WebOTX 提供機能ガイド

WebOTX 提供機能ガイド

バージョン: 7.1

版数: 第 8 版

リリース: 2010 年 9 月

目次

1.はじめに

WebOTX 提供機能ガイドは、WebOTX 上のアプリケーションをプログラミングする場合やWebOTXシステムの運用管理を行う場合に必要となる基本的な概念、WebOTX の提供する機能について説明します。

2.分散オブジェクト

WebOTX は、ネットワーク上の複数のコンピュータに配置されたソフトウェア部品を、業界で標準化された共通の呼び出し規約に従って連携動作させる、分散オブジェクト技術を備えたインフラストラクチャです。そのようなソフトウェア部品のことを、分散オブジェクトと呼びます。この分散オブジェクト技術を基盤とする WebOTX は、ネットワークを介してイントラネット、あるいはインターネット環境で分散しているアプリケーション・オブジェクトをクライアントーサーバ・モデルとしてサポートするために必要なサービス群を提供します。

オブジェクトを機能に応じて分割し、部品化することは、再利用性の向上が期待できます。これにより、ある機能の一部を修正してもアプリケーション全体を構築しなおさずに済むため、開発と保守の効率が上がります。さらに、部品化したソフトウェアを複数のコンピュータ上で組み合わせて実行することにより、資源の効率的な活用が可能になります。

WebOTX でサポートする分散オブジェクト技術は、規定の仕様に準拠しています。よってコンピュータの環境の相違やプログラミング言語の違い、ネットワーク上でのデータ/命令の送受信の仕組みなどを開発者から隠蔽することができます。規定の標準仕様に則ることで、開発者はそのような下層レベルの複雑な知識から解放され、そうした違いを気にすることなく、別の遠隔に位置するコンピュータで動作するオブジェクトを簡単に呼び出して利用することができます。

以上のような分散オブジェクトの技術分野において、WebOTX では OMG が策定した CORBA や、Sun が提唱する J2EE (Java RMI-IIOP) に基づき、広範な種類のサーバで構成したミドルウェアを提供しています。

2.1.Webサービス

一般的な Web サービスの概念と、それに対応した WebOTX の提供機能を紹介します。

2.1.1.Webサービスとは

Web サービスとは、広い意味で「Web の世界に存在するサービス」を指します。例えば、自動的にメールを返信する業務アプリケーションがあったとします。この業務アプリケーションには、配信先を登録したり、本文に相手の名前を挿入したりといったさまざまな機能が含まれているでしょう。このように、アプリケーションは 1 つ 1 つの機能をまとめることで動作するということができます。これらの機能を構成する要素 (コンポーネント)1 つ 1 つをサービスと言います。

これまでも、こうした様々な場所に点在するコンポーネントを連携させるコンポーネント技術は発達してきました。CORBA や COM などが代表的なコンポーネント技術ですが、複数コンポーネント技術が存在することで、コンポーネント技術間の連携が面倒なものとなっていました。もっと自由に、世界中、インターネット中に散らばっているいろいろなサービスを結び付けて、アプリケーションを作ることができたらいいのに、という考え方から生まれたのが「Web サービス」です。現在、Web サービスはコンポーネント技術の業界標準、よりオープンなビジネスソリューションを実現するための技術として捉えられ、広く普及しています。

Web サービスは、コンポーネント技術の標準を目指しているということで、データ交換の手段として最も標準的な言語仕様である「Extensible Markup Language (XML)」を採用しました。また、XML データをやり取りするための決まりとして「Simple Object Access Protocol (SOAP)」というプロトコルが、Web サービスのインタフェースを公開するための言語仕様として「Web Service Description Language (WSDL)」が、レジストリに登録されている Web サービスを検索するための技術とレジストリに Web サービスを登録するための技術である「Universal Description, Discover and Integration (UDDI)」が定義され、全てを XML のやり取りで解決するコンポーネント技術として確立されています。

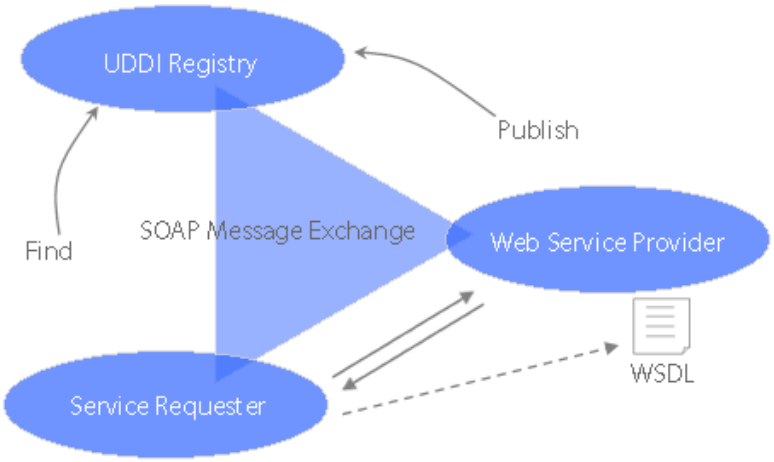


図.Web サービス技術の概念

2.1.2.Webサービスの基本仕様

Simple Object Access Protocol (SOAP)

SOAP は、XML 文書を交換するためのプロトコル仕様です。HTTP や SMTP などの既存のプロトコルに乗せて使う上位プロトコルです。WebOTX は SOAP 1.1、SOAP 1.2 仕様に対応した SOAP ランタイムを提供しています。

Web Services Description Language (WSDL)

WSDL は、Web サービスについての情報を XML で記述する方法を定めた仕様です。Web サービスの名前、ロケーション、インタフェースや Web サービスと通信する方法が記述されます。一般的に、Web サービスを作成したときに WSDL を作成します。Web サービスを公開する場合は、Web 上で WSDL を直接公開したり、UDDI に登録したり、あるいはその両方を行います。Web サービスのクライアントは、WSDL を元に作成します。WebOTX は、WSDL 1.1 仕様に対応しています。

Universal Description, Discovery and Integration (UDDI)

人間が目的のホームページを見つける時には「検索エンジン」を使います。同じように目的の Web サービスを検索するための仕様、それが UDDI です。UDDI は、Web サービスを登録するところとして「UDDI レジストリ」を定義し、UDDI レジストリを検索して Web サービスを探し出す方法を決めています。登録する情報は XML 形式で、検索に SOAP を使います。UDDI レジストリには、「UDDI ビジネスレジストリ」と「プライベートレジストリ」の 2 種類があり、前者はインターネット上に 1 つだけ存在するように見えるレジストリで、世界中の誰もが利用することができます。後者は企業内やポータルサイト内のようにある閉じられた空間の中に存在、または構築する UDDI レジストリです。

WebOTX の全てのエディションで UDDI 2.0/3.0 に対応した UDDI クライアントライブラリを提供します。UDDI クライアントライブラリを使うと、UDDI レジストリにアクセスして Web サービスを検索したり、Web サービスを登録したりすることができます。また、WebOTX UDDI Registry (オプション製品)では、UDDI 2.0/3.0 に対応した UDDI レジストリを提供します。

Web Services-Interoperability Organization Basic Profile (WS-I BP)

WS-I は Web サービスアプリケーションの相互接続性を高めていくことを目的として設立された団体で、そこで様々な Web サービス技術の仕様について議論されています。Basic Profile は SOAP、WSDL、UDDI の仕様のあいまいさを取り除き、相互接続性を高めるために Web サービスアプリケーションはどう作るべきかというガイドラインを定めている仕様です。WebOTX は WS-I BP 1.0 に対応し、WS-I BP 1.0 に沿った Web サービスアプリケーションの作成を支援します。

2.1.3.Webサービスの拡張仕様

Web Services Security (WS-Security)

WS-Security は、SOAP メッセージレベルにおけるセキュリティ機能を規定しています。トランスポートレイヤーでは SSL が利用できますので、用途に応じてどちらのどの機能を利用するかを検討する必要があります。

WebOTX では、OASIS Standard 仕様に準拠した WS-Security 機能を提供しています。WebOTX の WS-Security 対応機能を利用すると次のようなことができます。

- 認証
  ユーザネームトークンとデジタル署名による認証機能を提供します。ユーザ名とパスワードの照合は独自の仕組みによって実現していただくことができるほか、WebOTX の持つユーザ認証機構を利用することも可能です。また、ユーザ名やパスワードは暗号化することもできます。
- デジタル署名
  クライアントとサーバでやり取りされる双方向の SOAP メッセージに対して、それぞれの SOAP Body 要素、あるいはその中の特定の要素を対象としてデジタル署名をすることができます。これにより、メッセージの改ざん、なりすまし、否認を防止します。
- 暗号化
  クライアントとサーバでやり取りされる双方向の SOAP メッセージに対して、それぞれのオペレーション要素の内容(Content)、あるいはその中の特定のパラメータ要素の内容(Content)を暗号化することができます。これにより、メッセージの盗聴を防止します。
- タイムスタンプ
  クライアントとサーバでやり取りされる双方向の SOAP メッセージに対して、それぞれにタイムスタンプを付与することができます。これにより、リプレイ攻撃などを防止します。タイムスタンプへの署名や暗号化もすることができます。
- シングルサインオン
  SAML1.1 に対応した認証機構と連携して、SAML Assertion の付与、検証機能を提供します。これにより、Web サービスにおけるシングルサインオン環境の実現に貢献します。WebOTX は、「Sender-Vouches モデル」、「Holder-of-key モデル」におけるリクエスタとレシーバ間のアサーションの付与/検証、および署名の付与/検証を行います。

MEMO
SAML 1.1 に対応した認証機構製品として WebSAM SECUREMASTER V4.0 があります。

WebOTX が準拠している WS-Security 仕様は次のとおりです。

仕様名称	概要
Web Services Security: SOAP Message Security V1.0	SOAP メッセージへの XML 署名、XML 暗号の適用方法、認証情報(Token)の利用方法に関する仕様。
Web Services Security: Username Token Profile V1.0	ユーザ名、パスワードを用いた認証情報の利用方法に関する仕様。
Web Services Security: X.509 Token Profile V1.0	X.509 証明書を用いた認証情報の利用方法に関する仕様。
Web Services Security: SAML Token Profile V1.0	OASIS SAML 仕様に準拠したセキュリティトークンの利用方法に関する仕様。

Web Services-Interoperability Organization Basic Security Profile(WS-I Basic Profile)

WS-I Basic Security Profile は、Web サービスセキュリティの相互運用性を確保するガイドラインです。本プロファイルに現在も WS-I でプロファイル策定が続けられています。WebOTX の Web サービスセキュリティは Basic Securit Profile 1.0 Working Group Draft に対応しております。

今後のプロファイル策定の過程で、Working Group Draft 版から変更点が発生する可能性があるため、相互運用性の確保が必要となる Web サービスシステムを開発する場合は、正式バージョンへの配慮が重要となります。

2.2.J2EE

Java 2 Platform, Enterprise Edition (J2EE) は、エンタープライズ・アプリケーションの設計、開発、組み立て、配備の場面においてコンポーネント・ベースのアプローチを提供するアーキテクチャです。この J2EE アーキテクチャを満足するアプリケーションサーバは、再利用可能なコンポーネントのための多層分散アプリケーション・モデル、柔軟なトランザクション制御と一体化したセキュリティ・モデル、XML ベースのオープン・スタンダードとプロトコルの上に統合されたデータ交換技術による Web サービス支援を提供します。

J2EE テクノロジの進化は、これまでより迅速なマーケットへの革新的ビジネスソリューションを提案するだけでなく、特定のアプリケーションサーバ・ベンダの製品やプログラミング・インタフェースに縛られないプラットフォーム非依存の J2EE コンポーネント・ベース型ソリューションももたらします。

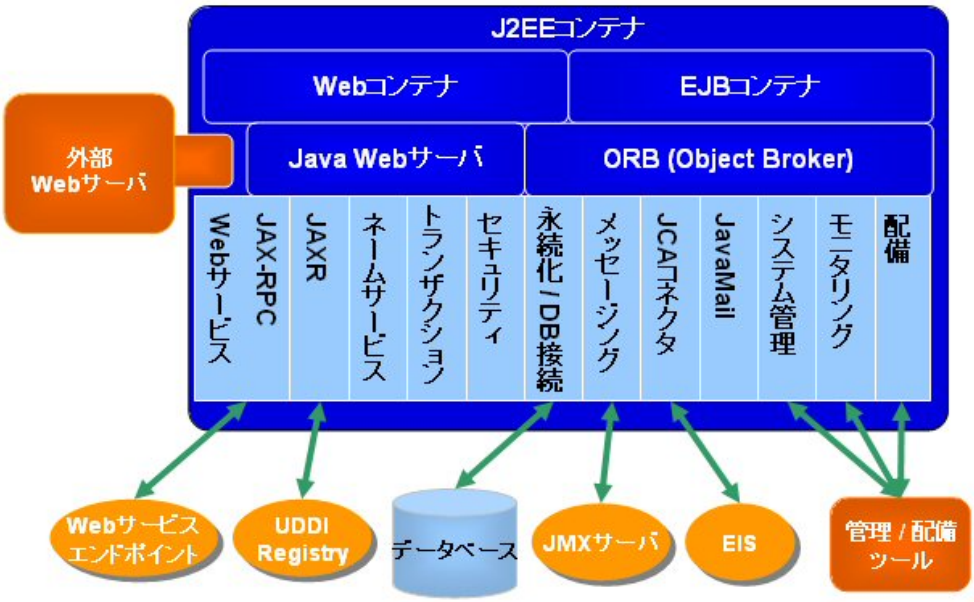
J2EE 仕様は Web と EJB を中心とするバージョン 1.2 から始まり、その次に非同期型アプリケーション・モデルの提供とベンダ間の相互接続性の解決に取り組んだバージョン 1.3 へと進化してきました。そして、今日では Web サービス技術の標準統合と運用管理の仕組みやインタフェースを一般化したバージョン 1.4 へと進んできました。

WebOTX と J2EE との関わりは、J2EE 技術あるいは J2EE という用語が生まれる前に遡ります。J2EE が誕生する前は、J2EE に含まれる各種仕様が単独で提案されていました。WebOTX は、当時の EJB 1.0 仕様から対応を始めています。そして、J2EE 1.2 が最初に提案されて以降、一貫して J2EE に準拠したアプリケーションサーバを製品化しています。

WebOTX は、特にミッションクリティカルな領域において、エンタープライズ・アプリケーションの配備、運用、実行シーンで堅牢な基盤を提供します。その基盤の中でも特徴的な優位点は、トランザクション制御と性能、実行時管理にあります。

2.2.1.J2EEアプリケーションサーバのアーキテクチャ

J2EE プラットフォームでは、アプリケーション動作のために多層の分散アプリケーション・モデルを提供します。アプリケーション・ロジックは、機能によってコンポーネントに分割されます。そして、1 つの J2EE アプリケーションを構成する各種コンポーネントが、そのコンポーネントの属す多層化 J2EE 環境内の特定の層へ応じたマシンにインストールされます。下の図は、WebOTX におけるアプリケーションサーバのアーキテクチャの概観を表しています。



- Web コンポーネントは、Web コンテナが位置する Web 層で動作します。
- EJB コンポーネントは、EJB コンテナが位置するビジネス層で動作します。
- データベースやレガシーシステム、UDDI などのレジストリサーバ内のソフトウェアは、エンタープラ

イズ情報システム層で動作します。

コンテナ

コンテナは、J2EE コンポーネントに対してトランザクション管理やセキュリティなどの下層サービスを提供する実行環境です。上の図では、Web コンテナと EJB コンテナの部分が相当します。2 つを合わせて J2EE コンテナと呼ぶ場合もあります。

JSP やサーブレットのような Web コンポーネントは、Web コンテナ内で動作します。WebOTX では、クライアントからの HTTP/S リクエストを受け付けるサーバとして、Web コンテナ内蔵の Web サーバを提供します。さらに外部の OS ネイティブな HTTP サーバと Web コンテナを連携させることも可能です。

EJB テクノロジに準じたコンポーネントである Enterprise Bean は、EJB コンテナ内で動作します。

Webサービス

J2EE 1.4 に準拠したアプリケーションサーバは、XML ベースの RPC のための Java API (JAX-RPC) で実装された Web サービスの実行エンジンが標準で利用できます。この JAX-RPC API で書かれたプログラムは、Web アプリケーションとしてコンテナに配備できます。さらに、コンテナ内で動作する J2EE アプリケーションやコンポーネントもまた、他の Web サービスへのクライアントとなることも可能です。アプリケーションは XML レジストリのための Java API (JAXR) を使ってコーディングすることにより、XML レジストリサーバへアクセスすることができます。WebOTX では XML レジストリサーバとして、UDDI 3.0 仕様に対応した UDDI Registry というサーバ製品を提供しています。また、WebOTX V7.1 では、JAX-RPC の後継仕様である JAX-WS も提供しています。JAX-WS では新しく XML/HTTP バインディングや非同期通信をサポートしています。

アプリケーションサービス

J2EE プラットフォームでは、コンテナがアプリケーションに各種サービスを提供するようにアーキテクチャが設計されています。

- ネームサービス
  ネームサービスは、JNDI を使用したオブジェクトを名前に結び付けるネーミングとディレクトリに関するサービスです。J2EE アプリケーションは、意図する JNDI 名をルックアップすることにより、JNDI サーバ内にあるオブジェクトの位置を特定できます。
  WebOTX の JNDI サーバは、CosNaming サービスを基盤に持ちます。CosNaming とは CORBA 仕様に準拠したネームサービスです。J2EE ではなく「2.3 CORBA」に基づく CORBA アプリケーションに対しては、CosNaming が提供されます。

- トランザクションサービス
  トランザクションは、それ以上分割できない、不可分な作業単位です。トランザクション管理を担当するサービス基盤は、各作業単位全体が他のプロセスからの干渉を受けずに完了するか、全体を完了できなければ、その単位で実行された処理全てが元に戻るように保証しています。
  J2EE アプリケーションに対しては、JTA (Java Transaction API) を提供しています。さらに J2EE プラットフォームには、アプリケーションにトランザクション制御を意識させなくする「コンテナ管理によるトランザクション」も提供されています。
  WebOTX におけるトランザクションサービスは、CORBA Transaction Service 仕様に準拠したサービスが下層で支えています。

- セキュリティサービス
  J2EE プラットフォームでのセキュリティサービスは、Web や Enterprise Bean のコンポーネントへそのアクセス権限を持つユーザによってのみアクセスできるように保証する設計が施されています。
  J2EE 1.4 仕様からは、さらにコンテナに対するセキュリティ規約が定義されました。これは、JACC (Java Authorization Contract for Containers) という仕様に基づきます。これにより、コンテナはクライアントの識別情報に基づきながら、コンテナの管理下にあるリソースやサービスに対してもアクセスを制限することができます。

外部システムへのアクセス

アプリケーションは、J2EE プラットフォーム内からサーバの外部に位置するシステムへアクセスすることができます。そのシステムの代表的な例がデータベースサーバです。アプリケーションは、リソースと呼ばれる接続オブジェクトを介して、それら外部システムへ接続します。そのリソースを構成する役割は、システム管理者が担当することになります。J2EE プラットフォームでは、次のような API とコンポーネントによって外部システムへアクセスする手段を提供しています。

- JDBC

データベース管理システムは、データの格納、構成、検索などを行うための機能を提供しています。ほとんどのビジネス・アプリケーションは、リレーショナルデータベースシステムを用いています。そのようなアプリケーションは、JDBC API 経由でデータベースサーバにアクセスします。
WebOTX は、JDBC 経由によるデータベース接続のために、データベースサーバの違いを抽象化する API として JDBC データソースを提供しています。JDBC データソースはその他にも豊富な機能を揃えています。詳細は「5.3 JDBC データソース」で説明されます。

- JMS
  JMS は非同期にメッセージを配信するための仕様です。J2EE では、バージョン 1.3 からアプリケーションサーバへ標準統合されました。JMS によるメッセージング・システムを利用することで、コンポーネントやアプリケーションの間でメッセージを伝達する手段を得られます。メッセージング・クライアントは、他のクライアントにメッセージを送信したり、他のクライアントからメッセージを受信したりすることが可能になります。
  WebOTX では、独自の JMS プロバイダを提供しています。詳細は「5.4 JMS」で説明されます。

- JCA
  JCA (J2EE Connector Architecture) は、ERP、CRM、他のレガシーシステムなどの様々な既存のエンタープライズ情報システム (EIS) と J2EE アプリケーションとの間の統合を可能にします。アプリケーションは、コネクタまたはリソースアダプタと呼ばれるポータブルな J2EE コンポーネントを介して EIS にアクセスできます。JCA は、そのようなシステムと対話するためのインフラストラクチャともいえます。
  WebOTX では、JMS サーバへの接続用リソースアダプタをバンドルしています。また、NEC のメインフレーム ACOS へ接続するための OLF/TP アダプタをオプショナルで製品化しています。

- JavaMail
  アプリケーションは、JavaMail API を使うことによって電子メールを送受信するために必要な SMTP サーバへの接続を可能にします。JavaMail API は、SMTP の他にも IMAP4、POP3 プロトコルによるアクセス手段も提供しています。

サーバ管理

J2EE 1.3 以前のプラットフォームでは、アプリケーションサーバの管理に関わる仕組みが標準化されていませんでした。そのため、各々のサーバベンダは、独自の実装方式によってサーバ管理を実現していました。

J2EE 1.4 では、それらの枠組みが規定されました。このことは、J2EE 1.4 にとっての大きな強化点です。アプリケーションサーバの管理仕様の一例として、サーバの実行時性能をモニタリングすることが可能になります。さらに、アプリケーションの配備についてもインタフェースが標準化されています。WebOTX には、アプリケーション配備のためのコマンドと GUI ツールを提供しています。

2.3.CORBA

WebOTX が提供する CORBA 実行環境である WebOTX Object Broker が提供する機能について説明します。

2.3.1.CORBAとは

はじめに、CORBA について説明します。

OMGとは

Object Management Group (OMG)は米国の民間の標準化団体です。会員の提案にもとづいた仕様策定を行っています。会員には、内外の主要なコンピュータメーカ、ソフトベンダやそれらのユーザ企業、他の標準化組織などが参加しています。

CORBAとは

Common Object Request Broker Architecture (CORBA)は OMG が規定した分散オブジェクトに関する標準仕様です。分散オブジェクトの開発時の仕様と実行時の仕様に分けられます。開発時の仕様には、記述するプログラミング言語に独立した Interface Definition Language (IDL)言語によるオブジェクトの外部仕様の表現方式と各種プログラミング言語からの利用方法を規定する言語マッピングがあります。実行時の仕様には、分散環境下に存在するオブジェクト間の通信プロトコルである General Inter-ORB Protocol/ Internet Inter-ORB Protocol (GIOP/ IIOP)などがあります。

CORBA を利用すると、オブジェクト指向言語である Java や C++では通信に関連するプログラミングを意識せずに、Java や C++のオブジェクトを呼び出す感覚で分散環境下での通信プログラムを記述することができます。引数や戻り値には整数や文字列などの基本型のほかに、構造体や配列などの複雑な型を利用できますので、通信電文の設計やバイトオーダ(メモリ上でのデータの格納順序)の変換などに時間を割く必要がありません。呼び出されるオブジェクトの記述言語と呼び出されるオブジェクトの記述言語は異なってもかまいません。クライアントを Java で記述し、サーバ側では Java はもちろんのこと、C++や COBOL などで記述することもできます。CORBA での通信は関数呼び出しと同様の動作となる要求応答型(リクエストレスポンス型)です。一方、ファイル転送や画像配信などのデータ送信方向が片道であるストリーム型通信には適していません。

IDLからのスタブ／スケルトンの生成

CORBA を利用したアプリケーションの開発は通常次の手順で行われます。

1. IDL 言語でオブジェクトの外部仕様であるインタフェースを記述する（記述物を IDL 定義と呼びます)。
2. IDL コンパイラで IDL 定義からスタブおよびスケルトンを生成します。
3. クライアントおよびサーバ(オブジェクト)の処理をそれぞれの開発言語で記述します。
4. クライアントではスタブを、サーバではスタブおよびスケルトンをプログラムと同時にコンパイルして使用します。

IDL コンパイラは IDL 定義ファイルから、クライアントおよびサーバで使用するスタブと呼ばれるプログラムと、サーバで使用するスケルトンというプログラムを生成します。クライアントではクライアントでの処理を記述したプログラムとスタブをコンパイルして使用します。また、サーバでは、サーバの処理を記述したプログラムとスタブおよびスケルトンをコンパイルして使用します。

実行時の通信概要

CORBA ではサーバオブジェクトの識別情報は Interoperable Object Reference (IOR)で表現されます。IOR 中にはサーバオブジェクトの IP アドレスあるいはホスト名と TCP ポート番号が含まれています。CORBA の実装内部ではこれらの情報を使ってサーバオブジェクトとの TCP コネクションを確立して通信を行います。主な通信は、処理要求であるリクエストメッセージと処理結果を返すレスポンスメッセージで行われます。

言語マッピングを使用するアプリケーションプログラムからは、関数呼び出しの延長として通信が行われますので、通信の詳細について意識する必要はありません。

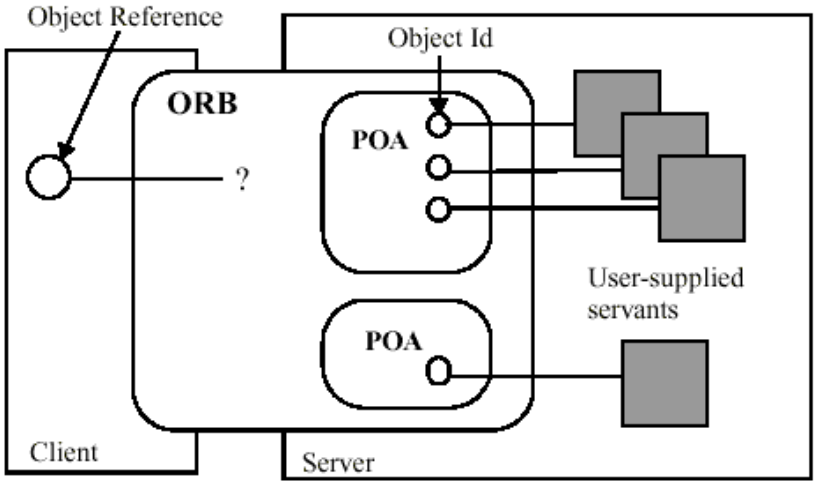
2.3.2.CORBA実行環境

Portable Object Adapter (POA)

Portable Object Adapter (POA)は、CORBA 2.2 で追加された標準的なオブジェクトアダプタであり、サーバアプリケーションのポータビリティを実現します。

POA の主な機能は、CORBA オブジェクトとサーバントのライフサイクルを管理し、クライアントからのオブジェクトに対する呼び出しを適切なサーバントにディスパッチすることです。サーバントとは、CORBA オブジェクトを実装した C++や Java などの言語上のオブジェクトのことです。

それまでの BOA と大きく異なる点は、複数の CORBA オブジェクトをひとつのサーバントに対応付けることができることです。例えば、データベースの個々のデータに対応する膨大な数の CORBA オブジェクトをひとつのサーバントで処理させることが可能になります。



- ThreadPolicy

要求をシングルスレッドで処理させるか、ORB に依存したマルチスレッドで処理させるかを設定します。

ORB_CTRL_MODEL:

ORB 依存のマルチスレッドで処理します。(既定値)

必ずスレッドセーフに処理を実装しなければなりません。

SINGLE_THREAD_MODEL:

シングルスレッドで処理します。

ORB_CTRL_MODEL の場合に WebOTX Object Broker が提供するマルチスレッドについて説明します。

C++版:次の 3 つのスレッド処理を選択できます。

1. スレッドを使用しません。(既定値)
2. 要求毎にスレッドを作成して処理します。
3. プールされたスレッドを使用して処理します。

選択は、プログラムで設定するか、レジストリやファイルまたは環境変数で行う方法があります。詳細については、アプリケーション開発ガイド、運用ガイドをごらんください。

Java 版:次の 2 つのスレッド処理を選択できます。

1. クライアント毎にスレッドを作成して処理します。(既定値)
2. プールされたスレッドを使用して処理します。

選択は、プログラムで設定するかプロパティやファイルで行う方法があります。詳細については、アプリケーション開発ガイド、運用ガイドをごらんください。

- LifespanPoilcy

CORBA オブジェクトを一時オブジェクトとして作成するか、永続オブジェクトとして作成するかを設定します。

TRANSIENT : 一時オブジェクト。サーバプロセスの生存期間内でのみ有効です。(既定値)

PERSISTENT: 永続オブジェクト。サーバプロセスの生存期間を超えて有効です。

PERSISTENT の場合、CORBA オブジェクトを作成して、その後サーバプロセスを停止・再起動しても、クライアントは同じリファレンスで引き続き同じオブジェクトにアクセスすることができます。

- IdAssignmentPolicy

CORBA オブジェクトのオブジェクト ID をアプリケーションで割り当てるか、ORB で自動的に割り当てるかを設定します。

SYSTEM_ID： ORB が自動的に割り当てます。(既定値)
USER_ID ： アプリケーションで割り当てます。

オブジェクト ID をデータベースアクセスのキーとして使用する場合など大量のオブジェクトに 1 つのサーバントを割り当てるときには、USER_ID を使用します。

- IdUniquenessPolicy

CORBA オブジェクトとサーバントの対応付けを 1 対 1 とするか、多対 1 とするかを設定します。

UNIQUE_ID ： 1 対 1 で対応付けます。(既定値)
MULTIPLE_ID： 多対 1 で対応付けます。

MULTIPLE_ID を設定する典型的なケースは、後述するデフォルトサーバントを使用する場合です。

- ServantRetentionPolicy

Active Object Map (AOM) を使用するか否かを設定します。AOM とは、CORBA オブジェクトとサーバントの対応付けを管理する対応表です。

RETAIN ： AOM を使用します。(既定値)
NON_RETAIN： AOM を使用しません。

NON_RETAIN を選択するには、後述するサーバントマネージャを作成して、アプリケーション自ら対応付けを行うか、デフォルトサーバントを使用することになります。このほかにも、RETAIN を選択してサーバントマネージャかデフォルトサーバントを実装することもできます。

- RequestProcessingPolicy

クライアントからの要求をどのサーバントにディスパッチするかを設定します。

USE_ACTIVE_OBJECT_MAP_ONLY： AOM だけを使用します。(既定値)
USE_DEFAULT_SERVANT ： デフォルトサーバントを使用します。
USE_SERVANT_MANAGER ： サーバントマネージャを使用します。

USE_DEFAULT_SERVANT でも USE_SERVANT_MANAGER の場合でも、ServantRetainPolicy に RETAIN が設定されているときには、まず AOM でサーバントを検索し、見つからない場合にだけディスパッチします。

- ImplicitActivationPolicy

暗黙的な活性化を行うか否かを設定します。

NO_IMPLICIT_ACTIVATION: 暗黙的な活性化を行いません。(既定値)

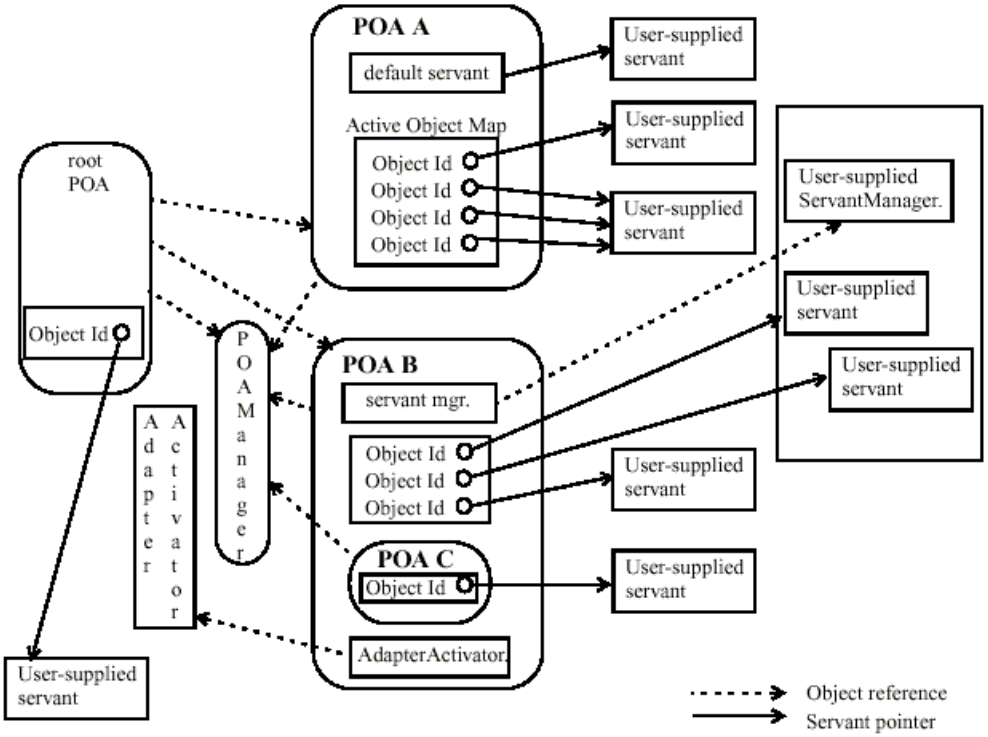
IMPLICIT_ACTIVATION : 暗黙的な活性化を行います。

NO_IMPLICIT_ACTIVATION を選択する場合、POA::activate_object() や POA::activate_object_with_id()により明示的に活性化しなければなりません。

IMPLICIT_ACTIVATION を選択する場合、まだ活性化されていないサーバントでも、POA::servant_to_reference()や POA::servant_to_id()の呼び出しのタイミングで暗黙的に活性化され、オブジェクトリファレンスやオブジェクト ID が返されます。

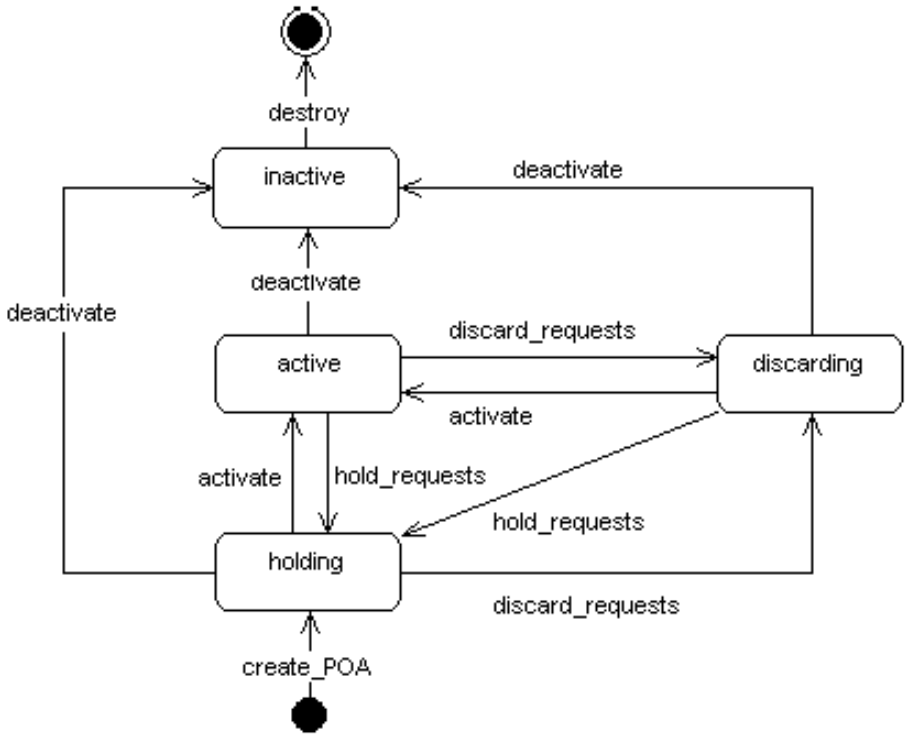
ORB はすべて既定値の POA ポリシをもつ rootPOA を提供しています。RootPOA は、ORB::resolve_initial_references("RootPOA")オペレーションで取得できます。

アプリケーションが独自の POA ポリシを持った POA を必要とする場合には、rootPOA を基点として階層的に自由に生成(POA::create_POA)することができます。



図中にあるPOA マネージャ、サーバントマネージャ、アダプタアクティベータ、そしてデフォルトサーバントについて簡単に説明します。

1. POA マネージャ
   POA マネージャは、POA に対するクライアントからの要求の流れを制御する管理オブジェクトです。4 つの状態があり、POA マネージャに対するオペレーション、
   ```
   OAManager::activate()
   OAManager::discard_requests()
   OAManager::hold_requests()
   OAManager::deactivate()
   ```
   により状態が遷移します。



HOLDING　… クライアントからの要求は POA にディスパッチせずにキューイングします。(初期値)
ACTIVE　… クライアントからの要求を POA にディスパッチします。
DISCARDING… クライアントからの要求は全て破棄しクライアントに TRANSIENT 例外を返却します。
INACTIVE　… クライアントからの要求は全て拒否します。

デフォルトの POA マネージャは rootPOA にあらかじめ関連付けられており、独自の POA を生成する際に POA::create_POA()の引数にその POA マネージャを引き継ぐか、引き継がずに新たな POA マネージャを関連づけるかを選択することができます。POA マネージャを分けることにより、いくつかの POA をグループ化して制御することができます。

2. サーバントマネージャ
サーバントマネージャとは、アプリケーション側で CORBA オブジェクトとサーバントの対応付けを行うもので、サーバントアクティベータとサーバントロケータの 2 種類があります。
サーバントアクティベータを使用するには、以下の 2 つのポリシが必須となります。

ServantRetentionPolicy : RETAIN
RequestProcessingPolicy: USE_SERVANT_MANAGER

クライアントから要求を受けると、まず AOM を検索し、見つからない場合にのみサーバントアクティベータを呼び出します。サーバントアクティベータが対応するサーバントを POA に返すと、AOM に登録されて以降の同じオブジェクトに対する要求のディスパッチは AOM の情報で解決されます。サーバントアクティベータは、ServantActivator インタフェースで定義されている incarnate()と etherealize()を実装しなければなりません。

一方、サーバントロケータを使用するには、以下の 2 つのポリシが必須となります。

ServantRetentionPolicy : NON_RETAIN
RequestProcessingPolicy: USE_SERVANT_MANAGER

AOM を使用しないため、クライアントからの要求ごとに呼び出され、サーバントを POA に返します。サーバントロケータは、ServantLocator インタフェースで定義されている preinvoke()と postinvoke()を実装しなければなりません。
サーバントアクティベータもサーバントロケータも、登録は POA::set_servant_manager()

で行います。

3. デフォルトサーバント
デフォルトサーバントとは、あるPOAのすべてのCORBAオブジェクトに対する要求をひとつのサーバントで処理する場合に使用します。この場合、以下の2つのポリシが必須となります。

IdUniquenessPolicy : MULTIPLE_ID
RequestProcessingPolicy: USE_DEFAULT_SERVANT

ServantRetentionPolicy を変えることによって、全ての要求をデフォルトサーバントで処理するか、AOM にない場合にだけデフォルトサーバントで処理するかを選択できます。デフォルトサーバントの実装の中で CORBA オブジェクトに依存した処理を行うためには、呼び出しの延長で`POACurrent`オブジェクトを`ORB::resolve_initial_references("POACurrent")`を使用して取得します。そして、`PortableServer::Current::get_object_id()`で対応するオブジェクト ID を取得します。デフォルトサーバントの登録は、`Portable::POA::set_servant()`で行います。

最後にCOBRAオブジェクトの数に応じたサーバタイプについて説明します。

・ CORBAオブジェクトの数が少ない場合
AOM だけを使用するポリシ設定が適しており、プログラミングも簡易にできます。オブジェクトの数が増えるとAOMの対応情報が増えるため、メモリ不足等の問題を引き起こすことがあります。また、永続オブジェクトを使用する場合には、起動時のオブジェクト活性化に要する時間を考慮する必要があります。

ServantRetentionPolicy : RETAIN
RequestProcessingPolicy: USE_ACTIVE_OBJECT_MAP_ONLY

・ CORBAオブジェクトの数が多い場合
オブジェクトの数が多く、実際にはそのうちの少数しか利用されないケースでは、全てのオブジェクトをあらかじめ活性化しておくよりも、必要時に活性化を行い AOM に登録するという、AOM とサーバントアクティベータ併用のパターンが適しています。

ServantRetentionPolicy : RETAIN
RequestProcessingPolicy: USE_SERVANT_MANAGER

ただし、このパターンでは、一度活性化されたオブジェクトはAOMに登録されるため、利用されるオブジェクトが多くなると最終的に AOM だけを使用するパターンと同様にメモリの問題が懸念されます。このような場合には、サーバントロケータを使用して、対応付けをアプリケーション側で完全に制御し、オブジェクトをキャッシュして管理するような仕組みを実装します。

デフォルトサーバントを使用すると全てのオブジェクトをひとつのサーバントで対応するためオブジェクトの数に依存せずにスケーラブルに対応できます。これは、データベースのキーをオブジェクトIDに対応づけるようなシステムに最適です。

RequestProcessingPolicy: USE_DEFAULT_SERVANNT

WebOTX Object Broker は、POA ポリシの組み合わせで実現できるいろいろなサンプルプログラムが用意しています。内容については、サンプル集をごらんください。

Dynamic Invocation Interface (DII)

動的起動インタフェース DII は、IDL コンパイラが生成するスタブ・スケルトンを使用せずに、実行時に動的に要求を組み立ててオブジェクトを呼び出す機能です。

CORBA 2.4 で追加された仕様についてはサポートしていません。

一般的には、次のようなケースで利用します。

- クライアントを開発する時点で、サーバが提供する interface が確定していない場合やインタフェースが頻繁に変更、追加されることが想定されている場合。

- 利用者にメニューからサービスを選択させて、サーバに処理を依頼するような汎用的なクライアントの場合。
- 一度に大量の要求を発行する必要のあるクライアントの場合。

DII では、クライアントは次のような手順でサーバ呼び出しを行います。

1. `Request` オブジェクト(`CORBA::Request`)を作成します。
2. `Request` オブジェクトに引数情報(`CORBA::NVList`)を設定します。
3. 要求を発行します。
4. `Request` オブジェクトから結果を取得します。

不特定のオブジェクトを扱うような場合には、インタフェース情報をインタフェースリポジトリに登録しておき、そこから取得した情報を `Request` オブジェクトの作成に使用します。

要求の発行には、4 つの方法があります。

- 同期オペレーション

  サーバに要求を送信して応答を待ち合わせます。
- oneway オペレーション

  サーバに要求を送信するだけで応答を待ち合わせません。
- 遅延同期オペレーション

  要求の送信と応答の受信を非同期に行います。
- 複数リクエストの発行

  oneway もしくは遅延同期オペレーションによる複数の要求を同時に送信します。

Request や NVList の生成や発行方法については、アプリケーション開発ガイドをご覧下さい。

Dynamic Skeleton Interface (DSI)

動的スケルトンインタフェース DSI は、IDL コンパイラが生成するスケルトンを使用せずにサーバ処理を実装し、クライアントからの要求を動的に処理する機能です。とくにインタフェースに依存しないサーバ処理を実装する場合に必要となります。

DSI では、サーバは次のような手順でクライアントからの要求処理を行います。

1. 言語マッピングで定められている `PortableServer::DynamicImplementation` クラスを継承してサーバ処理を `invoke()`に実装します。
2. クライアントからの要求の延長で `invoke()`が呼び出され、`ServerRequest` オブジェクト(`CORBA::ServerRequest`)を引数として渡されます。
3. `ServerRequest` オブジェクトからオペレーション名を取得します。
4. `ServerRequest` オブジェクトから引数情報(`CORBA::NVList`)を取得します。
5. オペレーション名に対応するサーバ処理を実行します。
6. `ServerRequest` オブジェクトに結果を設定します。
7. リターンします。

ServerRequest や NVList の生成や発行方法については、アプリケーション開発ガイドをご覧下さい。

DII の Request オブジェクトと DSI の ServerRequest オブジェクトは、共にオペレーションの引数情報を保持するために同じ NVList オブジェクトを使用しています。このため、クライアントからの要求を DSI で受け取り、その引数をそのまま DII でサーバに要求を発行するようなゲートウェイプログラムの実装にも利用することができます。後述する DynamicAny やインタフェースリポジトリを使用して、特定のインタフェースに依存しない処理を実装することもできます。

DynamicAny

DynamicAny は、実行時に IDL コンパイラが生成するコードを使用せずに動的に any 型を扱う機能です。CORBA 2.2 で最初に追加されましたが、CORBA 2.3 で大きく仕様が変更されています。WebOTX Object Broker Java(TM)は、CORBA 2.3 の仕様をサポートしています。

DynamicAny を使用することにより、次のことが可能になります。

- any に任意の型のデータを動的に構築する。
- any に格納された任意の型のデータを動的に走査し、構成要素を取得する。

例として、次のような IDL 型を動的に扱う場合について説明します。

```
// IDL
struct MyStruct {
    long member1;
    boolean member2;
}
```

このような構造体(MyStruct)を動的に生成して any に格納するには、次のような手順で行います。

1. ORB からファクトリ(DynAnyFactory)を取得します。

   ```
   ORB::resolve_initial_references("DynAnyFactory")
   ```

2. この構造体に対応する TypeCode を生成します。

   ```
   ORB::create_struct_tc()
   ```

3. DynAny を生成します。引数には 2 で生成した TypeCode を指定します。

   ```
   DynAnyFactory::create_dyn_any()
   ```

4. DynStruct に narrow します。
5. 構造体のメンバ値を設定します。

   ```
   DynStruct::insert_long()      // member1 の設定
   DynStruct::next()             // 次のメンバへの位置付け
   DynStruct::insert_boolean()   // member2 の設定
   ```

6. any を生成します。

   ```
   DynStruct::to_any()
   ```

逆に any に格納された MyStruct から member2 の値を取り出すには、次のような手順で行います。

1. ORB からファクトリ(DynAnyFactory)を取得します。

   ```
   ORB::resolve_initial_references("DynAnyFactory")
   ```

2. DynAny を生成します。引数には対象とする any を指定します。

   ```
   DynAnyFactory::create_dyn_any()
   ```

3. DynStruct に narrow します。

4. 構造体のメンバ値を取得します。

```
DynStruct::next()          // 次のメンバへの位置付け
DynStruct::get_boolean()  // member2 の取得
```

以上のように、IDL コンパイラが出力する支援コード(Java マッピングの場合は Helper や Holder を指す)を使用する必要がありません。

Interceptor (フック)

Interceptor は、CORBA 2.3 で追加されたクライアントと CORBA オブジェクトとの間の呼び出しルートにおいて割り込み処理を実現する機能です。リクエストレベルとメッセージレベルの割り込み処理を実装してリクエストの引数を変更したり、メッセージを変更したりすることができます。CORBA サービス(例えばセキュリティのアクセス制御)をプラグイン的に実装するような用途としても考えられています。ただし、この仕様にはあいまいな部分(例えば、native 型で宣言されながら各言語マッピングにはそれが定義されていない)が存在します。また、既に CORBA 2.5 で全く新しい仕様に変わることが分かっています。

WebOTX Object Broker の将来のバージョンではこの新しい仕様をサポートする予定ですが、従来から提供している WebOTX Object Broker 独自のフック機能が現バージョンでは利用できます。

フック機能は、Interceptor と同様のリクエストレベルとメッセージレベルに加えて、応答送信完了時に呼び出されるフックやクライアントとサーバ間のコネクション切断時に呼び出される、サーバ側のフックを登録できます。

詳細については、アプリケーション開発ガイドを参照してください。

Objects by Value

Objects by Value は、CORBA 2.3 で追加されたオブジェクトの値渡しを実現する機能です。

Value

Value は、interface と struct 定義の中間に位置し、オペレーションやメンバを定義することができます。valuetype という IDL キーワードを使用して定義します。

```
// IDL
valuetype Names {
    public Names next;
    private string name;
    private string kind;
    public Names oya;
    void print();
};
```

interface と異なるのは、interface の場合はそのオブジェクトリファレンスが渡される(参照渡し)のに対し、valuetype の場合にはそのコピーが渡される(値渡し)ことになります。そのため、オブジェクトに対するオペレーションはローカルな呼び出しとなり、メンバの値変更も、ローカルなオブジェクトに閉じた変更となります。

struct と異なるのは、継承・再帰定義が可能であることと、共有と null の概念をサポートしている点です。これにより、送信元の再帰的なリストや木構造をそのままのイメージで送信先に渡すことが可能になります。

一般的に、値渡しには通信コストがかかります。Value オブジェクトのメンバ情報をシリアライズして転送するためです。しかし、以降のオブジェクトへの呼び出しは全てローカルなものになるため、逆に通信コストはかかりません。その反面、値渡しされた Value オブジェクトに対してのアクセスとなるため、サーバ側での更新が反映されないことになります。Value オブジェクトは、このような特性をよく理解して利用することが必要です。

Value Box

Value Box は、メンバを一つだけ持つ Value のことです。

```
// IDL
valuetype Name string;
```

とくに有効な string と wstring については、標準で次の定義をサポートしています。

```
// IDL
module CORBA {
    valuetype StringValue string;
    valuetype WStringValue wstring;
}
```

あるオペレーションの引数に、メンバに string 型のデータをもつ struct が多数リンクされており、かつそのデータが全て同じ文字情報を持っているような場合、通常その個数分シリアライズされて送信され、送信先では異なる string 型のデータとして復元されてしまいます。しかし、string 型の代わりに Value Box である StringValue 型を使用すると同じ文字情報は最初のものだけがシリアライズされ、残りはすべてそこを参照する形でシリアライズされます。送信先においても、ただひとつの文字情報が復元されます。このように、共有の概念をプリミティブな型においても取り入れることが可能になります。

Abstract Interface

Abstract Interface は、あるオペレーションの引数にオブジェクトを指定する場合に、Value として渡すか、interface のオブジェクトリファレンスとして渡すかを、IDL 定義時ではなく、実際の呼び出しの時点で決定する仕組みを提供します。

abstract interface という IDL キーワードを使用して定義します。

```
// IDL
abstract interface Print {
    void print_op();
};
```

例えば、次のようなケースで利用することができます。

- サーバがクライアントに対し、何らかの情報を提供するオブジェクトを渡すような場合で、リアルタイムな情報を利用するクライアントに対しては参照渡しを行い、呼び出し時点で固定された情報で十分なクライアントに対しては値渡しをするような場合。
- セキュリティ上、信頼できないクライアントにはアクセス制御を通るように参照渡しを行い、信頼できるクライアントには値渡しをするような場合。

RMI over IIOP

RMI over IIOP は、CORBA 2.3 で追加された、Java 特有の機能です。

Java では分散オブジェクトプログラミングを可能にする Remote Method Invocation (RMI)が標準でサポートされていますが、CORBA で作成されたプログラムと通信することができません。RMI over IIOP は、RMI で作成されたプログラムのインタフェースを変更することなく、CORBA の標準プロトコルである IIOP を使って実行する機能です。これにより、RMI と CORBA の相互運用性を実現できます。

WebOTX Object Broker Java(TM)は、EJB の実行基盤としてサポートしています。

サーバプロセスの起動方法

WebOTX Object Broker での CORBA アプリケーションの起動方法には、クライアントの呼び出しによりサーバプロセスが自動的に起動される「自動起動」とクライアントの呼び出しを実行するまえにあらかじめサーバプロセスを立ち上げておく「手動起動」とがあります。自動起動は次の oadj (Java) および oad (C++)により行われます。

オブジェクト活性化デーモン (oad)

oad は呼び出しが発生した時点で該当するサーバプロセスが立ち上がっていない(応答が極端に低下した場合も含みます)ときに、サーバプロセスを自動的に立ち上げます。このとき、立ち上げを許可するファイル名を exeallow ファイルにあらかじめ記述しておく必要があります。このほか、特定のファイルの実行を不可能にする設定ファイルや、特定ホストからの要求のみ自動起動を可能にする設定などがあります。詳しくは、運用ガイドをごらんください。

Java用オブジェクト活性化デーモン (oadj)

oadj による自動起動を行うためには、instimpl コマンドでサーバプロセスを起動するためのコマンドイメージを登録しておく必要があります。自動起動を行う場合のオブジェクトリファレンスは、POA の LifespanPolicy を PERSISTENT に設定して作成します。

詳しくは、アプリケーション開発ガイド、運用ガイドをごらんください。

2.3.3.プロトコル、変換

GIOP/IIOP

WebOTX Object Broker は異なる ORB 間の接続のための CORBA 標準プロトコル General Inter-ORB Protocol/ Internet Inter-ORB Protocol (GIOP/ IIOP)を内部プロトコルとしても使用しています。このため、マルチベンダのシステムでも通信性能を損なうことがありません。現在 GIOP/ IIOP には、1.0、1.1、1.2 の 3 つのマイナーバージョンがありますが、GIOP/ IIOP はバージョン混在を前提に設計されていますので、これらのバージョンのいずれかをサポートしていれば WebOTX Object Broker と通信することができます。ただし、バージョン混在時は低いバージョンで通信されますので、一部機能が制限されることがあります。

GIOP には、バイトオーダ(整数型、浮動小数点型などのメモリ上の格納順序)を変換する機能がありますが、これについては、CORBA を使用することにより自動的に処理され、アプリケーションはこのことについて意識せずにプログラムを記述することができます。

コードセット

文字コードの変換をする機能です。`string` 型および `wstring` 型に適用されます。クライアントで SJIS、サーバで EUC など異なる文字コードを使用しているときに、変換を CORBA に任せることができます。文字コードの指定方法は運用ガイドまたはアプリケーション開発ガイド(プログラム開発者がプログラム中で指定する場合)をごらんください。

2.3.4.CORBAバージョンと追加機能およびWebOTX Object Brokerでの実装状況

CORBA の各バージョンの主な追加機能と WebOTX Object Broker での対応状況は次の表のとおりです。

CORBA バージョン	機能	WebOTX Object Broker	
		Java	C++
CORBA 2.0	GIOP/ IIOP 1.0	○	○
	インタフェースリポジトリ	○	○
	DII	○	○

	CDR	○	○
CORBA 2.2	GIOP/ IIOP 1.1	○	○
	POA	○	○
	コードセット	○	○
	DSI	○	○
	DynamicAny	○	×
	longlong 型	○	○
	long double 型	—	○
	fixed 型	○	○
CORBA 2.3	GIOP/ IIOP 1.2	○	○
	Objects By Value	○	○
	RMI over IIOP	○	—
CORBA 2.5	Portable Interceptors	○	○
CORBA 2.6	IIOP over SSL	○	○
	CSIv2	○	○

3.Webサーバ

アプリケーションサーバでの Web 層では、World Wide Web 上で活用されるコンテンツやアプリケーションのビジネスロジックを生成します。この章では、WebOTX の機能階層の中で、特に Web 層に含まれる、HTTP サーバと Web コンテナが提供する機能について説明します。

