 2.1 JavaBean と EJB 3.0 JavaBean のいずれを呼び出すかによって異なります。

注: 命名規則に関する詳細については、WebSphere Application Server インフォメーション・センターのEJB 3.0 アプリケーション・バインディングの概要を参照してください。

- EJB 2.1 JavaBean

  Integration Designer によって事前選択されているデフォルトの JNDI 名は、デフォルトの EJB 2.1 バインディングであり、**ejb/** の後にスラッシュで区切られたホーム・インターフェースが続く形式になっています。

  例えば、com.mycompany.myremotebusinesshome の EJB 2.1 JavaBean のホーム・インターフェースの場合、デフォルトのバインディングは以下のとおりです。

  ```
  ejb/com/mycompany/myremotebusinesshome
  ```

  EJB 2.1 の場合は、リモート EJB 呼び出しのみがサポートされます。

- EJB 3.0 JavaBean

  Integration Designer によって事前選択されているローカル JNDI 用のデフォルトの JNDI 名は、先頭に **ejblocal:** が付されたローカル・インターフェースの完全修飾クラス名になります。例えば、ローカル・インターフェース com.mycompany.mylocalbusiness の完全修飾インターフェースの場合、事前選択された EJB 3.0 JNDI は、以下のとおりです。

  ```
  ejblocal:com.mycompany.mylocalbusiness
  ```

  リモート・インターフェース com.mycompany.myremotebusiness の場合、事前選択された EJB 3.0 JNDI は、以下の完全修飾インターフェースになります。

  ```
  com.mycompany.myremotebusiness
  ```

  EJB 3.0 デフォルト・アプリケーション・バインディングの説明は、EJB 3.0 アプリケーション・バインディングの概要に記載されています。

  Integration Designer は、バージョン 3.0 のプログラミング・モデルを使用する EJB 用のデフォルトの JNDI の場所として短縮名を使用します。

  注: カスタム・マッピングが使用または構成されていたために、ターゲット EJB のデプロイ済み JNDI 参照が、デフォルトの JNDI バインディングの場所と異なっている場合は、ターゲット JNDI 名を正しく指定する必要があります。デプロイメント前に、Integration Designer で名前を指定することができます。あるいは、インポート・バインディングの場合は、(デプロイメント後に) ターゲット EJB の JNDI 名と一致するように管理コンソールで名前を変更することもできます。

EJB バインディングの作成について詳しくは、Integration Designer インフォメーション・センターで EJB バインディングの処理について扱われているセクションを参照してください。

JAX-WS データ・ハンドラー:

Enterprise JavaBeans (EJB) インポート・バインディングは、JAX-WS 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 Java オブジェクト・パラメーターに変換し、Java オブジェクト戻り値を応答ビジネス・オブジェクトに変換します。 EJB エクスポート・バインディングは、JAX-WS データ・ハンドラーを使用して、要求 EJB を要求ビジネス・オブジェクトに変換し、応答ビジネス・オブジェクトを戻り値に変換します。

このデータ・ハンドラーは、Java API for XML Web Services (JAX-WS) 仕様および Java Architecture for XML Binding (JAXB) 仕様を使用して、データを SCA 指定の WSDL インターフェースからターゲット EJB Java インターフェースに (また、逆も同様に) マップします。

注: 現行のサポートは JAX-WS 2.1.1 および JAXB 2.1.3 仕様に制限されています。

EJB バインディング・レベルで指定されたデータ・ハンドラーは、要求、応答、障害、およびランタイム例外を処理するために使用されます。

注: 障害については、faultBindingType 構成プロパティーを指定することによって、障害ごとに特定のデータ・ハンドラーを指定できます。この設定は、EJB バインディング・レベルで指定された値をオーバーライドします。

EJB バインディングに WSDL インターフェースがある場合、デフォルトで JAX-WS データ・ハンドラーが使用されます。このデータ・ハンドラーは、JAX-WS 呼び出しを表す SOAP メッセージからデータ・オブジェクトへの変換には使用できません。

EJB インポート・バインディングは、データ・ハンドラーを使用して、データ・オブジェクトを Java オブジェクト配列 (Object[]) に変換します。アウトバウンド通信中は、以下の処理が実行されます。

1. EJB バインディングは、WSDL で指定された内容と一致するように、期待されるタイプ、期待されるエレメント、およびターゲット・メソッド名を BindingContext に設定します。
2. EJB バインディングが、データ変換の必要なデータ・オブジェクトのための変換メソッドを呼び出します。
3. データ・ハンドラーは、(メソッド内部での定義の順で) メソッドのパラメーターを表す Object[] を返します。
4. EJB バインディングは Object[] を使用して、ターゲット EJB インターフェース上のメソッドを呼び出します。

また、バインディングは EJB 呼び出しからの応答を処理するために Object[] も作成します。

- Object[] の最初のエレメントは、Java メソッド呼び出しからの戻り値です。
- subsequent 値は、メソッドの入力パラメーターを表します。

これは In/Out タイプおよび Out タイプのパラメーターをサポートするために必要です。

Out タイプのパラメーターの場合、値は応答データ・オブジェクト内で返される必要があります。

データ・ハンドラーは Object[] 内で検出した値を処理および変換して、データ・オブジェクトに応答を返します。

データ・ハンドラーは xs:AnyType、xs:AnySimpleType、および xs:Any のほか、他の XSD データ型もサポートします。xs:Any へのサポートを使用可能にするには、以下の例に示すように Java コードの JavaBean プロパティーで **@XmlAnyElement (lax=true)** を使用します。

```
public class TestType {
 private Object[] object;

 @XmlAnyElement (lax=true)
 public Object[] getObject() {
  return object;
 )

 public void setObject (Object[] object) {
  this.object=object;
 )
)
```

これによって、TestType のプロパティー・オブジェクトが xs:any フィールドになります。xs:any フィールド内で使用される Java クラスの値は、**@XmlAnyElement** アノテーションを持つ必要があります。例えば Address がオブジェクト配列の設定に使用される Java クラスである場合、Address クラスはアノテーション **@XmlRootElement** を持つ必要があります。

注: JAX-WS 仕様で定義された XSD タイプから Java タイプへのマッピングをカスタマイズするには、ビジネス・ニーズに合うように JAXB アノテーションを変更します。JAX-WS データ・ハンドラーは xs:any、xs:anyType、および xs:anySimpleType をサポートします。

JAX-WS データ・ハンドラーには以下の制約が適用されます。

- データ・ハンドラーには、ヘッダー属性 **@WebParam** アノテーションのサポートが含まれない。
- ビジネス・オブジェクト・スキーマ・ファイル (XSD ファイル) の名前空間には、Java パッケージ名からのデフォルト・マッピングが含まれない。package-info.java のアノテーション **@XMLSchema** も機能

しません。名前空間で XSD を作成するための方法は、**@XmlType** および **@XmlRootElement** アノテーションを使用する方法だけです。**@XmlRootElement** は JavaBean タイプのグローバル・エレメントのターゲット名前空間を定義します。

- EJB インポート・ウィザードは、関連しないクラスについては XSD ファイルを作成しません。バージョン 2.0 では **@XmlSeeAlso** アノテーションがサポートされないため、子クラスが親クラスから直接参照されない場合、XSD は作成されません。この問題の解決方法は、そのような子クラスのために SchemaGen を実行することです。

  SchemaGen は、特定の JavaBean 用の XSD ファイルを作成するためのコマンド行ユーティリティー (場所は *WPS_Install_Home*/bin ディレクトリー) です。この解決方法を実現させるためには、これらの XSD をモジュールに手動でコピーする必要があります。

EJB 障害セレクター:

EJB 障害セレクターは、EJB 呼び出しの結果が、障害、ランタイム例外、または正常な応答のいずれかを判別します。

障害が検出された場合、EJB 障害セレクターがネイティブの障害名をバインディング・ランタイムに返すため、JAX-WS データ・ハンドラーが例外オブジェクトを障害ビジネス・オブジェクトに変換することができます。

正常な (障害のない) 応答の場合、EJB インポート・バインディングは、値を返すために、Java オブジェクト配列 (Object[]) をアセンブルします。

- Object[] の最初のエレメントは、Java メソッド呼び出しからの戻り値です。
- subsequent 値は、メソッドの入力パラメーターを表します。

これは In/Out タイプおよび Out タイプのパラメーターをサポートするために必要です。

例外シナリオの場合、バインディングは Object[] を組み立て、最初のエレメントはメソッドによって throw された例外を表します。

障害セレクターは以下のどの値でも返すことができます。

表 23. 戻り値

タイプ	戻り値	説明
障害	**ResponseType.FAULT**	渡された Object[] に例外オブジェクトが含まれる場合に返されます。
実行時例外	**ResponseType.RUNTIME**	例外オブジェクトが、メソッド上で宣言されたどの例外タイプとも一致しない場合に返されます。
通常応答	**ResponseType.RESPONSE**	その他のすべてのケースで返されます。

障害セレクターが **ResponseType.FAULT** の値を返す場合、ネイティブの障害名を返します。このネイティブの障害名は、対応する WSDL 障害名をモデルから判別し、正しい障害データ・ハンドラーを呼び出すためにバインディングによって使用されます。

EJB 関数セレクター:

EJB バインディングは、インポート関数セレクター (アウトバウンド処理の場合) またはエクスポート関数セレクター (インバウンド処理の場合) を使用して、呼び出す EJB メソッドを判別します。

インポート関数セレクター

アウトバウンド処理の場合、インポート関数セレクターは、EJB インポートにワイヤリングされている SCA コンポーネントによって呼び出される操作の名前に基づいて、EJB メソッド・タイプを派生させます。関数セレクターは、Integration Designer によって生成された JAX-WS アノテーション付きの Java クラスで @WebMethod アノテーションを探し、関連付けられているターゲット操作名を判別します。

- @WebMethod アノテーションが存在する場合、関数セレクターは @WebMethod アノテーションを使用して、WSDL メソッドのための正しい Java メソッド・マッピングを判別します。
- @WebMethod アノテーションがない場合、関数セレクターは、Java メソッド名は呼び出された操作名と同じであると想定します。

注: この関数セレクターは、EJB インポートの Java タイプ・インターフェースではなく、EJB インポートの WSDL タイプ・インターフェースでのみ有効です。

関数セレクターは EJB インターフェースのメソッドを表す java.lang.reflect.Method オブジェクトを返します。

関数セレクターは、ターゲット・メソッドからの応答を入れるために Java オブジェクト配列 (Object[]) を使用します。Object[] の最初のエレメントは、WSDL の名前を持つ Java メソッドで、Object[] の 2 番目のエレメントは入力ビジネス・オブジェクトです。

エクスポート関数セレクター

インバウンド処理の場合、エクスポート関数セレクターは、Java メソッドから呼び出されるターゲット・メソッドを派生させます。

エクスポート関数セレクターは、EJB クライアントによって呼び出される Java 操作名を、ターゲット・コンポーネントのインターフェースの操作名にマップします。メソッド名は、ストリングとして返され、ターゲット・コンポーネントのインターフェース・タイプに基づいて、SCA ランタイムによって解決されます。

EIS バインディング

エンタープライズ情報システム (EIS) のバインディングは、SCA コンポーネントと外部 EIS 間の接続を提供します。この通信を可能にするために使用する手段は、JCA 1.5 リソース・アダプターおよび Websphere Adapters をサポートする EIS エクスポートおよび EIS インポートです。

SCA コンポーネントでは、外部 EIS との間でのデータの転送が必要になる場合があります。このような接続を必要とする SCA モジュールを作成する場合は、SCA コンポーネントに加えて、特定の外部 EIS との通信のために EIS バインディングを使用したインポートまたはエクスポートを組み込む必要があります。

WebSphere Integration Developer のリソース・アダプターは、インポートまたはエクスポートのコンテキスト内で使用されます。インポートまたはエクスポートは外部サービス・ウィザードで開発しますが、これを開発するときにリソース・アダプターを組み込みます。アプリケーションが EIS システムのサービスを呼び出せるようにする EIS インポート、または EIS システムのアプリケーションが WebSphere Integration Developer で開発されたサービスを呼び出せるようにする EIS エクスポートは、特定のリソース・アダプターで作成されます。例えば、JD Edwards システムのサービスを呼び出すインポートを作成するには、JD Edwards アダプターを使用することになります。

外部サービス・ウィザードを使用すると、EIS バインディング情報が自動的に作成されます。また別のツールとして、バインディング情報を追加または変更するにはアセンブリー・エディターを使用できます。詳しくは、Integration Designer インフォメーション・センターを参照してください。

EIS バインディングが含まれる SCA モジュールをサーバーにデプロイすると、管理コンソールを使用して、バインディングに関する情報を表示したり、バインディングを構成したりできます。

EIS バインディングの概要:

EIS (エンタープライズ情報システム) バインディングを JCA リソース・アダプターと一緒に使用すると、エンタープライズ情報システム上のサービスにアクセスしたり、またはサービスを EIS で使用可能にすることができます。

以下は、Siebel システムと SAP システム間の接続情報を同期化する方法を説明する ContactSyncModule という名前の SCA モジュールの例です。

1. ContactSync という名前の SCA コンポーネントは、(Siebel Contact という名前の EIS アプリケーション・エクスポートを介して) Siebel への接続に対する変更を listen します。
2. ContactSync SCA コンポーネント自体は、SAP の接続情報を適宜更新するために、(EIS アプリケーション・インポートを介して) SAP アプリケーションを利用します。

接続情報の保管用に使用されるデータ構造は、Siebel システムと SAP システムでは異なるため、ContactSync SCA コンポーネントがマッピングを行う必要があります。

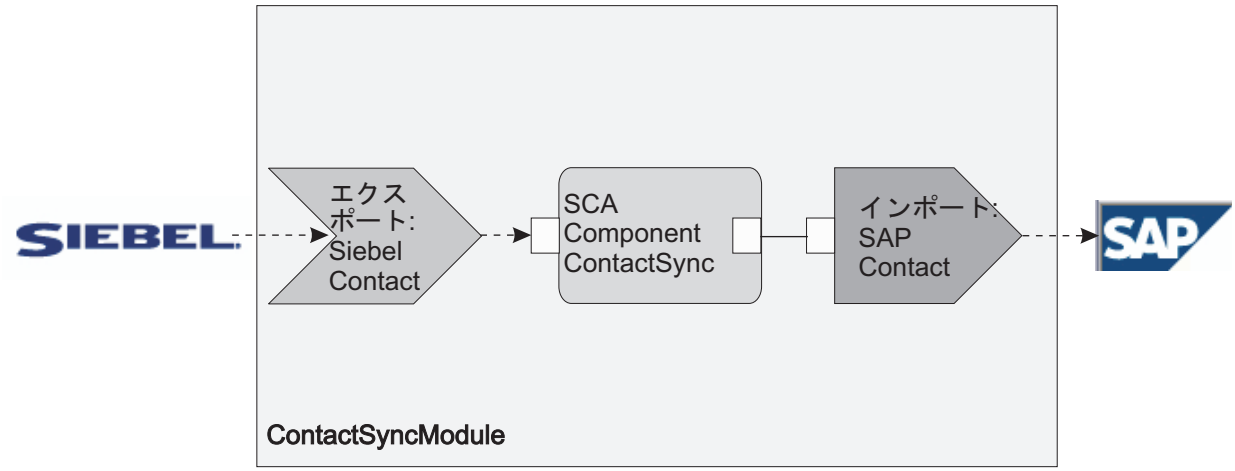


図 30. Siebel システムから SAP システムへのフロー

Siebel Contact エクスポートと SAP Contact インポートには、適切なリソース・アダプターが構成されます。

EIS バインディングの主な特徴:

EIS インポートとは、Service Component Architecture (SCA) モジュール内のコンポーネントが SCA モジュール外部で定義された EIS アプリケーションを使用できるようにする SCA インポートです。EIS インポートは、SCA コンポーネントから外部 EIS へのデータ転送に使用されます。EIS エクスポートは、外部 EIS から SCA モジュールへのデータ転送に使用されます。

インポート

EIS インポートのロールは、SCA コンポーネントと外部 EIS システムの間のすき間を埋めることです。外部アプリケーションを EIS インポートとして取り扱うことができます。この場合、EIS インポートはデータを外部の EIS に送信し、必要に応じて応答データを受信します。

EIS インポートは、SCA コンポーネントがモジュール外部のアプリケーションを統一された形式で認識できるようにします。これにより、コンポーネントは、SAP、 Siebel、PeopleSoft などの外部 EIS と、一貫性のある SCA モデルを使用して通信できます。

インポートのクライアント側には、EIS インポート・アプリケーションによって公開されるインターフェースがあります。ここには、1 つ以上のメソッドがあり、それぞれがデータ・オブジェクトを引数および戻り値として受け入れます。実装側には、リソース・アダプターによって実装される共通クライアント・インターフェース (CCI) があります。

EIS インポートのランタイム実装は、クライアント側のインターフェースとこの CCI を接続します。インポートにより、インターフェース上のメソッドの呼び出しが CCI 上の呼び出しにマップされます。

バインディングは 3 つのレベルで作成されます (インターフェース・バインディングがそれに含まれるメソッド・バインディングを使用し、さらにそのメソッド・バインディングがデータ・バインディングを使用します)。

インターフェース・バインディングは、インポートのインターフェースを、アプリケーションを提供する EIS システムとの接続に関連付けます。これは、インターフェースによって表されるアプリケーション・セットが EIS の特定のインスタンスによって提供されていて、このインスタンスには接続を介してアクセスできるという事実を反映しています。バインディング・エレメントは、接続を作成するための十分な情報を指定したプロパティーを持ちます (このプロパティーは、javax.resource.spi.ManagedConnectionFactory インスタンスの一部です)。

メソッド・バインディングは、メソッドを、EIS システムとの特定の対話に関連付けます。JCA では、この対話の特徴は、javax.resource.cci.InteractionSpec インターフェース実装のプロパティー・セットによって記述されます。メソッド・バインディングの対話エレメントには、これらのプロパティーと共にクラスの名前が含まれています。これにより、対話を実行するための十分な情報を提供します。メソッド・バインディングは、データ・バインディングを使用して、インターフェース・メソッドの引数と結果の EIS 表現へのマッピングを記述します。

EIS インポートのランタイム・シナリオを以下に示します。

1. インポート・インターフェースのメソッドが、SCA プログラミング・モデルを使用して呼び出されます。
2. EIS インポートに送信される要求には、メソッド名と引数が指定されています。
3. インポートは最初にインターフェース・バインディングの実装を作成し、次にインポート・バインディングのデータを使用して ConnectionFactory を作成し、この 2 つを関連付けます。したがって、インポート により、インターフェース・バインディングの setConnectionFactory が呼び出されることになります。
4. 呼び出されたメソッドと一致するメソッド・バインディングの実装が作成されます。
5. javax.resource.cci.InteractionSpec インスタンスが作成され、データが取り込まれます。次に、データ・バインディングを使用して、リソース・アダプターが認識できるフォーマットにメソッド引数がバインドされます。
6. CCI インターフェースを使用して対話が実行されます。

7. 呼び出しが戻されたら、データ・バインディングによって呼び出しの結果が作成され、結果が呼び出し元に戻されます。

エクスポート

EIS エクスポートのロールは、SCA コンポーネントと外部 EIS の間のすき間を埋めることです。外部アプリケーションを EIS エクスポートとして取り扱うことができます。この場合、外部アプリケーションはそのデータを定期的な通知という形で送信します。EIS エクスポートは、EIS からの外部要求を listen するサブスクリプション・アプリケーションであると考えることができます。EIS エクスポートを使用する SCA コンポーネントは、EIS エクスポートをローカル・アプリケーションとして認識します。

EIS エクスポートは、SCA コンポーネントがモジュール外部のアプリケーションを統一された形式で認識できるようにします。これにより、コンポーネントは、SAP、 Siebel、PeopleSoft などの EIS と、一貫性のある SCA モデルを使用して通信できます。

エクスポートは、EIS から要求を受け取るリスナー実装を特徴としています。リスナーは、リソース・アダプター固有のリスナー・インターフェースを実装します。また、エクスポートには、エクスポートを介して EIS に公開されるインターフェースを実装するコンポーネントも含まれます。

EIS エクスポートのランタイム実装は、インターフェースを実装するコンポーネントにリスナーを接続します。エクスポートにより、EIS 要求が、コンポーネント上の該当する操作の呼び出しにマップされます。バインディングは 3 つのレベルで作成されます (リスナー・バインディングがそれに含まれるネイティブ・メソッド・バインディングを使用し、さらにそのネイティブ・メソッド・バインディングがデータ・バインディングを使用します)。

リスナー・バインディングは、要求を受け取るリスナーを、エクスポートを介して公開されるコンポーネントに関連付けます。エクスポート定義には、コンポーネントの名前が含まれています。これにより、ランタイムがコンポーネントを検索し、コンポーネントに要求を転送します。

ネイティブ・メソッド・バインディングは、ネイティブ・メソッドまたはリスナーが受け取ったイベント・タイプを、エクスポートを介して公開されたコンポーネントによって実装される操作に関連付けます。リスナーで呼び出されるメソッドとイベント・タイプの間に関係はなく、すべてのイベントはリスナーの 1 つ以上のメソッドを使用して受信されます。ネイティブ・メソッド・バインディングは、エクスポートで定義された関数セレクターを使用してインバウンド・データからネイティブ・メソッド名を取り出し、データ・バインディングを使用して EIS のデータ・フォーマットをコンポーネントが認識できるフォーマットにバインドします。

EIS エクスポートのランタイム・シナリオを以下に示します。

1. EIS 要求は、リスナー実装のメソッドの呼び出しをトリガーします。
2. リスナーは エクスポートを検出して呼び出し、すべての呼び出し引数を渡します。
3. エクスポートは、リスナー・バインディングの実装を作成します。
4. エクスポートは、関数セレクターをインスタンス化してリスナー・バインディング上に設定します。
5. エクスポートは、ネイティブ・メソッド・バインディングを初期化してリスナー・バインディングに追加します。各ネイティブ・メソッド・バインディングについて、データ・バインディングも初期化されます。
6. エクスポートは、リスナー・バインディングを呼び出します。
7. リスナー・バインディングは、エクスポートされたコンポーネントを検出し、関数セレクターを使用してネイティブ・メソッド名を取得します。

8. この名前を使用して、ネイティブ・メソッド・バインディングを検出します。その後、ネイティブ・メソッド・バインディングによってターゲット・コンポーネントが呼び出されます。

アダプターの対話スタイルでは、EIS エクスポート・バインディングで、ターゲット・コンポーネントを非同期方式 (デフォルト) または同期方式のどちらでも呼び出すことができます。

リソース・アダプター

外部サービス・ウィザードを使用してインポートやエクスポートを開発する際に、リソース・アダプターを組み込みます。IBM Integration Designer に付属するリソース・アダプターは、 CICS、IMS、JD Edwards、PeopleSoft、SAP、Siebel の各システムへのアクセスに使用され、開発とテストのみを目的としています。これらのリソース・アダプターを、アプリケーションの開発とテスト以外の目的では使用しないでください。

デプロイしたアプリケーションを実行するには、ライセンスが交付されたランタイム・アダプターが必要になります。ただし、サービスを構築する際に、このアダプターをサービスに組み込むことができます。アダプターのライセンス交付により、組み込まれたアダプターを、ライセンスが交付されたランタイム・アダプターとして使用することができます。これらのアダプターは、Java EE Connector Architecture (JCA 1.5) に準拠しています。オープン・スタンダードである JCA は、EIS 接続のための Java EE 標準です。 JCA は、管理されたフレームワークを提供します。つまり、サービスの品質 (QoS) がアプリケーション・サーバーによって提供され、それによってライフサイクル管理とセキュリティーがトランザクションに対して提供されます。また、これらのアダプターは、Enterprise Metadata Discovery 仕様にも準拠しています (IBM CICS ECI リソース・アダプターおよび IBM IMS Connector for Java を除く)。

従来のアダプターのセットである WebSphere Business Integration Adapters もウィザードでサポートされています。

Java EE リソース

EIS モジュール (EIS モジュール・パターンに準拠する SCA モジュール) は、Java EE プラットフォームにデプロイできます。

EIS モジュールを Java EE プラットフォームにデプロイすると、アプリケーションが EAR ファイルとしてパッケージされ、サーバーにデプロイされるので、アプリケーションを実行する準備が整います。すべての Java EE 成果物およびリソースが作成され、アプリケーションが構成され、実行の準備が整います。

JCA 対話仕様および接続仕様の動的プロパティー:

EIS バインディングは、ペイロードに付随する明確に定義された子データ・オブジェクトを使用することによって指定された InteractionSpec および ConnectionSpec に対する入力を受け入れることができます。これにより、InteractionSpec を介したリソース・アダプターとの動的要求応答対話と、ConnectionSpec を介したコンポーネント認証が可能になります。

javax.cci.InteractionSpec は、リソース・アダプターとの対話要求の処理方法に関する情報を保持します。また、要求後に対話を行う方法に関する情報も保持します。対話によるこれらの両方向通信は、*会話*とも呼ばれます。

EIS バインディングはペイロードを必要とします。このペイロードは、**properties** という子データ・オブジェクトを格納するためのリソース・アダプターに対する引数になります。このプロパティー・データ・オブジェクトは、名前と値のペアを含みます。この名前は、特定のフォーマットでの対話仕様プロパティーの名前になります。フォーマット設定の規則は以下のとおりです。

- 名前はプレフィックス **IS** で開始し、その後にプロパティー名が続かなければなりません。例えば、**InteractionId** という JavaBeans プロパティーを持つ対話仕様は、プロパティー名を **ISInteractionId** と指定します。
- 名前と値のペアは、対話仕様プロパティーの名前と単純型の値を表します。

この例ではインターフェースによって、操作の入力は「**Account**」データ・オブジェクトと指定されます。このインターフェースは、値が **xyz** である **workingSet** という動的 InteractionSpec プロパティーを送受信するために、EIS インポート・バインディング・アプリケーションを呼び出します。

サーバーのビジネス・グラフまたはビジネス・オブジェクトには下位の「**properties**」ビジネス・オブジェクトが含まれていて、このビジネス・オブジェクトによってペイロードを持つプロトコル固有データを送信できます。この **properties** ビジネス・オブジェクトは組み込みのものであり、ビジネス・オブジェクトを構成するときに XML スキーマで指定する必要はありません。単に作成するだけで使用できます。XML スキーマに基づいて独自のデータ型を定義している場合は、必要な名前と値のペアを含む「**properties**」エレメントを指定する必要があります。

```
BOFactory dataFactory = (BOFactory) ¥
serviceManager.locateService("com/ibm/websphere/bo/BOFactory");
 //Wrapper for doc-lit wrapped style interfaces,
 //skip to payload for non doc-lit
 DataObject docLitWrapper = dataFactory.createByElement /
 ("http://mytest/eis/Account", "AccountWrapper");
```

ペイロードを作成します。

```
DataObject account = docLitWrapper.createDataObject(0);
  DataObject accountInfo = account.createDataObject("AccountInfo");
  //Perform your setting up of payload


  //Construct properties data for dynamic interaction

  DataObject properties = account.createDataObject("properties");
```

名前 workingSet に対して予想される値 (**xyz**) を設定します。

```
properties.setString("ISworkingSet", "xyz");


  //Invoke the service with argument

  Service accountImport = (Service) ¥
  serviceManager.locateService("AccountOutbound");
  DataObject result = accountImport.invoke("createAccount", docLitWrapper);

  //Get returned property
  DataObject retProperties = result.getDataObject("properties");

  String workingset = retProperties.getString("ISworkingSet");
```

ConnectionSpec プロパティーは動的コンポーネント認証に使用できます。上記と同じ規則が適用されます。ただし、プロパティー名のプレフィックスは **CS** にする必要があります (**IS** ではありません)。ConnectionSpec プロパティーは両方向ではありません。同じ **properties** データ・オブジェクトに IS プロパティーと CS プロパティーの両方を入れることができます。

ConnectionSpec プロパティーを使用するには、インポート・バインディングで指定する **resAuth** を「**Application**」に設定します。また、リソース・アダプターがコンポーネント許可をサポートする必要があります。詳しくは、J2EE Connector Architecture Specification の第 8 章を参照してください。

EIS バインディングを持つ外部クライアント:

サーバーは、EIS バインディングを使用して、外部クライアントとの間でメッセージを送受信できます。

外部クライアント (Web ポータルや EIS など) は、サーバーの SCA モジュールにメッセージを送信する必要があり、サーバー内からコンポーネントによって呼び出される必要もあります。

クライアントは、動的起動インターフェース (DII) または Java インターフェースを使用して、その他のアプリケーションの場合と同様に EIS インポートを呼び出します。

1. 外部クライアントが ServiceManager のインスタンスを作成し、その参照名を使用して EIS インポートを検索します。検索の結果は、サービス・インターフェースの実装です。
2. クライアントが入力引数 (総称データ・オブジェクト) を作成します。この引数は、データ・オブジェクト・スキーマを使用して動的に作成されます。このステップは、サービス・データ・オブジェクト DataFactory インターフェースの実装を使用して行われます。
3. 外部クライアントが EIS を呼び出して必要な結果を取得します。

代わりに、クライアントは Java インターフェースを使用して EIS インポートを呼び出すこともできます。

1. クライアントが ServiceManager のインスタンスを作成し、その参照名を使用して EIS インポートを検索します。検索の結果は、EIS インポートの Java インターフェースです。
2. クライアントは入力引数およびタイプト・データ・オブジェクトを作成します。
3. クライアントが EIS を呼び出して必要な結果を取得します。

EIS エクスポート・インターフェースは、外部の EIS アプリケーションで使用可能なエクスポート済み SCA コンポーネントのインターフェースを定義します。このインターフェースは、SAP や PeopleSoft などの外部アプリケーションが EIS エクスポート・アプリケーション・ランタイムの実装を介して呼び出すインターフェースと考えることができます。

エクスポートは、EISExportBinding を使用してエクスポート済みサービスを外部の EIS アプリケーションにバインドします。これにより、EIS サービス要求を listen するための SCA モジュールに含まれるアプリケーションをサブスクライブできます。EIS エクスポート・バインディングは、リソース・アダプターによって (Java EE コネクター・アーキテクチャー・インターフェースを使用して) 認識されるインバウンド・イベントの定義と SCA 操作の呼び出しの間のマッピングを指定します。

EISExportBinding では、外部の EIS サービスが Java EE コネクター・アーキテクチャー 1.5 のインバウンド規約に基づいている必要があります。EISExportBinding では、データ・ハンドラーまたはデータ・バインディングをバインディング・レベルまたはメソッド・レベルで指定する必要があります。

JMS バインディング

Java Message Service (JMS) プロバイダーにより、Java Messaging Service API およびプログラミング・モデルに基づいたメッセージングが可能になります。 JMS プロバイダーは、JMS 宛先への接続を作成しメッセージを送受信するための Java EE 接続ファクトリーを提供します。

以下の 3 つの JMS バインディングが提供されています。

- JMS JCA 1.5 に準拠したサービス統合バス (SIB) プロバイダー・バインディング (*JMS バインディング*)
- JMS 1.1 に準拠した非 JCA の汎用 JMS バインディング (*汎用 JMS バインディング*)

- WebSphere MQ JMS バインディング。WebSphere MQ に対し JMS プロバイダー・サポートを提供し、Java EE アプリケーションとの相互運用性を実現します。(*WebSphere MQ JMS バインディング*)

JMS エクスポートおよびインポート・バインディングにより、Service Component Architecture (SCA) モジュールは、外部 JMS システムに対する呼び出しを実行したり、外部 JMS システムからメッセージを受信したりできます。

また、WebSphere MQ バインディング (*WebSphere MQ バインディング*) もサポートされています。このバインディングにより、ネイティブ MQ ユーザーは、着信メッセージと発信メッセージの任意のフォーマットを処理できます (WebSphere MQ が必要です)。

JMS インポートおよびエクスポート・バインディングは、WebSphere Application Server の一部である JCA 1.5 ベースの SIB JMS プロバイダーを使用して JMS アプリケーションとの統合を行います。その他の JCA 1.5 ベースの JMS リソース・アダプターはサポートされません。

JMS バインディングの概要:

JMS バインディングは、Service Component Architecture (SCA) 環境と JMS システムの間の接続を提供します。

JMS バインディング

JMS インポート・バインディングおよび JMS エクスポート・バインディングの主なコンポーネントは、以下のとおりです。

- リソース・アダプター: SCA モジュールと外部 JMS システムの間の管理された両方向接続を使用可能にします。
- 接続: クライアントとプロバイダー・アプリケーションの間の仮想接続をカプセル化します。
- 宛先: 生成するメッセージの宛先またはコンシュームするメッセージの送信元を指定するためにクライアントが使用します。
- 認証データ: バインディングへのアクセスを保護するために使用します。

JMS インポート・バインディング

JMS インポート・バインディングでは、SCA モジュール内で使用する外部 JMS アプリケーションをインポートできます。JMS インポート・バインディングにより、SCA モジュール内のコンポーネントは、外部 JMS アプリケーションが提供するサービスと通信できるようになります。

JMS 宛先に関連する JMS プロバイダーへの接続を作成するには、JMS 接続ファクトリーを使用します。デフォルト・メッセージング・プロバイダーの JMS 接続ファクトリーを管理するには、接続ファクトリーの管理オブジェクトを使用します。

外部 JMS システムとの対話では、要求を送信し、応答を受信するための宛先が使用されます。

JMS インポート・バインディングでは、呼び出されている操作のタイプに応じた以下の 2 種類の使用シナリオがサポートされています。

- 片方向: JMS インポートは、インポート・バインディングに構成された送信宛先にメッセージを送信します。JMS ヘッダーの replyTo フィールドには何も設定しません。
- 両方向 (要求/応答): JMS インポートは、送信宛先にメッセージを送信し、その後 SCA コンポーネントから受信した応答を維持します。

(Integration Designer の「**応答相関スキーム**」フィールドを使用して) インポート・バインディングを構成して、要求メッセージ ID (デフォルト) または要求メッセージ相関 ID から応答メッセージ相関 ID がコピーされているようにすることができます。インポート・バインディングを構成して、応答と要求を相関させるために一時動的応答宛先を使用することもできます。一時宛先は要求ごとに作成され、インポートはこの宛先を使用して応答を受信します。

receive 宛先がアウトバウンド・メッセージの replyTo ヘッダー・プロパティーに設定されます。受信宛先で listen するためのメッセージ・リスナーをデプロイします。応答を受信すると、メッセージ・リスナーは応答をコンポーネントに返します。

片方向と両方向のいずれのシナリオを使用する場合も、動的および静的ヘッダー・プロパティーを指定できます。静的プロパティーは、JMS インポート・メソッド・バインディングから設定できます。これらのプロパティーの一部は、SCA JMS ランタイムで特別な意味を持ちます。

重要な点として、JMS は非同期バインディングであることに注意してください。呼び出し側コンポーネントが JMS インポートを同期的に呼び出すと (両方向操作の場合)、呼び出し側コンポーネントは、JMS サービスからの応答が返されるまでブロックされます。

図 31 は、インポートがどのように外部サービスにリンクされているのかを示しています。

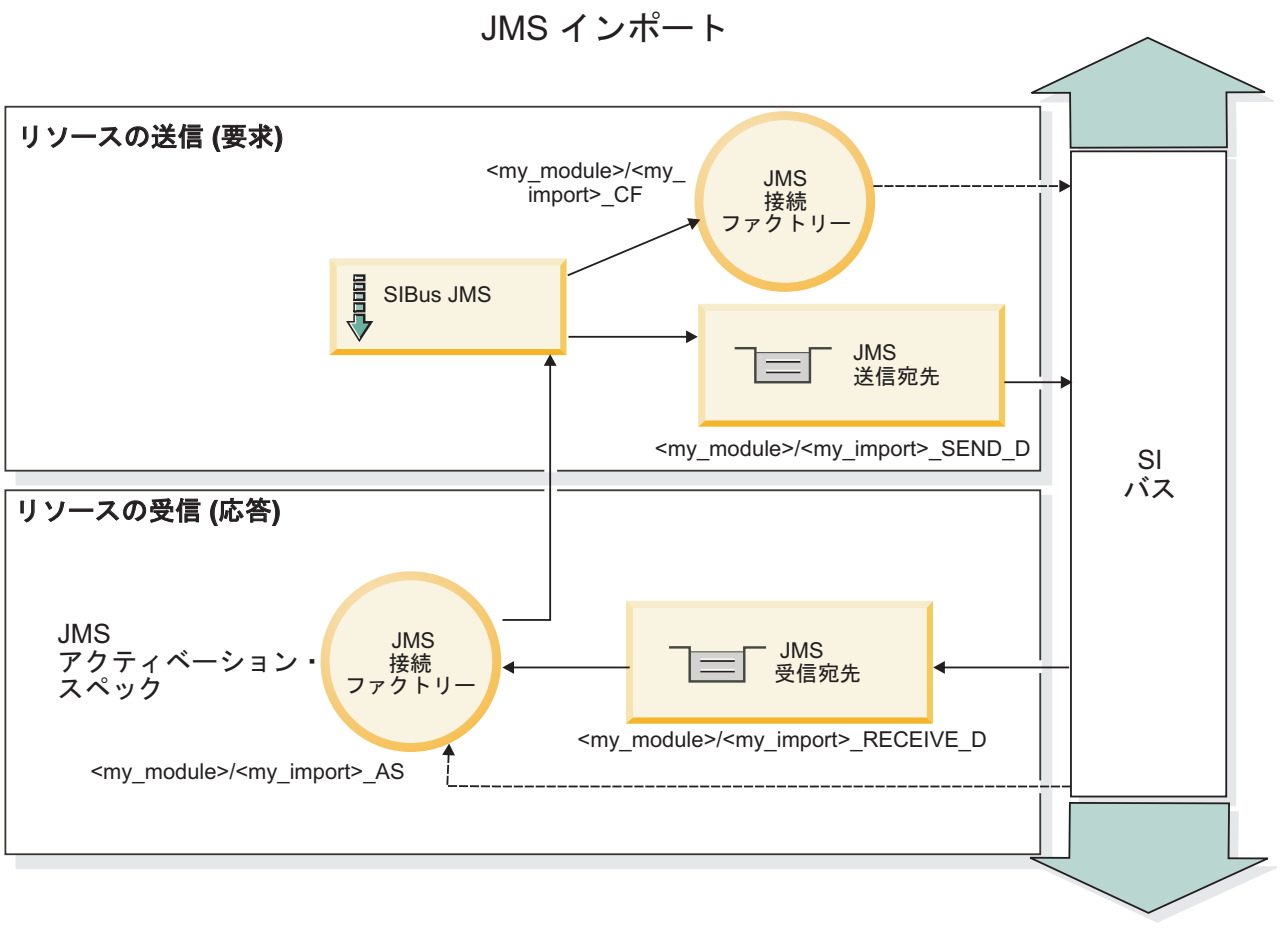


図 31. JMS インポート・バインディングのリソース

JMS エクスポート・バインディング

JMS エクスポート・バインディングは、SCA モジュールが外部 JMS アプリケーションにサービスを提供する手段を提供します。

JMS エクスポートの接続部分は、構成可能なアクティベーション・スペックです。

JMS エクスポートには、send 宛先と receive 宛先があります。

- receive 宛先は、ターゲット・コンポーネントに対する着信メッセージを格納する宛先です。
- send 宛先は、応答を送信する宛先です。ただし、着信メッセージで replyTo ヘッダー・プロパティーを使用してこの宛先をオーバーライドしている場合は、その宛先が優先されます。

エクスポート・バインディングで指定された receive 宛先に着信する要求を listen するため、メッセージ・リスナーがデプロイされます。send フィールドで指定された宛先は、呼び出されたコンポーネントが応答を返す場合にインバウンド要求に対する応答を送信するために使用されます。着信メッセージの replyTo フィールドで指定された宛先は、send で指定された宛先をオーバーライドします。

図 32 は、外部の要求側がどのようにエクスポートにリンクされているのかを示しています。

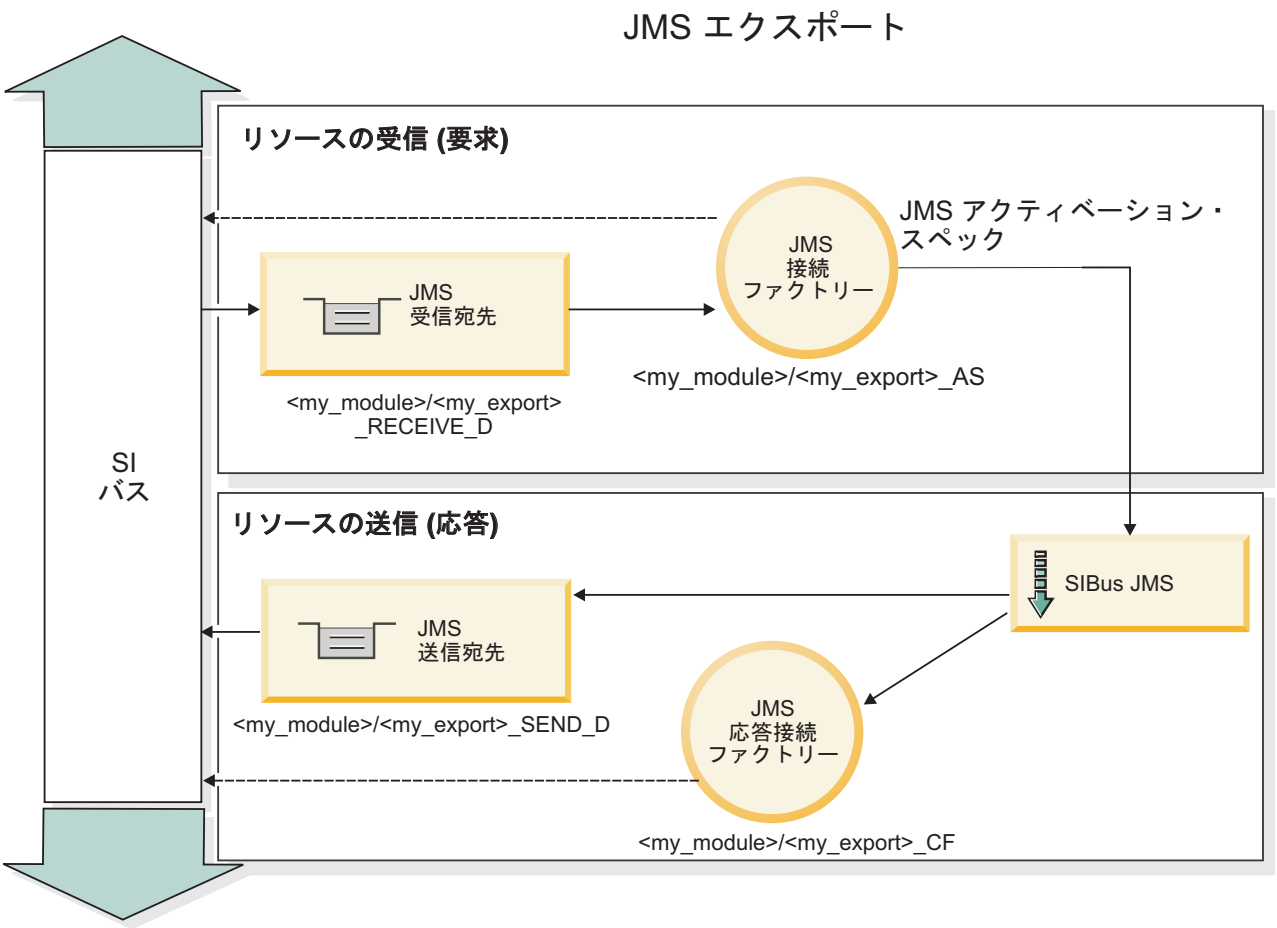


図 32. JMS エクスポート・バインディングのリソース

JMS 統合とリソース・アダプター:

Java Message Service (JMS) は、使用可能な JMS JCA 1.5 ベースのリソース・アダプターを介して統合を提供します。JMS 統合のための完全なサポートが、サービス統合バス (SIB) JMS リソース・アダプターに対して用意されています。

JCA 1.5 リソース・アダプター用の JMS プロバイダーは、外部 JCA 1.5 準拠 JMS システムと統合する場合に使用します。JCA 1.5 に準拠した外部サービスは、SIB JMS リソース・アダプターを使用してメッセージを送受信でき、Service Component Architecture (SCA) コンポーネントと統合します。

他のプロバイダー固有の JCA 1.5 リソース・アダプターの使用はサポートされません。

JMS モジュールは Java SE 環境にデプロイできません。それらのモジュールは Java EE 環境にのみデプロイできます。

JMS バインディングの主な特徴:

JMS のインポート・バインディングおよびエクスポート・バインディングの主な特徴は、ヘッダーと作成される Java EE リソースです。

特殊ヘッダー

特殊ヘッダーのプロパティーは、JMS インポートおよびエクスポートで使用され、ターゲットに対してメッセージの処理方法を指示します。

例えば、TargetFunctionName がネイティブ・メソッドから操作メソッドにマップされます。

Java EE リソース

JMS インポートおよびエクスポートを Java EE 環境にデプロイすると、さまざまな Java EE リソースが作成されます。

ConnectionFactory
クライアントが JMS プロバイダーとの接続を作成するために使用します。

ActivationSpec
インポートでは、要求に対する応答を受信するときに使用されます。エクスポートでは、メッセージング・システムとの対話でメッセージ・リスナーを表すメッセージ・エンドポイントを構成するときに使用されます。

宛先

- 送信宛先: インポートの場合は、要求または出力メッセージが送信される宛先になります。エクスポートの場合は、応答メッセージが送信される宛先になります。ただし、着信メッセージの JMSReplyTo ヘッダー・フィールドにより置き換えられた場合は、その宛先が優先されます。
- 受信宛先: 着信メッセージが格納される宛先です。インポートの場合は応答になり、エクスポートの場合は要求になります。
- コールバック宛先 : 相関情報の保管に使用される SCA JMS システム宛先です。この宛先に対して、読み取りまたは書き込みを行わないでください。

インストール・タスクでは、ConnectionFactory および 3 つの宛先を作成します。また、ActivationSpec を作成して、ランタイム・メッセージ・リスナーが受信宛先で応答を listen できるようにします。これらのリソースのプロパティーは、インポート・ファイルまたはエクスポート・ファイルに指定されます。

JMS ヘッダー:

JMS メッセージには、2 つのタイプのヘッダーが含まれます。1 つは JMS システム・ヘッダー、もう 1 つは複数の JMS プロパティーです。メディエーション・モジュールでは、いずれのタイプのヘッダーにも、サービス・メッセージ・オブジェクト (SMO) 内、または ContextService API を使用してアクセスできます。

JMS システム・ヘッダー

SMO では、JMS システム・ヘッダーは JMSHeader エレメントによって表されます。このエレメントには、JMS ヘッダーに通常あるすべてのフィールドが含まれます。これらのフィールドはメディエーション (または ContextService) で変更できますが、SMO に設定された一部の JMS システム・ヘッダー・フィールドは、システムまたは静的な値によってオーバーライドされるため、アウトバウンド JMS メッセージでは伝搬されません。

メディエーション (または ContextService) で更新可能な JMS システム・ヘッダーのキー・フィールドには以下があります。

- **JMSType** および **JMSCorrelationID** - 特定の事前定義メッセージ・ヘッダー・プロパティーの値
- **JMSDeliveryMode** - 送達モードの値 (persistent または nonpersistent。デフォルトは persistent)
- **JMSPriority** - 優先度の値 (0 から 9。デフォルトは JMS_Default_Priority)

JMS プロパティー

JMS プロパティーは、SMO ではプロパティー・リスト内のエントリーとして表されます。プロパティーは、メディエーション内で、または ContextService API を使用して追加、更新、または削除できます。

プロパティーは、JMS バインディングに静的に設定することもできます。静的に設定されたプロパティーは、動的に設定される (同じ名前の) 設定をオーバーライドします。

他のバインディング (例えば HTTP バインディング) から伝搬されるユーザー・プロパティーは、JMS バインディングでは JMS プロパティーとして出力されます。

ヘッダー伝搬の設定

JMS システム・ヘッダーおよびプロパティーの、インバウンド JMS メッセージからダウンストリームのコンポーネントへの伝搬、またはアップストリームのコンポーネントからアウトバウンド JMS メッセージへの伝搬は、バインディングのプロトコル・ヘッダー伝搬フラグで制御できます。

プロトコル・ヘッダー伝搬を設定すると、以下のリストに説明するようにヘッダー情報をメッセージまたはターゲット・コンポーネントに流すことができます。

- JMS エクスポート要求

  メッセージ内で受信した JMS ヘッダーは、コンテキスト・サービスを介してターゲット・コンポーネントに伝搬されます。メッセージ内で受信した JMS プロパティーは、コンテキスト・サービスを介してターゲット・コンポーネントに伝搬されます。
- JMS エクスポート応答

  コンテキスト・サービスに設定されたすべての JMS ヘッダー・フィールドは、JMS エクスポート・バインディングに設定された静的プロパティーによってオーバーライドされていない限り、アウトバウンド・メッセージ内で使用されます。コンテキスト・サービスに設定されたすべてプロパティーは、JMS エクスポート・バインディングに設定された静的プロパティーによってオーバーライドされていない限り、アウトバウンド・メッセージ内で使用されます。

- JMS インポート要求

  コンテキスト・サービスに設定されたすべての JMS ヘッダー・フィールドは、JMS インポート・バインディングに設定された静的プロパティーによってオーバーライドされていない限り、アウトバウンド・メッセージ内で使用されます。コンテキスト・サービスに設定されたすべてプロパティーは、JMS インポート・バインディングに設定された静的プロパティーによってオーバーライドされていない限り、アウトバウンド・メッセージ内で使用されます。
- JMS インポート応答

  メッセージ内で受信した JMS ヘッダーは、コンテキスト・サービスを介してターゲット・コンポーネントに伝搬されます。メッセージ内で受信した JMS プロパティーは、コンテキスト・サービスを介してターゲット・コンポーネントに伝搬されます。

JMS 一時動的応答宛先の相関スキーム:

一時動的応答宛先の相関スキームによって、送信された要求ごとに固有の動的キューまたはトピックが作成されます。

インポートに指定されている静的応答宛先は、一時動的宛先キューまたはトピックの性質を派生させるために使用されます。これは要求の **ReplyTo** フィールドで設定され、JMS インポートはその宛先で応答を listen します。応答が受信されると、その応答は非同期処理のために静的応答宛先に再キューイングされます。応答の **CorrelationID** フィールドは使用されないため、設定する必要はありません。

トランザクションの問題

一時動的宛先を使用する場合、応答は送信された応答と同じスレッドでコンシュームされる必要があります。要求はグローバル・トランザクションの外側で送信される必要があり、バックエンド・サービスによって受信される前、および応答が返される前にコミットされる必要があります。

パーシスタンス

一時動的キューは存続期間の短いエンティティーであるため、静的キューまたはトピックに関連付けられているのと同じレベルの永続性は保障されません。一時動的キューまたはトピックは、サーバーの再始動後も存続することはありません。メッセージも同様です。メッセージが静的応答宛先に再キューイングされると、メッセージに定義されている永続性を保持するようになります。

タイムアウト

インポートは、一定の時間、一時動的応答宛先で応答の受信を待機します。この時間間隔は、SCA 応答有効期限修飾子から取得されますが、これが設定されていない場合はデフォルトで 60 秒になります。待機時間を超過すると、インポートは `ServiceTimeoutRuntimeException` をスローします。

外部クライアント:

サーバーは、JMS バインディングを使用して、外部クライアントとの間でメッセージを送受信できます。

外部クライアント (Web ポータルやエンタープライズ情報システムなど) は、サーバーの SCA モジュールにメッセージを送信できます。また、サーバー内からコンポーネントによって外部クライアントを呼び出すこともできます。

JMS エクスポート・コンポーネントは、エクスポート・バインディングで指定された受信宛先に着信する要求を listen するためのメッセージ・リスナーをデプロイします。send フィールドで指定された宛先は、

呼び出されたアプリケーションが応答を返す場合にインバウンド要求に対する応答を送信するために使用されます。したがって、外部クライアントは、エクスポート・バインディングを使用してアプリケーションを呼び出すことができます。

JMS インポートは外部クライアントにバインドし、外部クライアントにメッセージを配信できます。このメッセージは、外部クライアントからの応答を要求してもしなくても構いません。

外部クライアントの使用:

外部クライアント (サーバー外部のクライアント) とサーバーにインストールされたアプリケーションの対話が必要になる場合があります。

外部クライアントとサーバー上のアプリケーションが対話する場合の、非常に簡単なシナリオを以下に示します。以下の図は、標準的な単純なシナリオを表しています。

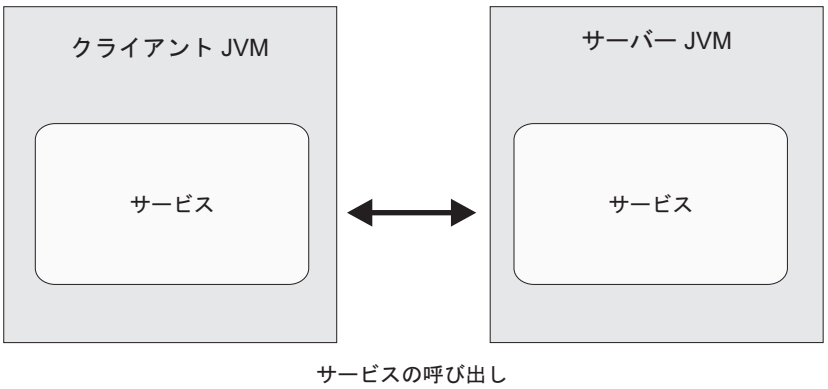


図 33. 単純なユース・ケース・シナリオ: 外部クライアントとサーバー・アプリケーションの対話

SCA アプリケーションには、JMS バインディングによるエクスポートが含まれています。これにより、外部クライアントからアプリケーションを使用できるようになります。

外部クライアントがサーバーとは別個の Java 仮想マシン (JVM) 内にある場合、JMS エクスポートとの接続および対話を作成するためにいくつかのステップを実行する必要があります。クライアントは、正しい値が指定された InitialContext を取得し、JNDI を使用してリソースを検索します。次にクライアントは、JMS 1.1 仕様 のクライアントを使用して宛先にアクセスし、宛先の送信メッセージと受信メッセージにアクセスします。

ランタイムにより自動作成されるリソースのデフォルトの JNDI 名は、このセクションの構成に関するトピックにリストされています。ただし、事前に作成済みのリソースがある場合は、その JNDI 名を使用します。

1. メッセージを送信するため、JMS の宛先と接続ファクトリーを構成します。
2. JNDI コンテキスト、SIB リソース・アダプターのポート、およびメッセージング・ブートストラッピング・ポートが正しいことを確認します。

   サーバーはいくつかのデフォルト・ポートを使用しますが、それより多くのサーバーがそのシステムにインストールされている場合は、他のサーバー・インスタンスとの競合を避けるために、インストール時に代わりのポートが作成されます。管理コンソールを使用して、サーバーが使用しているポートを調べることができます。「**サーバー**」 > 「**アプリケーション・サーバー**」 > ***your_server_ name*** > 「**構成**」を選択して、「**通信 (Communication)**」の下で「**ポート**」をクリックします。これで、使用するポートを編集できます。

3. クライアントは、正しい値が指定された初期コンテキストを取得して、JNDI を使用してリソースを検索します。
4. クライアントは、JMS 1.1 仕様に基づいて宛先にアクセスし、宛先の送信メッセージと受信メッセージにアクセスします。

JMS バインディングのトラブルシューティング:

JMS バインディングで発生した問題を診断し、修正できます。

実装例外

JMS のインポートおよびエクスポート実装は、さまざまなエラー状態に応じて、以下の 2 種類の例外のうちのいずれかを戻すことがあります。

- サービス・ビジネス例外: サービス・ビジネス・インターフェース (WSDL ポート・タイプ) で指定された障害が発生した場合に、この例外が戻されます。
- サービス・ランタイム例外: その他のすべてのケースで生成されます。ほとんどの場合、 cause 例外には元の例外 (JMSException) が含まれます。

  例えばインポートの場合、要求メッセージごとに 1 つの応答メッセージだけが戻されることを前提としています。そのため、複数の応答を受信した場合や遅延応答 (SCA の応答有効期限が切れた応答) を受信した場合は、サービス・ランタイム例外が throw されます。この場合、トランザクションはロールバックされ、応答メッセージはキューからバックアウトされるか、または Failed Event Manager によって処理されます。

主な障害状態

JMS バインディングの主な障害状態は、トランザクションの意味構造、JMS プロバイダー構成、またはその他のコンポーネントの既存動作への参照に基づいて判別されます。主な障害状態は以下のとおりです。

- JMS プロバイダーまたは宛先に接続できない。

  JMS プロバイダーに接続してメッセージを受信できない場合は、メッセージ・リスナーを開始することができません。この状態は、WebSphere Application Server ログに記録されます。正常に取得されるまで (または期限切れとなるまで)、永続メッセージは宛先に残ります。

  JMS プロバイダーに接続してアウトバウンド・メッセージを送信できない場合、送信操作を制御するトランザクションがロールバックされます。
- インバウンド・メッセージを解析できないか、アウトバウンド・メッセージを構成できない。

  データ・バインディングまたはデータ・ハンドラーが失敗すると、作業を制御するトランザクションがロールバックされます。
- アウトバウンド・メッセージを送信できない。

  メッセージを送信できないと、関連するトランザクションがロールバックされます。
- 複数のメッセージまたは予期しない遅延応答メッセージが返される。

  インポートの場合、要求メッセージごとに 1 つの応答メッセージだけが戻されることを前提としています。また、応答を受信できる有効期間は、要求の SCA 応答有効期限修飾子によって決まります。応答を受信したとき、または有効期間を超えたときに、相関レコードが削除されます。予期しない応答メッセージを受信した場合や遅れて応答を受信した場合は、サービス・ランタイム例外が throw されます。

- 一時動的応答宛先相関スキームを使用したとき、遅延応答によりサービス・タイムアウト・ランタイム例外が引き起こされる。

  JMS インポートは、SCA 応答有効期限修飾子によって決められた期間後にタイムアウトします。期間が設定されていない場合は、デフォルトで 60 秒になります。

Failed Event Manager に表示されない JMS ベースの SCA メッセージ

JMS の対話の失敗によって SCA メッセージが発生した場合は、Failed Event Manager でこのメッセージを見つけることになります。Failed Event Manager にこのようなメッセージが表示されない場合は、JMS 宛先の基礎となる SIB 宛先の最大配信失敗回数の値に **1** よりも大きい値が設定されているかどうかを確認してください。この値を **2** 以上に設定すると、JMS バインディングに対して SCA を呼び出す際に、Failed Event Manager と対話することができます。

例外の処理:

バインディングがどのように構成されているかによって、データ・ハンドラーまたはデータ・バインディングによる例外の処理方法が決まります。さらに、メディエーション・フローの特性により、そのような例外が throw された場合のシステムの動作が決まります。

データ・ハンドラーまたはデータ・バインディングがバインディングによって呼び出されるときには、さまざまな問題が発生する可能性があります。例えば、データ・ハンドラーが、ペイロードが破損しているメッセージを受信したり、誤った形式のメッセージを読み取るなどです。

バインディングによるそのような例外の処理方法は、データ・ハンドラーおよびデータ・バインディングの実装方法によって決まります。**DataBindingException** を throw するようにデータ・バインディングを設計することをお勧めします。

DataBindingException などのランタイム例外が throw された場合:

- メディエーション・フローがトランザクションとして構成されている場合、デフォルトでは、JMS メッセージは手動でやり直し、または削除できるように Failed Event Manager に保管されます。

  注: バインディングのリカバリー・モードを変更することによって、メッセージが Failed Event Manager に保管される代わりに、ロールバックされるようにできます。
- メディエーション・フローがトランザクションでない場合、例外はログに記録され、メッセージは失われます。

データ・ハンドラーの場合も同様です。データ・ハンドラーはデータ・バインディングによって呼び出されるため、データ・ハンドラー例外はデータ・バインディング例外にラップされます。したがって、**DataHandlerException** は、**DataBindingException** として報告されます。

汎用 JMS バインディング

汎用 JMS バインディングにより、サード・パーティーの JMS 1.1 準拠プロバイダーに対する接続が提供されます。汎用 JMS バインディングの操作は、JMS バインディングの操作と似ています。

JMS バインディングを介して提供されるサービスにより、Service Component Architecture (SCA) モジュールは、外部システムに対する呼び出しを実行したり、外部システムからメッセージを受信したりできます。このシステムとして、外部 JMS システムを使用できます。

汎用 JMS バインディングは、JCA 1.5 非準拠 JMS プロバイダー (JMS プロバイダーが JMS 1.1 をサポートし、オプションの JMS Application Server Facility を実装している場合) との統合を実現します。汎用

JMS バインディングは、JCA 1.5 をサポートしないが JMS 1.1 仕様の Application Server Facility をサポートする JMS プロバイダーをサポートしています (これには、Oracle AQ、TIBCO、SonicMQ、WebMethods、BEA WebLogic、WebSphere MQ などが含まれます)。 WebSphere の組み込み JMS プロバイダー (SIBJMS) である JCA 1.5 JMS プロバイダーは、このバインディングではサポートされていません。このプロバイダーを使用する場合は、 108 ページの『JMS バインディング』を使用してください。

SCA 環境内で JCA 1.5 非準拠の JMS ベース・システムと統合するときに、この汎用バインディングを使用します。これにより、ターゲットの外部アプリケーションがメッセージを送受信でき、SCA コンポーネントと統合できます。

汎用 JMS バインディングの概要:

汎用 JMS バインディングは、Service Component Architecture (SCA) 環境と JMS システム (JMS 1.1 に準拠し、オプションの JMS Application Server Facility を実装している場合) の間の接続を提供する非 JCA JMS バインディングです。

汎用 JMS バインディング

汎用 JMS インポートおよびエクスポート・バインディングの主な側面は以下のとおりです。

- リスナー・ポート: 非 JCA ベースの JMS プロバイダーがメッセージを受信し、それらをメッセージ・ドリブン Bean (MDB) にディスパッチできるようにします。
- 接続: クライアントとプロバイダー・アプリケーションの間の仮想接続をカプセル化します。
- 宛先: 生成するメッセージの宛先またはコンシュームするメッセージの送信元を指定するためにクライアントが使用します。
- 認証データ: バインディングへのアクセスを保護するために使用します。

汎用 JMS インポート・バインディング

汎用 JMS インポート・バインディングにより、SCA モジュール内のコンポーネントは、外部の JCA 1.5 非準拠の JMS プロバイダーが提供するサービスと通信できるようになります。

JMS インポートの接続部分は、接続ファクトリーです。接続ファクトリーとは、クライアントがプロバイダーへの接続を作成するために使用するオブジェクトであり、管理者によって定義された一連の接続構成パラメーターをカプセル化します。各接続ファクトリーは、ConnectionFactory、QueueConnectionFactory、または TopicConnectionFactory インターフェースのインスタンスです。

外部 JMS システムとの対話では、要求を送信し、応答を受信するための宛先が使用されます。

汎用 JMS インポート・バインディングでは、呼び出されている操作のタイプに応じた以下の 2 種類の使用シナリオがサポートされています。

- 片方向: 汎用 JMS インポートは、インポート・バインディングに構成された送信宛先にメッセージを送信します。JMS ヘッダーの replyTo フィールドには何も送信しません。
- 両方向 (要求/応答): 汎用 JMS インポートは、送信宛先にメッセージを送信し、その後 SCA コンポーネントから受信した応答を維持します。

  receive 宛先がアウトバウンド・メッセージの replyTo ヘッダー・プロパティーに設定されます。受信宛先で listen するためのメッセージ駆動型 Bean (MDB) をデプロイします。応答を受信すると、MDB は応答をコンポーネントに返します。