3.1.HTTPサーバ

WebOTX が対象とするクライアントは、World Wide Web のブラウザ(以下 Web ブラウザと呼びます)の Thin クライアント、Web ブラウザ内で動作する Applet クライアント、Web ブラウザと独立して動作する Rich クライアントがあります。Rich クライアントでもクライアントアプリケーションの配送管理を容易にするため、WebOTX (Standard/Enterprise Edition) は、ダウンローダを提供しています。

Web ブラウザは HTTP サーバと Hyper Text Transfer Protocol (HTTP) や Secure Sockets Layer (SSL) で暗号化された HTTPS プロトコルを使って通信を行います。HTTP プロトコルや HTTPS プロトコルでは、Uniform Resource Locators (URL) を使って、世界で一意のサーバ上の資源にアクセスします。URL は次のフォーマットを持っています。

プロトコル識別://ホストアドレス[:ポート]/ホスト内資源アドレス

プロトコル識別は HTTP プロトコルなら「http」、HTTPS プロトコルなら「https」を指定します。

ホストアドレスは IP アドレスまたはホスト名を指定します。ポートはプロトコルごとのポートで既定値として http は 80、https は 443 と決まっています。ホスト内資源アドレスはホスト内で資源を特定するために使用されるもので一般的にはファイルのディレクトリパスを指定します。

ポートを分けることで、1 台のサーバマシン上で本番用、テスト用のように複数の Web サーバを起動することができます。また、既定値以外のポートを使うことで一般の人からは発見しにくい Web サーバを運用することもできます。逆に Application Service Provider (ASP) や Internet Service Provider (ISP) のように 1 台のサーバマシンで複数のホストアドレスを提供したいという考え方があります。数千、数万の企業に対してデータセンタで Web サーバを提供するとしても、サーバ台数をそれだけ用意するのは困難です。このような場合、仮想 IP アドレスを使って、1 つの Web サーバが複数のホストサーバの振りをして動作します。

負荷分散装置を使った場合には別の概念の仮想 IP アドレスが出現します。HTTP サーバの可用性を高めるためにも、スケーラビリティを高めるためにも HTTP サーバを複数台用意して負荷分散を行い、ブラウザからは 1 つのサーバのように見せることは良く行われている技術です。この場合、各 HTTP サーバマシンの実際の IP アドレス(実 IP)に対して、負荷分散装置が外に向けて見せる IP アドレスを仮想 IP アドレスと呼びます。

WebOTX では HTTP サーバとして、Web コンテナに内蔵された Java ベースの HTTP サーバと、Apache Software Foundation の Apache HTTP Server 1.3.37/2.0.59 をバンドルしています。これらはインストール時の指定により選択することができます。(インストール時のデフォルトは Java ベースの HTTP サーバです。)

これらの HTTP サーバは SSL 2.0 および 3.0 をサポートしており、インターネットを使った安全な通信を提供します。

他にも次に示す HTTP サーバを利用することができますので、システム要件に合わせて Web サーバを選択することができます。

HTTP サーバ	Windows	HP-UX	Solaris	Linux
Microsoft Internet Information Server (IIS)	5.0、6.0	—	—	—
WebOTX Web Server (右は、Apache のバージョン)	1.3.37 以上 2.0.59 以上	1.3.37 以上 2.0.59 以上	1.3.37 以上 2.0.59 以上	1.3.37 以上 2.0.59 以上
Sun Java System Web Server	6.1	6.1	6.1	6.1
Sun ONE Web Server	6.0 以上	6.0 以上	6.0 以上	6.0 以上

3.2.Webコンテナ

Web コンテナは、Web ブラウザと通信を行いながら業務を行う Web アプリケーション Servlet/ JSP を動作させるための実行基盤です。Web アプリケーションは、Web コンテナの提供する各種機能を使って、クライアントからの入力データを解析し動的な HTML を生成するサーバサイドの Java アプリケーションです。

3.2.1.セッション管理

HTTP/HTTPS プロトコルでは Web ブラウザからの要求とサーバからの応答が一組で動作し、明確なセッションという概念がありません。一方、業務システムではクライアントとの複数回の通信の中で状態を変更して業務を遂行していくスタイルが一般的です。そのため HTTP/HTTPS プロトコル上で論理的なセッションを管理する必要性が出てきました。

HTTP/HTTPS プロトコルのセッション管理は、Cookie、URL Rewriting、Hidden TAG を使うのが一般的です。これらは論理セッションであり、解放プロトコルが存在しません。上位でログオフを作る、またはタイムアウトでセッションの消滅を行うことになります。ログオフ操作は永久に行われないかもしれませんので、業務要件に合わせて、タイムアウト時間を設定し、資源の解放漏れがおこらないようにする必要があります。

Web コンテナでは、Web コンテナとしてのセション管理に Cookie を利用するか、URL Rewriting を利用するかを選択できます。この選択は、Webコンテナの「運用管理コンソール」でおこないます。Web サーバは、Web コンテナの設定した Cookie もしくは URL の値を Web ブラウザに引き継ぎ、Web ブラウザから送信されたデータとともに Cookie もしくは URL の値を Web コンテナに引き渡します。これらの値は Web コンテナ上で動作する Web アプリケーションから参照することもできます。また、Web アプリケーションで、独自の ID を設定することもできます。

Web クライアントが携帯電話などの場合、Cookie を処理することができません。この場合には、URL Rewriting を利用するか、Hidden TAG を利用することになります。

3.2.2.URL

Servlet/JSP の特定は URL により行われます。業務システムでは静的なメニューページやログイン Servlet から順番にターゲット業務を行う Servlet へと遷移していくのが一般的です。WebOTX では URL 単位の閉塞・閉塞解除機能を提供しています。Servlet/ JSP を動的に置換すると後続の Servlet/ JSP との間で矛盾を起こすことがありえますが、この機能を使えば処理中のトランザクションがなくなった時点で、安全に Web アプリケーションの置換を行うことができます。

3.2.3.Webサーバのクラスタ化

負荷分散装置（サーバスイッチ等）を用いて、複数の Web サーバをクラスタ化し、Web アプリケーションで買い物かごのようにセション情報を管理するシステムに対応します。Web アプリケーションへの特別なコーディングなしで、Web サーバのクラスタ化に対応ができます。

この場合、Web アプリケーションでセションオブジェクトに保存する情報は、シリアライズ可能なオブジェクトになります。また、Web アプリケーションが Web サーバのクラスタ化に対応していることを示すために、配備記述子（web.xml）に<distributable>要素の指定をします。

3.2.4.認証APIの利用

通常、BASIC 認証や FORM 認証では、クライアントからユーザ名とパスワードを受け取り、認証処理を行います。 しかし、場合によっては、Servlet/JSP のプログラムで認証をするほうが便利なことがあります。

Servlet/JSP のプログラムでユーザ名とパスワードを指定し、認証 API を呼び出すことにより、BASIC 認証および FORM 認証を行うことができます。

3.2.5.Webコンテナの機能構造

Web コンテナは、WebOTX エディションによって 2 つの異なる機能構造となっています。1 つは、Web Edition / Standard-J Edition で提供する 1 つの Java 仮想マシン上で動作する Web コンテナです。もう 1 つは、Standard / Enterprise Edition で提供する WebOTX アプリケーションサーバへのプラグインとしての構造です。

Web Edition / Standard-J EditionのWebコンテナ

このエディションでは、Standard Edition / Enterprise Edition よりも軽量なアーキテクチャになっており、1つのドメインで1つの Web コンテナプロセス（Java VM プロセス）が動作します。EJB を利用しない Web アプリケーションを動作させるのであれば、Standard Edition / Enterprise Edition よりも高速に動作します。また、メモリ消費量も削減されます。その反面、システム障害に対する多重化のような高度な機能を持ちません。

Standard Edition / Enterprise EditionのEJBコンテナ

このエディションでは、信頼性重視型として高度な運用環境と連携するモードを選択できます。堅牢な実行環境の上層に位置することにより、大規模システムにも対応、拡張でき、システム障害に対する多重化が可能です。Web アプリケーションから EJB を利用する形態の J2EE アプリケーションであれば、Web コンテナと EJB コンテナが同一の JavaVM 上で動作するので、高速に動作することができます。

4.APサーバ

アプリケーションサーバでのビジネス層では、アプリケーションの種別に応じた固有のビジネスロジックに対応し、トランザクション管理、スレッド/プロセス管理、アクセス制御管理などのシステムレベルのサービスを提供します。WebOTX では、Java ベースの EJB コンテナから Java、C/C++言語で書かれた CORBA コンポーネントを扱うハイエンドな動作環境までを提供しています。CORBA コンポーネントは、WebOTX Standard / Enterprise Edition で利用できます。EJB コンテナは、それらのエディションに加えて、Standard-J Edition から利用可能です。

この層に位置する EJB コンテナは、EIS リソースを使用して Web 層のコンポーネントからやスタンドアロン型アプリケーション・クライアントからの要求にサービスする Enterprise Bean コンポーネントをホストします。この章では、WebOTX のコンポーネント動作サービス機能階層の中で、特に EJB コンテナが提供する機能について説明します。

4.1.EJBコンテナ

J2EE プラットフォーム（Java 2 Platform, Enterprise Edition）では、多層エンタープライズ・アプリケーションをサポートする技術が指定されています。これらの技術は大きく別けて、コンポーネント、サービス、通信の 3 つの分野に分かれています。

EJB コンテナには、Enterprise Bean コンポーネントのトランザクション管理やライフサイクル管理、セキュリティ管理などをサポートする機能があります。さらに、JDBC データソースを介したデータベースサーバへのアクセス、JCA リソースアダプタを介したエンタープライズ情報システムへのアクセスなどを提供します。

4.1.1.Enterprise Beanコンポーネント

EJB アーキテクチャは、エンタープライズ・アプリケーションのビジネスロジックを含むコンポーネントを開発し EJB コンテナに配備するためのサーバサイド技術です。このコンポーネントは、拡張性に富み、トランザクション型で、複数ユーザに対するセキュリティが適用されます。これらに適用するサービス提供元が EJB コンテナにあたります。

Enterprise Bean には、Session Bean と Entity Bean、メッセージドリブン Bean の 3 種類があります。Session Bean と Entity Bean には、ホームインタフェースとコンポーネントインタフェースがあります。コンポーネントインタフェースは、EJB 1.1 以前の仕様ではリモートインタフェースと呼ばれていました。ホームインタフェースは、Bean の生成、検索、削除などのメソッドを定義します。コンポーネントインタフェースは、Bean のビジネスロジックを定義します。メッセージドリブン Bean には、2 つのインタフェースはありません。

Session Bean は、クライアントに代わってサービスを提供するために作成され、通常は単一のクライアントーサーバ・セッションの期間中にのみ生存します。Session Bean はトランザクション型にすることができますが、EJB コンテナがダウンすると回復不能です。

Session Bean には、ステートレスのものと、複数のメソッドやトランザクション間で会話状態を維持できるステートフルのタイプがあります。状態維持のためのオブジェクト管理は、EJB コンテナが行いますが、オブジェクト自体の永続データは、Session Bean 自身が行う必要があります。

Entity Bean は、データストア内で維持されるデータを表した永続化オブジェクトです。Entity Bean は自身で永続性を管理するか、その機能を EJB コンテナに委譲するかが選択できます。

Entity Bean は、プライマリキーで識別されます。たとえ EJB コンテナがダウンしても Entity Bean オブジェクトや、そのプライマリキー、リモート・リファレンスは残ります。

EJB 2.0 から追加されたメッセージドリブン Bean によって、クライアントが非同期でビジネスロジックにアクセスできようになりました。メッセージドリブン Bean は、待機している JMS のキューやトピックから受信した非同期メッセージによって初めてアクティブになります。クライアントは、直接メッセージドリブン Bean にアクセスせず、その代わりに、メッセージを JMS サーバにキューやトピックとして送信します。

EJB コンテナには、Enterprise Bean の生成や、他のアプリケーション・コンポーネントが Enterprise Bean にアクセスできるようにネーミング・サービスへ Bean をバインドしたり、許可されたクライアントだけが Enterprise Bean にアクセスすることを保証したり、Bean の状態を永続化領域に保存したり、Bean インスタンスをキャッシュしたり、必要に応じて Bean インスタンスを活性化、非活性化する責任があります。

4.1.2.EJBコンテナのサービス

EJB 2.1 コンテナは、全てのコンポーネントに J2SE 1.4 API を提供します。それに加えて、J2EE 1.4 API も使用できます。次に示す項目は、WebOTX の EJB コンテナで利用できる J2EE 1.4 の API 実装です。

- EJB (Enterprise JavaBeans) 2.1
- JMS (Java Message Service) 1.1
- JTA (Java Transaction API) 1.0.1B
- JavaMail 1.3.1
- JAF (JavaBeans Activation Framework) 1.0.2
- JAXP (Java API for XML Processing) 1.2.5
- JCA (J2EE Connector Architecture) 1.5
- WSEE (Web Services for J2EE) 1.1
- JAX-RPC (Java API for XML-based RPC) 1.1
- SAAJ (SOAP Attachments API for Java) 1.2
- JAXR (Java API for XML Registries) 1.0
- J2EE Management 1.0
- JMX (Java Management Extensions) 1.2
- JACC (Java Authorization Service Provider Contract for Containers) 1.0

これらの API サービスの他、アプリケーション・プログラミングが簡素化でき、アプリケーション間のカスタマイズを可能にさせるサービスも用意されています。

ネーミング・サービス

ネーミング・サービスは、アプリケーション・クライアントや Enterprise Bean、Web コンポーネントに JNDI ネーミング環境へのアクセスを提供します。このネーミング環境では、コンポーネントのソースコードを変更することなくコンポーネントをカスタマイズすることができます。EJB コンテナは、コンポーネントの実行環境を実装し、これをコンポーネントに JNDI ネーミング・コンテキストとして提供します。

コンポーネントでは、JTA UserTransaction オブジェクトのようなシステム提供オブジェクトと、EJB 参照、環境エントリ、JDBC データソースやメール送信などのユーザ定義オブジェクトの名前を指定してネーミング・サービスにアクセスできます。EJB コンテナは、そのアクセスに備えて名前サーバに定義オブジェクトをバインドしておく役割を持ちます。

ユーザ定義オブジェクトは、「java:comp/env」という名前空間の先に格納されます。この下には、JNDI サービスによってアプリケーション毎にサブコンテキストが割り当てられます。これにより、たとえ異なるアプリケーションで同一の登録名が与えられていても、それらが実際の登録時に衝突することなくアプリケーション間の独立性が維持できます。

JNDI サービスに関する詳細機能は、「5.2 JNDI」を参照してください。

配備サービス

配備サービスでは、コンポーネントをカスタマイズしてパッケージ化された J2EE アプリケーションを配備ツールから受け取り、パッケージ内の配備情報を解析した後、その情報に従って EJB コンテナ上で Enterprise Bean コンポーネントを実行可能状態にするものです。

コンポーネントのカスタマイズや、パッケージ化、コンテナ配備に関する詳細機能は、「8. アプリケーション配備」を参照してください。

トランザクションサービス

アプリケーションはトランザクションにより、それ以上分割できない一連の作業ユニットに分割されます。トランザクションをサポートするシステムでは、各ユニット全体は他のプロセスからの干渉を受けずに完了することが保証されます。ユニットは、全体を完了できる場合はコミットされます。全体を完了できなければ、そのユニットで実行された処理全体がシステムによって元に戻されます(ロールバック)。

Enterprise Bean では、Bean 管理とコンテナ管理という 2 つのトランザクション境界設定方式が用意されています。Bean 管理による場合は、Bean プログラム自身が全てのトランザクション管理工程をコード化しなければなりません。一方、コンテナ管理による場合は、EJB コンテナが配備時のトランザクション属性に基づいて管理します。コンテナが管理する範囲は、トランザクションの開始と終了処理だけでなく、トランザクショナルなオブジェクトの寿命全体を通じてトランザクションコンテキストを維持します。これにより、分散環境におけるトランザクションの場合に、コンポーネント供給者の責任分担とタスクが大幅に簡素化されます。

トランザクション属性は、Enterprise Bean がコンテナ管理によるトランザクション境界設定を使う Bean のメソッドに関連付けられた値です。Bean のトランザクション処理過程を最適化するために、メソッドごとに別の属性を与えることができます。

- Required
  クライアントがトランザクション中であるとき、そのトランザクションでメソッドを実行します。
  クライアントがトランザクション中でないとき、新たにトランザクションを開始してメソッドを実行し、メソッドの実行が終了したら、開始したトランザクションをコミットします。

- RequiresNew
  クライアントがトランザクション中であるとき、コンテナはメソッドを実行する前に、クライアントのトランザクションを中断させ、新たにトランザクションを開始してメソッドを実行します。メソッドの実行が終了したら、開始したトランザクションをコミットして、中断しているクライアントのトランザクションを復帰させます。
  クライアントがトランザクション中でないとき、新たにトランザクションを開始してメソッドを実行し、メソッドの実行が終了したら、開始したトランザクションをコミットします（Required と同様）。

- Mandatory
  クライアントがトランザクション中であるとき、そのトランザクションでメソッドを実行します(Required と同様)。
  クライアントがトランザクション中でないとき、コンテナはクライアントに対し、`TransactionRequiredException` 例外を送出します。

- NotSupported
  クライアントがトランザクション中であるとき、コンテナはメソッドを実行する前に、クライアントのトランザクションを中断させ、トランザクションなしでメソッドを実行します。メソッドの実行が終了したら、中断しているクライアントのトランザクションを復帰させます。クライアントがトランザクション中でないとき、そのままトランザクションなしでメソッドが実行されます。

- Supports
  クライアントがトランザクション中であるとき、そのトランザクションでメソッドを実行します（Required と同様）。
  クライアントがトランザクション中でないとき、そのままトランザクションなしでメソッドが実行されます(NotSupported と同様)。

- Never
  クライアントがトランザクション中であるとき、コンテナは `RemoteException` 例外を送出します。
  クライアントがトランザクション中でないとき、そのままトランザクションなしでメソッドが実行されます(NotSupported と同様)。

トランザクションサービス自身に関する詳細機能は、「5.5 JTA」と「5.6.6 トランザクションサービス」を参照してください。

セキュリティサービス

J2EE プラットフォームのセキュリティサービスは、リソースがその使用権限を持つユーザにのみアクセスされることを保証するように設計されています。アクセス制御は、認証と承認のステップに分かれています。

EJB コンテナには、宣言とプログラムによるものの 2 つのセキュリティ方式が用意されています。宣言によるセキュリティは、コンポーネント内に含まれる配備記述子という実行時属性によって指定されます。プログラムによるセキュリティは、プログラム内でアクセスされるセキュリティ機構を指します。EJB 仕様では、その標準 API を用意しています。

アクセス制御のステップのうちの認証は、通常、Web コンテナ側の HTTP 基本認証やフォームベース認証などを使用します。認証対象は、ユーザ名などのプリンシパルとなります。EJB コンテナを認証する方法はありませんが、Web から EJB にアクセスする場合、Web コンテナで認証されたプリシパルが EJB コンテナに伝播されます。EJB コンテナは、渡されたプリンシパルが呼び出されたメソッドを起動可能であるかを検査します。承認は、セキュリティ・ロールの概念に基づいています。ロールとは運用者が定義する論理的なユーザ・グループです。ロールは、配備記述子内のプリシパルにマッピングされます。Enterprise Bean の各メソッドに対しては、呼出し元が属するロールを関係付けることが可能です。

4.1.3. 通信技術

クライアントーサーバ間や、各種サーバに収容されている共用オブジェクト間の通信機構として OMG (Object Management Group) プロトコルをサポートしています。このプロトコルを使用すると、EJB コンテナに収容されたオブジェクトと、CORBA 技術を使用して

開発されたリモート・オブジェクトが相互にアクセスできます。その主要な通信サービスが RMI-IIOP です。

RMI-IIOP は、IIOP を介した RMI API の実装です。RMI-IIOP を使用すると、アプリケーションが Java プログラミング言語でリモート・オブジェクトにアクセスできます。コンテナ間の IIOP リモート呼び出しでは、セキュリティを必要とする場合があります。J2EE 仕様では、RMI-IIOP を介して行われる安全な呼び出しのために、CORBA 仕様の一部である「CSIv2 (Common Secure Interoperability)」プロトコルをサポートします。CSIv2 では、メッセージの一貫性や機密性の保護や、SSL/TLS などの利用した認証に対応しています。

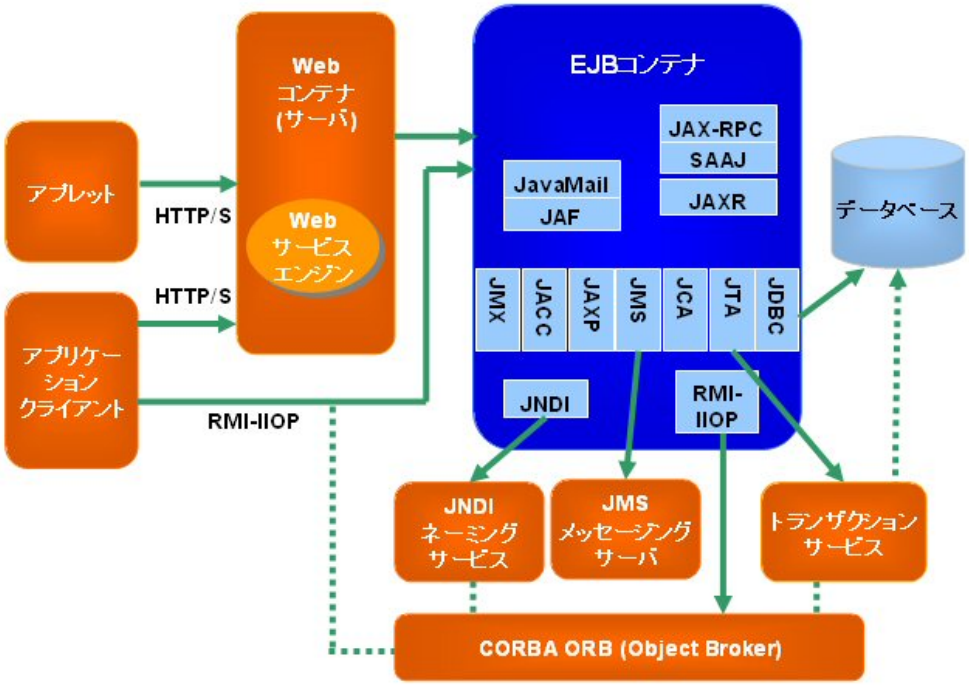
これらの通信技術は、WebOTX Object Broker という名前の CORBA 製品を用いています。Object Broker は、業界最高レベルの性能を発揮します。

4.1.4.WebOTX EJBコンテナの機能構造

EJB コンテナは、WebOTX エディションによって 2 つの異なる機能構造となっています。1 つは、Standard-J で提供する 1 つの Java 仮想マシン上で動作する単独 EJB サーバです。もう 1 つは、Standard/Enterprise で提供する WebOTX アプリケーションサーバへのプラグインとしての構造です。

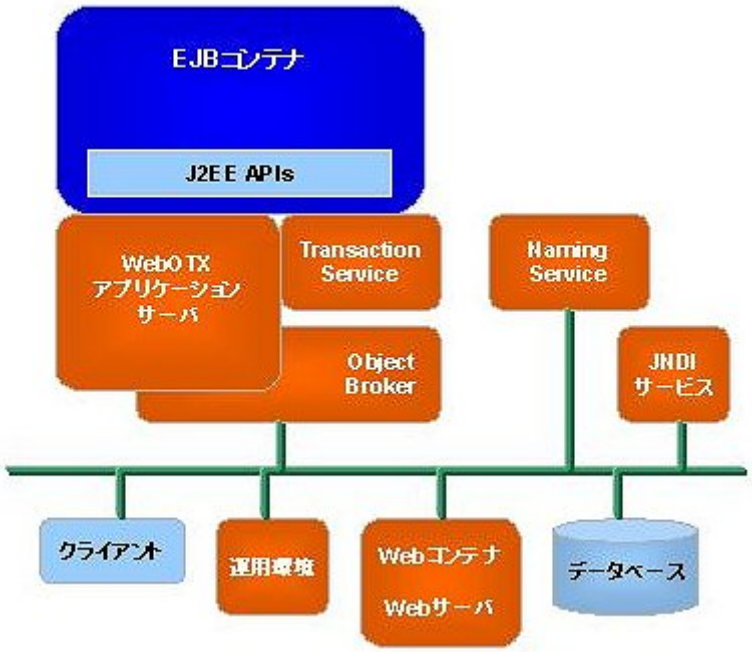
Standard-J EditionのEJBコンテナ

このエディションでは、性能重視型として Standard/Enterprise Edition よりも軽量なアーキテクチャになっています。また、メモリ消費量も削減されます。サポートする OS は、Windows 2000/2003、HP-UX 11/11i、Red Hat Enterprise Linux ES/AS、Solaris 8/9 と幅広く利用できます。その反面、システム障害に対する多重化のような高度な機能を持ちません。また、実行状態のモニタリングは、Standard / Enterprise Edition よりも対象項目が少ないものとなっています。以下に、Standard-J Edition 全体における EJB コンテナのシステムを図示します。



Standard/Enterprise EditionのEJBコンテナ

このエディションでは、信頼性重視型として高度な運用環境と連携します。さらに、堅牢な実行環境の上層に位置することにより、大規模システムにも対応、拡張できます。以下に、Standard/Enterprise Edition における EJB コンテナの関係を図示します。



実行時の特徴として、マルチプロセス動作が可能になります。

Standard/Enterprise Edition では、1 つの EJB コンポーネントを複数プロセス上で動作させることができます。このとき、クライアントからアクセスされるインスタンスの種類によってオブジェクトとプロセスのマッピングが異なります。

- ホームインタフェースとステートレス Session Bean に対するアクセスは、その EJB が動作している任意のプロセスにマッピングされます。つまりメソッド呼出しごとに処理を行うプロセスが変わります。これにより、メソッド呼び出しが複数プロセスに分散されます。
- ステートフル Session Bean と Entity Bean の場合に対するアクセスは、インスタンスが固有の情報をもつため、特定のプロセスにマッピングされます。つまりメソッド呼出しごとに処理を行うプロセスが変わることはありません。異なるプロセス上で実行させるにはホームインタフェースからコンポーネントインタフェースの参照を create メソッドやファインダーメソッドで再取得する必要があります。

4.1.5.EJBコンテナの付加価値機能

WebOTX EJB コンテナは EJB 2.1 仕様の機能を網羅しています。さらに付加価値機能として、下記の機能を提供しています。

- Bean インスタンスプールとキャッシュ
- コンカレンシー制御
- トランザクションコミットオプション

Beanインスタンスプールとキャッシュ

プールとキャッシュの機能

プールとは、複数の Bean インスタンスをコンテナが制御し、複数のクライアントからのリクエストを効率的に処理するために Bean インスタンスが配置されるメモリ領域を指します。プールにはクライアントからのリクエストに対してサービスを提供するために、全てのリクエストごとに Bean をインスタンス化し、放棄するオーバーヘッドを軽減させる目的があります。

一方、キャッシュとは、クライアントに迅速なサービスを提供するために特定の ID を保持した Bean インスタンスが配置されるメモリ領域を指します。クライアントのリクエストにより生成された Bean インスタンスはキャッシュに常駐し、アクセスを高速化させる目的があります。

EJBの種類とプール、キャッシュの関係

プールとキャッシュは EJB の種類によって、その Bean の性質から使用の可否が決まっています。

	ステートレス Session Bean	ステートフル Session Bean	Entity Bean	メッセージドリブン Bean
プール	○	－	○	○
キャッシュ	－	○	○	－