インポート・バインディングは、要求メッセージ ID (デフォルト) または要求メッセージ相関 ID から応答メッセージ相関 ID がコピーされていることを期待するように (Integration Designer の「**応答相関スキーム**」フィールドを使用して) 構成することができます。

片方向と両方向のいずれのシナリオを使用する場合も、動的および静的ヘッダー・プロパティーを指定できます。静的プロパティーは汎用 JMS インポート・メソッド・バインディングから設定できます。これらのプロパティーのなかには、SCA JMS ランタイムにとって特殊な意味を持つものがあります。

重要な点として、汎用 JMS は非同期バインディングであることに注意してください。呼び出し側コンポーネントが汎用 JMS インポートを同期的に呼び出すと (両方向操作の場合)、呼び出し側コンポーネントは、JMS サービスからの応答が返されるまでブロックされます。

図 34 は、インポートがどのように外部サービスにリンクされているのかを示しています。

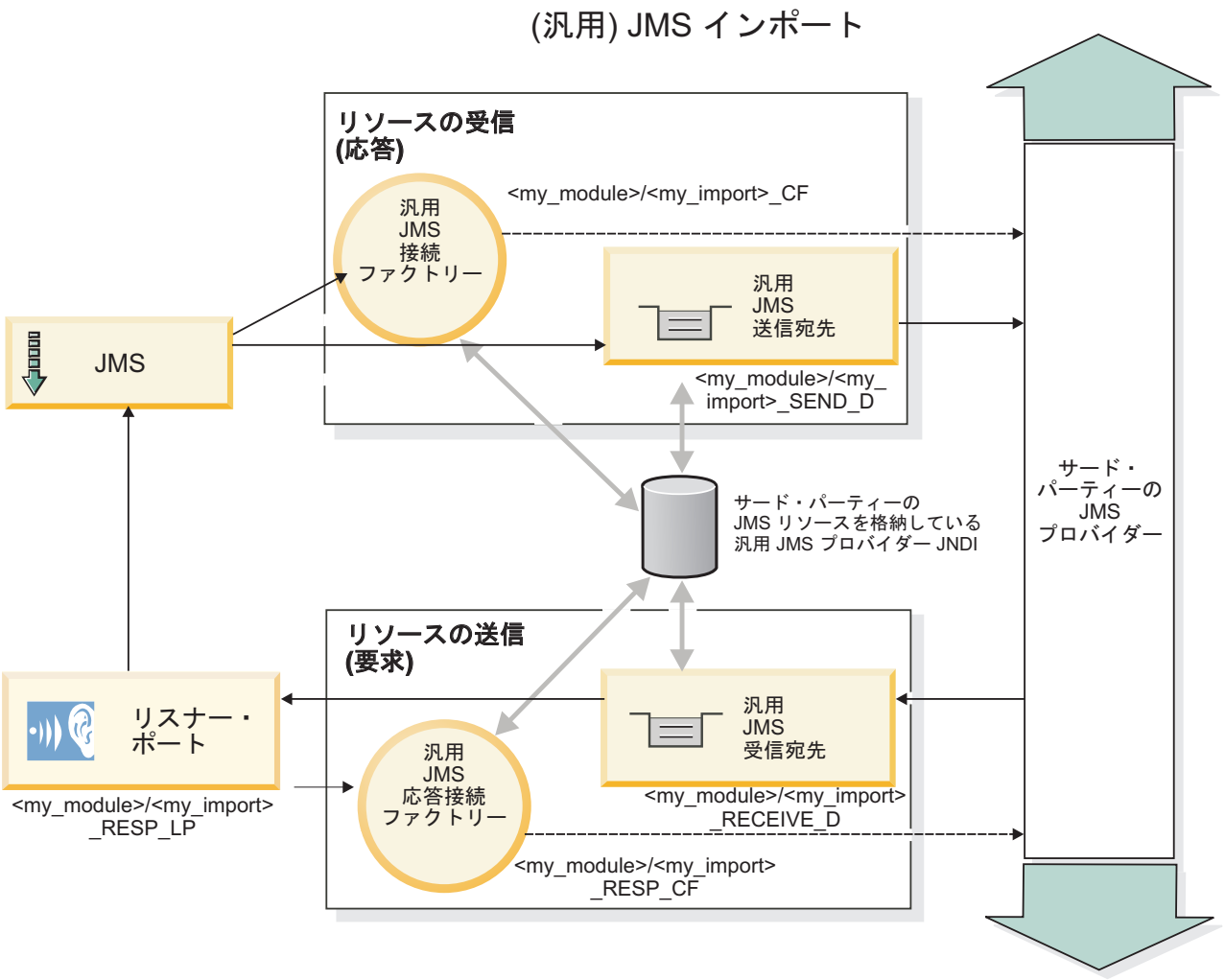


図 34. 汎用 JMS インポート・バインディング・リソース

汎用 JMS エクスポート・バインディング

汎用 JMS エクスポート・バインディングは、SCA モジュールが外部 JMS アプリケーションにサービスを提供する手段を提供します。

JMS エクスポートの接続部分は、ConnectionFactory および ListenerPort から構成されます。

汎用 JMS エクスポートには、send 宛先と receive 宛先があります。

- receive 宛先は、ターゲット・コンポーネントに対する着信メッセージを格納する宛先です。
- send 宛先は、応答を送信する宛先です。ただし、着信メッセージで replyTo ヘッダー・プロパティーを使用してこの宛先をオーバーライドしている場合は、その宛先が優先されます。

エクスポート・バインディングで指定された receive 宛先に着信する要求を listen するため、MDB がデプロイされます。

- send フィールドで指定された宛先は、呼び出されたコンポーネントが応答を返す場合にインバウンド要求に対する応答を送信するために使用されます。
- 着信メッセージの replyTo フィールドで指定された宛先は、send フィールドで指定された宛先をオーバーライドします。
- 要求/応答シナリオの場合、応答が要求 message ID を応答メッセージの correlation ID フィールドにコピーする (デフォルト) ことを期待するように、(Integration Designer の「**応答相関スキーマ**」フィールドを使用して) インポート・バインディングを構成できます。または、応答が要求の correlation ID を応答メッセージの correlation ID フィールドにコピーすることもできます。

121 ページの図 35 は、外部の要求側がどのようにエクスポートにリンクされているのかを示しています。

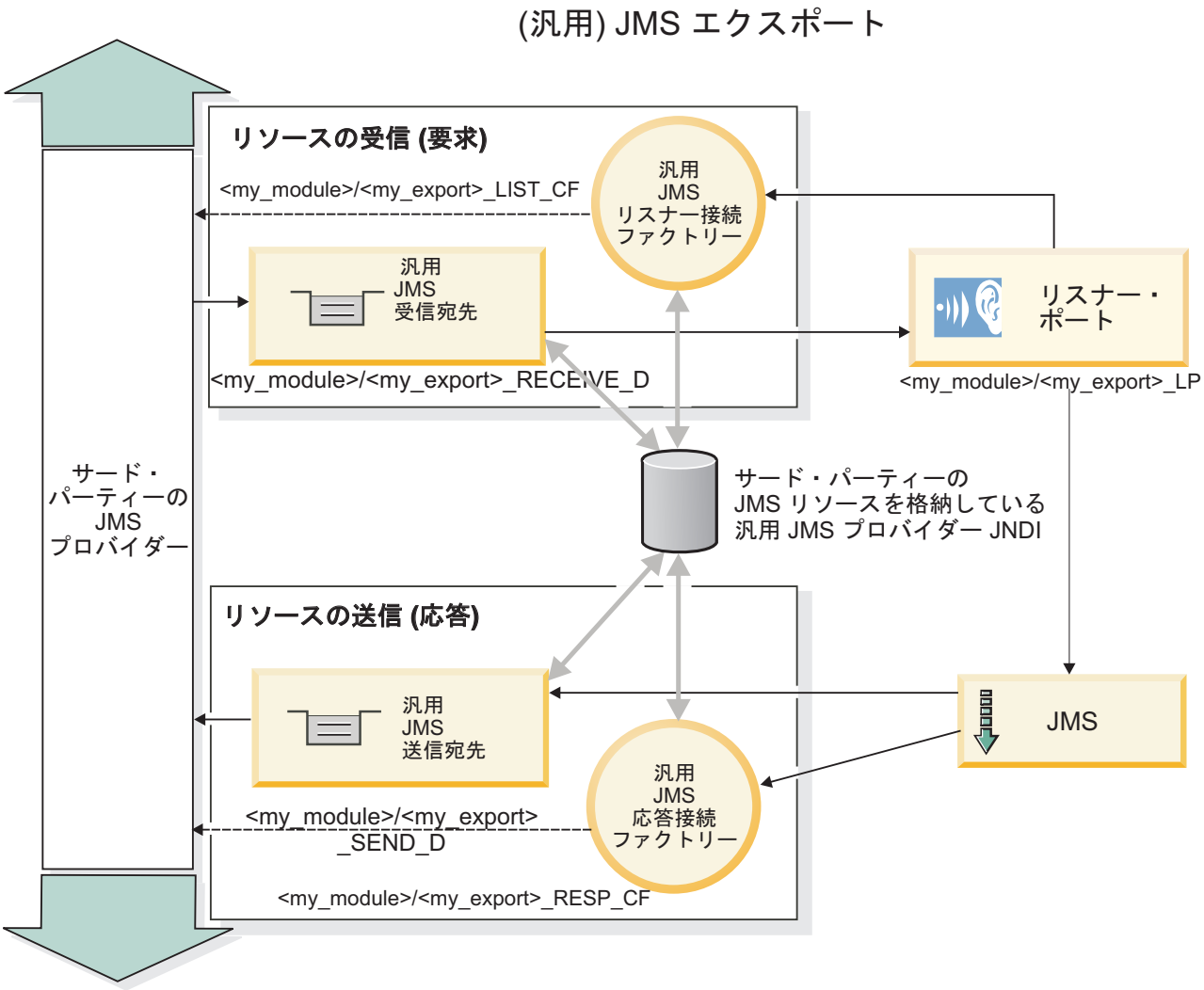


図 35. 汎用 JMS エクスポート・バインディング・リソース

汎用 JMS バインディングの主な機能:

汎用 JMS インポート・バインディングとエクスポート・バインディングの機能は、WebSphere 埋め込み JMS と MQ JMS インポート・バインディングの機能と整合性があります。主な機能は、ヘッダー定義および既存の Java EE リソースへのアクセスです。ただし、汎用という性質上、JMS プロバイダー固有の接続オプションはありません。また、このバインディングでは、デプロイメントおよびインストール時にリソースを生成できる機能が限定されています。

汎用インポート

MQ JMS インポート・アプリケーションと同様に、汎用 JMS 実装は非同期であり、3 つの呼び出し (片方向、両方向 (要求/応答ともいう)、コールバック) をサポートしています。

JMS インポートがデプロイされるとき、ランタイム環境によって提供されるメッセージ駆動型 Bean がデプロイされます。MDB は要求メッセージに対する応答を listen します。 MDB は、要求と共に送信される宛先に関連付けられています (この宛先を listen します)。この宛先は、JMS メッセージの replyTo ヘッダー・フィールドで指定されます。

汎用エクスポート

汎用 JMS エクスポート・バインディングは、結果の戻りの処理方法が EIS エクスポート・バインディングとは異なります。汎用 JMS エクスポートは、着信メッセージに指定された replyTo 宛先に応答を明示的に送信します。この宛先が指定されていない場合は、送信宛先が使用されます。

汎用 JMS エクスポートがデプロイされるときに、メッセージ駆動型 Bean (汎用 JMS インポートに使用されるものとは異なる MDB) がデプロイされます。この MDB は受信宛先で着信要求を listen し、次にその要求が SCA ランタイムで処理されるようにディスパッチします。

特殊ヘッダー

特殊ヘッダーのプロパティーは、汎用 JMS インポートとエクスポートで使用され、ターゲット・バインディングに対してメッセージの処理方法を指示します。

例えば、TargetFunctionName プロパティーは、呼び出されているエクスポートのインターフェース内の操作名を識別する際に、デフォルトの関数セレクターによって使用されます。

注: インポート・バインディングを構成して、各操作名に TargetFunctionName ヘッダーを設定することができます。

Java EE リソース

JMS バインディングを Java EE 環境にデプロイすると、いくつかの Java EE リソースが作成されます。

- インポートの場合は受信宛先 (応答、両方向のみ)、エクスポートの場合は受信宛先 (要求) で listen するためのリスナー・ポート
- outboundConnection (インポート) および inboundConnection (エクスポート) の汎用 JMS 接続ファクトリー
- 送信宛先 (インポート) および受信宛先 (エクスポート、両方向のみ) の汎用 JMS 宛先
- responseConnection の汎用 JMS 接続ファクトリー (両方向のみ、オプション。これ以外の場合は、インポートに outboundConnection が使用され、エクスポートに inboundConnection が使用される)
- 受信宛先 (インポート) および送信宛先 (エクスポート) の汎用 JMS 宛先 (両方向のみ)
- SIB コールバック・キュー宛先にアクセスするときに使用されるデフォルトのメッセージング・プロバイダー・コールバック JMS 宛先 (両方向のみ)
- コールバック JMS 宛先にアクセスするときに使用されるデフォルトのメッセージング・プロバイダー・コールバック JMS 接続ファクトリー (両方向のみ)
- 応答処理中に使用される要求メッセージに関する情報を格納するための SIB コールバック・キュー宛先 (両方向のみ)

インストール・タスクにより、インポート・ファイルおよびエクスポート・ファイルの情報から ConnectionFactory、3 つの宛先、および ActivationSpec が作成されます。

汎用 JMS ヘッダー:

汎用 JMS ヘッダーは、すべての汎用 JMS メッセージ・プロパティーを含むサービス・データ・オブジェクト (SDO) です。これらのプロパティーは、インバウンド・メッセージからのプロパティーか、アウトバウンド・メッセージに適用されるプロパティーにすることができます。

JMS メッセージには、2 つのタイプのヘッダーが含まれます。1 つは JMS システム・ヘッダー、もう 1 つは複数の JMS プロパティーです。メディエーション・モジュールでは、いずれのタイプのヘッダーにも、サービス・メッセージ・オブジェクト (SMO) 内、または ContextService API を使用してアクセスできます。

以下のプロパティーが、methodBinding に静的に設定されます。

- JMSType
- JMSCorrelationID
- JMSDeliveryMode
- JMSPriority

また、汎用 JMS バインディングでは、JMS および MQ JMS バインディングと同じ方法で JMS ヘッダーとプロパティーの動的変更がサポートされます。

一部の汎用 JMS プロバイダーでは、アプリケーションにより設定できるプロパティーとその組み合わせが制限されます。詳しくは、ご使用のサード・パーティー製品資料を参照してください。ただし、methodBinding、ignoreInvalidOutboundJMSProperties にプロパティーが追加され、このプロパティーにより、すべての例外を伝搬することが可能です。

汎用 JMS ヘッダーとメッセージのプロパティーは、基本 Service Component Architecture の SCDL バインディング・スイッチがオンになっている場合にのみ使用されます。スイッチがオンになっているとき、コンテキスト情報が伝搬されます。デフォルトでは、このスイッチはオンになっています。コンテキスト情報の伝搬を回避するには、値を **false** に変更します。

コンテキスト伝搬を使用可能にすると、ヘッダー情報をメッセージまたはターゲット・コンポーネントに流すことができます。コンテキスト伝搬のオン/オフを切り替えるには、インポート・バインディングおよびエクスポート・バインディングの contextPropagationEnabled 属性に **true** または **false** を指定します。以下に例を示します。

```
<esbBinding xsi:type="eis:JMSImportBinding" contextProgagationEnabled="true">
```

デフォルトは **true** です。

汎用 JMS バインディングのトラブルシューティング:

汎用 JMS バインディングで発生した問題は、診断して修正することができます。

実装例外

汎用 JMS のインポート実装とエクスポート実装は、さまざまなエラー状態に応じて、以下の 2 種類の例外のいずれかを戻すことがあります。

- サービス・ビジネス例外: サービス・ビジネス・インターフェース (WSDL ポート・タイプ) で指定された障害が発生した場合に、この例外が戻されます。
- サービス・ランタイム例外: その他のすべてのケースで生成されます。ほとんどの場合、 cause 例外には元の例外 (JMSException) が含まれます。

汎用 JMS メッセージの有効期限のトラブルシューティング

JMS プロバイダーによる要求メッセージには、有効期限があります。

要求の有効期限 とは、要求メッセージの JMSExpiration 時刻に達したときの、JMS プロバイダーによる要求メッセージの有効期限を指します。その他の JMS バインディングの場合と同様に汎用 JMS バインディングも、インポートによって格納されたコールバック・メッセージの有効期限を発信要求と同じ有効期限に設定することにより、要求の有効期限を処理しています。コールバック・メッセージの有効期限の通知では、要求メッセージの有効期限が切れたことが示され、クライアントにはビジネス例外により通知する必要があります。

ただし、コールバックの宛先をサード・パーティーのプロバイダーに移動すると、このタイプの要求有効期限はサポートされません。

応答の有効期限 とは、応答メッセージの JMSExpiration 時刻に達したときの、JMS プロバイダーによる応答メッセージの有効期限を指します。

サード・パーティーの JMS プロバイダーの正確な期限満了動作は定義されていないため、汎用 JMS バインディングの応答の有効期限はサポートされていません。ただし、応答を受信した場合は、その応答が有効期限に達していないことを確認できます。

アウトバウンド要求メッセージの場合、JMSExpiration の値は、待機時間と、asyncHeader (設定されている場合) 内の requestExpiration の値を基に計算されます。

汎用 JMS 接続ファクトリー・エラーのトラブルシューティング

汎用 JMS プロバイダーで特定のタイプの接続ファクトリーを定義すると、アプリケーションの開始時にエラー・メッセージを受信することがあります。この問題を回避するには、外部接続ファクトリーを変更します。

アプリケーションの起動時に、以下のエラー・メッセージを受け取ります。

```
MDB リスナー・ポート JMSConnectionFactory タイプが JMSDestination タイプと
一致しません (MDB Listener Port JMSConnectionFactory type
does not match JMSDestination type)
```

この問題は、外部接続ファクトリーの定義時に発生します。特に、JMS 1.1 (統合) 接続ファクトリー (Point-to-Point 通信およびパブリッシュ/サブスクライブ通信の両方をサポート可能な接続ファクトリー) ではなく JMS 1.0.2 トピック接続ファクトリーを作成すると、例外が throw されることがあります。

この問題を解決するには、以下のステップを実行します。

1. 使用している汎用 JMS プロバイダーにアクセスします。
2. 定義されている JMS 1.0.2 トピック接続ファクトリーを JMS 1.1 (統合) 接続ファクトリーに置き換えます。

新規に定義した JMS 1.1 接続ファクトリーを使用してアプリケーションを起動すると、エラー・メッセージは表示されないはずです。

Failed Event Manager に表示されない汎用 JMS ベースの SCA メッセージ

汎用 JMS の対話の失敗によって SCA メッセージが発生した場合は、Failed Event Manager でこのメッセージを見つけることになります。Failed Event Manager にこのようなメッセージが表示されない場合は、基礎となるリスナー・ポートの最大再試行回数プロパティーの値が 1 以上になっているかどうかを確認してください。この値を 1 以上に設定すると、汎用 JMS バインディングに対して SCA を呼び出す際に、Failed Event Manager と対話することができます。

例外の処理:

バインディングがどのように構成されているかによって、データ・ハンドラーまたはデータ・バインディングによる例外の処理方法が決まります。さらに、メディエーション・フローの特性により、そのような例外が throw された場合のシステムの動作が決まります。

データ・ハンドラーまたはデータ・バインディングがバインディングによって呼び出されるときには、さまざまな問題が発生する可能性があります。例えば、データ・ハンドラーが、ペイロードが破損しているメッセージを受信したり、誤った形式のメッセージを読み取るなどです。

バインディングによるそのような例外の処理方法は、データ・ハンドラーおよびデータ・バインディングの実装方法によって決まります。**DataBindingException** を throw するようにデータ・バインディングを設計することをお勧めします。

データ・ハンドラーの場合も同様です。データ・ハンドラーはデータ・バインディングによって呼び出されるため、データ・ハンドラー例外はデータ・バインディング例外にラップされます。したがって、**DataHandlerException** は、**DataBindingException** として報告されます。

DataBindingException 例外などのランタイム例外がスローされた場合:

- メディエーション・フローがトランザクションとして構成されている場合、デフォルトでは、手動でやり直し、または削除できるように JMS メッセージは Failed Event Manager に保管されます。

  注: バインディングのリカバリー・モードを変更することによって、メッセージが Failed Event Manager に保管される代わりに、ロールバックされるようにできます。
- メディエーション・フローがトランザクションでない場合、例外はログに記録され、メッセージは失われます。

データ・ハンドラーの場合も同様です。データ・ハンドラーはデータ・バインディングによって呼び出されるため、データ・ハンドラー例外はデータ・バインディング例外内に生成されます。したがって、**DataHandlerException** は **DataBindingException** として報告されます。

WebSphere MQ JMS バインディング

WebSphere MQ JMS バインディングは、WebSphere MQ JMS ベースのプロバイダーを使用する外部アプリケーションとの統合を提供します。

WebSphere MQ JMS エクスポートおよびインポート・バインディングは、サーバー環境から外部 JMS または MQ JMS システムと直接統合する場合に使用します。 これにより、サービス統合バスの MQ リンクまたはクライアント・リンク機能を使用する必要がなくなります。