ステートレス Session Bean は会話情報を保持しないため、プールを用いてインスタンススワッピングを行い少ないメモリ消費で複数のクライアントから利用されます。キャッシュは状態を保持する Bean インスタンスが置かれる領域であるため、情報をもたないステートレス Session Bean では利用されません。

ステートフル Session Bean は特定のクライアントに継続してサービスを提供するため、キャッシュを利用します。インスタンスは特定のクライアントのためにその都度、生成・削除されるので、プールは利用されません。

Entity Bean はホームのファインダーメソッドを呼び出す際に、その処理をするオブジェクトとしてプールに存在するインスタンスを一時的に利用します。その後、クライアントから利用される時点で特定の ID が付与されてキャッシュにインスタンスが移動します。よって、プールとキャッシュの両方が利用されます。

メッセージドリブン Bean は JMS のデスティネーションから配信されるメッセージを `onMessage()` メソッドで受け取り処理するのにプールのみを利用します。

コンカレンシー制御（CMP Entity Beanのみ）

複数トランザクションが稼動する環境でリソースに関するロックを取得する方法はコンカレンシー制御と呼ばれます。コンカレンシー制御には、ペシミスティック・ロック(悲観的排他)とオプティミスティック・ロック(楽観的排他)の 2 種類があります。WebOTX では、この 2 つの制御方法をサポートしています。

ロックの違い

- ペシミスティック・ロック

ペシミスティック・ロックは、「自分が操作している情報は他の人も操作する可能性がある」という悲観的な視点に立っており、トランザクション開始時に排他ロックを獲得し、それをトランザクション終了時まで保持する排他方法です。つまり、ロックを獲得したアプリケーションのみがその行にアクセスすることができ、その行を更新したい他のアプリケーションは待たされることになります。

- オプティミスティック・ロック

オプティミスティック・ロックは、「自分が操作している情報は他の人が操作する可能性が低い」という楽観的な視点に立っており、更新時にリソースに変更がないことを確認し、変更がなければ排他ロックを獲得してリソースを更新し、変更があればアプリケーションにエラーを返す排他方法です。データベースから値を読み取るだけであれば他のアプリケーションに影響を与えないため、ロック待ちを解消できるという利点があります。

トランザクションコミットオプション（CMP Entity Beanのみ）

EJB 2.1 仕様ではトランザクションコミットオプションを 3 つ定義しています。これらは、コミットオプション A・コミットオプション B・コミットオプション C と呼ばれています。WebOTX では、コミットオプション B と C をサポートしています。

コミットオプションの違い

- コミットオプション B：

Bean インスタンスはキャッシュに保持され、プライマリキーは関連付けられていますが永続化フィールド値は関連付けされておらず、データベースとの同期が取られていない状態です。クライアントがビジネスメソッドを呼び出したとき、EJB コンテナは対応する Bean がキャッシュに存在するかどうかプライマリキーを用いて確認を行います。キャッシュに残存していればそのキャッシュされた Bean をそのまま再利用し、永続化フィールド値とデータベースの同期を取るためにキャッシュされた Bean の `ejbLoad()` が呼び出されます。もしキャッシュに残存していなければ、コミットオプション C 同様に Bean インスタンスをプールから推移し、プライマリキーの関連付けから行われます。

- コミットオプション C：

キャッシュ領域を利用しないため、Bean インスタンスは常にプールで待機しています。そのため、クライアントからビジネスメソッドが呼び出された場合は、まず ejbActivate()が呼び出され Bean インスタンスにプライマリキーが関連付けられます。その後、ejbLoad()により Bean インスタンスにプライマリキー以外の永続化フィールドとデータベースとの同期が取られます。また、クライアントからの呼び出しが終わると Bean に関連付けられたプライマリキーが解放され、Bean インスタンスはプールへ戻されます。

4.1.6.運用操作の概略

WebOTX アプリケーションサーバは、サーバ側アプリケーションのマルチプロセスをサポートする他、強力な実行時動作属性を指定できます。 この指定は、WebOTX 運用環境に含まれる運用管理ツールを操作して与えます。 Standard-J Edition の EJB コンテナは、運用管理ツールに相当する GUI ツールはありませんが、豊富なオプションを持つコマンドを提供しています。実行時動作属性は、サーバ内に XML ファイルとして格納されます。 一方、Standard/Enterprise の EJB コンテナは、その指定項目の多くが運用管理ツールからも定義できます。

このような運用情報は、アプリケーションサーバー般のシステム・パラメータや属性値を対象にしていますが、さらに EJB コンポーネントには、コンポーネント単位で実行時動作属性を持つことができます。この定義は、配備記述子と呼ばれ、XML 形式としてコンテナ内に登録するファイル中に納められています。配備記述子は、EJB 仕様で定められる標準のものと、WebOTX 固有の定義の 2 種類があります。

配備記述子は、特定の実行環境に配備する方法を表しています。「配備する」とは、実行環境にコンポーネントを登録、起動し、クライアントからのアクセスが可能な状態にさせることを言います。それとは反対に、実行環境からコンポーネントの状態を停止させ、削除することを「配備解除する」と言います。配備記述子は通常、配備ツールによって自動的に生成されます。配備記述子の要素には、コンポーネントのプログラム・コードに直接含まれていないコンポーネント動作に関する情報が定義されています。配備記述子要素の詳細については、「運用編」で説明されます。

EJBコンポーネント配備の流れ

EJB コンポーネントを EJB コンテナに配備するためには、運用管理コマンドを利用する方法の他に、配備ツールや統合運用管理ツールの GUI ツールを用いる方法があります。ここでは配備ツールを使った場合を例に、配備の流れを説明します。

まず、配備ツールでコンポーネントを読み込み、ツールの画面から必要に応じて配備記述子をカスタマイズします。次に配備先ホスト名を指定して配備実行します。

この時、配備ツールは WebOTX EJB コンテナ固有の情報も暗黙に生成します。そして、クライアントーサーバ間通信で必要となる Enterprise Bean のスタブとスケルトンを生成します。これらを 1 つの Java アーカイブファイルに再編成して、EJB コンテナに転送します。配備先のホストはローカル、リモートを問いません。

Standard/Enterprise Edition では、サーバアプリケーションの管理体系が、「TP システム」、「アプリケーショングループ」、「プロセスグループ」に階層化されています。配備ツールは、TP システム内のアプリケーショングループかプロセスグループを配備先に指定できます。プロセスグループは、アプリケーションに名付けられた「表示名」にマッピングされ、配備時に配備ツールがそのグループを自動生成します。

さらに、Standard/Enterprise Edition では、WebOTX 運用管理ツールを使って配備された EJB コンポーネントをシステム固有な詳細かつ最適な状態に設定を施すことができます。

このように配備されたコンポーネントは、EJB コンテナによって情報解析が行われ、起動初期化で環境エントリやリソース参照などを JNDI 名前空間に登録します。

データベースの利用

EJB コンテナは、いくつかの目的のため、データベース・システムを必要とします。1 つは、コンテナが永続化を管理する Entity Bean を扱うためです。2 つめは、Enterprise Bean に対して JDBC データソースのコネクションファクトリを提供するためです。最後に、コンテナ管理によるタイマーサービスを行うためです。データベース・システムを EJB コンテナが利用するにあたり、運用者は WebOTX のシステム構成情報の一部としてデータベース・システム固有の定義を設定する必要があります。データベース・システムは、Oracle、SequeLink、Cloudscape などをサポートします。

これらの設定方法は、運用管理コマンドか運用管理ツールを使って行います。運用者がサーバに格納された定義情報ファイルを直接編集してもかまいませんが、その場合は、ドメインを手動で再起動する必要があります。設定項目の詳細は、「運用編」で説明されます。

Entity Beanが永続化のために利用するデータベース

Entity Bean インスタンスとバックエンドの永続的な記憶領域間でエンティティの状態を転送するためのプロトコルのことを、オブジェクトによる永続性と言います。WebOTX EJB コンテナは、その記憶領域としてデータベース・システムを用います。

Bean 管理による永続性(略して、BMP)では、データベースアクセスのための呼出しを Bean クラスまたは、そのヘルパー・クラスで直接コード化します。

コンテナ管理による永続性(略して、CMP)では、Bean はデータベースに格納するフィールドを識別するだけで、永続化実装コードは

記述しません。代わりに、配備ツールがデータベースアクセスのための呼出しコードを生成します。これにより、永続化管理がコンテナに委譲されます。

CMP の利点としては、Entity Bean のコードが単純化できることが挙げられます。その反面、複数の結合を表したような複雑な状態オブジェクトを管理できない場合もあります。また、CMP によって生成されるデータベースへのアクセス・コードは、非常に汎用化されたものであるため、データベース・システムに特化した最適コードと比較して高速なアクセス性能が期待できません。このような場合、BMP を使用することを勧めます。

4.1.7.新機能 ー タイマーサービス

ビジネスのワークフローをモデル化するアプリケーションは、しばしば特定の時刻や一定の時間間隔に合わせたイベントに依存することがあります。タイマーサービスは、Enterprise Bean によって使用される通知やイベントのスケジュール化のために提供される、EJB コンテナでの永続的でトランザクショナルな通知サービスです。ステートフル Session Beans 以外の全ての Enterprise Bean タイプがタイマーサービスから通知を受け取ることができます。

コンテナは、Enterprise Bean に「時間/時刻」をトリガーにコールバックを行います。コールバックを受けてビジネスロジックを実行するタイミングは、次の中から指定できます。

- 特定の時刻　　（例：23 時 30 分）
- 一定時間の経過後　（例：12 時間後）
- 一定の時間間隔　　（例：8 分間隔）

これらの組み合わせも可能です。

タイマーは永続性が保証されているため、サーバがクラッシュしてもシャットダウンしてもタイマー情報は消滅せずに保持され、再起動後に継続してタイマーが作動します。これは、タイマーを永続化するためにデータベースへ情報を格納しているからです。

タイマーサービスは、Enterprise Bean プログラムにとって、より高度な処理制御の管理基盤となえます。タイマーサービスが基盤として提供されることにより、例えば、監査処理や夜間バッチ処理などに有効利用できます。

4.2.TPモニタ

WebOTX　Standard Edition および Enterprise Edition はシステムを安定稼動させるために、メインフレームで培われた OLTP 技術をアプリケーションサーバ実行基盤(TP モニタ)に取り込んでいます。これにより、サーバのコンピュータ資源(CPU、メモリ)を効率良く利用することができ、一時的なソフトウェア障害に対してもサービス全体を停止することなく安定してサービスを継続できるようにしています。

以下に WebOTX の提供する TP モニタの特徴について説明します。

4.2.1.アプリケーション管理

WebOTX では OLTP 技術を取り込んだ TP モニタによりアプリケーションサーバ上で動作するアプリケーションが管理されます。アプリケーションは以下の単位でグルーピングされ管理されます。TP モニタはこれら構成単位毎に起動、停止などの運用制御や状態監視、障害リカバリを行います。

TPシステム

TP システムは複数の業務アプリケーションを管理します。 ドメイン単位で 1 つの TP システムが構成されます。マルチドメイン構成にすることにより WebOTX では 1 つのマシン上に複数の TP システムを起動させることが可能です。これにより、クラスタ運用を行った場合、ダウンしたサーバのシステムをそのまま稼働中のバックアップサーバに引き継ぐことができます。 1 つのマシン上に作成できる TP システムの最大数は、プラットフォームによって異なります。以下にその最大値を示します。

OS	最大システム数
Windows	20
HP-UX	105
Solaris	105
Linux	105

アプリケーショングループ

アプリケーショングループとは、開始・終了などの運用を共にするアプリケーション群をグルーピングしたものです。例えば、「受発注業務」「在庫数管理業務」といったような業務毎にツリーを分けて管理するといった方法を用います。こうすることで、「受発注業務」は 10 時から 17 時まで動作させ、その後は停止させる。「在庫数管理業務」は 24 時間動作させるなどのアプリケーショングループ単位での独立した運用が行えるようになります。

プロセスグループ

プロセスグループとは、ある特定のサービスを提供するプロセス群です。同一のプロセスグループに登録されているコンポーネントは同一のプロセス上でロードされます。 WebOTX はビジネスロジックを記述した 1 つまたは複数のコンポーネントをプロセスグループに登録し、プロセスとして実行します。プロセスグループ単位に実行多重度(プロセス数、スレッド数) などを設定することができます。

モジュール(コンポーネント)

モジュール(コンポーネント)とは、アプリケーションのモジュールファイルを指します。C++で作られた CORBA アプリケーションの場合は `dll` ファイルや `sl` ファイルなどのダイナミックリンクライブラリ、Java の場合は `jar` などの Java アーカイブファイルになります。J2EE アプリケーションの場合は J2EE アプリケーションとしてアーカイブされた `ear` ファイルを指します。アプリケーションの配備はこのモジュール単位に行ないます。

オブジェクト(インタフェース)

オブジェクトとは、CORBA や EJB のインタフェースを実装した実行オブジェクトです。CORBA アプリケーションの場合は IDL で記述したインタフェース単位にオブジェクトが生成されます。EJB の場合は各 Bean 単位でオブジェクトが生成されます。

オペレーション

オペレーションとは、CORBA アプリケーションの場合、IDL ファイルで定義した operation であり、C++や Java アプリケーションではクライアントから参照できる public クラスメンバ関数になります。EJB の場合、Bean クラスで実装した public メソッドになります。オペレーションは、クライアントから呼び出し可能なサービスであり、サーバアプリケーションのオブジェクトにて実装を行います。クライアントからのオペレーションコールにより、その実装を持っているサーバアプリケーションのオペレーションが実行されます。

共有コンポーネント

複数のコンポーネント間で共有するライブラリは共有コンポーネントとして登録することができます。共有コンポーネントとして登録されたライブラリや CORBA インタフェースファイル(if ファイル)は全てのコンポーネントから参照可能となります。また、ライブラリパスやクラスパスなどの設定は WebOTX で自動的に行なうので参照を行なうための環境変数の定義は必要ありません。複数のコンポーネントから参照が行なわれるため、共有コンポーネントを置換する場合は、利用している全てのコンポーネントが停止している必要があります。以下に共有コンポーネントとして登録できるファイル一覧を示します。

- モジュールファイル

  Java の場合は jar としてアーカイブされたモジュール（EJB からも参照可能）

  C++の場合は共有ライブラリとして作成されたモジュール（「`.dll`」、「`.sl`」、「`.so`」）

- CORBA インタフェースファイル
  複数のコンポーネントが共通して利用する CORBA インタフェースファイル（`.if`）
- CORBA 常駐オブジェクト
  常駐オブジェクトとして作成されたモジュールファイル

常駐オブジェクト

1 つのプロセスグループの中に複数のコンポーネントが登録されている場合、オペレーションの呼び出しタイミングなどのコールバックルーチンを定義した特別なインタフェースをもつライブラリを常駐オブジェクトとして登録することができます。常駐オブジェクトは、プロセスのスレッド初期化時にスレッドに対して 1 つ作成されるため、各コンポーネントで共通で利用するリソースなどの初期化処理を行うことが可能です。WebOTX ではサーバオブジェクトとは別に事前に生成することができるオブジェクトを用意しました。それが常駐オブジェクトです。常駐オブジェクトは、WebOTX の提供する API を用いて全てのモジュールからオブジェクトを参照することが可能です。常駐オブジェクトが実装可能なコールバックルーチンは次のとおりです。

- オペレーション開始時
- オペレーション正常終了時
- オペレーション異常終了時
- 例外発生時

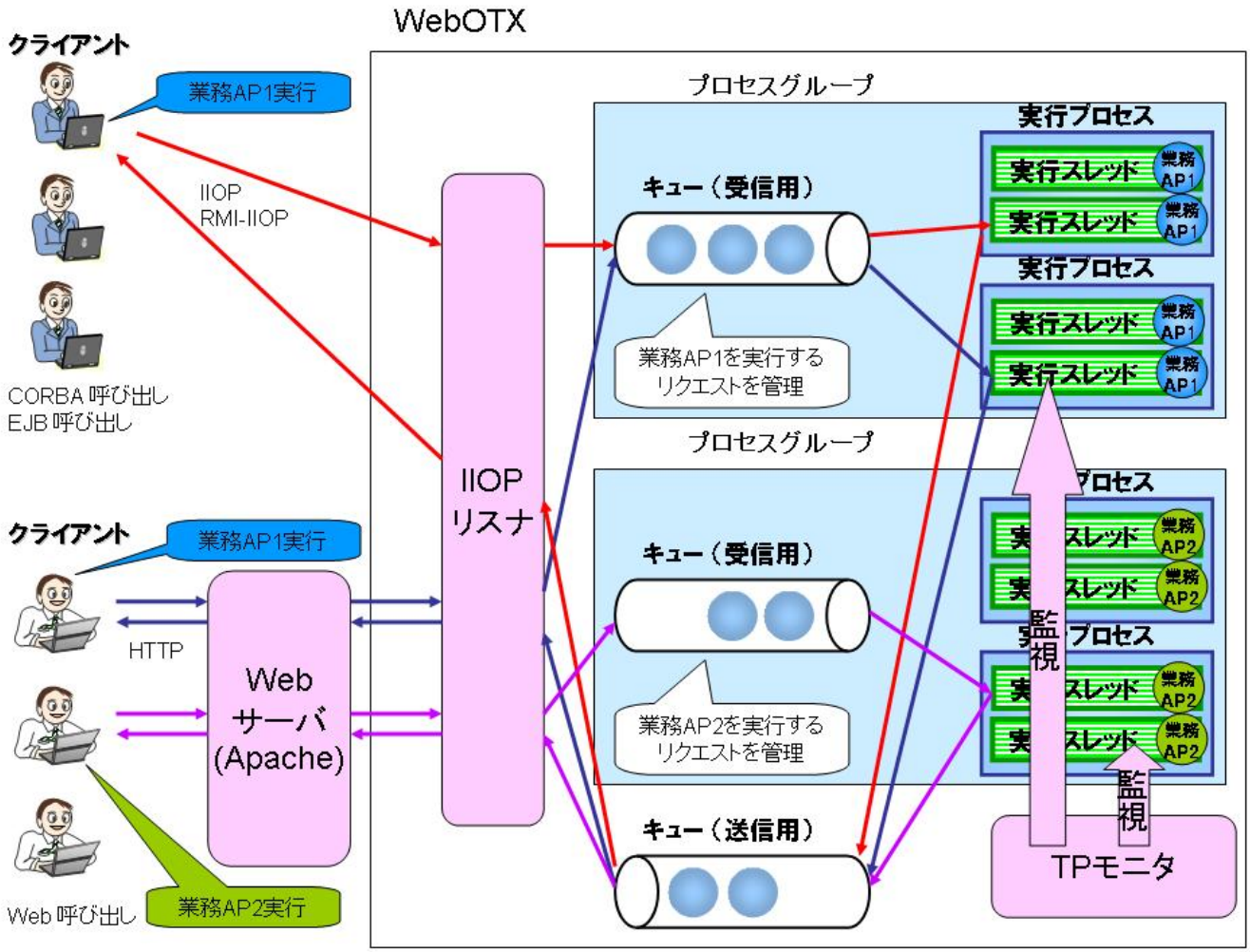
4.2.2.リソース制御

TP モニタは TP システムを管理するために必要なリソース(主にメモリ)をサービス開始時に事前に確保します。事前にリソースを確保することにより、運用中にリソース不足による動作不安定になることを避けるためです。システム規模に応じたリソース消費量にするために WebOTX では TP システム単位で以下のシステム・パラメータの設定が行えます。

パラメータ	Standard Edition		Enterprise Edition	
	既定値	最大値	既定値	最大値
アプリケーショングループ数	20	100	100	985
プロセスグループ数	20	100	100	985
コンポーネント数	200	3844	1000	3844
インタフェース数	200	10000	1000	10000
オペレーション数	200	10000	1000	10000

4.2.3.流量制御

クライアントからのオペレーション要求に対して WebOTX の処理フローについて以下に示します。



WebOTX のアプリケーションサーバは、サーバアプリケーションの制御を行なう TP モニタおよび、クライアントのオペレーション要

求、応答処理を行なうIIOPリスナ、実際に業務ロジックを実行するプロセスグループ群から構成されます。また、プロセスグループ単位で実行多重度を超えた要求があった場合に要求を保留するためのキューを用意しており、要求を同時実行可能とするために、プロセスグループ単位に実行多重度(プロセス数、スレッド数)を設定することができます。プロセスグループの各アプリケーションはWebOTXの制御部分(ベースライブラリ、オブジェクトマネージャ)とユーザが作成したコンポーネントから構成されています。以下にその構成部分について説明します。

- ベースライブラリ
  OLTP 機能を提供するライブラリです。リスナへの要求の送受信、プロセス・スレッド管理、障害監視、優先度制御などを行います。
- オブジェクトマネージャ
  CORBAやEJBの制御部分を行うライブラリです。ベースライブラリからの要求をCORBAやEJBの呼び出しに変換しユーザコンポーネントを実行し、各種コールバックAPIを提供します。EJBの実行基盤であるEJBコンテナもここに含まれます。
- サーバコンポーネント
  ユーザが作成したCORBAもしくはEJBコンポーネントです。クライアントからの要求に対してのビジネスロジックを提供します。

IIOPリスナはクライアントからのオペレーション要求を受け付けると、要求を処理するプロセスグループを判別し、該当プロセスグループのキューに要求をキューイングします。ベースライブラリにより、該当プロセスグループのうち処理実行待ちのスレッドがその要求を取得し、オブジェクトマネージャに処理を渡します。オブジェクトマネージャでオペレーション要求に応じたユーザコンポーネントの該当オペレーションの呼び出しを行います。ユーザコンポーネントで実際のビジネスロジックが処理され、オペレーションの処理結果はオブジェクトマネージャ、ベースライブラリを通してIIOPリスナの持つ送信用キューにキューイングされます。IIOPリスナは送信用キューに処理結果がキューイングされると、該当クライアントに応答を返します。

クライアントから実行多重度を超えた要求があった場合、要求はキューイングされ空きスレッドができるまで処理を保留します。キューイングできる要求数もアプリケーション毎に設定できます。キューイング上限を超えた要求に対してはクライアントにエラーを返します。サーバの資源に応じてプロセスグループの実行多重度(プロセス・スレッド)、キューイング数を適切に設定することにより、想定外の大量のトランザクション要求に対しても、サーバリソースが枯渇して動作が不安定となることを回避し、リソースの範囲内で安定した動作を行わせることができます。またトランザクション毎にプライオリティ(実行優先度)を設定でき、高負荷で処理がキューイングされている状態においても、重要な処理は優先的に行うことが可能です。

4.2.4.多重実行

WebOTXでは、アプリケーションを複数のプロセスに分散して実行するマルチプロセス動作を行うことができます。各プロセスはシングルスレッドでもマルチスレッドでも動作させることができます。すべてのプロセス上のスレッド群は、単一のキューに対してデータの到着を待ち合わせるので、空きスレッドに効率良くトランザクション要求をディスパッチすることができます。

マルチプロセス実行させることにより、アプリケーションの一時的な障害に対してサービスの継続が可能になります。データや環境、タイミングなどの要因でアプリケーション障害が発生しても、そのプロセスだけが異常終了し、他のプロセスは影響を受けません。したがって、システム全体ではサービスを中断させることがありません。

しかしマルチプロセス構成はマルチスレッド構成に比べてより多くのリソースを必要とします。システムで多数のプロセスが起動することによりマシンのリソース不足を招き、サーバ自体が不安定になる恐れがあります。マルチプロセスは予期せぬアプリケーション障害に備えての多重化とし、クライアントからの同時実行に備えての多重化はできるだけマルチスレッドで行う構成が望ましいといえます。

また、システム運用中に発生した想定外の大量のトランザクション要求に対して、動的にアプリケーション多重度(スレッド、プロセス)を増加させることが可能です。これにより、大量のトランザクション要求に対してサーバ側が処理しきれずに要求がキューに滞留しレスポンス悪化することが避けられます。また、負荷が下がったときに安全に実行多重度を元に戻すこともできます。

[Windows(32bits)の場合]

次の測定値はWindows(32bits)のアプリケーションプロセスで最低限必要となるメモリ使用量です。システム構築時のチューニングの参考としてください。実際のアプリケーションではユーザロジックで確保しているメモリ使用量(ヒープサイズやスレッドスタックサイズ)を考慮に入れる必要があります。また、メモリ使用量はOSやJavaのバージョンにより異なります。

・マルチプロセスシングルスレッド構成の場合

メモリ使用量(JavaAP) = 33MB × プロセス数

メモリ使用量(C++AP) = 10MB × プロセス数

・シングルプロセスマルチスレッド構成の場合

スレッド数	メモリ使用量(JavaAP)	メモリ使用量(C++AP)
1	33MB	10MB
2	35MB	11MB
5	38MB	14MB
10	43MB	19MB
20	54MB	29MB
50	85MB	60MB
100	138MB	112MB

注）メモリ使用量はタスクマネージャの仮想メモリサイズ

・マルチプロセスマルチスレッドの場合
1 プロセス当たりのメモリ使用量を測定し、プロセス数を掛けて見積もってください。
例）5 プロセス 10 スレッド構成の場合

メモリ使用量(JavaAP) = 43MB × 5 プロセス = 215MB

メモリ使用量(C++AP) = 19MB × 5 プロセス = 95MB

[HP-UX 11iv2 の場合]

HP-UX 11iv2 で Java AP をの最低限必要となるメモリ使用量は、およそ次の計算式で見積もり可能です。

メモリ使用量(JavaAP) = 300MB × プロセス数 + 40MB

式中の 300MB は空実装の場合の実測値です。40MB はプロセス間で共有で使用しているメモリです。あくまでおおよその計算式ですので、正確にはユーザの AP を使用して検証機で測定する必要があります。

Windows(32bits)の場合と比べてメモリ使用量が多いですが、これは WebOTX の実装の影響ではなく OS の影響です。

4.2.5.ステートレス・ステートフル

サーバアプリケーションのオブジェクトを実行する場合、オブジェクト内で状態を保持するか否かを指定する事ができます。状態を保持しないオブジェクトをステートレスといい、保持するオブジェクトをステートフルといいます。

ステートレス:
ステートレスの場合、アプリケーションが動作するプロセスが起動する時に、オブジェクトのインスタンスを一度に生成してプールしておきます。これらはオペレーションの処理が終了しても消滅することはありません。実際に消滅するのは、プロセスが終了する時です。そのため、2 度目以降のオペレーション呼び出しでは生成のオーバーヘッドが発生せずスループットをあげることができます。

ステートフル:
ステートフルの場合、オブジェクトのインスタンスは、クライアントからの処理要求(ファクトリオブジェクトのオブジェクト生成要求)が来た時に初めて生成されます。生成されたインスタンスは、要求を出したクライアントと 1 対 1 にマッピングされます。そして同一のクライアントから要求が来た場合は、それを同一のインスタンスで処理することになります。

WebOTX では CORBA アプリケーションの場合は、動作するアプリケーションの特性に応じてステートレス、ステートフルを設定することができます。ただしステートフルなアプリケーションはクライアント単位でオブジェクトが生成されるため、消費リソースやレスポンス面でステートレスより劣ります。できるだけアプリケーションはステートレスで作成することを推奨します。

EJB アプリケーションの場合は Bean の種類によってステートレス、ステートフルは決定されますので設定を変更することはできません。

Bean の種類	オブジェクト	ステート
ステートレス Session Bean	home	ステートレス
	remote	ステートレス
ステートフル Session Bean	home	ステートレス
	remote	ステートフル

Entity Bean	home	ステートレス
	remote	ステートフル
メッセージ・ドリブン Bean	--	ステートレス

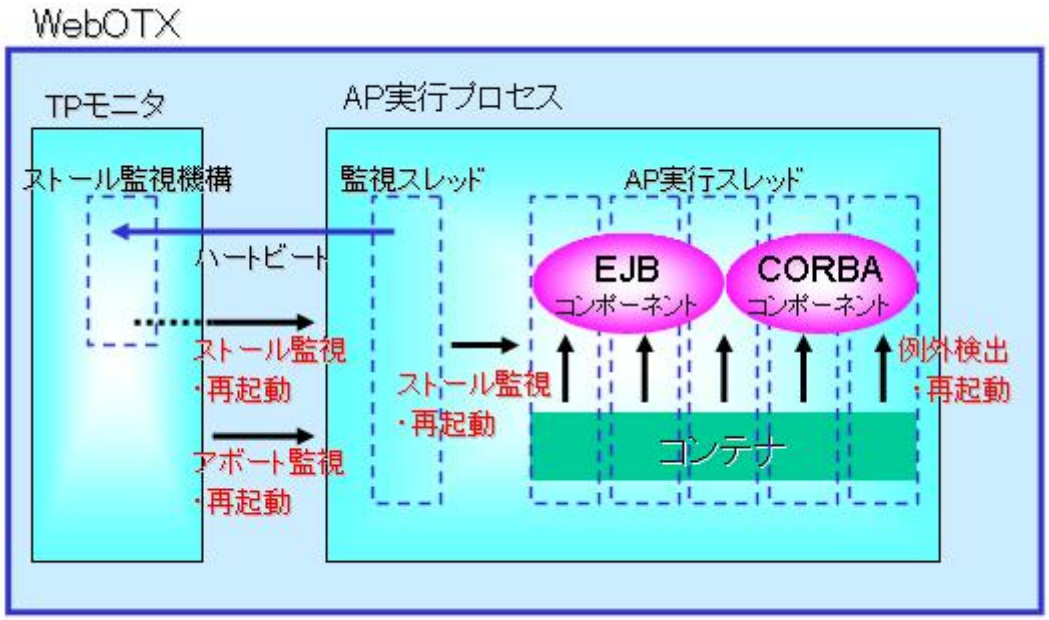
4.2.6.プロセス障害監視

WebOTX では OLTP 技術を取り込んだ TP モニタによりアプリケーションサーバ上で動作するアプリケーションが管理されます。アプリケーションは以下の単位でグルーピングされ管理されます。TP モニタはこれら構成単位毎に起動、停止などの運用制御や状態監視、障害リカバリを行います。

- アプリケーションアボート監視
  クライアントからのトランザクション要求に対してサーバアプリケーションが不正な処理を行い例外が発生した場合に、該当スレッドを閉塞し、例外が発生したことをコールバック API によりアプリケーションに通知します。アプリケーションは、データベースのロールバック前に障害情報の保存などの後処理を行うことができます。

- アプリケーションストール監視
  クライアントからのトランザクション要求に対して指定した時間が経過してもサーバアプリケーションが応答を返さない場合に、アプリケーションを異常終了させて該当プロセスを閉塞します。これによりクライアントに応答が返らない事態やサーバの資源を無制限に消費する事態を回避できます。

- 異常終了プロセス再起動
  上記事象により、アプリケーションプロセスが異常終了した場合、そのプロセスが使用していたリソース(コネクション、共有メモリ、データベース資源)を WebOTX が解放します。さらに、設定に応じてプロセスの再起動を行います。障害発生後もそれ以前と同じ処理能力を維持することができます。



4.2.7.キュー制御

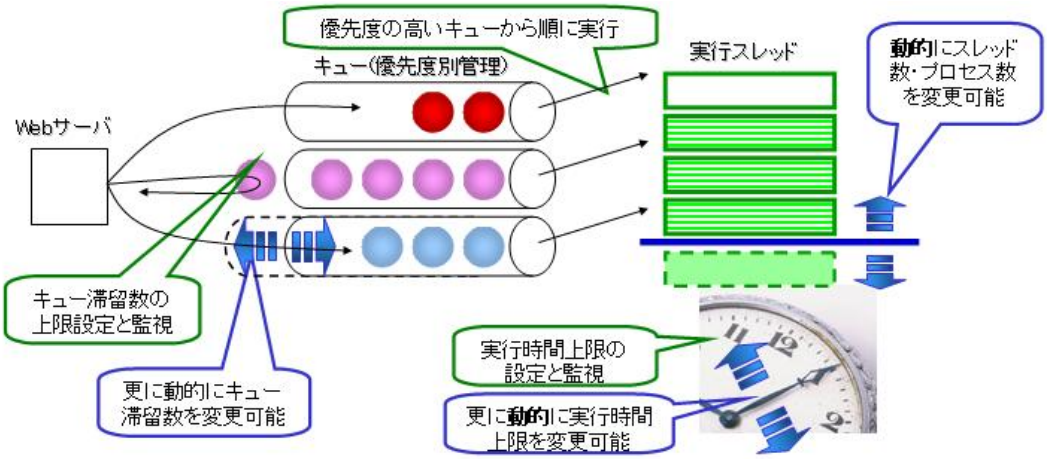
WebOTX では予測できない急激な負荷変動に対しても重要度の高い業務やユーザに対してサービスを継続させるためにキュー制御を行なっています。WebOTX のキュー制御について説明します。

- プロセスグループ単位のキュー制御
  WebOTX ではプロセスグループ単位でキュー管理を行なっています。これによりある 1 つのプロセスグループの負荷が高くても、他のプロセスグループの実行には影響を与えません。

- プライオリティ管理
  同一プロセスグループの負荷が高いとき通常であれば、そのプロセスグループ全体に影響が及びます。そこで WebOTX ではオペレーション単位で実行優先度を設定することが可能です。実行優先度が高いオペレーション要求は、低いオペレーション要求がキューイングされた状態でも優先的に実行させることが可能です。

- キュー滞留数閾値設定
  高負荷時は一時的にキューに滞留しますが、慢性的にスローダウンが発生するとキュー滞留が増加し、システム全体のレスポンスを悪化させてしまいます。そのような事態にならないように、キュー滞留数に閾値を設けることが可能です。閾値設定が行

なわれると、それ以上の要求は受け付けずクライアントにエラーを返します。なおこのキュー滞留数の閾値は運用中に動的に変更可能です。



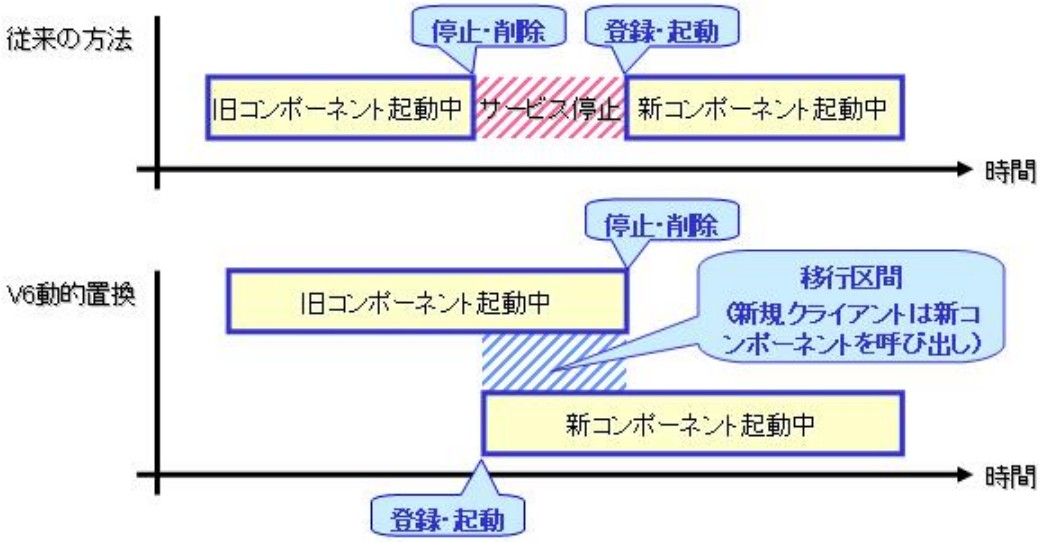
4.2.8.アプリケーションの動的配備

WebOTX では、アプリケーションをコンポーネント単位で動的に登録、変更、削除することができます。V6 では動的置換について強化を行なっています。

下図にあるように従来の方法ではまず旧コンポーネントを停止してから削除しそのあと登録・起動を行なっていましたがこの方法では以下の問題点がありました。

- 停止してから起動までサービスの停止時間がある。
- 置換後旧コンポーネントは利用できないため、クラスタなどで複数のサーバにまたがってアプリケーションが存在する場合に新旧のアプリケーションが混在することがあり、思わぬ結果になる危険性がある。

この問題を解決するために旧コンポーネントと新コンポーネントが共存する移行期間を設けるようにしました。移行期間中は旧コンポーネント、新コンポーネント共に呼び出すことが可能となります。全てのクライアントで旧コンポーネントの利用が完了した段階で、旧コンポーネントを削除できます。



4.2.9.性能情報

WebOTX ではシステムの運用状態を取得するための各種情報を採取することができます。

ジャーナル

ジャーナルは WebOTX の稼働状況を評価するための性能及び統計情報を各種レポートとして以下の項目について採取します。

- アクティビティレポート

指定された時間間隔における WebOTX 全体の動作状況を表示します。表示項目は CPU 使用量(秒)、CPU 使用率(%)、オペレーション処理数(件数、%)、スループット、レスポンス時間(最大、最小、平均)です。

- トランザクション・サマリレポート

アプリケーショングループ、プロセスグループ単位に発生したオペレーション処理状況を表示します。表示項目は CPU 時間の合計(秒、使用率)、オペレーション処理件数の合計(件数、%)、レスポンス時間(最大、最小、合計)です。

- CPU 時間タイムチャート

全体、アプリケーショングループ、プロセスグループ単位に、一定時間間隔におけるオペレーション処理と CPU 時間の推移を表示します。表示項目は単位時間あたりの CPU 時間の合計とそのグラフ化です。

- トランザクション件数タイムチャート

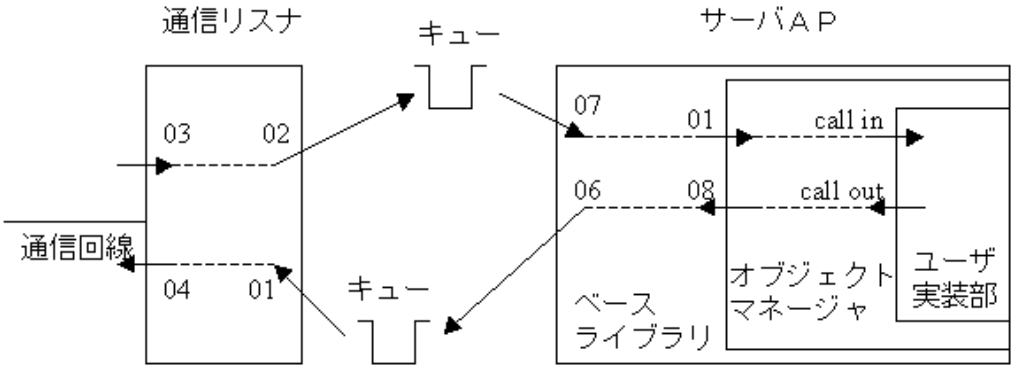
全体、アプリケーショングループ、プロセスグループ単位に、一定時間間隔におけるオペレーション処理件数の推移を表示します。表示項目は単位時間あたりのオペレーション件数の合計とそのグラフ化です。

- トランザクション件数レスポンス時間グラフ

全体、アプリケーショングループ、プロセスグループ単位に、一定レスポンス時間内で処理されたオペレーション処理数の一覧表示です。表示項目はオペレーション処理件数とそのグラフ化です。

イベントジャーナル

イベントジャーナルは個々のオペレーション処理に関して処理の流れを詳細に採取します。システムの詳細な性能分析をすることが可能になります。以下にイベントジャーナルの主な採取ポイントを以下に示します。



コンポーネント	番号	採取ポイント
通信リスナ (IIOP リスナ)	01	レスポンスをキューアウト時
	02	リクエストをキューイン時
	03	リクエストを受信完了時
	04	リクエストを送信完了時
ベースライブラリ	01	オブジェクトマネージャ呼び出し時
	06	レスポンスキューイング時
	07	リクエストキューアウト時
	08	オブジェクトマネージャ戻り時
オブジェクトマネージャ	call in	ユーザオペレーション呼び出し時
	call out	ユーザオペレーション戻り時

オペレーションジャーナル

オペレーションジャーナルは、システムの稼働状況・統計をオペレーション単位でレポート出力します。
オペレーションごとの実行時間情報(平均値・最大値・最小値)、オペレーションの実行回数、プロセスグループごとの稼動スレッド数(起動中のスレッドのうち実際にオペレーションを実行しているスレッド数)を知ることができます。それぞれの情報について、全体の統計と単位時間ごとの推移がレポート出力されます。

R6.3 からはこれに加えて、オペレーションの CPU 使用時間（ユーザモード、カーネルモード）の平均・最大・最小、プロセスグループの CPU 使用率・CPU 使用時間(ユーザモード、カーネルモード)をレポート出力します。
またオペレーション情報のサマリに「実行時間上限推奨値」が追加されました。実行時間上限を設定する際の参考にしてください。

4.2.10.プロセス間での情報共有

サーバアプリケーションの各プロセス間で情報を引き継ぎ、共有を行うために WebOTX では以下の機能を提供しています。

- プロセス共有プロパティ
  プロセス共有プロパティは、プロセス間で共有となるデータを管理するために使用する機能です。一般的にいう「共有メモリ」と同様の機能です。WebOTX が共有プロパティを操作するための API を提供します。作成、削除、排他についても WebOTX 内部で管理するためサーバアプリケーション内で意識する必要はありません。

4.2.11.アプリケーション初期トランザクション

WebOTX ではサーバアプリケーション起動前に、複数のプロセスグループ間で利用するグローバルデータなどの初期化を行うためにアプリケーション初期トランザクションを定義できます。アプリケーション初期トランザクションはアプリケーショングループ起動時に配下のプロセスグループのプロセス起動前に運用管理ツールで定義した関数が呼び出されます。

4.2.12.データベース連携

WebOTX で使用するデータベースに対し、データベースの接続・切断処理を自動的に行います。データベース自動接続はプロセス初期化処理の後で行い、切断はプロセス終了処理の前に行います。さらにオペレーション実行結果によりオペレーション実行後、オペレーションの実行結果に応じて、正常終了の場合はコミットを異常終了の場合はロールバックを行います。

4.2.13.名前サービスへのリファレンス登録

WebOTX の CORBA アプリケーションではクライアントがサーバアプリケーションのオブジェクトリファレンスを取得するのに名前サービスを利用しますが、オブジェクトリファレンスを名前サービスに登録する名前についてはインタフェース毎に任意に設定することができます。また登録、削除の契機ですがサーバアプリケーションの起動、停止にあわせる方式と永続的に登録する方式があります。なお EJB アプリケーションについては、JNDI名により登録する名前が決定されるため、配備を行なうと自動的に設定が行なわれます。

登録する名前の設定

名前サービスに登録する名前についてはインタフェース毎に任意に設定することができます。corbaname 形式の URL で複数設定することができますので、オブジェクトリファレンスを登録する名前サーバが分散させることも、1 つの名前サーバに複数の名前で登録することもできます。インタオペラブル名前サービスをサポートする他社の名前サーバに対しての登録も可能です。

リファレンスの登録契機

WebOTX におけるオブジェクトリファレンスの名前サーバへの登録契機は、以下の 2 種類をサポートしています。

- アプリケーションと同期して登録、削除(一時的)
  サーバアプリケーションの起動時にオブジェクトリファレンスを登録終了時に削除します。
- アプリケーションとは独立して、永続的に登録(永続的)
  サーバアプリケーションの起動、停止とは独立して、永続的にオブジェクトリファレンスを登録します。登録は運用管理ツールもしくはコマンドを用いて行います。WebOTX V6.2 ではこちらが既定値となっています。

	一時的登録	永続時的登録
名前サーバへのオブジェクトリファレンス登録/削除	名前サーバへのオブジェクトリファレンス登録/削除はサーバアプリケーションと同期するため、アプリケーション起動時に名前サーバへの登録が行われます。このため、WebOTX サービス起動時に全てのアプリケーションにおいて名前サーバへの登録処理が行われるため、	一度名前サーバへの登録を行えば、サーバアプリケーションの状態に関係なく登録は行われているため、WebOTX サービス起動時の負荷が軽減されます。

	AP サーバ、名前サーバへの負荷が高くなります。	
サーバアプリケーション未起動時のクライアント動作	サーバアプリケーション未起動時は名前サーバにオブジェクトリファレンスが登録されていないため、クライアントからのアクセスは名前サーバからオブジェクトを取得できないというエラーになります。既に取得済みのオブジェクトを利用した場合は、呼び出しを行ったときにエラーとなります。	サーバアプリケーション未起動時も名前サーバよりオブジェクトリファレンスを取得できるため、クライアントのアクセスは全て呼び出しを行ったときにエラーとなります。
マルチサーバ構成における負荷分散	マルチサーバ構成における負荷分散として名前サーバのラウンドロビン負荷分散が利用できます。クライアントが名前サーバにアクセスする契機でアクセスする AP サーバが振り分けられます。以降そのオブジェクトを利用する限り同一サーバにアクセスします。	マルチサーバ構成における負荷分散としてオペレーション呼び出しでのラウンドロビン負荷分散が利用できます。クライアントがオペレーション呼び出しを行う度に AP サーバが振り分けられます。呼び出しを行う度に実行する AP サーバが振り分けられるためオブジェクトに状態を保持し、複数のオペレーションにまたがって値を参照、更新する(ステートフル)場合は対応できません。

4.2.14.ログ採取

WebOTX ではシステムの動作を確認できる各種のログやトレースを提供しています。また提供 API を用いることによりユーザロジックの中でログ出力、トレース出力ができ、障害解析や性能分析に役立てることができます。

WebOTX V6.2 ではログについて、障害解析により原因究明を行ないやすくするための強化を行なっています。

- 通常時ログ(V5 と同等)
  V5 と同様のログを採取できます。従来どおりサイズやレベルの設定が行なえます。
- 起動時ログ(V6.2 以降)
  アプリケーションの起動ができなかった場合や初期化処理に問題があった場合の問題解析を容易にするためアプリケーション起動時のログを通常ログとは切り離して作成することにしました。これにより起動時のログは確実に残し、起動後のログにより上書きされることを防ぐことができます。
- log4j アペンダ提供
  V5 まではユーザロジック中でログ出力を行ないたい場合、オブジェクトマネージャが提供するログ出力 APIを利用しなければ出力されませんでしたが、Java で一般的にログ出力 API として用いられている log4j の API を利用しても出力できるように log4j アペンダを提供します。これにより log4j で出力したログもサーバ AP ログにマージされて出力されます。
- Java スタックダンプ採取(V6.2 以降)
  Java アプリケーションで以下の契機でスタックダンプを出力するようになりました。これにより不具合の原因究明をより行ないやすくできます。
  - ユーザロジック中で catch していない例外が発生した場合
  - ストール監視のための実行時間上限を設定し、実行時間がその上限を越えた場合
  - アプリケーションを強制終了させたときに、終了要求に応じないスレッドがある場合
- ログの退避機能(V6.2 以降)
  現在動作していないプロセスのログは save ディレクトリ配下に退避するようにしました。

4.2.15.仮想宛先（VD）

WebOTX ではメインフレームで培った技術を活かすために、仮想宛先(VD)をサポートしています。VD を利用した、帳票印刷や非同期トランザクション実行をサポートします。

WebOTX では次の VD をサポートしています。

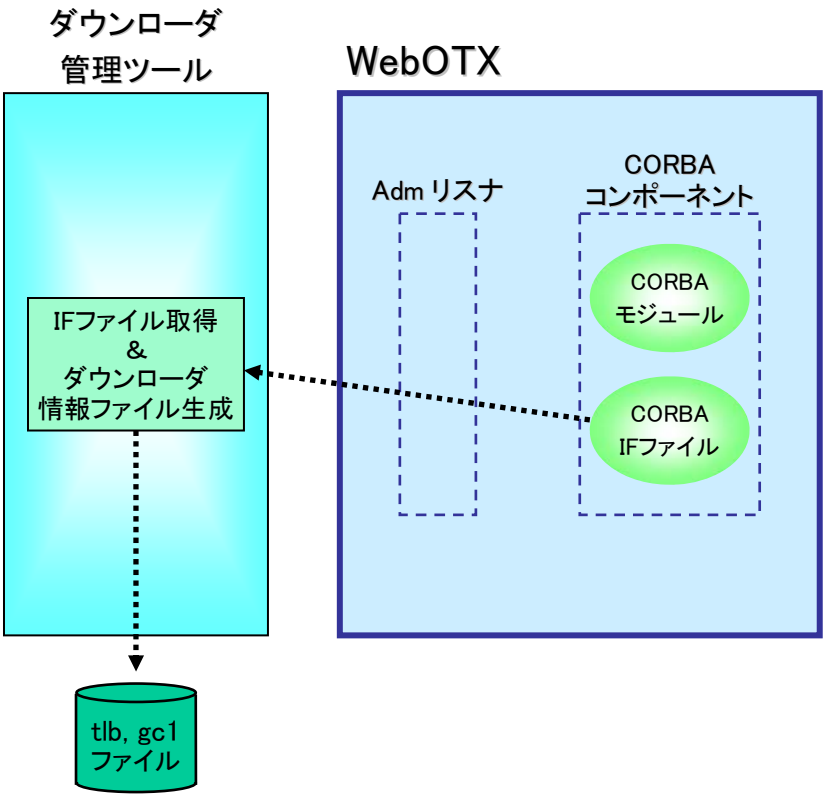
- 端末型 VD
  主に帳票印刷に利用します、サーバ AP より端末型 VD に帳票電文イメージが送信されると VD にその電文をキューイングしま

す。WebOTX Print Kit などのクライアントがその電文を受信するまで VD 内で電文を保持します。

- トランザクション型 VD
  主にサーバ AP より他のサーバ AP へ非同期にオペレーション実行要求を行なうときに利用します。要求元サーバ AP よりオペレーション実行要求の電文をトランザクション型 VD でキューイングし、呼び出されたサーバ AP がその電文を引き取るまで、VD 内で電文を保持します。
- 間接型 VD
  トランザクション型 VD を拡張した VD です。トランザクション型 VD では全ての非同期要求はシリアライズして処理されます。それぞれの非同期処理に順序性がある場合はシリアライズされなければならないのですが、順序性が無い場合、間接型 VD を利用することで処理を並列に実行させることができます。なお VD の振り分け方針については次の 2 通りがあります。
  - ラウンドロビン ： ラウンドロビンに振り分けを行ないます。
  - 滞留数優先 ： 滞留数の一番小さいトランザクション型 VD に振り分けを行ないます。

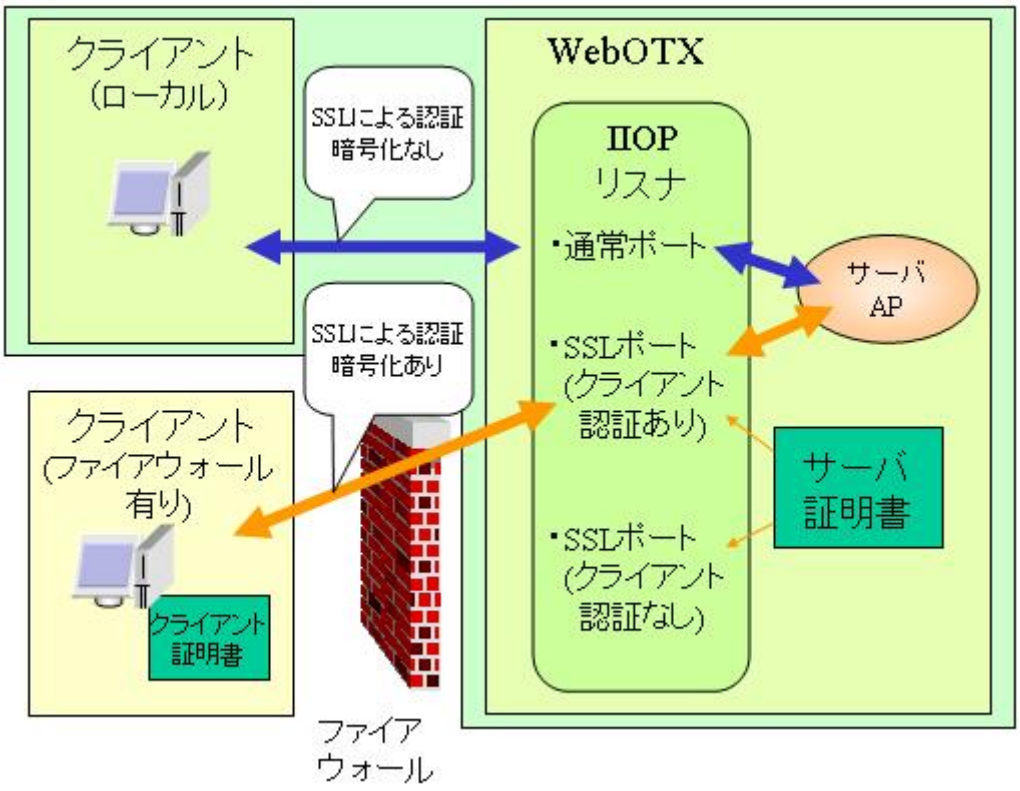
4.2.16.VB連携

WebOTX の CORBA アプリケーションは Microsoft Visual Basic から呼び出すことが可能です。これを実現するために CORBA インタフェースを COM インタフェースにマッピングさせています。このマッピングを行なうためにダウンローダ管理ツールおよび、Adm リスナを提供しています。



4.2.17.SSLとの連携

「Open SSL」もしくは「SecureWare/セキュリティパック」を WebOTX 実行環境に適用することにより、WebOTX のクライアントーサーバ間通信を Secure Socket Layer (SSL)プロトコルにより行えます。SSL は共通鍵暗号化方式と公開鍵暗号化方式、デジタル証明書を併用した認証および暗号化のプロトコルで、WebOTX のクライアントーサーバ間通信においてクライアント、サーバの認証および電文の暗号化を行うことができます。WebOTX の SSL 通信についての特徴を以下に説明します。



クライアント・サーバの認証

クライアント、サーバでそれぞれ認証を行うための鍵識別子を設定します。クライアントアプリケーションは起動引数もしくはAPIを利用して鍵識別子を指定します。サーバアプリケーションは TP システムの設定として統合運用管理ツールより設定を行います。設定を行うと、クライアントとサーバのコネクション開設時にお互いの認証を行います。またサーバアプリケーションでは API を利用することによりクライアントを識別するために証明書情報を取得することも可能です。

ポートの設定

WebOTX の IIOP リスナでは SSL を使用した通信と使用しない通信を同時にサポートするため、通信用のポート番号をそれぞれ独立して設定することが可能です。これにより、ファイアウォールを超える通信は SSL を利用して、ローカル内での通信は SSL を利用しない通信と振り分けて利用することが可能です。

- 通常ポート
  SSL を利用しない場合に使用するポート番号です。
- SSL クライアント認証付きポート
  SSL を利用し暗号化を行い、なおかつクライアント、サーバ認証を行う場合のポート番号です。
- SSL クライアント認証なしポート
  SSL を利用し暗号化は行うが、サーバ側でクライアントの認証は行わない場合のポート番号です