コンポーネントがインポートによって WebSphere MQ JMS ベースのサービスと対話するときには、WebSphere MQ JMS インポート・バインディングはデータの送信先の宛先と応答を受信できる宛先を使用します。JMS メッセージとの間のデータ変換は、JMS データ・ハンドラーまたはデータ・バインディング・エッジ・コンポーネントを介して行います。

SCA モジュールが WebSphere MQ JMS クライアントにサービスを提供するときには、WebSphere MQ JMS エクスポート・バインディングは、要求の受信と応答の送信ができる宛先を使用します。 JMS メッセージとの間のデータ変換は、JMS データ・ハンドラーまたはデータ・バインディングを介して行います。

関数セレクターは、呼び出すターゲット・コンポーネントに対する操作へのマッピングを提供します。

WebSphere MQ JMS バインディングの概要:

WebSphere MQ JMS バインディングは、WebSphere MQ JMS プロバイダーを使用する外部アプリケーションとの統合を提供します。

WebSphere MQ 管理用タスク

WebSphere MQ システム管理者は、基盤となる WebSphere MQ キュー・マネージャーを作成する必要があります。このキュー・マネージャーは、WebSphere MQ JMS バインディングが含まれるアプリケーションを実行する前に、これらのバインディングによって使用されます。

WebSphere MQ JMS インポート・バインディング

WebSphere MQ JMS インポートにより、SCA モジュール内のコンポーネントは、 WebSphere MQ JMS ベースのプロバイダーが提供するサービスと通信できるようになります。サポートされているバージョンの WebSphere MQ を使用していることを確認してください。詳細なハードウェアおよびソフトウェア要件については、IBM サポート・ページを参照してください。

WebSphere MQ JMS インポート・バインディングでは、呼び出されている操作のタイプに応じた以下の 2 種類の使用シナリオがサポートされています。

- 片方向: WebSphere MQ JMS インポートは、インポート・バインディングに構成された送信宛先にメッセージを送信します。 JMS ヘッダーの replyTo フィールドには何も送信しません。
- 両方向 (要求/応答): WebSphere MQ JMS インポートは、送信宛先にメッセージを送信します。

  receive 宛先が replyTo ヘッダー・フィールドに設定されます。受信宛先で listen するためのメッセージ駆動型 Bean (MDB) をデプロイします。応答を受信すると、MDB は応答をコンポーネントに返します。

  インポート・バインディングは、要求メッセージ ID (デフォルト) または要求メッセージ相関 ID から応答メッセージ相関 ID がコピーされていることを期待するように (Integration Designer の「**応答相関スキーム**」フィールドを使用して) 構成することができます。

片方向と両方向のいずれのシナリオを使用する場合も、動的および静的ヘッダー・プロパティーを指定できます。静的プロパティーは JMS インポート・メソッド・バインディングから設定できます。 これらのプロパティーの一部は、SCA JMS ランタイムで特別な意味を持ちます。

重要な点として、WebSphere MQ JMS は非同期バインディングであることに注意してください。呼び出し側コンポーネントが WebSphere MQ JMS インポートを同期的に呼び出すと (両方向操作の場合)、呼び出し側コンポーネントは、JMS サービスからの応答が返されるまでブロックされます。

127 ページの図 36 は、インポートがどのように外部サービスにリンクされているのかを示しています。

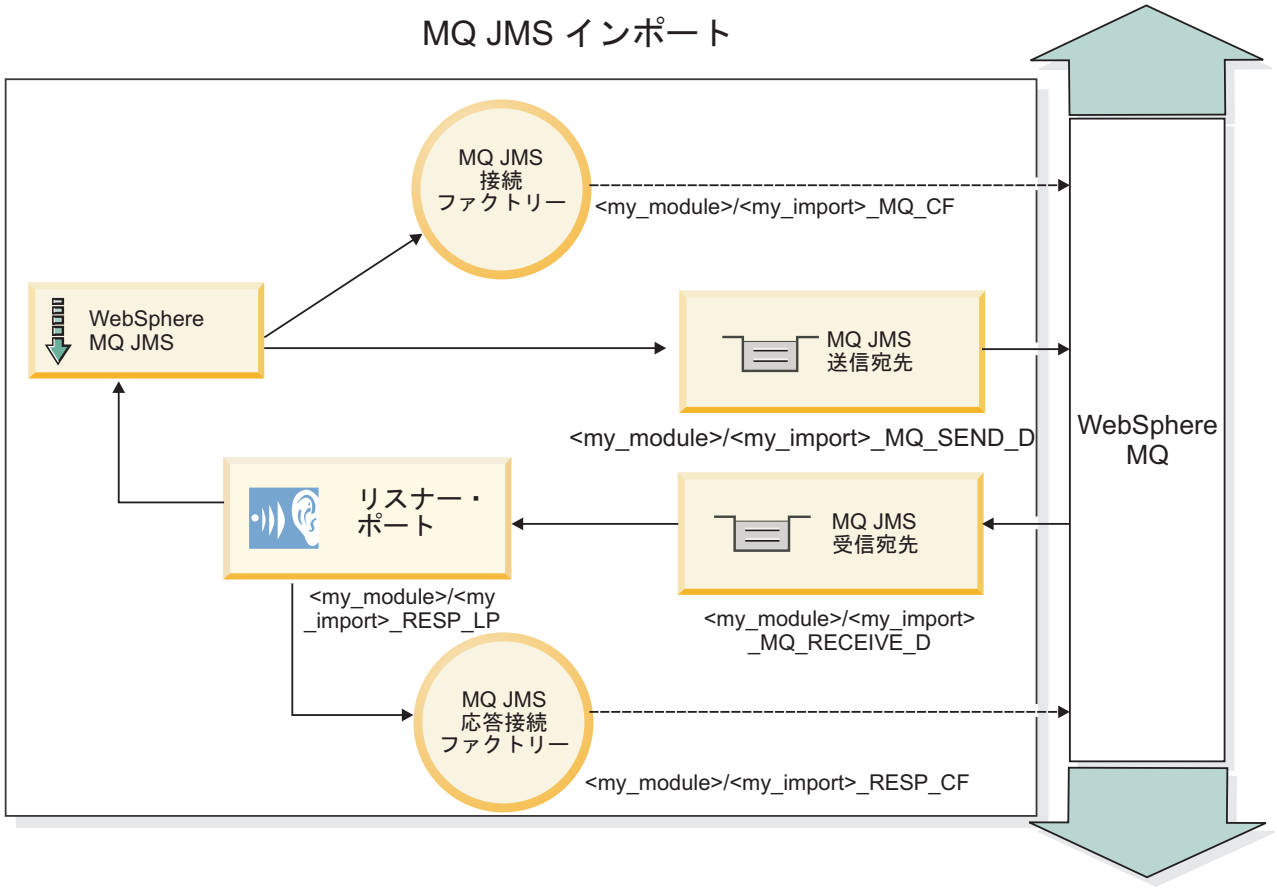


図 36. WebSphere MQ JMS インポート・バインディングのリソース

WebSphere MQ JMS エクスポート・バインディング

WebSphere MQ JMS エクスポート・バインディングは、SCA モジュールが WebSphere MQ ベースの JMS プロバイダーで外部 JMS アプリケーションにサービスを提供する手段を提供します。

エクスポート・バインディングで指定された receive 宛先に着信する要求を listen するため、MDB がデプロイされます。send フィールドで指定された宛先は、呼び出されたコンポーネントが応答を返す場合にインバウンド要求に対する応答を送信するために使用されます。応答メッセージの replyTo フィールドで指定された値は、send フィールドで指定された宛先をオーバーライドします。

128 ページの図 37 は、外部の要求側がどのようにエクスポートにリンクされているのかを示しています。

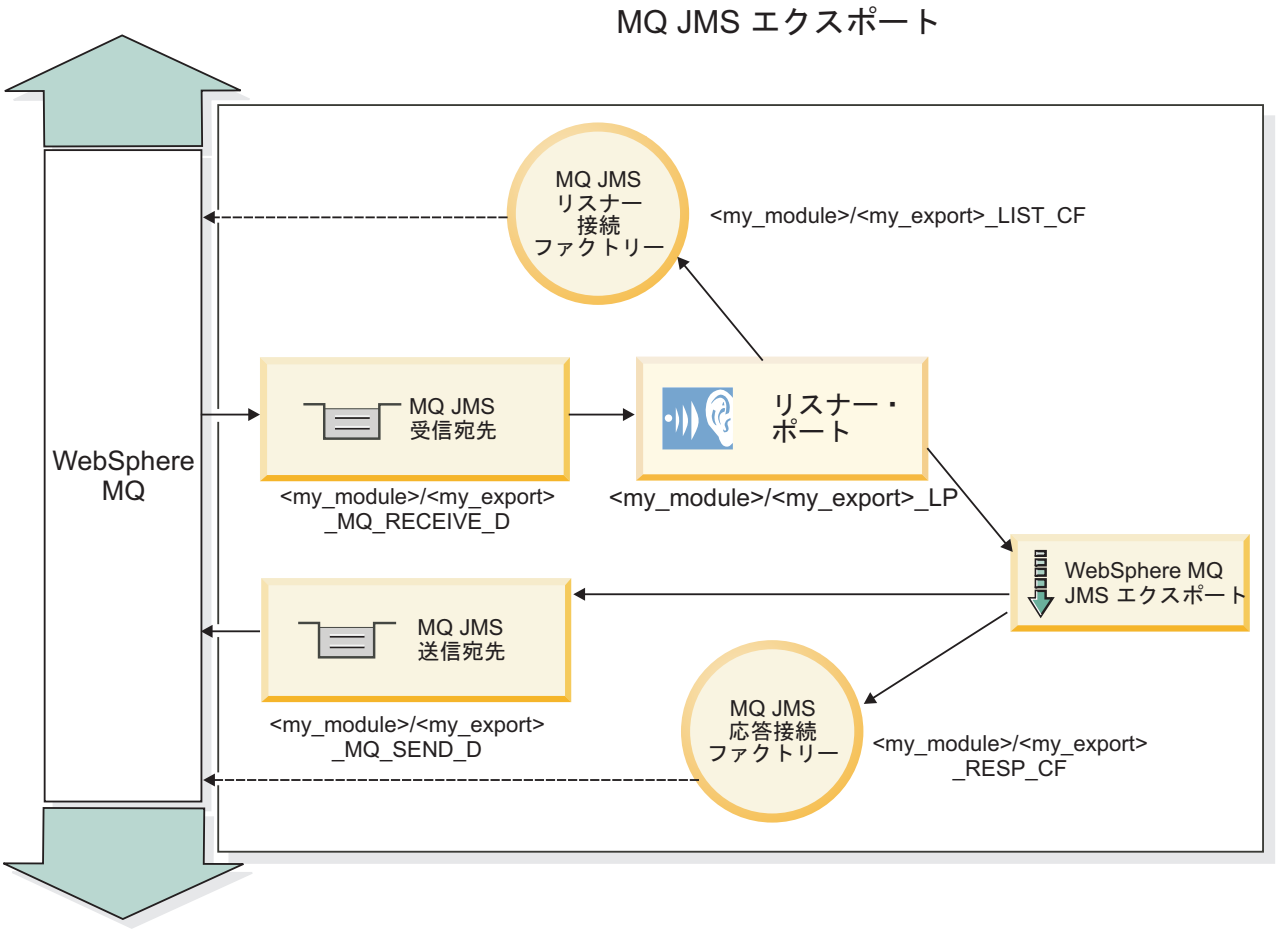


図 37. WebSphere MQ JMS エクスポート・バインディングのリソース

注: 127 ページの図 36 および 図 37 は、旧バージョンの IBM Business Process Manager のアプリケーションが外部サービスにリンクされる方法を示しています。 IBM Business Process Manager バージョン 7.0 用に開発されたアプリケーションについては、リスナー・ポートと接続ファクトリーではなくアクティベーション・スペックが使用されます。

WebSphere MQ JMS バインディングの主な特徴:

WebSphere MQ JMS バインディングの主な特徴は、ヘッダー、Java EE 成果物、および作成される Java EE リソースです。

ヘッダー

JMS メッセージ・ヘッダーには、いくつかの定義済みフィールドがあります。これらのフィールドには、クライアントとプロバイダーの両方でメッセージの識別と送付に使用される値が格納されます。バインディング・プロパティーを使用して、固定値でこれらのヘッダーを設定するか、またはヘッダーを実行時に動的に指定することができます。

JMSCorrelationID
関連メッセージへのリンクです。通常、このフィールドは、応答の対象となるメッセージのメッセージ ID のストリングに設定されます。

TargetFunctionName
このヘッダーは、呼び出された操作を識別する際に、指定されたいずれかの関数セレクターによって使

用されます。TargetFunctionName JMS ヘッダー・プロパティーを JMS エクスポートに送信されたメッセージ内に設定することにより、この関数セレクターが使用可能になります。このプロパティーは、JMS クライアント・アプリケーションに直接設定することも、JMS バインディングが指定されたインポートを JMS エクスポートに接続する際に設定することもできます。この場合、JMS インポート・バインディングを構成し、操作名とのインターフェースにおいて、各操作ごとに TargetFunctionName ヘッダーを設定する必要があります。

相関スキーム

WebSphere MQ JMS バインディングは、要求メッセージと応答メッセージを相関させる方法を決定するためのさまざまな相関スキームを提供します。

RequestMsgIDToCorrelID
JMSMessageID は JMSCorrelationID フィールドにコピーされます。これはデフォルト設定です。

RequestCorrelIDToCorrelID
JMSCorrelationID は JMSCorrelationID フィールドにコピーされます。

Java EE リソース

MQ JMS インポートを Java EE 環境にデプロイすると、いくつかの Java EE リソースが作成されます。

パラメーター

MQ 接続ファクトリー
クライアントが MQ JMS プロバイダーとの接続を作成するために使用します。

応答接続ファクトリー
送信宛先が受信宛先とは異なるキュー・マネージャー上にある場合に、SCA MQ JMS ランタイムが使用します。

アクティベーション・スペック
MQ JMS アクティベーション・スペックは、1 つ以上のメッセージ駆動型 Bean に関連付けられており、これらの Bean がメッセージを受信するのに必要な構成を提供します。

宛先

- 送信宛先:
  - インポート: 要求または出力メッセージが送信される宛先です。
  - エクスポート: 応答メッセージが送信される宛先です。ただし、着信メッセージの JMSReplyTo ヘッダー・フィールドにより置き換えられた場合は、その宛先が優先されます。
- 受信宛先:
  - インポート: 応答メッセージまたは着信メッセージが格納される宛先です。
  - エクスポート: 着信メッセージまたは要求メッセージが格納される宛先です。

JMS ヘッダー:

JMS メッセージには、2 つのタイプのヘッダーが含まれます。1 つは JMS システム・ヘッダー、もう 1 つは複数の JMS プロパティーです。メディエーション・モジュールでは、いずれのタイプのヘッダーにも、サービス・メッセージ・オブジェクト (SMO) 内、または ContextService API を使用してアクセスできます。

JMS システム・ヘッダー

SMO では、JMS システム・ヘッダーは JMSHeader エレメントによって表されます。このエレメントには、JMS ヘッダーに通常あるすべてのフィールドが含まれます。これらのフィールドはメディエーション (または ContextService) で変更できますが、SMO に設定された一部の JMS システム・ヘッダー・フィールドは、システムまたは静的な値によってオーバーライドされるため、アウトバウンド JMS メッセージでは伝搬されません。

メディエーション (または ContextService) で更新可能な JMS システム・ヘッダーのキー・フィールドには以下があります。

- **JMSType** および **JMSCorrelationID** - 特定の事前定義メッセージ・ヘッダー・プロパティーの値
- **JMSDeliveryMode** - 送達モードの値 (persistent または nonpersistent。デフォルトは persistent)
- **JMSPriority** - 優先度の値 (0 から 9。デフォルトは JMS_Default_Priority)

JMS プロパティー

JMS プロパティーは、SMO ではプロパティー・リスト内のエントリーとして表されます。プロパティーは、メディエーション内で、または ContextService API を使用して追加、更新、または削除できます。

プロパティーは、JMS バインディングに静的に設定することもできます。静的に設定されたプロパティーは、動的に設定される (同じ名前の) 設定をオーバーライドします。

他のバインディング (例えば HTTP バインディング) から伝搬されるユーザー・プロパティーは、JMS バインディングでは JMS プロパティーとして出力されます。

ヘッダー伝搬の設定

JMS システム・ヘッダーおよびプロパティーの、インバウンド JMS メッセージからダウンストリームのコンポーネントへの伝搬、またはアップストリームのコンポーネントからアウトバウンド JMS メッセージへの伝搬は、バインディングのプロトコル・ヘッダー伝搬フラグで制御できます。

プロトコル・ヘッダー伝搬を設定すると、以下のリストに説明するようにヘッダー情報をメッセージまたはターゲット・コンポーネントに流すことができます。

- JMS エクスポート要求

  メッセージ内で受信した JMS ヘッダーは、コンテキスト・サービスを介してターゲット・コンポーネントに伝搬されます。メッセージ内で受信した JMS プロパティーは、コンテキスト・サービスを介してターゲット・コンポーネントに伝搬されます。

- JMS エクスポート応答

  コンテキスト・サービスに設定されたすべての JMS ヘッダー・フィールドは、JMS エクスポート・バインディングに設定された静的プロパティーによってオーバーライドされていない限り、アウトバウンド・メッセージ内で使用されます。コンテキスト・サービスに設定されたすべてプロパティーは、JMS エクスポート・バインディングに設定された静的プロパティーによってオーバーライドされていない限り、アウトバウンド・メッセージ内で使用されます。

- JMS インポート要求

  コンテキスト・サービスに設定されたすべての JMS ヘッダー・フィールドは、JMS インポート・バインディングに設定された静的プロパティーによってオーバーライドされていない限り、アウトバウンド・メッセージ内で使用されます。コンテキスト・サービスに設定されたすべてプロパティーは、JMS

インポート・バインディングに設定された静的プロパティーによってオーバーライドされていない限り、アウトバウンド・メッセージ内で使用されます。

- JMS インポート応答

  メッセージ内で受信した JMS ヘッダーは、コンテキスト・サービスを介してターゲット・コンポーネントに伝搬されます。メッセージ内で受信した JMS プロパティーは、コンテキスト・サービスを介してターゲット・コンポーネントに伝搬されます。

外部クライアント:

サーバーは、WebSphere MQ JMS バインディングを使用して、外部クライアントとの間でメッセージを送受信できます。

外部クライアント (Web ポータルやエンタープライズ情報システムなど) は、エクスポートによって、アプリケーションの SCA コンポーネントにメッセージを送信できます。また、アプリケーション内の SCA コンポーネントがインポートによって外部クライアントを呼び出すこともできます。

WebSphere MQ JMS エクスポート・バインディングは、エクスポート・バインディングで指定された receive 宛先に着信する要求を listen するためのメッセージ駆動型 Bean (MDB) をデプロイします。send フィールドで指定された宛先は、呼び出されたアプリケーションが応答を返す場合にインバウンド要求に対する応答を送信するために使用されます。したがって、外部クライアントは、エクスポート・バインディングを介してアプリケーションを呼び出すことができます。

WebSphere MQ JMS インポートは外部クライアントにバインドし、外部クライアントにメッセージを配信できます。このメッセージは、外部クライアントからの応答を要求してもしなくても構いません。

WebSphere MQ を使用して外部クライアントと対話する方法について詳しくは、WebSphere MQ インフォメーション・センターを参照してください。

WebSphere MQ JMS バインディングのトラブルシューティング:

WebSphere MQ JMS バインディングで発生した問題を診断し、修正できます。

実装例外

MQ JMS のインポートおよびエクスポート実装は、さまざまなエラー状態に応じて、以下の 2 種類の例外のうちのいずれかを戻すことがあります。

- サービス・ビジネス例外: サービス・ビジネス・インターフェース (WSDL ポート・タイプ) で指定された障害が発生した場合に、この例外が戻されます。
- サービス・ランタイム例外: その他のすべてのケースで生成されます。ほとんどの場合、 cause 例外には元の例外 (JMSException) が含まれます。

  例えばインポートの場合、要求メッセージごとに 1 つの応答メッセージだけが戻されることを前提としています。そのため、複数の応答を受信した場合や遅延応答 (SCA の応答有効期限が切れた応答) を受信した場合は、サービス・ランタイム例外が throw されます。この場合、トランザクションはロールバックされ、応答メッセージはキューからバックアウトされるか、または Failed Event Manager によって処理されます。

Failed Event Manager に表示されない JMS ベースの WebSphere MQ SCA メッセージ

WebSphere MQ JMS の対話の失敗によって SCA メッセージが発生した場合は、Failed Event Manager でこのメッセージを見つけることになります。Failed Event Manager にこのようなメッセージが表示されない

場合は、基礎となるリスナー・ポートの最大再試行回数プロパティーの値が **1** 以上になっているかどうかを確認してください。この値を **1** 以上に設定すると、MQ JMS バインディングに対して SCA を呼び出す際に、Failed Event Manager と対話することができます。