4.2.18.運用アシスタント

運用アシスタント機能は、定期的に採取した情報をもとにシステムの稼働状況を学習し、そこでの分析をもとに適切な運用を支援する、自律運用機能です。運用アシスタントにより以下の設定の最適化と、適正値算出に関するコスト削減をはかることができます。稼働状況の変化に合わせて、システムの設定を動的に変更させることも可能です。

多重度の最適化支援
システムの稼働状況から、設定されている多重度(プロセス数/スレッド数)が適切かどうか分析します。多重度は少なすぎると性能問題を引き起こし、多すぎると不必要にリソースを消費してしまいます。
多重度が少なすぎる/多すぎると判断された場合は、運用管理ツールなどを通じてオペレータに通知します。またオプション設定により、システムの稼働状況に合わせて多重度を自動設定することもできます。

実行時間上限の適正値算出
計算では求めづらく、また設定箇所の多い「実行時間上限」を、運用アシスタント機能により自動的に算出します。「実行時間上限」はオペレーションのストール障害復旧の要となる重要な設定です。オペレータは一つの操作だけで、全オペレーションに対して適切な実行時間上限を設定することができます。

スローダウン障害監視

オペレーションのスローダウン障害を検出し、運用管理ツールなどを通じてオペレータに通知します。一般的な監視機構とは違い、オペレーションごとの閾値設定なしでスローダウン障害を検出できます。

スローダウン状態が長期化している場合は、追加して警告を通知します。このとき自動的にスタックトレースを採取し、スローダウン障害発生時の情報の消失を防ぎ、障害調査を容易にします。

5.アプリケーションサービス

5.1.Webサービス

5.1.1.Webサービス基本仕様

Web サービスの基本仕様は次のとおりです。

Simple Object Access Protocol (SOAP)

WebOTX は、SOAP 1.1、SOAP 1.2 仕様に対応した SOAP ランタイムを提供します。SOAP ラインタイムは、トランスポートプロトコルとして HTTP に対応しており、SOAP メッセージの送受信・解析・生成を Java で行うための API を提供します。

Web Services Description Language (WSDL)

WebOTX は、WSDL 1.1 仕様に対応しており、Java プログラムから WSDL を生成する機能、WSDL から SOAP ランタイムを使用して SOAP メッセージを送受信・解析・生成するための Java ソースコードを生成する機能を提供します。

Universal Description, Discovery and Integration (UDDI)

WebOTX は、全 Edition で UDDI 2.0 と 3.0 に対応した UDDI クライアントライブラリを提供します。UDDI クライアントライブラリを使うと、UDDI レジストリにアクセスして Web サービスを検索したり、Web サービスを登録したりすることができます。また、WebOTX UDDI Registry(オプション製品)では、UDDI 2.0 と 3.0 に対応した UDDI レジストリを提供します。

Web Services-Interoperability Organization Basic Profile (WS-I BP)

WebOTX は WS-I BP 1.0 に対応し、WS-I BP 1.0 に沿った Web サービスアプリケーションの開発を支援します。

5.1.2.J2EE対応

J2EE 1.4 に規定された次の仕様に対応した機能を提供します。

SOAP with Attachments API for Java (SAAJ)

Java 言語で SOAP メッセージや添付ファイルつきの SOAP メッセージを作成したり、読み込んだりするときに使用する標準 API です。

Java API for XML-Based RPC (JAX-RPC)

SOAP、WSDL を使って Java 言語で RPC を実現するときに使用する標準 API や Web サービスアプリケーションの実装方法などについて定義した仕様です。

Java API for XML Registries (JAXR)

UDDI や ebXML などの XML ベースのレジストリサービスにアクセスするときに使用する Java 言語の標準 API です。

Web Services for J2EE (WSEE)

J2EE サーバ（Web コンテナ/EJB コンテナ）で Web サービスアプリケーションを動作させるときの Web サービスアプリケーションの実装方法、Web サービスアプリケーションのポータビリティについて定めている仕様です。

また、次の機能も提供します。

Java API for XML Web Service (JAX-WS)

JAX-RPC の後継仕様で、XML/HTTP バインディングや非同期通信が可能になっています。

Java Architecture for XML Binding (JAXB)

Java と XML のデータバインディングのための API です。

5.1.3.プロバイダ

SOAP で RPC を行う時、EJB や CORBA のバックエンドサーバアプリケーションに対して SOAP でダイレクトにアクセスするためのプロトコル変換を自動で行うプロバイダ機能を提供します。

5.1.4.開発環境

WebOTX 開発環境は、統合された J2EE の開発環境の中で Web サービスの開発支援を行います。既存のビジネスロジックを Web サービス化する「Web サービス作成ウィザード」を提供し、Web サービスアプリケーションに必要なファイルである、サービスエンドポイントインタフェース、プロキシ・タイ・クライアントクラス、WSDL、各種配備記述子、をウィザードに答えていく簡単な操作だけで生成することができます。同時に、Web アプリケーションや EJB アプリケーションのアーカイブを自動で作成する機能を提供します。

5.1.5.セキュリティ

WS-Security

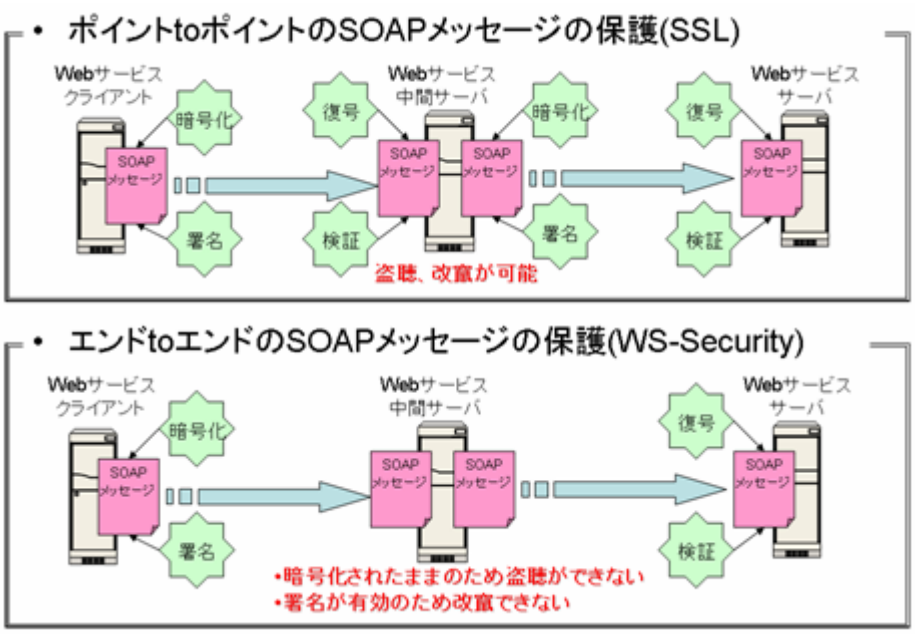
WS-Security は、Web サービス間通信におけるセキュリティ技術です。WS-Security は、Web サービス間の通信において、完全性(改竄検知)、秘匿性(盗聴防止)、送信者認証(認証、および否認防止)といったセキュリティ要件を保証します。これらのセキュリティ要件を実現する方法として、WS-Security では SOAP メッセージへの XML 署名、XML 暗号の適用方法、SOAP メッセージ送信者の認証情報である Token の利用方法を規定しています。WS-Security は、それに準拠する製品相互の互換性を確保し、エンド to エンドなセキュリティを実現することが特長です。

WebOTX では、OASIS(Organization for the Advancement of Structured Information Standards)で標準化が完了した次の OASIS Standard 仕様に準拠した WS-Security 機能をサポートしています。

仕様名称	概要
Web Services Security: SOAP Message Security V1.0	SOAP メッセージへの XML 署名、XML 暗号の適用方法、認証情報(Token)の利用方法に関する仕様。
Web Services Security: Username Token Profile V1.0	ユーザ名、パスワードを用いた認証情報の利用方法に関する仕様。
Web Services Security: X.509 Token Profile V1.0	X.509 証明書を用いた認証情報の利用方法に関する仕様。
Web Services Security: SAML Token Profile	OASIS SAML 仕様に準拠したセキュリティトークンの利用方法に関する仕様。

WS-Security は、Web サービス間通信のセキュリティを実現する一技術であり、十分なセキュリティを提供することを保証しません。高いセキュリティレベルを必要とするシステムにおいては、他のセキュリティ技術(例えば SSL/TSL など)の併用を検討する必要があります。

SSL と WS-Security



図のように、広くインターネット通信で利用されている SSL 技術を用いた場合、中間サーバ上で復号が可能なため、盗聴が可能です。SSL は、通信ポイント間のセキュリティ確保を目的とした技術であるため、ポイント to ポイントのセキュリティ技術と呼ばれます。一方 WS-Security は、復号が可能なのは受信者のみで、中間サーバ上では復号が不可能であるため、送信者と想定した受信者間のすべての経路でのセキュリティを確保します。このような特徴をもつため、WS-Security はエンド to エンドのセキュリティ技術と呼ばれます。

5.2.JNDI

5.2.1.JNDIとは

あらゆるコンピューティングシステムの基本機能は、ネームサービスです。ネームサービスによって、名前とオブジェクトを関連付け、名前をもとにしてオブジェクトを検出する方法が提供されます。従来のシステムでは、多くの場合、ネームサービスは単体では機能しません。通常は、ファイルシステム、ディレクトリサービス、データベース、デスクトップ、メールシステム、スプレッドシート、カレンダなど、その他のサービスと統合されます。たとえば、ファイルシステムには、ファイルとディレクトリ用のネームサービスが組み込まれています。また、スプレッドシートには、セルとマクロ用のネームサービスが組み込まれています。

Java Naming and Directory Interface (JNDI) は、Java プラットフォーム用の標準拡張機能で、Java テクノロジに対応したアプリケーションに、企業内の複数のネーミングおよびディレクトリサービスへの統一したインタフェースを提供します。 JNDI は、Java Enterprise API セットの一部として、異機種システムの混在したネーミングおよびディレクトリサービスへのスムーズな接続を可能にします。開発者は、この業界標準機能を使用することで、強力かつ移植性の高いディレクトリ対応アプリケーションを構築することができます。

J2EE では、JNDI がさまざまな場面で用いられています。例えば、EJB のホームインタフェースの取得、J2EE アプリケーションで使用するリソースの取得、環境エントリ取得などで使用されます。

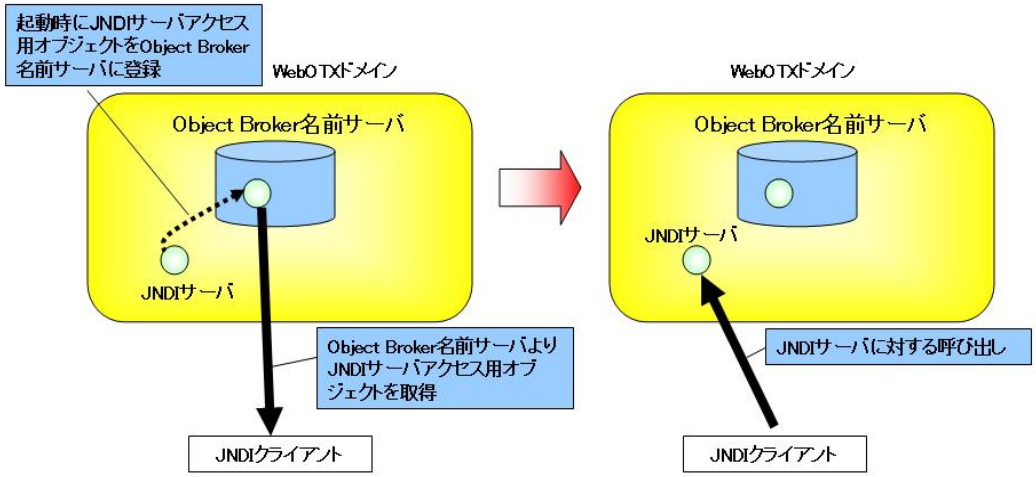
JNDI アーキテクチャは、JNDI API および JNDI SPI から構成されます。JNDI API とは、JNDI を使用するためのインタフェースを定義しているクラスライブラリで、JDK 1.3 以上ではシステムのクラスライブラリに含まれています。JNDI SPI とは、JNDI の実装を提供するクラスライブラリで、JNDI の提供ベンダから提供されます。JNDI の利用者は環境プロパティによってどの SPI を使用するかを指定することにより、使用する JNDI の実装を選択して使用します。

5.2.2.WebOTXの提供するJNDIについて

動作概要

WebOTX の JNDI サーバは、RMI-IIOP 通信を行うサーバプロセスで、Object Broker の CosNaming サーバをバックエンドの名前サーバとして使用します。 Object Broker の CosNaming サーバは、JNDI サーバにアクセスするための CORBA オブジェクトの参照の取得のために用いられます。

JNDI サーバは起動時に CosNaming サーバの特定のコンテキスト配下に自身のホスト名で自身の RMI-IIOP サーバオブジェクトを登録します。 JNDI サーバを使用するクライアントは CosNaming サーバの特定のコンテキスト配下から JNDI サーバオブジェクトの参照を取得します。そして、JNDI サーバオブジェクトに対してバインド、アンバインドなどの名前空間にアクセスするメソッド要求を実行します。



CosNaming サーバを使用せずに直接 JNDI サーバにアクセスする方法もあります。両者は JNDI クライアントの接続先 URL の形式によって切り替わります。直接 JNDI サーバにアクセスする場合は通信先の JNDI サーバのホスト名、ポート番号を URL に指定します。

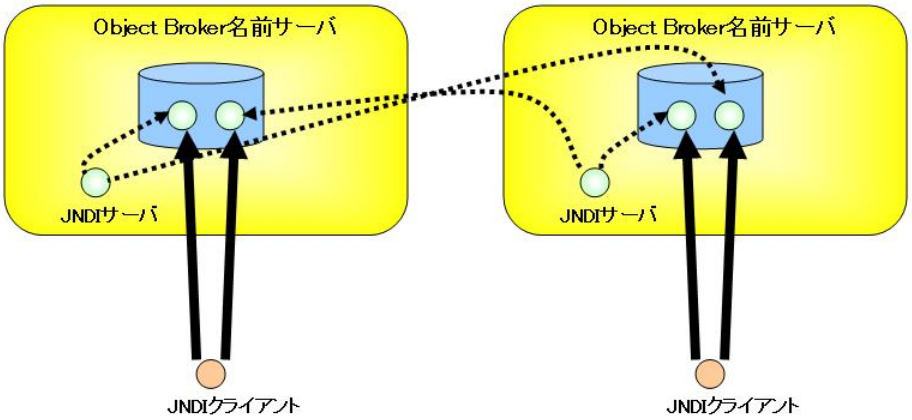
二重化

WebOTX の JNDI サーバは複数のサーバで同じ名前空間を管理することができます。これを二重化と呼びます。

JNDI サーバの二重化を行うためには、各 JNDI サーバが使用する CosNaming サーバを同一の設定にします。このように設定して各 JNDI サーバを起動すると、CosNaming サーバの特定のコンテキスト配下に複数の JNDI サーバの参照が登録されます。これらの JNDI サーバは同じ名前空間を管理することになります。

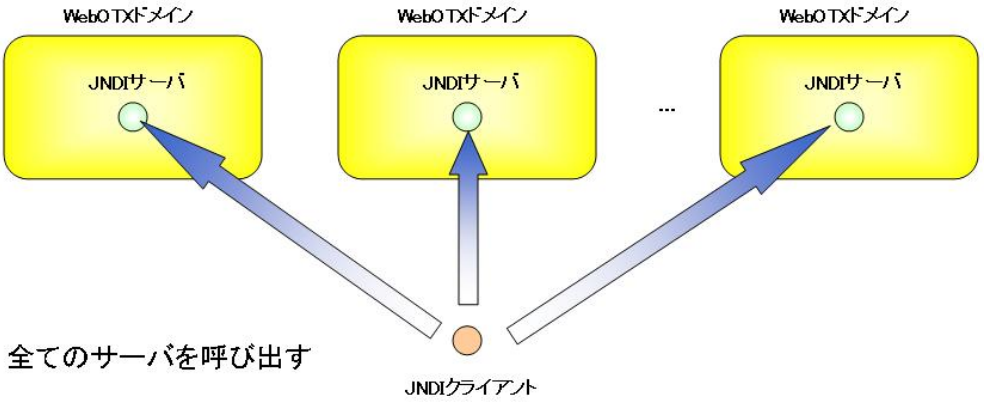
ただし CosNaming サーバを 1 つしか指定していない場合、JNDI サーバ全てがその CosNaming サーバに依存することになってしまいます。よって通常はそれぞれの JNDI サーバで動作している CosNaming サーバを全て列挙して指定します。このように指定すると JNDI サーバは起動時に。列挙した全ての CosNaming サーバに自身の RMI-IIOP サーバオブジェクトを登録します。このため一つの CosNaming サーバが停止しても、別の CosNaming サーバから JNDI サーバのリファレンスを取得することができます。

クライアント側でも同様に接続先 URL に複数の URL を列挙して指定することにより、動作中の CosNaming サーバから JNDI サーバのリファレンスを取得します。



二重化を行った場合、各 JNDI サーバで保持している情報が同じものでなければなりません。この同期処理は JNDI クライアントが行います。つまり bind、unbind などの名前空間の情報を変更するようなメソッドを実行する際は JNDI クライアントはその名前空間を管理している JNDI サーバ全てに対してメソッド実行を行います。

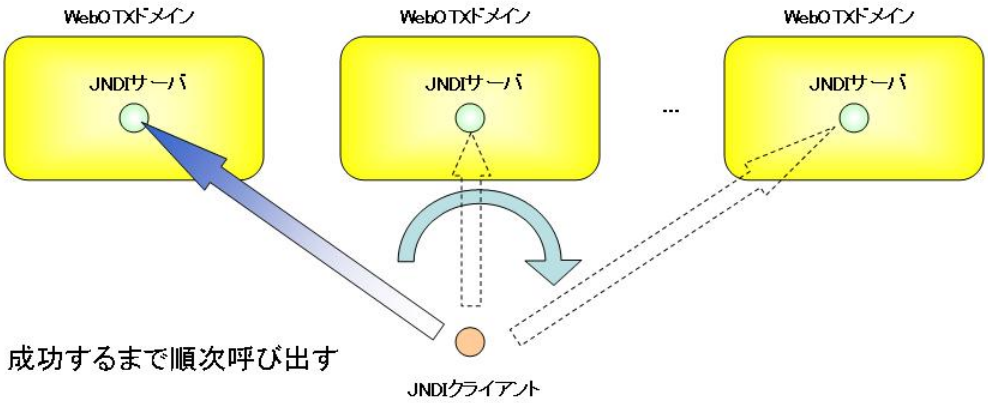
名前の登録、削除



このとき、それぞれの JNDI サーバに対するメソッド呼び出しは非同期に行われるため、メソッド実行が複数の JNDI サーバに対して行われることによる性能劣化は最小限に抑えられています。

lookup などの名前空間の情報を参照する処理は、どれかひとつの JNDI サーバに対してメソッド実行を行います。障害などによりその JNDI サーバが停止している場合は、別の JNDI サーバに対してメソッド実行を行います。

名前の参照



これらの処理は WebOTX JNDI が提供する SPI クラスの中で行われます。つまり、JNDI クライアント側では通常の JNDI メソッドを呼び出すだけです。

JNDI サーバは名前空間の情報をメモリ中に保持しているため、障害などによって再起動された場合、登録されていた名前が全て消えてしまいます。このため、再起動時に CosNaming サーバの特定のコンテキスト配下に登録されている JNDI サーバから名前空間の情報を全て取得し、自身の名前空間の情報とマージします。これによって他の JNDI サーバとの同期を行います。

CosNaming サーバを使用せずに JNDI サーバにアクセスする場合は、接続先 URL には二重化対象の JNDI サーバのホスト名、ポート番号をすべて列挙します。すると CosNaming サーバから JNDI サーバのリファレンスを取得する代わりに URL に指定された JNDI サーバに対して二重化を行います。

5.3.JDBCデータソース

JDBC コネクション（java.sql.Connection インスタンス）の取得は、データベースとの物理的なコネクションの接続処理をともないますので、非常に時間を要します。そのため、JDBC コネクションのプール管理を行うことが一般的です。 また、アプリケーションを記述するうえでは、データベースや接続プロトコルによらず、統一的なインタフェースが提供されていることが望まれます。 JDBC データソースでは、これらの要求を満たすための機能を備えた、JDBC コネクションのファクトリインタフェースを提供します。

5.3.1.コネクションプール機能

JDBC データソースは、JDBC Optional Package で規定されているコネクションプール機能を提供します。
JDBC データソースは、JDBC コネクションを解放することなく、内部のプールに管理して再利用します。このため、アプリケーションは、接続にかかるコストを気にすることなく、業務処理を行うたびに、毎回、JDBC コネクションを取得することができます。

JDBC データソースでは、次のとおり、プール管理するコネクションの数を指定することができます。

- 最大プールサイズ［maxPoolSize］
  コネクションプールから同時に払い出されるコネクション数の上限を指定することができます。
- 最小プールサイズ［minPoolSize］
  コネクションプールに常時保持しておくコネクションの数を指定できます。
- 初期プールサイズ［initialPoolSize］
  最初のコネクション払い出しの際に複数のコネクションをまとめてプールすることができます。

また、JDBC データソースでは、次のとおり、プール管理されているコネクションの状態監視を行うことができます。

- 無通信時間の上限［maxIdleTime］
  コネクションが一定時間使用されなかった場合に、そのコネクションを破棄することができます。
- データベースサーバの状態監視オプション［checkServerOption］
  定期的に SQL 命令を発行してデータベースの状態を確認し、異常を検知した場合に、コネクションプール

内のコネクションを破棄することができます。

- コネクションのガベージコレクト機能の動作オプション［checkGarbageOption］
  コネクションのクローズ漏れを検知し、クローズやロールバックといった後処理を行うと同時に、そのコネクションが払い出された場所をログに出力することができます。この機能は、デバッグ用の機能です。

ほかに、次のとおり、プール管理されているコネクションの解放契機を調整することができます。

- コネクション解放までの待ち合わせ時間［shrinkDelayTime］
  minPoolSize プロパティで指定した数を超えて払い出されたコネクションの解放を一定時間遅らせることができます。それにより、多くのコネクションが同時に利用される負荷が高い状況で、コネクションの接続および切断の頻度を下げることができます。

このように、きめ細やかなコネクションプールの管理を行うことができます。

これらの設定に応じて、コネクションの接続および切断契機は次のようになります。

接続契機

initialPoolSize に、0 より大きい数を指定した場合は、指定した数のコネクションが、アプリケーションプロセス起動時に接続されます。ただし、CORBA や単体の Java アプリケーションの場合には、プロセス起動時には接続されません。これらの場合、アプリケーションがコネクション取得 API をプロセス内で最初に呼び出した際に、指定した数のコネクション接続が行われます。
initialPoolSize に 0 を指定した場合は、アプリケーションがコネクション取得 API を呼び出した際に、コネクションプール内に未使用のコネクションがない場合に、コネクション接続が行われます。
また、コネクション接続のための API 呼び出しまたは運用操作を実行することで、maxPoolSize で指定された数まで接続を行うことができます。

切断契機

コネクションのクローズを行った時、または、トランザクションが完了した時に、次のいずれかの条件を満たした場合に切断されます。

- minPoolSize で指定した数を超えて払い出されたコネクションで、shrinkDelayTime で指定した時間が経過した
- 使用されないまま maxIdleTime で指定した時間が経過した
- コネクション一括破棄のための API 呼び出しまたは運用操作を実行した
- checkServerOption の指定により、データベースサーバ側の障害が検出された
- checkGarbageOption の指定により、コネクションのクローズ漏れが検出された
- JDBC ドライバからコネクション障害が通知された
- JDBC データソースのコネクション返却処理で障害が発生した

5.3.2.分散トランザクション連携機能

アプリケーションは、Java Transaction API (JTA)を使用して、2 フェーズコミットに参加することができます。 複数のデータベースにまたがったトランザクションを実行するような場合には、Transaction サービスによって、データの一貫性が保証されます。 特に、EJB のコンテナでトランザクションの管理を行うように設定した場合には、アプリケーションでは、分散トランザクションに参加するための特別な処理を記述する必要はありません。

また、useMultiUsersPerTransaction プロパティを指定することで、接続先のデータベースが同じでユーザアカウントが異なる JDBC コネクションを同一グローバルトランザクション内で同時利用することができます。

5.3.3.JNDI サーバとの連携機能

JDBC データソースを JNDI サーバに登録するための運用管理機能を提供しています。 JDBC データソースの運用管理コマンド、または、WebOTX の統合運用管理コマンドや統合運用管理ツールを使用することで、JDBC データソースのプロパティ情報を登録し、JNDI サーバに登録することができます。一度 JNDI サーバに登録された JDBC データソースの情報は、JNDI サーバを再起動した場合にも保持されます。この機能を使うことによって、アプリケーションは、JNDI API や javax.sql.DataSource インタフェースといった Java 標準のイ

ンタフェースを使用してデータベースにアクセスすることができますので、移植性の高いビジネスロジックを記述することができます。

5.3.4.コネクション制御機能

JDBC データソースでは、defaultAutoCommit プロパティで、コネクションを払いだす際の自動コミットモードのデフォルト値を指定することができます。デフォルトでは、コネクションを払い出す際に、必ず自動コミットとなるように調整されます。アプリケーションで必ずコミットを発行する場合に、デフォルト値を非自動コミットに設定することで、自動コミットモード変更処理のプログラミングを行う必要がなくなります。ただし、JTA のトランザクションを実行する場合は例外で、必ず、非自動コミットになります。

また、JDBC データソースでは、JDBC 3.0 以降の Oracle の JDBC ドライバとバージョン 5.3 の SequeLink を利用する場合に、maxStatements プロパティで、ステートメントのプール数を指定することができます。この指定を行うことで、性能を改善することができます。

そのほか、JDBC データソースでは、コネクションの接続に失敗した場合のリトライ回数(connectRetryMax プロパティ) やリトライ間隔 (connectRetryInterval プロパティ)、初期接続の接続リトライ有無 (reconnectInitialPool プロパティ) を指定することができます。

5.3.5.コネクションの共有範囲

JDBC データソースでは、コネクションの共有範囲を変更することができます。特別な指定を行わない場合、jndiName プロパティが同じ JDBC データソースインスタンスから払い出されたコネクションは、Java VM (※1) 内で共有されます。このため、同じ Java VM 内で動作する複数のアプリケーション間でコネクションを共有することができ、容易に、コネクションの数を最小限に抑えることができます。これに対して、useStaticPool プロパティに false を設定すると、同一の JDBC データソースのインスタンスを使用する範囲内に閉じてコネクションが共有されます。

※ 1:正確には、Java のクラスローダ内での共有となります。同一の Java VM 内であっても、Web コンテナではコンテキスト毎、EJB コンテナでは EJB 毎、アプレットではコードベース毎に、複数のクラスローダが生成されます。

5.3.6.データベースと JDBC ドライバの対応バージョン

JDBC データソースは、次のデータベースと JDBC ドライバをサポートします。

データベース

- Oracle9i Database Release 1 (9.0.1)
- Oracle9i Database Release 2 (9.2.0)
- Oracle Database 10g Release 1 (10.1.0)
- Oracle Database 10g Release 2 (10.2.0)
- Oracle Database 11g Release 1 (11.1.0)
- DB2 Universal Database 8.1.4
- DB2 V9.1
- Microsoft SQL Server 2000
- Microsoft SQL Server 2005
- Sybase Adaptive Server Enterprise 12.5
- SequeLink 5.0
- Cloudscape 3.0.3
- Apache Derby 10.2.2.0
- Apache Derby 10.3.1.4
- PostgreSQL 7.3.2 (JDBC ドライバ 7.3-113) ～