誤用例: WebSphere MQ バインディングとの比較

WebSphere MQ JMS バインディングは、WebSphere MQ に対してデプロイされている JMS アプリケーションと相互協調処理するよう設計されています。これにより、メッセージは JMS メッセージ・モデルに基づいて公開されます。これに対し、WebSphere MQ インポートおよびエクスポートは、ネイティブ WebSphere MQ アプリケーションと相互協調処理することができ、WebSphere MQ メッセージ本体の内容全体をメディエーションに公開するように設計されています。

以下のシナリオでは、WebSphere MQ バインディングではなく WebSphere MQ JMS バインディングを使用して作成する必要があります。

- JMS メッセージ駆動型 Bean (MDB) を SCA モジュールから呼び出す。この MDB は、WebSphere MQ JMS プロバイダーに対してデプロイされています。WebSphere MQ JMS インポートを使用します。
- SCA モジュールを Java EE コンポーネント・サーブレットまたは EJB から JMS を介して呼び出すことができるようにする。WebSphere MQ JMS エクスポートを使用します。
- WebSphere MQ 上で転送中の JMS MapMessage の内容のメディエーションを実行する。WebSphere MQ JMS エクスポートとインポート、および適切なデータ・ハンドラーまたはデータ ・バインディングを組み合わせて使用します。

WebSphere MQ バインディングと WebSphere MQ JMS バインディングの相互協調処理が予期される状況があります。特に、Java EE WebSphere MQ アプリケーションと非 Java EE WebSphere MQ アプリケーション間をブリッジングする場合は、 WebSphere MQ エクスポートと WebSphere MQ JMS インポート (あるいはこの逆) を、適切なデータ・バインディングまたはメディエーション・モジュール (あるいはこの両方) と組み合わせて使用します。

例外の処理:

バインディングがどのように構成されているかによって、データ・ハンドラーまたはデータ・バインディングによる例外の処理方法が決まります。さらに、メディエーション・フローの特性により、そのような例外が throw された場合のシステムの動作が決まります。

データ・ハンドラーまたはデータ・バインディングがバインディングによって呼び出されるときには、さまざまな問題が発生する可能性があります。例えば、データ・ハンドラーが、ペイロードが破損しているメッセージを受信したり、誤った形式のメッセージを読み取るなどです。

バインディングによるそのような例外の処理方法は、データ・ハンドラーおよびデータ・バインディングの実装方法によって決まります。**DataBindingException** を throw するようにデータ・バインディングを設計することをお勧めします。

データ・ハンドラーの場合も同様です。データ・ハンドラーはデータ・バインディングによって呼び出されるため、データ・ハンドラー例外はデータ・バインディング例外にラップされます。したがって、**DataHandlerException** は、**DataBindingException** として報告されます。

DataBindingException 例外などのランタイム例外がスローされた場合:

- メディエーション・フローがトランザクションとして構成されている場合、デフォルトでは、手動でやり直し、または削除できるように JMS メッセージは Failed Event Manager に保管されます。

注: バインディングのリカバリー・モードを変更することによって、メッセージが Failed Event Manager に保管される代わりに、ロールバックされるようにできます。

- メディエーション・フローがトランザクションでない場合、例外はログに記録され、メッセージは失われます。

データ・ハンドラーの場合も同様です。データ・ハンドラーはデータ・バインディングによって呼び出されるため、データ・ハンドラー例外はデータ・バインディング例外内に生成されます。したがって、**DataHandlerException** は **DataBindingException** として報告されます。

WebSphere MQ バインディング

WebSphere MQ バインディングは、WebSphere MQ アプリケーションと Service Component Architecture (SCA) との接続を提供します。

WebSphere MQ エクスポートおよびインポート・バインディングは、サーバー環境から WebSphere MQ ベースのシステムと直接統合する場合に使用します。これにより、サービス統合バスの MQ リンクまたはクライアント・リンク機能を使用する必要がなくなります。

コンポーネントがインポートによって WebSphere MQ サービスと対話するときには、WebSphere MQ インポート・バインディングはデータの送信先のキューと応答を受信できるキューを使用します。

SCA モジュールが WebSphere MQ クライアントにサービスを提供するときには、WebSphere MQ エクスポート・バインディングは、要求の受信と応答の送信ができるキューを使用します。関数セレクターは、呼び出すターゲット・コンポーネントに対する操作へのマッピングを提供します。

MQ メッセージとの間のペイロード・データの変換は、MQ 本体データ・ハンドラーまたはデータ・バインディングを使用して行われます。 MQ メッセージとの間のヘッダー・データの変換は、MQ ヘッダー・データ・バインディングを使用して行われます。

サポートされる WebSphere MQ バージョンについては、Web ページIBM Business Process Manager のシステム要件のシステム要件を参照してください。

WebSphere MQ バインディングの概要:

WebSphere MQ バインディングにより、ネイティブ MQ ベースのアプリケーションとの統合が実現します。

WebSphere MQ 管理用タスク

WebSphere MQ システム管理者は、基盤となる WebSphere MQ キュー・マネージャーを作成する必要があります。このキュー・マネージャーは、WebSphere MQ バインディングが含まれるアプリケーションを実行する前に、これらのバインディングによって使用されます。

WebSphere 管理用タスク

Websphere の MQ リソース・アダプターの「**ネイティブ・ライブラリー・パス (Native library path)**」プロパティーを、サーバーでサポートされる WebSphere MQ バージョンに設定して、サーバーを再始動する必要があります。これにより、サポートされるバージョンの WebSphere MQ のライブラリーが使用されることが保証されます。詳細なハードウェアおよびソフトウェア要件については、IBM サポート・ページを参照してください。

WebSphere MQ インポート・バインディング

WebSphere MQ インポート・バインディングにより、SCA モジュール内のコンポーネントは、外部の WebSphere MQ ベースのアプリケーションが提供するサービスと通信できるようになります。サポートされているバージョンの WebSphere MQ を使用していることを確認してください。詳細なハードウェアおよびソフトウェア要件については、IBM サポート・ページを参照してください。

外部 WebSphere MQ システムとの対話では、要求を送信し、応答を受信するためのキューが使用されます。

WebSphere MQ インポート・バインディングでは、呼び出されている操作のタイプに応じた以下の 2 種類の使用シナリオがサポートされています。

- 片方向: WebSphere MQ インポートは、インポート・バインディングの「**送信宛先キュー (Send destination queue)**」フィールドに構成されたキューにメッセージを送信します。 MQMD ヘッダーの replyTo フィールドには何も送信しません。
- 両方向 (要求/応答): WebSphere MQ インポートは、「**送信宛先キュー (Send destination queue)**」フィールドに構成されたキューにメッセージを送信します。

  receive キューは、replyTo MQMD ヘッダー・フィールドに設定されます。 受信キューで listen するためのメッセージ駆動型 Bean (MDB) をデプロイします。応答を受信すると、MDB は応答をコンポーネントに返します。

  インポート・バインディングは、要求メッセージ ID (デフォルト) または要求メッセージ相関 ID から応答メッセージ相関 ID がコピーされていることを期待するように (「**応答相関スキーム**」フィールドを使用して) 構成することができます。

重要な点として、 WebSphere MQ は非同期バインディングであることに注意してください。呼び出し側コンポーネントが WebSphere MQ インポートを同期的に呼び出すと (両方向操作の場合)、呼び出し側コンポーネントは、WebSphere MQ サービスからの応答が返されるまでブロックされます。

135 ページの図 38 は、インポートがどのように外部サービスにリンクされているのかを示しています。

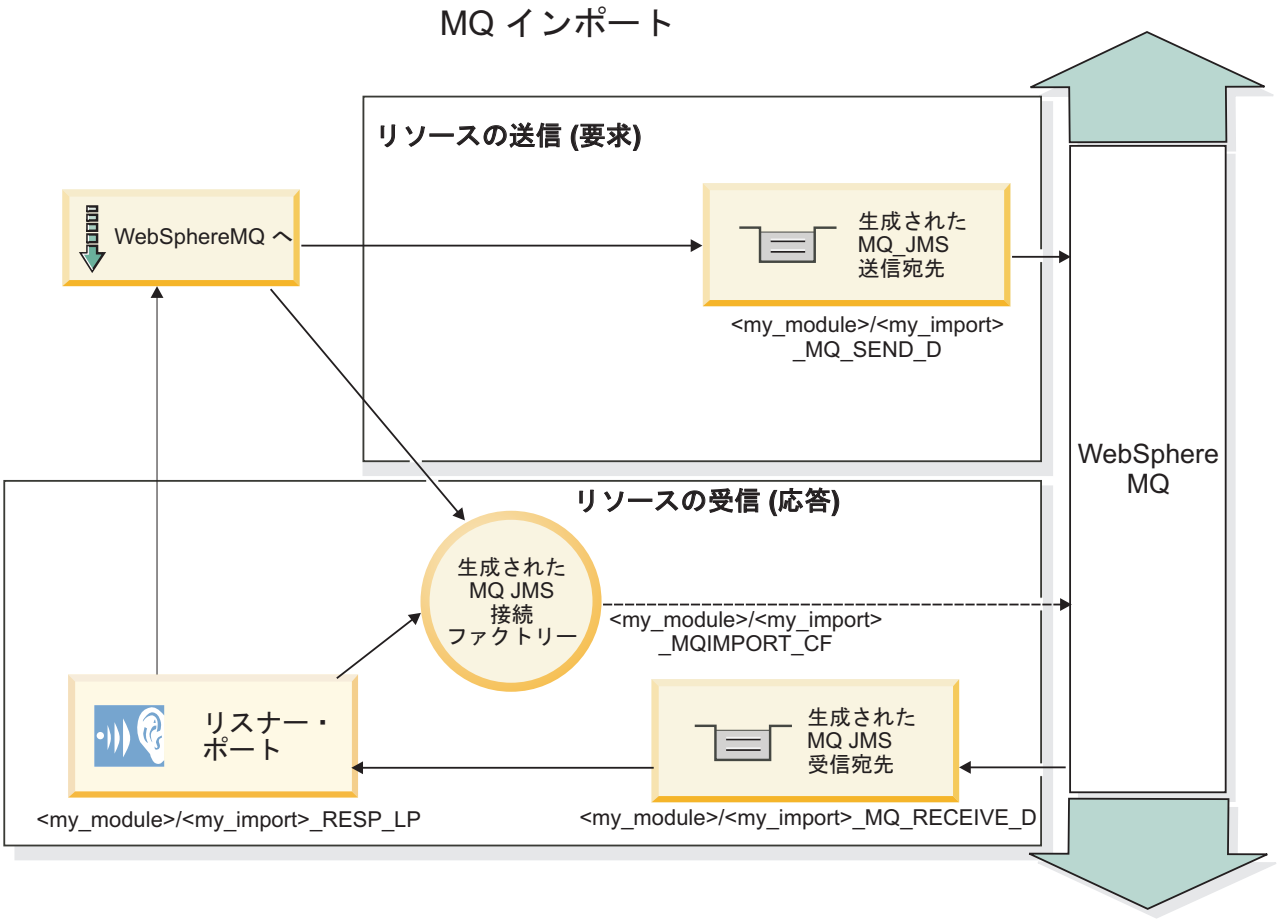


図 38. WebSphere MQ インポート・バインディングのリソース

WebSphere MQ エクスポート・バインディング

WebSphere MQ エクスポート・バインディングは、SCA モジュールが外部の WebSphere MQ ベースにアプリケーションにサービスを提供する手段を提供します。

エクスポート・バインディングで指定された「**受信宛先キュー (Receive destination queue)**」に着信する要求を listen するため、MDB がデプロイされます。「**送信宛先キュー (Send destination queue)**」 フィールドで指定されたキューは、呼び出されたコンポーネントが応答を返す場合にインバウンド要求に対する応答を送信するために使用されます。応答メッセージの replyTo フィールドで指定されたキューは、「**送信宛先キュー (Send destination queue)**」フィールドで指定されたキューをオーバーライドします。

136 ページの図 39 は、外部の要求側がどのようにエクスポートにリンクされているのかを示しています。

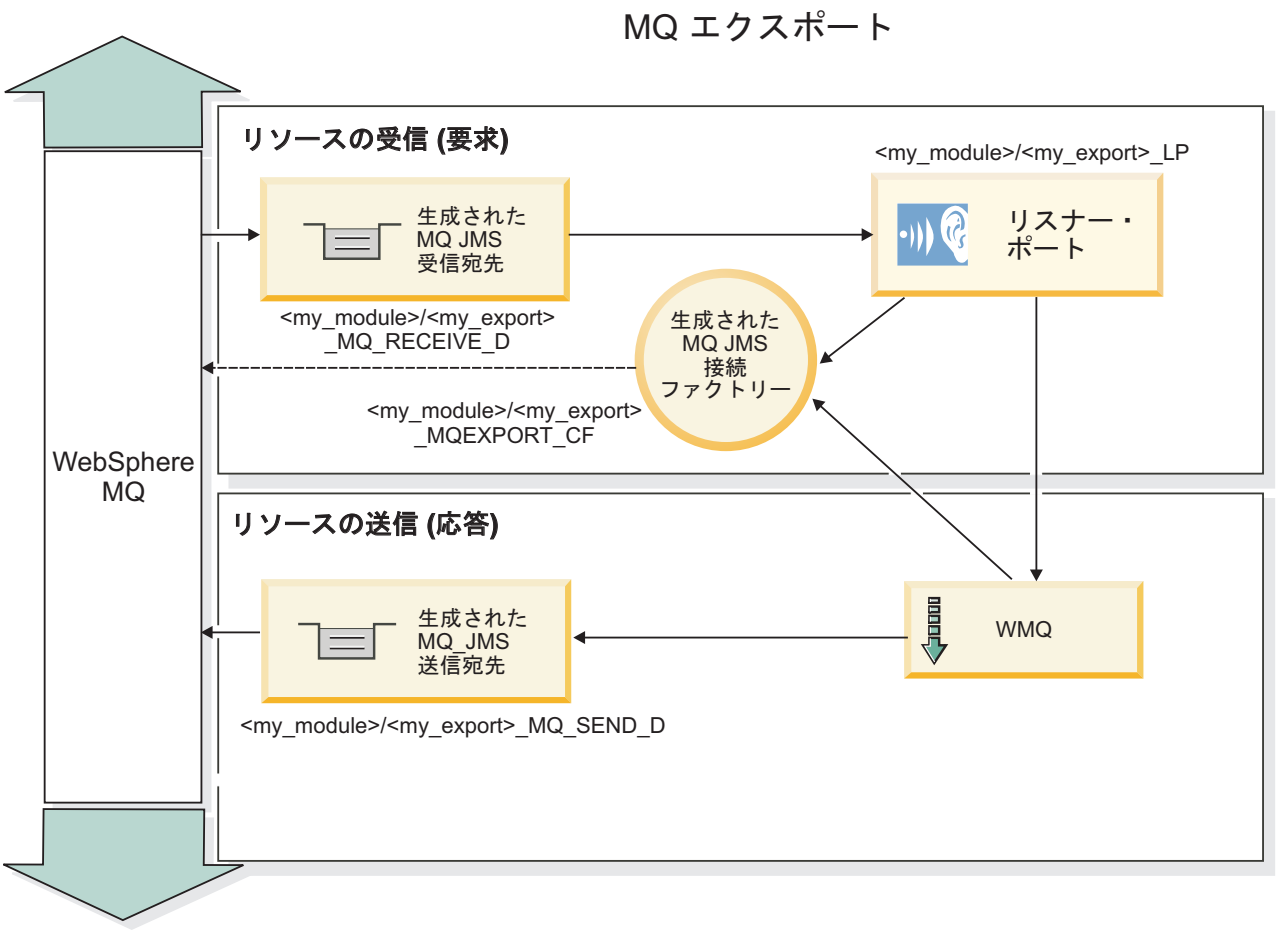


図 39. WebSphere MQ エクスポート・バインディングのリソース

注: 135 ページの図 38 および 図 39 は、旧バージョンの IBM Business Process Manager のアプリケーションが外部サービスにリンクされる方法を示しています。 IBM Business Process Manager バージョン 7.0 用に開発されたアプリケーションについては、リスナー・ポートと接続ファクトリーではなくアクティベーション・スペックが使用されます。

WebSphere MQ バインディングの主な特徴:

WebSphere MQ バインディングの主な特徴は、ヘッダー、Java EE 成果物、および作成される Java EE リソースです。

相関スキーム

WebSphere MQ 要求/応答アプリケーションでは、応答メッセージと要求を相関付けるさまざまな手法の 1 つを使用できます。これらの手法には、MQMD MessageID フィールドと CorrelID フィールドが関連します。ほとんどの場合、要求側はキュー・マネージャーに MessageID を選択させ、応答アプリケーションがこれを応答の CorrelID にコピーすることを想定します。多くの場合、要求側と応答アプリケーションは、使用される相関手法を暗黙的に認識しています。応答アプリケーションは、要求の Report フィールドに設定されているさまざまなフラグの指示に従って、これらのフィールドを処理することもあります。

WebSphere MQ メッセージのエクスポート・バインディングを構成するときには、以下のオプションを使用できます。

応答の MsgId のオプション:

新規 MsgID (New MsgID)
キュー・マネージャーが応答の固有の MsgId を選択できるようにします (デフォルト)。

要求の MsgID からコピーする (Copy from Request MsgID)
要求の MsgId フィールドから MsgId フィールドをコピーします。

SCA メッセージからコピーする (Copy from SCA message)
SCA 応答メッセージの WebSphere MQ ヘッダーに含まれる値を MsgId に設定します。この値が存在しない場合は、キュー・マネージャーに新規 Id を定義させます。

Report オプションを使用 (As Report Option)
要求の MQMD の Report フィールドを検査して、MsgId の処理方法を認識します。MQRO_NEW_MSG_ID オプションと MQRO_PASS_MSG_ID オプションがサポートされています。これらのオプションは、それぞれ「新規 MsgID (New MsgID)」および「要求の MsgID からコピーする (Copy from Request MsgID)」と同様に動作します。

応答の CorrelId のオプション:

要求の MsgID からコピーする (Copy from Request MsgID)
要求の MsgId フィールドから CorrelId フィールドをコピーします (デフォルト)。

要求の CorrelID からコピーする (Copy from Request CorrelID)
要求の CorrelId フィールドから CorrelId フィールドをコピーします。

SCA メッセージからコピーする (Copy from SCA message)
SCA 応答メッセージの WebSphere MQ ヘッダーに含まれる値を CorrelId に設定します。または、この値が存在しない場合は、空白のままにします。

Report オプションを使用 (As Report Option)
要求の MQMD の Report フィールドを検査して、CorrelId の処理方法を認識します。MQRO_COPY_MSG_ID_TO_CORREL_ID オプションと MQRO_PASS_CORREL_ID オプションがサポートされています。これらのオプションは、それぞれ「要求の MsgID からコピーする (Copy from Request MsgID)」および「要求の CorrelID からコピーする (Copy from Request CorrelID)」と同様に動作します。

WebSphere MQ メッセージのインポート・バインディングを構成するときには、以下のオプションを使用できます。

要求の MsgId のオプション:

新規 MsgID (New MsgID)
キュー・マネージャーが要求の固有の MsgId を選択できるようにします (デフォルト)。

SCA メッセージからコピーする (Copy from SCA message)
SCA 要求メッセージの WebSphere MQ ヘッダーに含まれる値を MsgId に設定します。この値が存在しない場合は、キュー・マネージャーに新規 Id を定義させます。

応答相関のオプション:

応答の CorrelID を MsgId からコピーする (Response has CorrelID copied from MsgId)
応答メッセージの CorrelId フィールドが、要求の MsgId ごとに設定されていることを想定します (デフォルト)。

応答の MsgID を MsgId からコピーする (Response has MsgID copied from MsgId)
応答メッセージの MsgId フィールドが、要求の MsgId ごとに設定されていることを想定します。

応答の CorrelID を CorrelId からコピーする (Response has CorrelID copied from CorrelId)
応答メッセージの CorrelId フィールドが、要求の CorrelId ごとに設定されていることを想定します。

Java EE リソース

WebSphere MQ バインディングを Java EE 環境にデプロイすると、いくつかの Java EE リソースが作成されます。

パラメーター

MQ 接続ファクトリー
クライアントが WebSphere MQ プロバイダーとの接続を作成するために使用します。

応答接続ファクトリー
送信宛先が受信宛先とは異なるキュー・マネージャー上にある場合に、SCA MQ ランタイムが使用します。

アクティベーション・スペック
MQ JMS アクティベーション・スペックは、1 つ以上のメッセージ駆動型 Bean に関連付けられており、これらの Bean がメッセージを受信するのに必要な構成を提供します。

宛先

- 送信宛先: インポートの場合は、要求または出力メッセージが送信される宛先です。エクスポートの場合は、応答メッセージが送信される宛先です。ただし、着信メッセージの MQMD ReplyTo ヘッダー・フィールドにより置き換えられた場合は、その宛先が優先されます。
- 受信宛先: 応答/要求または着信メッセージが格納される宛先です。

WebSphere MQ ヘッダー:

WebSphere MQ ヘッダーには、Service Component Architecture (SCA) メッセージへの変換に関する特定の規則が組み込まれています。

WebSphere MQ メッセージは、システム・ヘッダー (MQMD)、場合によっては 1 つ以上のその他の MQ ヘッダー (システムまたはカスタム)、 およびメッセージ本体で構成されます。メッセージ内に複数のメッセージ・ヘッダーがある場合、ヘッダーの順序が意味を持ちます。

各ヘッダーには、次のヘッダーの構造を記述する情報が含まれています。 MQMD は最初のヘッダーを記述します。

MQ ヘッダーの解析方法

MQ ヘッダーの解析には、MQ ヘッダー・データ・バインディングが使用されます。以下のヘッダーは自動的にサポートされます。

- MQRFH
- MQRFH2
- MQCIH
- MQIIH

MQH で始まるヘッダーについては、処理方法が異なります。ヘッダーの特定のフィールドは解析されません。これらのフィールドは未解析バイトとして維持されます。

その他の MQ ヘッダーについては、カスタム MQ ヘッダー・データ・バインディングを作成して解析することができます。

MQ ヘッダーへのアクセス方法

製品内で MQ ヘッダーにアクセスするには、以下の 2 つの方法のいずれかを使用します。

- メディエーション内でサービス・メッセージ・オブジェクト (SMO) を使用
- ContextService API を使用

MQ ヘッダーは、内部では SMO MQHeader エレメントによって表されます。 MQHeader は MQControl を拡張するヘッダー・データのコンテナーですが、anyType の値エレメントが含まれています。これには MQMD、MQControl (MQ メッセージ本体制御情報)、およびその他の MQ ヘッダーのリストが含まれます。

- MQMD は、WebSphere MQ メッセージ記述の内容を表します。ただし、本体の構造とエンコードを定義する情報は含まれません。
- MQControl には、メッセージ本体の構造とエンコードを定義する情報が含まれています。
- MQHeaders には、MQHeader オブジェクトのリストが含まれています。

MQ ヘッダー・チェーンはアンワインドされるため、SMO 内部では各 MQ ヘッダーが独自の制御情報 (CCSID、Encoding、および Format) を持つことになります。ヘッダーは簡単に追加または削除できます。他のヘッダー・データを変更する必要はありません。

MQMD のフィールドの設定

MQMD を更新するときは、コンテキスト API を使用するか、またはメディエーション内でサービス・メッセージ・オブジェクト (SMO) を使用します。以下のフィールドは、自動的にアウトバウンド MQ メッセージに伝搬されます。

- エンコード
- CodedCharacterSet
- フォーマット
- レポート
- 有効期限
- フィードバック
- 優先順位
- パーシスタンス
- CorrelId
- MsgFlags

以下のプロパティーがアウトバウンド MQ メッセージに伝搬されるように、インポートまたはエクスポートの MQ バインディングを構成します。

MsgID
「**要求メッセージ ID オプション**」に 「SCA メッセージからのコピー」を設定します。

MsgType
「**要求/応答操作のメッセージ・タイプを MQMT_DATAGRAM または MQMT_REQUEST に設定**」チェック・ボックスのチェック・マークを外します。

ReplyToQ
「**要求メッセージの応答先キューのオーバーライド**」チェック・ボックスのチェック・マークを外します。

ReplyToQMgr
「**要求メッセージの応答先キューのオーバーライド**」チェック・ボックスのチェック・マークを外します。

バージョン 7.0 以降では、JNDI 宛先定義のカスタム・プロパティーを使用して、コンテキスト・フィールドをオーバーライドすることが可能です。送信宛先上の値 SET_IDENTITY_CONTEXT を使用してカスタム・プロパティー MDCTX を設定し、以下のフィールドをアウトバウンド MQ メッセージに伝搬します。

- UserIdentifier
- AppIdentityData

送信宛先上の値 SET_ALL_CONTEXT を使用してカスタム・プロパティー MDCTX を設定し、以下のプロパティーをアウトバウンド MQ メッセージに伝搬します。

- UserIdentifier
- AppIdentityData
- PutApplType
- PutApplName
- ApplOriginData

アウトバウンド MQ メッセージに伝搬されないフィールドもあります。以下のフィールドは、メッセージの送信時にオーバーライドされます。

- BackoutCount
- AccountingToken
- PutDate
- PutTime
- Offset
- OriginalLength

WebSphere MQ バインディングへの MQCIH の静的な追加:

IBM Business Process Manager では、メディエーション・モジュールを使用することなく、MQCIH ヘッダー情報を静的に追加することができます。

メッセージに MQCIH ヘッダー情報を追加するには、いくつかの方法があります (例えば、ヘッダー・セッター・メディエーション・プリミティブを使用するのも、1 つの方法です)。このヘッダー情報を静的に追加すると、追加のメディエーション・モジュールを使用しなくてもよいので便利です。静的なヘッダー情報 (CICS® プログラム名、トランザクション ID、その他のデータ形式ヘッダー詳細情報など) を定義し、WebSphere MQ バインディングの一部として追加することができます。

MQCIH ヘッダー情報を静的に追加するには、WebSphere MQ、MQ CICS Bridge、CICS をすべて構成する必要があります。

Integration Designer を使用して、MQCIH ヘッダー情報に必要な静的値を WebSphere MQ インポートに設定できます。

到着したメッセージが WebSphere MQ インポートによって処理される際に、MQCIH ヘッダー情報がメッセージ内に存在するかどうかが確認されます。MQCIH が存在する場合、WebSphere MQ インポート内に

定義されている静的値を使用して、メッセージ内の対応する動的値がオーバーライドされます。MQCIH が存在しない場合、メッセージ内に MQCIH が作成され、WebSphere MQ インポート内で定義されている静的値が追加されます。

WebSphere MQ インポート内に定義されている静的値には、メソッドごとに固有の値が設定されます。同じ WebSphere MQ インポート内のさまざまなメソッドごとに、異なる静的 MQCIH 値を指定することができます。

この機能は、MQCIH に特定のヘッダー情報が含まれていない場合にデフォルト値を提供する目的では使用されません。この理由は、WebSphere MQ インポートで定義された静的な値によって、着信メッセージで指定された対応する値が上書きされるためです。

外部クライアント:

IBM Business Process Manager は、WebSphere MQ バインディングを使用して、外部クライアントとの間でメッセージを送受信できます。

外部クライアント (Web ポータルやエンタープライズ情報システムなど) は、エクスポートによって、アプリケーションの SCA コンポーネントにメッセージを送信できます。また、アプリケーション内の SCA コンポーネントがインポートによって外部クライアントを呼び出すこともできます。

WebSphere MQ エクスポート・バインディングは、エクスポート・バインディングで指定された receive 宛先に着信する要求を listen するためのメッセージ駆動型 Bean (MDB) をデプロイします。send フィールドで指定された宛先は、呼び出されたアプリケーションが応答を返す場合にインバウンド要求に対する応答を送信するために使用されます。したがって、外部クライアントは、エクスポート・バインディングを介してアプリケーションを呼び出すことができます。

WebSphere MQ インポートは外部クライアントにバインドし、外部クライアントにメッセージを配信できます。このメッセージは、外部クライアントからの応答を要求してもしなくても構いません。

WebSphere MQ を使用して外部クライアントと対話する方法について詳しくは、WebSphere MQ インフォメーション・センターを参照してください。

WebSphere MQ バインディングのトラブルシューティング:

WebSphere MQ バインディングで発生する障害または失敗の状態を診断し、このような状態を修正できます。

主な障害状態

WebSphere MQ バインディングの主な障害状態は、トランザクションの意味構造、WebSphere MQ 構成、またはその他のコンポーネントの既存の動作への参照に基づいて判別されます。主な障害状態は以下のとおりです。

- WebSphere MQ キュー・マネージャーまたはキューに接続できない。

  WebSphere MQ に接続してメッセージを受信できない場合は、MDB リスナー・ポートを開始することができません。この状態は、WebSphere Application Server ログに記録されます。正常に取得されるまで (または WebSphere MQ により期限切れとなるまで)、永続メッセージは WebSphere MQ キューに残ります。

  WebSphere MQ に接続してアウトバウンド・メッセージを送信できない場合は、送信操作を制御するトランザクションがロールバックされます。

- インバウンド・メッセージを解析できないか、アウトバウンド・メッセージを構成できない。

  データ・バインディングが失敗すると、作業を制御するトランザクションがロールバックされます。
- アウトバウンド・メッセージを送信できない。

  メッセージを送信できないと、関連するトランザクションがロールバックされます。
- 複数の応答メッセージまたは予期しない応答メッセージが返される。

  インポートの場合、要求メッセージごとに 1 つの応答メッセージだけが戻されることを前提としています。そのため、複数の応答を受信した場合や遅延応答 (SCA の応答有効期限が切れた応答) を受信した場合は、サービス・ランタイム例外が throw されます。この場合、トランザクションはロールバックされ、応答メッセージはキューからバックアウトされるか、または Failed Event Manager によって処理されます。

誤用例: WebSphere MQ JMS バインディングとの比較

通常、WebSphere MQ インポートおよびエクスポートは、ネイティブ WebSphere MQ アプリケーションと相互協調処理し、WebSphere MQ メッセージ本体の内容全体をメディエーションに公開するように設計されています。一方、WebSphere MQ JMS バインディングは、WebSphere MQ に対してデプロイされている JMS アプリケーションと相互協調処理するよう設計されています。これにより、メッセージは JMS メッセージ・モデルに基づいて公開されます。

以下のシナリオでは、WebSphere MQ バインディングではなく WebSphere MQ JMS バインディングを使用して作成する必要があります。

- JMS メッセージ駆動型 Bean (MDB) を SCA モジュールから呼び出す。この MDB は、WebSphere MQ JMS プロバイダーに対してデプロイされています。WebSphere MQ JMS インポートを使用します。
- SCA モジュールを Java EE コンポーネント・サーブレットまたは EJB から JMS を介して呼び出すことができるようにする。WebSphere MQ JMS エクスポートを使用します。
- WebSphere MQ 上で転送中の JMS MapMessage の内容のメディエーションを実行する。WebSphere MQ JMS エクスポートとインポート、および適切なデータ・バインディングを組み合わせて使用します。

WebSphere MQ バインディングと WebSphere MQ JMS バインディングの相互協調処理が予期される状況があります。特に、Java EE WebSphere MQ アプリケーションと非 Java EE WebSphere MQ アプリケーション間をブリッジングする場合は、 WebSphere MQ エクスポートと WebSphere MQ JMS インポート (あるいはこの逆) を、適切なデータ・バインディングまたはメディエーション・モジュール (あるいはこの両方) と組み合わせて使用します。

未配布メッセージ

構成エラーなどが原因で WebSphere MQ がメッセージを対象の宛先に配信できない場合、メッセージは指定されている送達不能キューに送信されます。

このとき、メッセージ本体の先頭には、送達不能ヘッダーが追加されます。このヘッダーには、失敗の原因、元の宛先、およびその他の情報が含まれています。

Failed Event Manager に表示されない MQ ベースの SCA メッセージ

WebSphere MQ の対話の失敗によって SCA メッセージが発生した場合は、Failed Event Manager でこのメッセージを見つけることになります。Failed Event Manager にこれらのメッセージが表示されない場合は、基盤となる WebSphere MQ 宛先の最大配信失敗回数の値が 1 よりも大きいことを確認してくださ

い。この値を 2 以上に設定すると、WebSphere MQ バインディングに対する SCA 呼び出しの間の Failed Event Manager との対話が可能になります。

誤ったキュー・マネージャーに再生される MQ 失敗イベント

事前定義の接続ファクトリーをアウトバウンド接続に使用する場合、接続プロパティーは、インバウンド接続に使用されるアクティベーション・スペックで定義されている接続プロパティーと一致していなければなりません。

事前定義の接続ファクトリーは、失敗イベントの再生時に接続を作成するために使用されるので、メッセージをもともと受信したのと同じキュー・マネージャーを使用するように構成する必要があります。

例外の処理:

バインディングがどのように構成されているかによって、データ・ハンドラーまたはデータ・バインディングによる例外の処理方法が決まります。さらに、メディエーション・フローの特性により、そのような例外が throw された場合のシステムの動作が決まります。

データ・ハンドラーまたはデータ・バインディングがバインディングによって呼び出されるときには、さまざまな問題が発生する可能性があります。例えば、データ・ハンドラーが、ペイロードが破損しているメッセージを受信したり、誤った形式のメッセージを読み取るなどです。

バインディングによるそのような例外の処理方法は、データ・ハンドラーおよびデータ・バインディングの実装方法によって決まります。**DataBindingException** を throw するようにデータ・バインディングを設計することをお勧めします。

データ・ハンドラーの場合も同様です。データ・ハンドラーはデータ・バインディングによって呼び出されるため、データ・ハンドラー例外はデータ・バインディング例外にラップされます。したがって、**DataHandlerException** は、**DataBindingException** として報告されます。

DataBindingException 例外などのランタイム例外がスローされた場合:

- メディエーション・フローがトランザクションとして構成されている場合、デフォルトでは、手動でやり直し、または削除できるように JMS メッセージは Failed Event Manager に保管されます。

  注: バインディングのリカバリー・モードを変更することによって、メッセージが Failed Event Manager に保管される代わりに、ロールバックされるようにできます。
- メディエーション・フローがトランザクションでない場合、例外はログに記録され、メッセージは失われます。

データ・ハンドラーの場合も同様です。データ・ハンドラーはデータ・バインディングによって呼び出されるため、データ・ハンドラー例外はデータ・バインディング例外内に生成されます。したがって、**DataHandlerException** は **DataBindingException** として報告されます。

バインディングの制限

バインディングには、その使用時に以下に示すいくつかの制限があります。

MQ バインディングの制限:

MQ バインディングは、その使用にいくつかの制限があります。これらの制限を以下に示します。

パブリッシュ/サブスクライブ方式のメッセージ配布機能なし

WMQ 自体はパブリッシュ/サブスクライブをサポートしますが、メッセージ配布のパブリッシュ/サブスクライブ方式は、現時点では、MQ バインディングによってサポートされていません。ただし、MQ JMS バインディングは、この配布方式をサポートしています。

共用受信キュー

複数の WebSphere MQ エクスポート・バインディングおよびインポート・バインディングでは、構成されている受信キュー上のメッセージはすべて、そのエクスポートまたはインポートの対象であると期待します。インポート・バインディングとエクスポート・バインディングを構成するときは、以下の点に考慮する必要があります。

- 各 MQ インポートにはそれぞれ異なる受信キューが必要です。これは、MQ インポート・バインディングでは、その受信キュー上のすべてのメッセージは自身が送信した要求に対する応答であると想定されるためです。受信キューが複数のインポートで共用されている場合、誤ったインポートで応答が受信される場合があり、その応答は元の要求メッセージに相関されません。
- 各 MQ エクスポートにはそれぞれ異なる受信キューが必要です。これは、そうでない場合、どのエクスポートで特定の要求メッセージが取得されるのか予測できないためです。
- MQ インポートと MQ エクスポートで、同じ送信キューを指すことはできます。

JMS、MQ JMS、および汎用 JMS バインディングの制限:

JMS および MQ JMS バインディングには、いくつかの制限があります。

デフォルトのバインディングを生成する意味

JMS、MQ JMS、および汎用 JMS バインディングを使用する場合の制限については、以下のセクションで説明します。

- デフォルトのバインディングを生成する意味
- 応答相関スキーム
- 双方向言語サポート

バインディングを生成するときに、自分で値を入力することを選択しなかった場合は、いくつかのフィールドにデフォルトとして値が自動的に入力されます。例えば、接続ファクトリー名が自動的に作成されます。アプリケーションをサーバー上に置き、このアプリケーションにクライアントからリモートでアクセスすることが分かっている場合は、デフォルトを採用するのではなく、バインディングの作成時に JNDI 名を入力してください。これらの値を実行時に管理コンソールから管理することになる可能性が高いためです。

ただし、デフォルトを確定した後で、リモート・クライアントからアプリケーションにアクセスできないことが分かった場合は、管理コンソールを使用して接続ファクトリーの値を明示的に設定できます。接続ファクトリー設定のプロバイダー・エンドポイント・フィールドを見つけて、<server_hostname>:7276 (デフォルトのポート番号を使用している場合) などの値を追加します。

応答相関スキーム

要求/応答操作でメッセージの相関をとるために使用される CorrelationId To CorrelationId 応答相関スキームを使用する場合は、メッセージ内に動的相関 ID が必要です。

メディエーション・フロー・エディターを使用してメディエーション・モジュール内に動的相関 ID を作成するには、JMS バインディングを使用するインポートの前に XSLT ノードを追加します。XSLT マッピ

ング・エディターを開きます。ターゲット・メッセージには、既知の Service Component Architecture ヘッダーを使用できます。ソース・メッセージ内の固有 ID が入っているフィールドをドラッグして、ターゲット・メッセージの JMS ヘッダー内にある相関 ID にドロップします。

双方向言語サポート

実行時の Java Naming and Directory Interface (JNDI) 名に対しては、ASCII 文字のみがサポートされます。

共用受信キュー

複数のエクスポート・バインディングとインポート・バインディングでは、構成されている受信キュー上のメッセージはすべて、そのエクスポートまたはインポートの対象であると期待します。インポート・バインディングとエクスポート・バインディングを構成するときは、以下の点に考慮する必要があります。

- 各インポート・バインディングにはそれぞれ異なる受信キューが必要です。これは、インポート・バインディングでは、その受信キュー上のすべてのメッセージは自身が送信した要求に対する応答であると想定されるためです。受信キューが複数のインポートで共用されている場合、誤ったインポートで応答が受信される場合があり、その応答は元の要求メッセージに相関されません。
- 各エクスポートにはそれぞれ異なる受信キューが必要です。これは、そうでない場合、どのエクスポートで特定の要求メッセージが取得されるのか予測できないためです。
- インポートとエクスポートで、同じ送信キューを指すことはできます。

ビジネス・オブジェクト

コンピューター・ソフトウェア業界では、ビジネス・オブジェクトを使用して、アプリケーション処理の対象となるビジネス・データを自然に表現するプログラミング・モデルとフレームワークをいくつか開発してきました。

一般に、これらのビジネス・オブジェクトには以下のような特徴があります。

- 業界標準を使用して定義される
- データベース表またはエンタープライズ情報システムにデータを透過的にマップする
- リモート呼び出しプロトコルをサポートする
- アプリケーション・プログラミング用のデータ・プログラミング・モデルの基盤を提供する

ツールの観点からは、Integration Designer が、異なるドメインからのさまざまなビジネス・エンティティーを表すための共通ビジネス・オブジェクト・モデルを開発者に提供します。開発者が開発時にこのモデルを使用すると、ビジネス・オブジェクトを XML スキーマ定義として定義できるようになります。

XML スキーマ定義で定義されたビジネス・データは、実行時に Java ビジネス・オブジェクトとして表現されます。このモデルにおけるビジネス・オブジェクトは、初期版のサービス・データ・オブジェクト (SDO) 仕様に大まかに基づいており、ビジネス・データの操作に必要なプログラミング・モデル・アプリケーション・インターフェース一式が用意されています。

ビジネス・オブジェクトの定義

Integration Designer でビジネス・オブジェクト・エディターを使用して、ビジネス・オブジェクトを定義することができます。ビジネス・オブジェクト・エディターは、ビジネス・オブジェクトを XML スキーマ定義として格納します。

XML スキーマを使用してビジネス・オブジェクトを定義した場合、以下のような利点があります。

- XML スキーマには、標準ベースのデータ定義モデルと、異機種のシステム間とアプリケーション間の相互運用を実現するための基礎が用意されています。XML スキーマは Web サービス記述言語 (WSDL) と連携して使用され、コンポーネント間、アプリケーション間、システム間の、標準ベースのインターフェース規約を提供します。
- XML スキーマは、リッチ・データ定義モデルを使用してビジネス・データを表現します。このモデルには、複合タイプ、単純タイプ、ユーザー定義タイプ、タイプの継承、カーディナリティーなどのフィーチャーが含まれています。
- ビジネス・オブジェクトは、ビジネス・インターフェースと Web サービス記述言語で定義されたデータを使用して定義することができます。また、業界標準組織の XML スキーマや別のシステムまたはアプリケーションの XML スキーマを使用して定義することもできます。Integration Designer は、こうしたビジネス・オブジェクトを直接インポートすることができます。

Integration Designer は、データベースとエンタープライズ情報システムからビジネス・データをディスカバーし、このビジネス・データの標準ベース XML スキーマのビジネス・オブジェクト定義を生成するための機能もサポートしています。この方法で生成されたビジネス・オブジェクトは、*アプリケーション固有*のビジネス・オブジェクトと呼ばれることがよくあります。これは、企業の情報システムで定義されたビジネス・データの構造と同じような構造がこのビジネス・オブジェクトで使用されるためです。

さまざまな情報システムのデータをプロセスで操作する場合は、ビジネス・データのさまざまな異なる表現 (CustomerEIS1 および CustomerEIS2、または OrderEIS1 および OrderEIS2 など) を 1 つの正規表現 (Customer または Order など) に変換することが重要になる場合があります。このような正規表現は、*汎用*ビジネス・オブジェクトと呼ばれることがよくあります。

ビジネス・オブジェクト定義は、多くの場合 (特に、汎用ビジネス・オブジェクトの場合)、複数のアプリケーションで使用されます。こうしたビジネス・オブジェクト定義の再使用をサポートするため、Integration Designer では、ビジネス・オブジェクトをライブラリー内で作成して複数のアプリケーション・モジュールに関連付けることができます。

Service Component Architecture (SCA) アプリケーション・モジュールによって提供されて使用されるサービスの規約と、アプリケーション・モジュール内のコンポーネントの作成に使用される規約は、Web サービス記述言語 (WSDL) を使用して定義されます。 WSDL により、規約の操作とビジネス・オブジェクトの両方を記述することができます。これらの操作とビジネス・オブジェクトは、ビジネス・データを表現するために XML スキーマによって定義されます。