PostgreSQL 8.3 (JDBC ドライバ 8.3dev-601)

(JTA 連携を行う場合はバージョン 8.1.0 以降)

大規模システムや高い性能が要求されるシステムでは、Cloudscape と PostgreSQL 以外のデータベースをお使いになることをお薦めします。PostgreSQL を使用して分散トランザクション連携を行う場合には、PostgreSQL 8.1 とバージョン 8.1-404 ～ PostgreSQL 8.3 とバージョン 8.3dev-601 の JDBC ドライバをお使いください。

JDBCドライバ

データベースに含まれる JDBC ドライバの他に、次の JDBC ドライバをサポートします。

- SequeLink バージョン 5.0 以降

その他の製品についても、JDBC 2.0 または、JDBC 3.0、JDBC 4.0 の仕様に準拠している JDBC ドライバであれば、使用することができます。ただし、十分な評価を行ってください。WebOTX の J2EE の互換性テスト (CTS) では、DataDirect Connect for JDBC バージョン 3.6 を使用した実績があります。そのほか、MySQL Connector /J 5.0 の評価実績があります。

どの JDBC ドライバをご利用になる場合でも、アプリケーションでは、その種別を意識した記述を行う必要はありません。JDBC データソースの定義情報を変更し、JNDI サーバに登録し直すことで、使用する JDBC ドライバを切り替えることができます。

5.4.JMS

JMS とは複数のアプリケーションがメッセージの交換を行うことで通信を可能とするエンタープライジングメッセージシステム(メッセージ指向ミドルウェア)です。JMS では、メッセージ作成、送受信など汎用的な Java インタフェースが提供されています。このインタフェースを利用して、容易にアプリケーションを作成することができます。

WebOTX JMS は、サンマイクロシステムズ社発行の規約「Java(TM) Message Service Version 1.1」に準拠した機能を提供しています。また、JCA 1.5 に準拠したリソースアダプタ機能、JMX 1.2 に対応した運用管理機能をサポートしています。

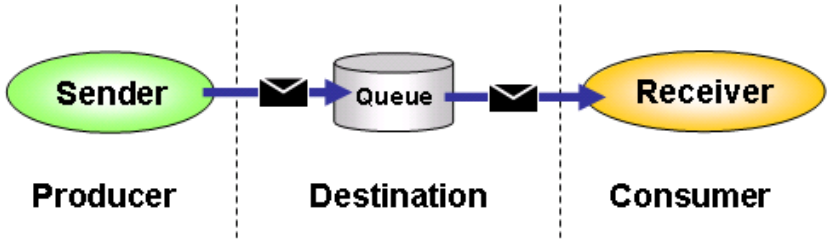
5.4.1.JMSの概要

JMS の概要について説明します。

メッセージングモデル

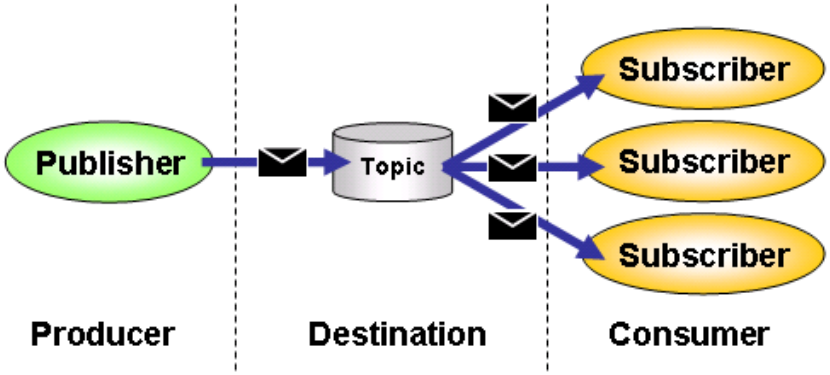
WebOTX JMS では、次の 2 つのメッセージングモデルを提供します。配信対象のコンシューマアプリケーションの数に応じて、お使いになるモデルを選択してください。

ポイントツーポイント(PTP)
1 つのプロデューサアプリケーションが、ただ 1 つのコンシューマアプリケーションにメッセージを送信する 1 対 1 通信のモデルです。デスティネーションオブジェクトを拡張したキューと呼ばれる名前付のオブジェクトを使います。プロデューサはこのキューにメッセージを送信し、コンシューマはこのキューからメッセージを受信します。



パブリッシュ／サブスクライブ(Pub/Sub)
1 つのプロデューサアプリケーションが、複数のコンシューマアプリケーションにメッセージを送信す

る 1 対n通信のモデルです。デスティネーションオブジェクトを拡張したトピックと呼ばれる名前付オブジェクトを使います。ポイントツーポイントとは異なり同じメッセージを複数のコンシューマが受信することができます。



メッセージタイプ

WebOTX JMS では、次の 5 つのメッセージタイプを提供します。 JMS クライアントアプリケーション間で交換したいデータの構成に応じて選択することができます。

BytesMessage
バイト配列のメッセージです。コンシューマは受け取ったバイト配列の内容を解釈できる必要があります。

TextMessage
String 型のメッセージです。

MapMessage
「String オブジェクトの名前」と「Java プログラミング言語のプリミティブデータ型の値」のセットのメッセージです。名前を指定してランダムに値を読み込むことができます。

StreamMessage
Java プログラミング言語のプリミティブデータ型の値を含む Stream のメッセージです。ランダムに値を読み書きすることはできません。プロデューサが書き込んだ順番でコンシューマは読み込む必要があります。

ObjectMessage
Java プログラミング言語（Java オブジェクト）の直列化可能なオブジェクトのメッセージです。

メッセージの永続性モード

WebOTX JMS では、メッセージ配信に関して、次の 2 つの永続性モードを提供します。メッセージの重要性に関して、2 つのモードを選択することができます。

NON_PERSISTENT モード
メッセージは永続ストレージに格納されず、回線障害等の要因でロストし、配信されないことがあります。ただし永続ストレージにアクセスするためのオーバーヘッドが無いので高速なメッセージ配信を行うことができます。

PERSISTENT モード
メッセージは永続ストレージに格納され、信頼性の高い配信を行うことができます。ただし永続ストレージへのメッセージ格納のオーバーヘッドは発生します。

WebOTX JMS では、永続化すべきデータの格納先として、標準で提供しているファイルベースのデータストア、もしくは JDBC 対応データベースを利用することができます。

メッセージの確認応答

PERSISTENT モードの場合、プロデューサから送信されたメッセージは、WebOTX JMS によって永続スト

レージに格納された後、コンシューマに送信されます。コンシューマでは、受信したメッセージに対して acknowledge メソッドを呼び出して、メッセージ受信完了を WebOTX JMS に通知する必要があります。この通知を確認応答と呼びます。

WebOTX JMS では、コンシューマからの確認応答を受けて、該当するメッセージを確認応答受信待ちリストから削除し、メッセージの永続情報を削除します。コンシューマは必要に応じて WebOTX JMS に「確認応答を行っていないメッセージ」の再送要求を行うことができます。

WebOTX JMS では、次の 4 つの確認応答モードを提供します。

AUTO_ACKNOWLEDGE
WebOTX JMS で自動的に確認応答を行います。このモードでは、メッセージを受信するたびに毎回確認応答を発行します。コンシューマが明示的に確認応答を行う必要はありません。ただしメッセージの再送要求を行うことはできません。

CLIENT_ACKNOWLEDGE
コンシューマが WebOTX JMS 提供のインタフェースを呼び出し、確認応答を行います。配信されたメッセージそれぞれに確認応答を行うこともできますが、配信された複数のメッセージに対して一括して確認応答を行うこともできます。また必要に応じてメッセージの再送要求を行うことができます。

DUPS_OK_ACKNOWLEDGE
WebOTX JMS で自動的に確認応答を行います。このモードの動作は JMS の実装に依存しており、WebOTX JMS では、10 件のメッセージを受信すると確認応答を発行します。また必要に応じてメッセージの再送要求を行うことができます。

WebOTX V5 では、このモードは、AUTO_ACKNOWLEDGE と同様の動作（メッセージを受信するたびに毎回確認応答を発行）をしていました。

NO_ACKNOWLEDGE
確認応答が不要な WebOTX JMS 独自の拡張モードです。確認応答のための通信が行われないため高速な配信が行えますが信頼性は低下します。メッセージの再送要求も行うことはできません。

メッセージの同期受信と非同期受信

WebOTX JMS ではコンシューマがメッセージを受信するときのモードとして次の 2 つを提供します。

同期受信
メッセージが配信されるまでコンシューマアプリケーションはブロックされます。受信すべきメッセージがない場合、メッセージなしでブロックが解けるように待ち合わせることもできます。

非同期受信
メッセージが配信されている、いないにかかわらずコンシューマアプリケーションはブロックされません。ただし、配信されたメッセージを受け取るためのリスナを登録する必要があります。メッセージが配信されるとこのリスナに通知され、コンシューマアプリケーションはメッセージを受信することができます。

コンシューマがメッセージ受信を待ち合わせている間に回線障害など、システムに障害が発生した場合、待ち合わせしているスレッドに異常が通知されない場合があります。WebOTX JMS ではシステムに障害が発生した場合にアプリケーションに異常を通知する機能を提供しています。コンシューマは、メッセージを受信するために使うコネクションオブジェクトごとに例外リスナを登録することで該当するコネクションに障害が発生した場合、例外通知を受けることができます。この例外通知を受け取って必要な復旧処理を行うことができます。

トランザクション

JMS でのトランザクションは、プロデューサからのメッセージ送信、コンシューマでのメッセージ受信を最小単位としています。

プロデューサからのコミット要求でコンシューマへのメッセージ配信が開始され、コンシューマからのコミット要求で JMS サーバ内のメッセージが削除されます。また、プロデューサからロールバックが要求されると送信したメッセージは破棄され、受信したメッセージは JMS サーバ内で保持されます。ロールバックされたメッセージは、トピックの場合は同じサブスクライバに対して再送されますが、キューの場合はどのレシーバに再送されるかは規定されていません。

WebOTX JMS ではトランザクションを制御する方法として以下の 2 つの方法を提供します。1 つ目は、JMS のローカルトランザクションを使用する方法で、2 つ目は分散トランザクションを使用する方法です。

それぞれの方法について以下に説明します。

ローカルトランザクション
JMS でローカルトランザクションを使用する場合は、Session オブジェクト生成時に、transacted パラメータに true を指定して、トランザクションセッションを生成する必要があります。トランザクションは最初にメッセージを送受信した時点で自動的に開始され、以降はコミット／ロールバックした時点で自動的に次のトランザクションを開始します。

分散トランザクション
JMS を分散トランザクションに参加させるには、JMS が提供するリソースアダプタを使用する必要があります。また、2 フェーズコミットメントを使用する場合は、使用するリソースアダプタをトランザクションサービスの JCA リソースとして登録する必要があります。

トランザクションセッションでは WebOTX JMS がトランザクション処理を行うため独自の確認応答の制御を行います。したがってクライアントアプリケーションはメッセージの確認応答の制御を行うことはできません。確認応答モードの設定は無視されます。

トランザクションセッションの使い方については「アプリケーション開発ガイド（JMS） JMSのトランザクション機能を使用したプログラム」を参照してください。

メッセージのフィルタリング

WebOTX JMS のそれぞれのメッセージには付加情報として、メッセージを識別したりするメッセージの属性を表す標準のヘッダフィールドと、アプリケーション固有の情報を含めることができるプロパティフィールドがあります。

コンシューマでは受け取ったメッセージのヘッダフィールド、プロパティフィールドに格納されている情報をもとにメッセージのフィルタリングを行い必要なメッセージだけを受け取ることができます。フィルタリングにはセレクタと呼ばれる単純な構文の文字列を使います。

フィルタリングを使って不要なメッセージの配信を抑制することでネットワークへの負荷を軽減する効果も得られます。

メッセージのフィルタリングの使い方については「アプリケーション開発ガイド（JMS） メッセージのフィルタリングを使用したプログラム」を参照してください。

5.4.2.メッセージのスケジューリング

WebOTX JMS では、メッセージの配信時間をスケジューリングする独自の拡張機能を提供しています。プロデューサは、将来のある特定の時間にメッセージをコンシューマに配信することができます。

また、WebOTX JMS では「確認応答を行っていないメッセージ」の再配信時間を遅延させる機能を提供しています。コンシューマは障害などでメッセージを受信する準備ができていないときにメッセージの再配信を遅延させ、その間にメッセージを受信するための準備を行うことができます。

メッセージのスケジューリングの使い方については「リファレンスマニュアル（JMS） 拡張インタフェース」を参照してください。

WebOTX JMS V6 以降では、V5 で規定した拡張インタフェースの API 仕様を変更しています。V5 で作成したアプリケーションは変更する必要があります。

5.4.3.メッセージフロー制御

メッセージが滞留するような状況（高負荷、コンシューマ遅延、コンシューマ不在）になると、メモリ不足によりパフォーマンス低下やプロセスダウンなどの問題を引き起こす可能性があります。WebOTX JMS では、このような問題を未然に防ぐために、次のようなメッセージフロー制御に対応しています。

JMSサーバでの滞留抑止

サーバ単位での制御

すべての送信先で滞留するメッセージを監視して、JMS サーバ内での滞留を抑止することができます。
サーバ全体で滞留可能なメッセージの最大数と総メモリ量を設定して制御します。
この他に、1 つのメッセージで許容するサイズを制限する機能や、滞留しているメッセージの有効期限を

チェックする間隔を設定して定期的にメモリを回復させる機能も提供します。

送信先レベルでの制御

送信先に滞留するメッセージを監視して、JMS サーバ内での滞留を抑止することができます。送信先単位で、滞留可能なメッセージの最大数、総メモリ量、そしてその制限到達時の振る舞いを設定して制御します。制限到達時の振る舞いには、以下のものがあります。

FLOW_CONTROL : プロデューサからの送信をスローダウンします。
REMOVE_OLDEST : 滞留中の最古のメッセージを破棄します。
REMOVE_LOW_PRIORITY: 滞留中の最低プライオリティのメッセージを破棄します。
REJECT_NEWEST : メッセージの受付を拒否します。

これらの振る舞いは、WebOTX V5 の送信先生成時に指定した「メッセージ廃棄ポリシ」に相当しますが、WebOTX V5 の LIFO に対応する振る舞いはサポートしていません。

メモリ監視による制御

メモリ使用量を監視して、4 つの状態（green：使用可能なメモリが十分にある、yellow：JMS サーバのメモリが減っている、orange：JMS サーバのメモリが不十分である、red：JMS サーバのメモリが不足している）に応じたメモリ回復のための制御として、GC (Garbage Collector)の実行、メッセージの受付抑止、JMS クライアントの接続拒否を行います。状態毎のメモリ使用率、メッセージフローカウントを設定して制御します。
それぞれの状態に遷移した際に行われる制御は次のとおりです。

green 指定されたメッセージフローカウントに到達すると受付を抑止します。
GC は行いません。

yellow 指定されたメッセージフローカウントに到達すると受付を抑止します。
GC を 1 回行います。

orange 指定されたメッセージフローカウントに到達すると受付を抑止します。
GC を 5 回行います

red 指定されたメッセージフローカウントに到達すると受付を抑止します。
GC を 10 回行います。

JMSクライアントでの滞留抑止

コネクションレベルでの制御

コンシューマでのメッセージ処理が遅延すると、JMS クライアントに配信され続けるメッセージが JMS クライアント内に滞留することになります。これを抑止するためには、コネクション上のすべてのコンシューマ群での滞留可能なメッセージ数の制限値をコネクションファクトリに設定して制御します。JMS サーバは、この制限値に到達するとメッセージの配信を停止し、下回ると配信を再開します。

コンシューマレベルでの制御

個々のコンシューマ単位に、滞留可能なメッセージ数の制限値と配信を再開するポイントをコネクションファクトリに設定して制御します。JMS サーバは、この制限値に到達するとメッセージの配信を停止し、配信再開値を下回ると配信を再開します。
この制御は、マルチコンシューマ間の負荷分散や、混在するコンシューマ間で相互に影響を与えないようにするために有効です。

その他のフロー制御

制御メッセージのフロー促進

JMS サーバとクライアントの間では、メッセージの送受信だけでなく、確認通知のような数多くの制御データもやり取りされています。そのため、JMS サーバからクライアントに対するメッセージ配信が連続するような高負荷が続くと、確認通知が遅延することによってメッセージの滞留につながる場合があります。
この問題を回避するためには、メッセージを連続して配信する数の制限値をコネクションファクトリに設定して制御します。JMS サーバは、この制限値に到達すると一時的にメッセージの配信を停止して制御データのフローを促進します。
制御データを数多く使用するのは、次のような機能を利用する場合です。

CLIENT_ACKNOWLEDGE、AUTO_ACKNOWLEDGE モード

メッセージの永続化

トランザクション

キューブラウザ

Broker への接続／切断繰り返し

5.4.4.マルチコンシューマ負荷分散

WebOTX JMS では、接続している複数のコンシューマへの負荷分散配信機能を提供します。

Queueの場合

負荷分散の対象とするアクティブなコンシューマの数とそのバックアップとなるコンシューマの数を送信先に設定すると次のように制御します。

- アクティブなコンシューマの数とそのバックアップとなるコンシューマの数の合計値を超えるコンシューマの接続は拒否します。
- 当該 Queue に接続した順序で、設定した最大数個分のアクティブなコンシューマに対してメッセージを 1 個以上バッチ的に順次配信します。その数はメッセージの滞留状況に応じて変化しますが、コンシューマで許される滞留数を超えることはありません。
- アクティブなコンシューマへの配信が失敗した場合、バックアップの先頭のものをアクティブなコンシューマとして扱います。

WebOTX V5 では Queue に接続している複数のコンシューマへの配信は不定でした。

Topicの場合

Standard/Enterprise Edition において、Message-Driven Bean(MDB)や JMS の非同期受信を行う COBRA アプリケーションを TP モニタ上で起動する場合、その多重度の設定によっては複数のコンシューマが動作します。WebOTX JMS では、自動的に複数のコンシューマをひとつのグループとして扱い、Topic からの配信を同じグループの中のひとつのコンシューマに対してだけ行うように制御します。

グルーピングするには、次の情報を一致させる必要があります。

- クライアント識別子
- サブスクリプション識別子（DurableSubscriber の場合）
- セレクタ

クライアント識別子は、次の方法で設定することができます。

- コネクションファクトリリソースの属性に設定（統合運用管理ツール、運用管理コマンド）
- プログラム上で`Connection.setClientID()`メソッドで設定　※CORBA アプリケーションのみ
- 配備記述子で設定（配備ツール）　※MDB のみ

CORBA アプリケーションの場合は、明示的に設定しなくても、デフォルトでアプリケーショングループ名_プロセスグループ名が設定されます。MDB の場合は、クライアント識別子の共有を許可するフラグと共に必ず明示的に設定する必要があります。

5.4.5.セキュリティ

WebOTX JMS では、ユーザ認証、アクセス制御、メッセージの暗号化の 3 つのセキュリティ機能を提供します。

ユーザ認証

JMS クライアントからのコネクション確立時に、指定したユーザ名とパスワードをあらかじめ定義されたユーザ管理情報と照合して認証を行います。

アクセス制御

JMS クライアントからの操作（コネクション、送信先、送信先自動作成）に対して、あらかじめ定義されたアクセス制御情報を照合して承認を行います。

メッセージの暗号化

Java Secure Socket Extension (JSSE)に基づいた、サーバの自己署名型証明書によるメッセージの SSL 暗号化に対応しています。メッセージを暗号化することで、機密性の高い重要な情報が外部に漏れないようにすることができます。

5.4.6.データストア

WebOTX JMS では、永続化すべきデータの格納先として、標準で提供しているファイルベースのデータストア、もしくは JDBC 対応データベースを利用することができます。

データストアには、次のような情報が格納されます。

- メッセージ
- 送信先
- 永続サブスクリプション
- トランザクション

5.4.7.コネクション制御スレッドのプール管理

WebOTX JMS では、コネクションサービスで JMS クライアントと JMS サーバとの間で確立される TCP コネクションを管理しています。これらのコネクションに必要なスレッドは、プール管理して有効に利用できるようになっています。プール管理はスレッドの最小数と最大数の 2 つを指定して制御します。

コネクションサービスは、サービスタイプとプロトコルタイプ別に 4 種類 (jms、admin、ssljms、ssladmin) 存在します。

コネクションを確立してスレッドが必要になると、確保したスレッドをスレッドプールに順次追加していきます。最小数を超えると、コネクションを解放するタイミングに不要になったスレッドを、最小数を下回らない範囲で解放します。一時的な高負荷で最大数に到達した場合、それ以上の新たなスレッドは確保せずに、スレッドの空きができるまで待ち合わせます。

また、プール管理されたスレッドリソースを、TCP コネクションとどのように対応づけるかを管理モデルとして選択できます。

- dedicated ： スレッド専用モデル
  1 つのコネクションに対して、受信用と送信用の 2 つのスレッドを固定的に割り付ける方式です。高いパフォーマンスを得ることができますが、実際に接続できるコネクションの数は最大数の半分になります。
- shared ： スレッド共有モデル
  1 つのコネクションに対して、固定的にスレッドを割り当てず、実際に送受信処理が必要になった時点でスレッドを動的に割り当てる方式です。動的な割り当てを行う監視スレッドは、監視対象コネクション数のプロパティに指定された複数のコネクションの入出力事象を監視します。

5.4.8.送信先の自動生成

WebOTX JMS では、送信先の自動生成をサポートしています。環境を変えながらの開発を柔軟に行うような場合に有効です。

この機能を利用する場合には、あらかじめアクセス制御で許可しておく必要があります。

5.4.9.再配信の遅延と回数制限

ロールバックされたメッセージの再配信する時間を遅延(V5 からの機能)させる機能に加えて、再配信を有限回に制限することができる機能です。

コンシューマが処理できないメッセージ自体の問題が原因で、再配信が延々と繰り返し行われるような状況に陥り、システム全体のスループットが低下してしまう事態を未然に防ぐような場合に利用します。

制限値に到達したメッセージは破棄されますが、破棄せずに別な送信先に蓄積することもできます。(破棄メッセージの転送機能)再配信の遅延時間と回数は、JMS サーバのプロパティで設定するか、JMS サービスの属性として設定してください。

遅延の設定については、WebOTX V5 よりサポートしているコンシューマプログラム向けに用意している API で行います。また、Message-Driven Bean(MDB)では、コネクションファクトリの属性か配備記述子で

設定を行います。

5.4.10.破棄メッセージの転送機能

再配信の回数制限値を超過したメッセージや、有効期限に到達した未配信のメッセージはデフォルトでは破棄されますが、これらをあらかじめ用意した送信先に転送することができる機能です。

この送信先からメッセージを取得するプログラムを作成し、ロギングやリカバリを行う処理を組み込むことが可能になります。

転送先は、再配信超過のものと有効期限切れのものがあり、それぞれ、JMS サーバのプロパティで設定するか、JMS サービスの属性として設定してください。

5.4.11.サーバクライアント間の相互監視

WebOTX JMS は、サーバとクライアント間の相互通信に TCP/IP プロトコルを利用していますが、データ受信を主体としている場合、ネットワークの異常を検知できないような事象（簡単な例だと、相手側のケーブルが抜けたような場合）が発生する場合があり、次のような問題を引き起こす可能性があります。

- JMS サーバが、クライアントの異常を検知できずに、クライアントに関するリソースがパージできない。
- JMS クライアントが、サーバの異常を検知できずに、メッセージ配信を待ち続けてしまい、JMS サーバのクラスタ切り替えに対応できない。

このような問題を防ぐために、クライアントアプリケーションのライブラリレベルで、JMS サーバとの接続を定期的にチェックする機能を提供しています。この機能はデフォルトで動作しますが、監視間隔は変更することができます。

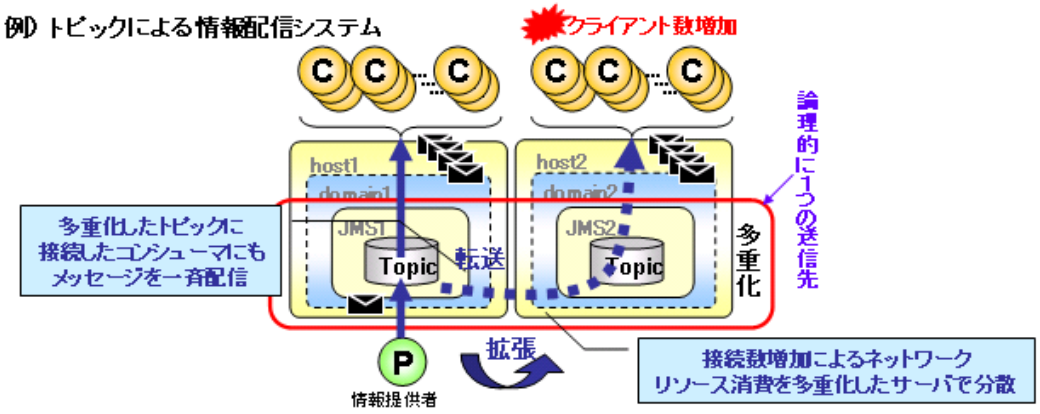
5.4.12.JMSサーバクラスタ

JMS サーバの多重化を行うための機能です。JMS クライアントの接続数や、送受信メッセージ数の増加に応じて、JMS 利用システムを拡張することが可能になります。また、Message-Driven Bean(MDB)利用アプリケーションや、ESB によるメッセージ処理の負荷分散や異常時のサービス継続を実現します。

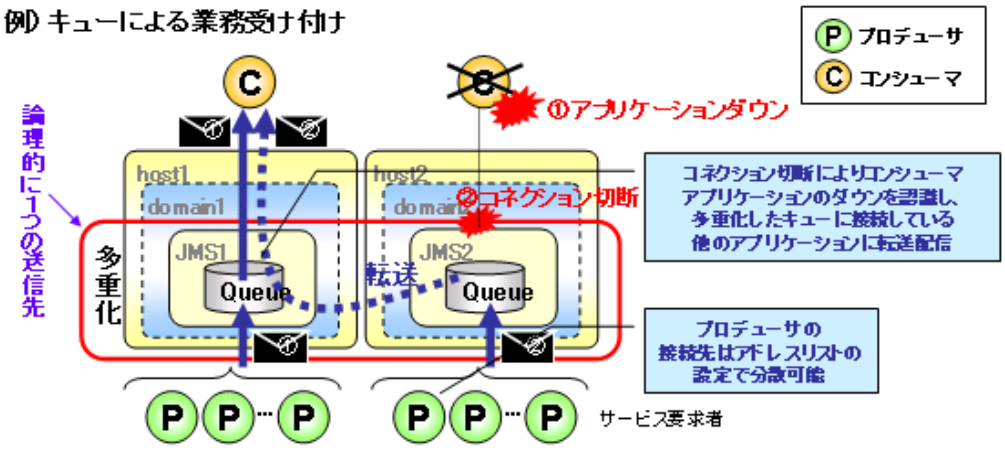
例えば、

- トピックによる情報配信を行うシステムで、クライアント数の増加によるネットワークリソース消費を多重化したサーバに分散
- キューによる業務受付を行うシステムで、コネクション切断により、コンシューマアプリケーションのダウンを検出し、多重化したキューに接続している他のアプリケーションに転送