ビジネス・オブジェクトの操作

Service Component Architecture (SCA) には、アプリケーション・モジュールを定義するためのフレームワーク、SCA が提供するサービス、SCA がコンシュームするサービス、アプリケーション・モジュールのビジネス・ロジックを提供するコンポーネント構成が用意されています。ビジネス・オブジェクトは、アプリケーション内で重要な役割を担っています。ビジネス・オブジェクトにより、サービス規約とコンポーネント規約を記述するためのビジネス・データと、コンポーネントで操作するビジネス・データが定義されます。

以下の図は、SCA アプリケーション・モジュールを示したものです。この図では、さまざまな場所で開発者がビジネス・オブジェクト使用して作業を行っています。

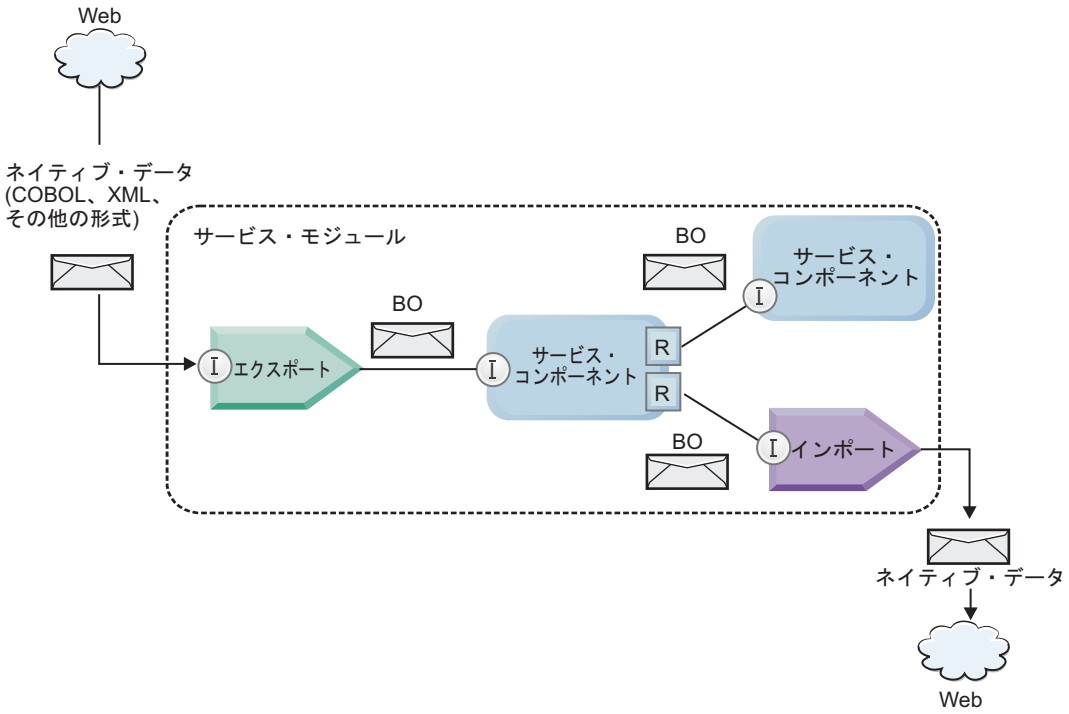


図 40. ビジネス・オブジェクトは、アプリケーション内のサービス間を流れるデータを表します。

注: このトピックでは、ビジネス・オブジェクトが SCA アプリケーション・モジュールでどのように使用されるかについて説明します。Java インターフェースを使用している場合は、SCA アプリケーション・モジュールで Java オブジェクトを処理することもできます。

ビジネス・オブジェクト・プログラミング・モデル

ビジネス・オブジェクト・プログラミング・モデルは、以下の項目を表す一連の Java インターフェースから構成されています。

- ビジネス・オブジェクトの定義とインスタンス・データ
- ビジネス・オブジェクト上での操作をサポートする一連のサービス

ビジネス・オブジェクト・タイプの定義は、commonj.sdo.Type インターフェースと commonj.sdo.Property インターフェースによって表されます。ビジネス・オブジェクト・プログラミング・モデルには、XML スキーマの複合タイプ情報を Type インターフェースにマップし、複合タイプ定義内の各エレメントを Property インターフェースにマップするための一連のルールが用意されています。

ビジネス・オブジェクト・インターフェースは、commonj.sdo.DataObject インターフェースによって表されます。ビジネス・オブジェクト・プログラミング・モデルには、タイプは指定されていません。そのため、同じ commonj.sdo.DataObject インターフェースを使用して、Customer や Order などの異なるビジネス・オブジェクト定義を表すことができます。設定可能なプロパティーと、各ビジネス・オブジェクトから取得できるプロパティーについては、各ビジネス・オブジェクトに関連する XML スキーマ内のタイプ情報によって定義されます。

ビジネス・オブジェクト・プログラミング・モデルの動作は、サービス・データ・オブジェクト 2.1 仕様に基づきます。詳しい情報については、次の Web ページで SDO 2.1 for Java の仕様、チュートリアル、および Javadoc を参照してください: http://osoa.org/display/Main/Service+Data+Objects+Specifications

ビジネス・オブジェクト・サービスは、ビジネス・オブジェクト上のさまざまなライフサイクル操作 (作成、同等化、解析、直列化など) をサポートしています。

ビジネス・オブジェクト・プログラミング・モデルの詳細については、ビジネス・オブジェクト・サービスを使用したプログラミングと『パッケージ com.ibm.websphere.bo』を参照してください。

バインディング、データ・バインディング、データ・ハンドラー

147 ページの図 40 のように、SCA アプリケーション・モジュールが提供するサービスの呼び出しに使用されるビジネス・データは、ビジネス・オブジェクトに変換されます。これにより、SCA コンポーネントでビジネス・データを操作できるようになります。同様に、SCA コンポーネントによって操作されるビジネス・オブジェクトは、外部サービスで必要なデータ・フォーマットに変換されます。

サービスのエクスポートとインポートで使用されるバインディングにより、データ・フォーマットが自動的に正しい形式に変換される場合があります (Web サービス・バインディングなど)。または、非ネイティブの形式を DataObject インターフェースによって表されるビジネス・オブジェクトに変換するためのデータ・バインディングまたはデータ・ハンドラーを開発者側で準備できる場合もあります (JMS バインディングなど)。

データ・バインディングとデータ・ハンドラーの開発について詳しくは、 44 ページの『データ・ハンドラー』と 46 ページの『データ・バインディング』を参照してください。

コンポーネント

SCA コンポーネントは、Web サービス記述言語と XML スキーマを組み合わせて使用することにより、サービスの提供とコンシュームに関する規約を定義します。SCA がコンポーネント間で受け渡しをするビジネス・データは、DataObject インターフェースを使用してビジネス・オブジェクトとして表されます。SCA は、これらのビジネス・オブジェクトのタイプが、呼び出し対象のコンポーネントによって定義されたインターフェース規約と互換性があるかどうかを検証します。

ビジネス・オブジェクトを操作するためのプログラミング・モデル抽象化は、コンポーネントによって異なります。 POJO コンポーネントとメディエーション・フロー・コンポーネントの Custom プリミティブは、ビジネス・オブジェクト・プログラミングのインターフェースとサービスを直接使用して Java プログラミングを使用可能にすることにより、ビジネス・オブジェクトの直接操作を可能にしています。多くのコンポーネントで、ビジネス・オブジェクトを操作するための高水準の抽象化が用意されていますが、ビジネス・オブジェクトのインターフェースとサービスにおけるカスタムの動作を定義するための Java コードのスニペットも用意されています。

Interface Flow Mediation と Business Object Map コンポーネントの組み合わせ、またはメディエーション・フロー・コンポーネントとその XML マップ・プリミティブの組み合わせを使用して、ビジネス・オブジェクトを変換することができます。これらのビジネス・オブジェクト変換機能は、アプリケーション固有のビジネス・オブジェクトを汎用的なビジネス・オブジェクトに変換 (あるいはその逆) する場合に便利です。

特殊なビジネス・オブジェクト

サービス・メッセージ・オブジェクトとビジネス・グラフは、特定の用途に使用される 2 種類の専門化されたビジネス・オブジェクトです。

サービス・メッセージ・オブジェクト

サービス・メッセージ・オブジェクト (SMO) は、サービス呼び出しに関連するデータ・コレクションを表すためにメディエーション・フロー・コンポーネントによって使用される特殊なビジネス・オブジェクトです。

SMO には、固定された最上位階層があります。この階層は、ヘッダー、コンテキスト、本文、添付ファイル (存在する場合) から構成されています。

- ヘッダーは、特定のプロトコルまたはバインディングを使用して、サービス呼び出しに関連する情報を渡します。SOAP ヘッダーや JMS ヘッダーなどがあります。
- コンテキスト・データは、メディエーション・フロー・コンポーネントによる処理中に、呼び出しに関する追加の論理情報を渡します。通常、この情報は、クライアントによって送受信されるアプリケーション・データの一部ではありません。
- SMO の本文は、ペイロード・ビジネス・データを渡します。このデータは、標準的なビジネス・オブジェクトの形式で、コア・アプリケーション・メッセージまたは呼び出しデータを表現します。

SMO は、添付ファイルが設定された SOAP を使用して、Web サービス呼び出し用の添付データを渡すこともできます。

メディエーション・フローは、要求のルーティングやデータ変換などのタスクを実行します。SMO は、ヘッダーとペイロードの内容を 1 つの統一された構造で組み合わせて表現します。

ビジネス・グラフ

ビジネス・グラフは、統合のシナリオにおけるデータ同期をサポートする場合に使用される特殊なビジネス・オブジェクトです。

ここでは、特定の発注データを表示する 2 つのエンタープライズ情報システムを例として考えてみます。一方のシステムで発注データが変更されると、発注データを同期化するためのメッセージがもう一方のシステムに送信されます。ビジネス・グラフは、注文データの変更された部分だけを他方のシステムに送信し、変更のタイプを定義するための変更概要情報を注釈として発注データに追加するという概念をサポートしています。

この例では、発注データの 1 明細行項目が削除され、予定出荷日プロパティーが更新されていることが、発注ビジネス・グラフから他方のシステムに通知されます。

ビジネス・グラフは、Integration Designer の既存のビジネス・オブジェクトに簡単に追加することができます。ビジネス・グラフが最も頻繁に使用されるのは、WebSphere アダプターを使用していて、WebSphere InterChange Server アプリケーションのマイグレーションをサポートする場合のシナリオです。

ビジネス・オブジェクト解析モード

Integration Designer には、モジュールとライブラリーのプロパティーが用意されています。このプロパティーを使用すると、ビジネス・オブジェクト用の XML 解析モード (EAGER または LAZY) を構成することができます。

- このオプションを「*EAGER*」に設定すると、XML バイト・ストリームの解析処理が優先的に実行され、ビジネス・オブジェクトが作成されます。
- このオプションを「*LAZY*」に設定すると、ビジネス・オブジェクトは通常どおりに作成されますが、XML バイト・ストリームの実際の解析処理は、ビジネス・オブジェクト・プロパティーにアクセスしたときにのみ、部分的または遅延して実行されます。

どちらの XML 解析モードの場合も、非 XML データの解析処理は常に優先的に実行され、ビジネス・オブジェクトが作成されます。

ビジネス・オブジェクト構文解析モード選択時の考慮事項

ビジネス・オブジェクト構文解析モードは、実行時の XML データの解析方法を決定します。ビジネス・オブジェクト構文解析モードは、モジュールまたはライブラリーの作成時にそれらに対して定義されます。モジュールまたはライブラリーの解析モードは変更できますが、その影響を認識しておく必要があります。

ビジネス・オブジェクト・ランタイム・フレームワークのバージョンは、モジュールおよびライブラリーのレベルで設定されます。バージョン 7 よりも前のバージョンの IBM Integration Designer で作成されたモジュールは、イーガー解析モードで実行されます (変更の必要はありません)。デフォルトでは、IBM Integration Designer バージョン 7 以降のバージョンで作成されるモジュールおよびライブラリーは、いくつかの要因に応じて最適な解析モードが指定されます。この要因としては、ワークスペース内の既存プロジェクトの解析モード、同じソリューション内の従属プロジェクトやその他のプロジェクトの解析モードなどがあります。モジュールまたはライブラリーのビジネス・オブジェクト構文解析モードは実装に合うように変更できますが、以下の考慮事項を認識しておく必要があります。

考慮事項

- レイジー・ビジネス・オブジェクト構文解析モードでは、XML データはより高速に処理されます。ただし、イーガー・モードとレイジー・モードの間には、モジュールまたはライブラリーの構成を変更する場合は、互換性についての相違点があるため注意が必要です。これらの相違点は、モジュールの実行時動作に影響を及ぼします。どちらの解析モードがアプリケーションに最適であるかについては、『レイジー構文解析モードとイーガー解析モードの利点 (Benefits of using lazy versus eager parsing mode)』を参照してください。
- モジュールには、実行時の解析モードを 1 つだけ構成できます。ライブラリーでは、一方の解析モードをサポートするよう構成することも、両方の解析モードをサポートするよう構成することもできます。両方の解析モードをサポートするよう構成されたライブラリーは、イーガー解析モードを使用するモジュールと、レイジー解析モードを使用するモジュールの両方から参照される可能性があります。実行時のライブラリーの解析モードは、そのライブラリーを参照するモジュールによって決まります。実行時、モジュールは解析モードを宣言し、その解析モードはモジュールと、そのモジュールが使用するすべてのライブラリーによって使用されます。
- 異なる解析モードが構成されたモジュールおよびライブラリーは、以下の場合に互換性があります。
  - レイジー解析モードが構成されたモジュールおよびライブラリーは、レイジー解析モードのみ、またはイーガー解析モードとレイジー解析モードの両方を使用するライブラリーと互換性があります。
  - イーガー解析モードが構成されたモジュールおよびライブラリーは、イーガー解析モードのみ、またはイーガー解析モードとレイジー解析モードの両方を使用するライブラリーと互換性があります。
  - レイジー解析モードとイーガー解析モードが構成されたライブラリーは、レイジー解析モードとイーガー解析モードの両方を使用するライブラリーとのみ互換性があります。
- SCA バインディングを使用して通信する対話モジュールには、同じフレームワークを使用してください。異なる解析モードを使用するモジュール同士が通信すると、パフォーマンス上の問題が発生する可能性があります。

LAZY 解析モードと EAGER 解析モードを使用する場合のそれぞれの利点

XML 解析を行う場合、LAZY 解析モードが適しているアプリケーションもあれば、EAGER 解析モードによってパフォーマンスが向上するアプリケーションもあります。両方の解析モードでベンチマーク評価を行い、アプリケーションの特定の特性に最も適したモードを判断することをお勧めします。

このセクションでは、それぞれの解析モードに適したアプリケーションのタイプに関する一般的なガイダンスについて説明します。

- LAZY 解析モードが適したアプリケーション

  大きな XML データ・ストリームを解析するアプリケーションの場合、LAZY 解析モードを使用するとパフォーマンスが向上する可能性があります。XML バイト・ストリームのサイズが大きくなり、アプリケーションがアクセスするバイト・ストリームからのデータ量が小さくなるにつれて、パフォーマンス上の利点が大きくなります。

  注: ビジネス・オブジェクトの LAZY 解析モードは、WebSphere Process Server バージョン 7.0.0.3 以降、および IBM Process Server でサポートされています。メディエーション・フロー・コンポーネントを含むモジュールおよびメディエーション・モジュールはサポートされません。

- EAGER 解析モードが適したアプリケーション

  次に示すアプリケーションの場合、EAGER 解析モードを使用するとパフォーマンスが向上する可能性があります。

  – 非 XML データ・ストリームを解析するアプリケーション
  – BOFactory サービスを使用して作成されたアプリケーション
  – 非常に小さな XML メッセージを解析するアプリケーション

アプリケーションのマイグレーションと開発に関する考慮事項

元は EAGER 解析モードで開発されたアプリケーションを LAZY 解析モードを使用するように構成する場合、または 1 つのアプリケーションで LAZY 解析モードと EAGER 解析モードを切り替える場合は、2 つのモードの違いと、モードを切り替える場合の考慮事項について注意してください。

エラー処理

解析対象の XML バイト・ストリームの形式が正しくない場合、解析例外が発生します。

- EAGER モードで XML 解析を行う場合、インバウンド XML ストリームからビジネス・オブジェクトが解析されると同時に、この解析例外が発生します。
- LAZY モードによる XML 解析が構成されている場合、ビジネス・オブジェクト・プロパティーが読み込まれ、無効な形式の XML の一部が解析されたときに、解析例外が潜在的に発生します。

無効な形式の XML を処理する場合は、以下のいずれかの方法で処理を行ってください。

- エンタープライズ・サービス・バスをエッジにデプロイし、インバウンド XML を検証する。
- ビジネス・オブジェクトへのアクセスを行う箇所に LAZY エラー検出ロジックを記述する。

例外スタックと例外メッセージ

XML 解析における EAGER モードと LAZY モードはそれぞれ基礎となる実装が異なっているため、ビジネス・オブジェクト・プログラミングのインターフェースとサービスによってスローされるスタック・トレースには、すべて同じ例外クラス名が設定されます。ただし、同じ例外メッセージ、または実装固有の例外クラスの同じラップ・セットがこのスタック・トレースに記録されるとは限りません。

XML 直列化形式

XML 解析で LAZY モードを使用すると、XML を直列化する際に、変更されていない XML をインバウンドのバイト・ストリームからアウトバウンドのバイト・ストリームにコピーする場合のパフォーマンスを最適化することができます。この最適化によってパフォーマンスが向上しますが、ビジネス・オブジェクト

全体が LAZY 解析モードで更新されている場合や EAGER 解析モードで実行されていた場合は、アウトバウンドの XML バイト・ストリームが異なる形式で直列化されることがあります。

XML 直列化形式は、構文的に正確に同じではない場合がありますが、ビジネス・プロジェクトによって提供される意味値は、解析モードに関わらず常に等価になります。また、意味的に等価なアプリケーションの場合、異なる解析モードで実行されていても、アプリケーション間で安全に XML を渡すことができます。

ビジネス・オブジェクト・インスタンス・バリデーター

XML 解析で LAZY ビジネス・オブジェクト・モードのインスタンス・バリデーターを使用すると、特にプロパティー値のファセット検証において、ビジネス・オブジェクトの検証精度がさらに高くなります。この改善により、EAGER 解析モードでは検出されない追加の問題が LAZY 解析モードのインスタンス・バリデーターで検出され、より詳細なエラー・メッセージが出力されるようになります。

バージョン 602 の XML マップ

バージョン 6.1 より前の WebSphere Integration Developer で開発されたメディエーション・フローには、XML 解析の LAZY モードで直接実行できないマップまたはスタイル・シートを使用する XSLT プリミティブが含まれている場合があります。XML 解析の LAZY モードで使用するためにアプリケーションをマイグレーションすると、XSLT プリミティブに関連付けられたマップ・ファイルをマイグレーション・ウィザードで自動的に更新し、新しいモードで実行することができます。ただし、手動で編集されたスタイル・シートを XSLT プリミティブが直接参照している場合、このスタイル・シートはマイグレーションされず、XML 解析の LAZY モードで実行することはできません。

非公開のプライベート API

非公開でプライベート、かつ実装固有のビジネス・オブジェクト・プログラミング・インターフェースの機能を使用しているアプリケーションの場合、解析モードを切り替えるとコンパイルが失敗する可能性があります。EAGER 解析モードの場合、こうしたプライベート・インターフェースは、通常、Eclipse Modeling Framework (EMF) によって定義されるビジネス・オブジェクトの実装クラスです。

すべての場合において、プライベート API をアプリケーションから削除することを強くお勧めします。

サービス・メッセージ・オブジェクト EMF API

IBM Integration Designer のメディエーション・コンポーネントには、com.ibm.websphere.sibx.smobo パッケージに組み込まれている Java のクラスとインターフェースを使用してメッセージの内容を操作するための機能が用意されています。XML 解析の LAZY モードの場合、com.ibm.websphere.sibx.smobo パッケージの Java インターフェースを引き続き使用することができますが、Eclipse Modeling Framework (EMF) のクラスとインターフェースを直接参照しているメソッドや EMF インターフェースから継承されたメソッドは失敗する可能性があります。

ServiceMessageObject とその内容を、XML 解析の LAZY モードで EMF オブジェクトにキャストすることはできません。

BOMode サービス

BOMode サービスは、現在実行している XML 解析が EAGER モードと LAZY モードのどちらで実行されているかを確認する場合に使用します。

マイグレーション

バージョン 7.0.0.0 よりも前のアプリケーションについては、すべて EAGER モードで XML 解析が実行されます。このアプリケーションが BPM ランタイム・マイグレーション・ツールを使用してマイグレーションされたランタイムである場合も、引き続き EAGER モードで XML 解析が実行されます。

バージョン 7.0.0.0 よりも前のアプリケーションで XML 解析の LAZY モードを使用できるように構成するには、最初に Integration Designer を使用して、対象のアプリケーションの成果物をマイグレーションする必要があります。マイグレーションが終了したら、XML 解析の LAZY モードを使用するようにアプリケーションを構成します。

Integration Designer の成果物のマイグレーションについては、『ソース成果物のマイグレーション (Migrating source artifacts)』を参照してください。解析モードの設定については、『モジュールとライブラリーのビジネス・オブジェクト解析モードの構成 (Configuring the business object parsing mode of modules and libraries)』を参照してください。