といった利用法が考えられます。



V7.1 では、ドメイン単位に 1 つの JMS サーバを起動しますので、JMS サーバの多重化を行う場合は、多重化する数だけドメインが必要になります。



クラスタを構成する JMS サーバは、他のすべての JMS サーバと相互に接続して、送信先の生成／削除、コンシューマの登録／削除といった情報を交換します。

クライアントは、コネクションファクトリの「アドレスリスト」を利用して、クラスタ内の 1 つの JMS サーバと直接通信し、その JMS サーバが唯一の JMS サーバであるかのようにメッセージの送受信を行います。実際には、クライアントが接続している JMS サーバが他の JMS サーバと協調動作し、接続中のすべてのクライアントに対してメッセージ配信サービスを提供します。

クライアントの接続先は、JMS クライアントランタイムが、コネクションファクトリに設定された「アドレスリスト」から、プライオリティ(リストに記述された順番)、あるいは、ランダムに決定します。これにより、複数の JMS サーバ間でクライアントコネクションを分散させることが可能となります。

以降、JMS サーバクラスタ構成でのメッセージの送受信や、JMS サーバ間での情報の送受信の動作について説明します。

JMSサーバクラスタ内のメッセージ送受信

JMS サーバクラスタ構成では、各 JMS サーバが送信先とコンシューマに関する以下の情報を共有しています。

- クラスタ内のすべての物理的な送信先の名前、タイプ、および属性
- 各コンシューマの名前、場所、および配信対象メッセージ
- 送信先や、コンシューマに対する更新（削除、追加、または再設定）

これらの情報によって、各 JMS サーバは、自身に接続しているプロデューサからのメッセージをリモートのコンシューマに配信することができるようになります。

一方、クライアントは、1 つの JMS サーバに接続し、それがクラスタ内の唯一の JMS サーバであるかのようにメッセージの送受信を行います。このクライアントが接続した JMS サーバを「ホームブローカ」と呼びます。

上記の通り、各 JMS サーバはコンシューマの情報は共有しますが、プロデューサの情報は共有しません(プロデューサが追加されても、その情報はクラスタ内には伝播しません)。プロデューサから送信されたメッセージは、ホームブローカ上の送信先にのみ存在し、ホームブローカは、ローカル、リモートにかかわらず、受信可能なコンシューマが存在すればメッセージを配信するようになっています。

クライアントの接続先は、一度接続すると以降は固定で、動的に変更されません。

JMSサーバクラスタ内の送信先

物理的な送信先は、以下のような作成方法の違いによって、クラスタ内での伝播情報が異なります。

- 運用管理コマンドや、統合運用管理ツールから作成した送信先
- 自動作成された送信先
- 一時送信先

次に、伝播するものの一覧と、その範囲を示します。

送信先種別	JMS サーバクラスタ内での情報伝播	
運用管理コマンド／統合運用管理ツールで作成	作成	伝播する。
	削除	伝播する。
自動作成	作成	**プロデューサが接続する送信先が存在しない場合**: 伝播しない。プロデューサのホームブローカ上にのみ送信先が作成される。 **コンシューマが接続する送信先が存在しない場合**: 伝播する。
	削除	**管理コマンドでの削除**: 伝播する。 **自動削除**: 伝播しない。自動作成された送信先は、以下の場合には、JMS サーバ毎に削除される。 ・ 一定時間、コンシューマまたはメッセージがない場合 ・ JMS サーバの再起動時に、その送信先にメッセージがない場合
一時	作成	伝播する。
	削除	伝播する。

送信先の属性

送信先の属性は、クラスタ内の JMS サーバに伝播するものとしないものがあります。次に、その一覧を示します。

server.jms-service.jms-physical-destination.*physical-destination-name* :

属性名		JMS サーバクラスタ内での情報伝播
コンシューマへのフロー制限数	consumerFlowLimitNum	伝播する
制限到達時の振る舞い	limitBehavior	伝播する
アクティブコンシューマの最大数	maxNumActiveConsumers	伝播する
バックアップコンシューマの最大数	maxNumBackupConsumers	伝播する
ローカル配信のみ	isLocalOnly	伝播する
ローカル配信優先	localDeliveryPreferred	伝播する
再配信時の順序保証	supportOrderedRedelivery	伝播する
1 メッセージ当たりの最大サイズ	maxBytesPerMsg	伝播しない
メッセージの最大数	maxNumMsgs	伝播しない
プロデューサの最大数	maxNumProducers	伝播しない
メッセージの最大合計サイズ	maxTotalMsgBytes	伝播しない

JMSサーバクラスタの同期化

JMS サーバクラスタ環境では、以下の情報が、すべての起動中の JMS サーバに即座に伝播するようになっています。

- クラスタ内のすべての物理的な送信先の名前、タイプ、および属性

- 各コンシューマの名前、場所、および配信対象メッセージ
- 送信先や、コンシューマに対する更新（削除、追加、または再設定）

しかし、障害などで停止していた JMS サーバはこれらの変更通知を受け取ることができません。

この問題に対処するため、クラスタ内に、変更情報を記録するための JMS サーバとして、「マスターブローカ」を設定することができます。再起動する JMS サーバは、マスターブローカが記録した変更情報を参照して、必要な更新を行ってから起動するた設定情報の同期を取ることが可能です。

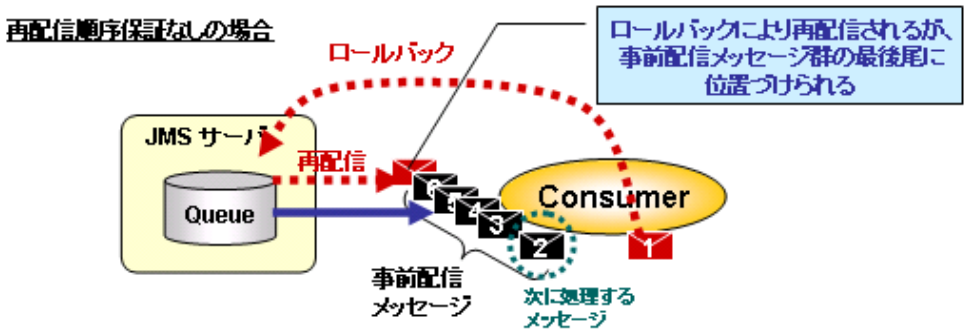
マスターブローカは、JMS サーバクラスタ内で 1 つ設定します。マスターブローカを設定した場合、クラスタ内の JMS サーバはマスターブローカである JMS サーバの起動を待ち合わせます。また、マスターブローカに障害が発生した場合、以下の操作は行うことができません。それ以外のメッセージ送受信などの処理はそのまま継続できます。

- 物理的な送信先の生成／削除／プロパティ更新
- 永続サブスクリプションの削除

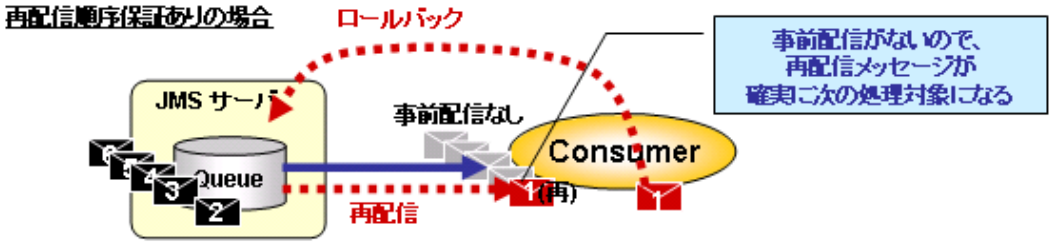
マスターブローカを設定しなかった場合、障害発生などで停止している JMS サーバには、設定変更を反映することができません。

5.4.13.再配信メッセージの順序保証

JMSサーバは、効率よくメッセージを配信するために、「5.4.3メッセージフロー制御」で説明した機能により、メッセージをあらかじめコンシューマに事前配信しています。そのため、トランザクションのロールバックによってメッセージの再配信が行われた場合、そのメッセージは事前配信されたメッセージ群の最後尾に位置づけられてしまい、配信順序が狂ってしまいます。



このように、トランザクションのロールバックが発生した場合でも配信順序を保証するための機能が「再配信メッセージの順序保証」です。フロー制御の設定は無効になり、メッセージの受信が確定するまでは、事前配信を行わないようにします。設定は、物理的な送信先単位に行います。



再配信メッセージの順序保証が利用できるのは、次の環境に制限されます。

- 送信先タイプはキューであること
  JMS サーバクラスタ構成では、送信先の「ローカル配信のみ（isLocalOnly）」の設定を有効（true）にして、ローカルコンシューマ（送信先が作成されたブローカに接続しているコンシューマ）だけにメッセージを配信するようにおく必要があります。
- コンシューマのメッセージ受信はトランザクション内で行われること
- 送信先に接続するコンシューマ数は 1 であること
  送信先に接続できるコンシューマ数は、1 に制限されます。また、Message-Driven Bean (MDB)の場合は、多重度を 1 に制限しておく必要があります。

再配信メッセージの順序保証は、WebOTX V7.11.02 からのサポートです。

【参考】属性値について

説明中の各属性の詳細については、「運用編(コンフィグレーション)」-「JMS に関する設定」を参照してください。

5.5.JTA

JTA(Java Transaction API)とは、アプリケーションやアプリケーションサーバからトランザクションの開始や完了といったトランザクションに対する操作を行うための API です。

トランザクションを実行するためには、アプリケーション、DBMS に代表されるリソース、アプリケーションサーバが協調動作する必要があります。JTA は、それらの各種コンポーネントが協調して動作するための仲介者となります。

WebOTX では最新仕様 1.0.1B に対応しているため、WebOTX 上で動作するアプリケーションでは、リソースの種類や Transaction サービスの実装などを意識せずにトランザクションに参加することができます。

また WebOTX では、JTA と Transaction サービス間のインタフェースを規定する JTS(Java Transaction Service)に対応しています。J2EE で標準となっている JTA インタフェースと CORBA 共通サービスで規定されている Object Transaction Service(OTS)との連携を実現します。

WebOTXが提供するTransactionサービスでは、JTA、およびOTSを利用したトランザクションの管理を一手に行なっています。Transactionサービスの提供する機能、およびOTSについての詳細については、「CORBAサービス」内の 「トランザクションサービス」の章をご参照ください。

5.6.CORBAサービス

ここでは、CORBA アプリケーションから利用できる部品であるサービスについて説明します。

5.6.1.名前サービス

名前サービスは、OMG が公布する「Naming Service Specification 1.1 (formal/2001-02-65)」を実装しています。CORBA では通信相手であるオブジェクトを識別するためにオブジェクトリファレンスを使用します。クライアントは通信をする前に通信相手のオブジェクトリファレンスを入手しなければなりません。オブジェクトリファレンスを受け渡す方法にはいくつかありますが、もっとも一般的な方法は名前サービスを使用することです。オブジェクトリファレンスとその名前をあらかじめ名前サービスに登録しておき、クライアントは名前をもとにオブジェクトリファレンスを入手します。

5.6.2.インタフェースリポジトリ

インタフェースリポジトリは厳密には CORBA Service ではありませんが、CORBA アプリケーションが他のサービス同様に利用できるためここにあげています。CORBA 2.3 仕様を実装しています。実行時にオブジェクトのインタフェース情報を取得するために使用します。

スタブやスケルトンを使用する通常のアプリケーションでは必要ありません。主に DII や DSI などの利用時に使用します。データの登録は IDL コンパイラが生成するファイルを用いて行います。

5.6.3.セキュリティサービス

セキュリティを実現するサービスとして、次の IIOP over SSL、CSIv2 が提供されています。

IIOP over SSL

Security Service 1.2 仕様にしたがっています。IIOP 通信で使用する通常の TCP/IP ソケットの代わりに SSL を使用する通信方式です。通信データが SSL により保護されます。IIOP over SSL で通信を行うためには、SSL で使用する「証明書」が必要です。

CSIv2

セキュリティ認証情報を IIOP プロトコルで伝播するために CORBA 2.6 で規定された仕様です。

5.6.4.イベントサービス

「CORBA Service (98-12-09)」にしたがっています。イベントサービスは、イベントチャネルオブジェクトを媒介として、any 型のデータを 1 イベントとする非同期通信を提供します。データの供給側(サプライア)、データの消費側(コンシューマ)とも、プル型とプッシュ型のデータ交換が可能です。プッシュ型では、供給者がクライアントとなり、消費側オブジェクトを呼び出します。プル型では、消費者がクライアントとなり、供給側オブジェクトを呼び出します。なお、イベントチャネルを介して通信しますので、供給者、消費者でプッシュ型、プル型を混在させて使用することができます。

5.6.5.ノティフィケーションサービス

Notification Service 1.0 仕様にしたがっています。ノティフィケーションサービスは、非同期通信を実現するイベントサービスを拡張したサービスです。イベントサービスが提供する機能のほかに、サービス品質の設定やフィルタリングといった機能を提供します。

サービス品質の設定では、優先度が高い順にコンシューマにイベントを送信することや、イベントチャネルにキューイングされてから一定の期間が経過したイベントを自動的に廃棄すること、コネクション情報とイベントメッセージの永続性を保証して、システム障害などでイベントを失わないようにすることができます。また、イベントのフィルタリングを行うことで、コンシューマに送信するイベントの種類を限定することができます。

詳細は、WebOTX Notification Service に関する記述をごらんください。

5.6.6.トランザクションサービス

Transaction サービスは CORBA 共通サービスの 1 つである Object Transaction Service(OTS) 1.4 仕様に準拠しており、WebOTX のアプリケーションに対してトランザクションの原子性、一貫性、独立性、永続性を提供します。これにより、アプリケーションは分散されたデータベースなどの複数のリソースに対する更新を安全に行うことができます。また JTA の章でも記載したとおり、WebOTX が提供する Transaction サービスでは J2EE 仕様の JTA インタフェースと OTS との連携を規定する JTS にも対応しており、Transaction サービスで JTA、および OTS を利用したトランザクションの管理を一手に行なっています。

また、ACOS とオープンサーバのリアルタイム連携を実現するためのランタイムライブラリ「ACOS Access Toolkit」が提供する JDBC ドライバと連携することで、本来は 2 フェーズコミットメントをサポートしていない ACOS 上のトランザクションを WebOTX が管理する 2 フェーズコミットメントトランザクションに参加させることができるように強化を図っています。概要については、後述する「ACOS 上トランザクションの 2 フェーズコミットメント参加」、詳細については「運用編(運用操作)」をご参照ください。

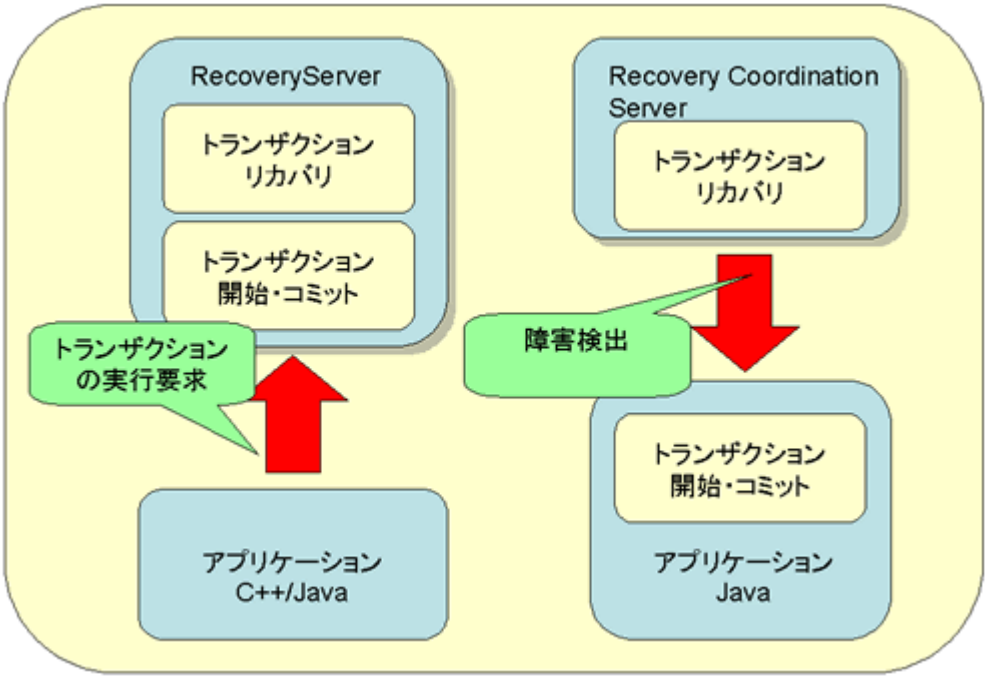
提供機能

トランザクションサービスの役割

WebOTX Transaction Service では従来の Recovery Server によるトランザクションに加え、Recovery Coordination Server を導入して、トランザクションを高速に管理する機能を提供しています。従来の Recovery Server は CORBA Transaction Service の仕様に準拠しており、高い相互接続性があります。しかし、トランザクションの開始・終了時にアプリケーションと Recovery Server との間で CORBA Transaction Service が規定する通信が必要でした。Recovery Coordination Server では、トランザクションの開始や終了処理をアプリケーションプロセス内で実行し、大幅な性能の改善を図りました。ただし、CORBA Transaction Service 仕様を一部拡張しているため、相互接続性はありません。

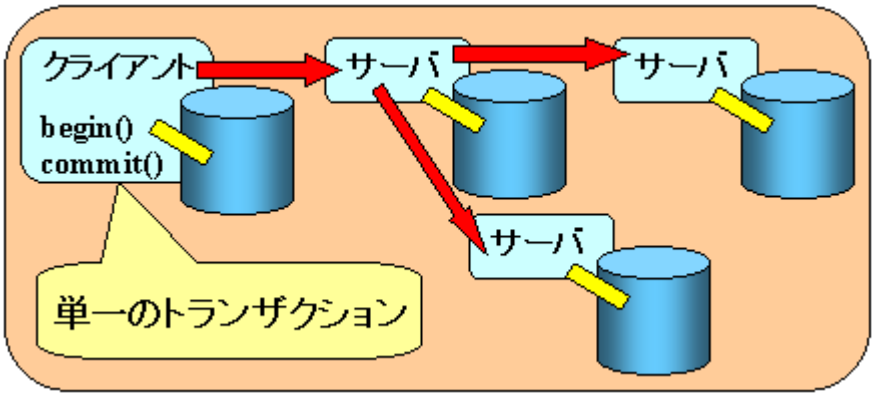
V7.1 では、従来別プロセスとして動作させていた Recovery Coordination Server をドメイン管理プロセス内で動作させるようにしました。これにより、メモリ使用量の削減やドメイン起動時間の短縮が図られています。

WebOTX 上で動作するアプリケーションは堅牢なプロセスになるため、サーバトランザクションを実行する際に Recovery Coordination Server を使用するのが最も適した形になります。



フラットトランザクションとネスティッドトランザクション

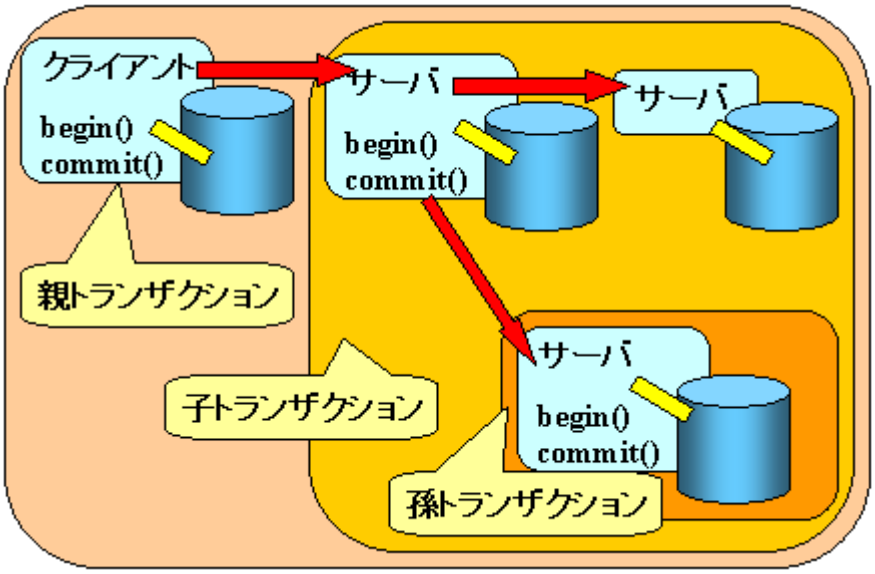
フラットトランザクションは、ネットワーク全体で単一のトランザクションとしてリソースの コミット、ロールバックを行う単純な形態です。そのため、あるアプリケーションで障害が発生した場合はトランザクション全体がロールバックしてしまいますが、トランザクションの 一貫性(consistency)を保つという点ではこの形態を使用するのが一般的です。



ネスティッドトランザクションは、全体のトランザクションを多段の サブトランザクションにツリー上に分割して実行する形態です。サブトランザクション内のリソースに障害があってロールバックされても、全体のトランザクションには影響を与えないようにすることができます。一方、サブトランザクションのコミットはトップレベルのトランザクションがコミットされるまで実行されません。

Transaction サービスの提供するネスティッドトランザクションは CORBA トランザクションサービスに完全に準拠していますが、仕様自身に相互接続性がありません。したがって WebOTX の Transaction サービスと、それ以外の CORBA トランザクションサービスとの間でサブトランザクションを実行できるという保証はありませんので注意してください。

ネスティッドトランザクションは Recovery Server のみのサポートです。Recovery Coordination Server はサポートしておりません。



トランザクションコンテキストの管理方法

Transactionサービスが提供するトランザクションサービスでは、トランザクションの管理 のためにトランザクションコンテキストを使います。

トランザクションコンテキストは、トランザクションに参加しているアプリケーションやリソース、トランザクションの状態といったトランザクションに関する情報をもっています。Transaction サービスあるいはアプリケーションプログラムは、トランザクションコンテキスト内の情報を変更/参照することにより、トランザクションの制御を行います。例えば、アプリケーションをトランザクションに参加させるには、アプリケーションをトランザクションコンテキストに関連付けることで実現します。

アプリケーションがトランザクションコンテキストを管理する方法には次の 2 種類があります。

- 間接的なコンテキスト管理
- 直接的なコンテキスト管理

間接的なコンテキスト管理とは、アプリケーションとトランザクションコンテキストの関連付けを Transaction サービスが自動的に管理します。アプリケーションは Transaction サービスが提供するインタフェースを使用するだけでトランザクションの制御を行うことができます。

直接的なコンテキスト管理とは、アプリケーションがトランザクションコンテキストを直接参照したり、変更したりする方法です。直接的なコンテキスト管理を使うと、アプリケーションとトランザクションコンテキストの関連付けはアプリケーションの責任になりますが、きめ細かなトランザクション制御を行うことができます。

これらのコンテキスト管理を、同時に使用することもできます。詳細は「アプリケーション開発ガイド」の第 4.4 章を参照してください。

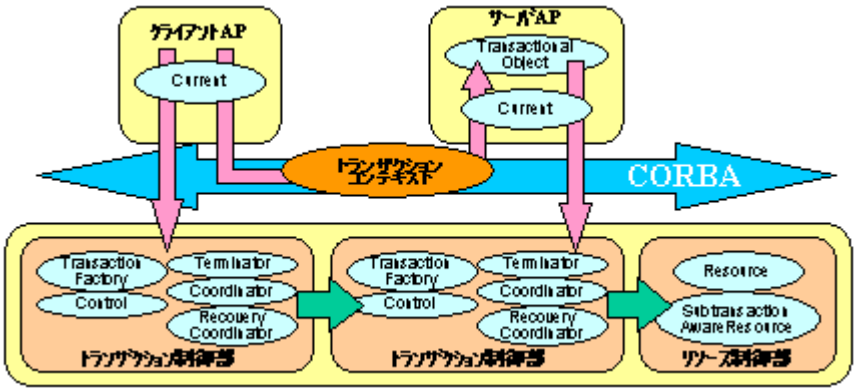
トランザクションコンテキストの伝播

トランザクションコンテキストを伝達することで、クライアントアプリケーションが呼び出したサーバオブジェクトを、クライアントが開始したトランザクションに参加させることができます。サーバオブジェクトは、伝達されたトランザクションコンテキストを使ってトランザクションに参加し、トランザクションの制御を行うことができるようになります。このトランザクションコンテキストの伝達を**伝播**といいます。伝播には次の 2 種類があります。

- 暗黙的な伝播
- 明示的な伝播

暗黙的な伝播は、Transactionサービスが伝播を行います。間接的なコンテキスト管理によりアプリケーションのスレッドにトランザクションコンテキストが関連付けられているときにサーバオブジェクトが呼び出されるとトランザクションサービスは暗黙の伝播を行います。アプリケーションは 伝播のために特別な処理を行う必要はありません。伝播が行われるかどうかは、サーバオブジェクトのOTSPolicyによって決まります。

明示的な伝播は、アプリケーションが伝播を行います。アプリケーションがサーバオブジェクトのオペレーションを呼び出す時、引数にトランザクションコンテキストを指定することで行います。



Transaction サービスで提供するインタフェース。

詳細は API リファレンス編をご参照ください。

OTSPolicyによる伝播の管理

WebOTX では、CORBA の Object Transaction Service(OTS)の最新仕様である 1.4 に対応しています。OTS 1.2/1.3 と同様に OTSPolicy を設定することにより、トランザクションとして動作させることができます。なお OTS 1.2 からトランザクションの伝播はサーバアプリケーションが登録された POA の Policy によって設定されます。OTSPolicy は REQUIRES、FORBIDS、ADAPTS の 3 種類があります。

- REQUIRES
  オブジェクトは必ずトランザクション内で動作する必要があることを示します。そのため、REQUIRES が設定されたオブジェクトを呼び出すクライアントはトランザクションを開始しておかなければなりません。トランザクションを開始していない状態で呼び出すと CORBA::TRANSACTION_REQUIRED 例外が発生します。
- FORBIDS
  オブジェクトは必ずトランザクション外で動作する必要があることを示します。クライアントでトランザクションが開始されている場合、サーバアプリケーションを呼び出すかどうかは NonTxTargetPolicy によって決定します。WebOTX Transaction Service は NonTxTargetPolicy の既定値を PERMIT としているため、トランザクションコンテキストを伝播させずにサーバアプリケーションの呼び出しを行います。NonTxTargetPolicy を PREVENT に設定した場合、トランザクションが開始されていると CORBA::INVALID_TRANSACTION 例外が発生します。
- ADAPTS
  オブジェクトはトランザクション内でもトランザクション外でも動作できることを示します。クライアントでトランザクションが開始されていればトランザクションコンテキストを伝播し、トランザクションが開始されていなければトランザクションコンテキストを伝播せずにサーバアプリケーションを呼び出します。OTS 1.1 の CosTransactions::TransactionalObject を呼び出す場合は OTSPolicy が ADAPTS であるものとみなして動作します。

ルート POA や明示的に OTSPolicy を設定しない子 POA の OTSPolicy は FORBIDS として動作します。OTS 1.2 ではサーバアプリケーションは CosTransactions::TransactionalObject を継承していてもトランザクションとしては動作しませんが、オブジェクトマネージャへの登録時に ADAPTS を設定することで OTS 1.1 のアプリケーションをそのまま動作させることができます。クライアントアプリケーションは OTSPolicy が設定されていないサーバアプリケーションを呼び出す場合 CosTransactions::TransactionalObject を継承しているかチェックを行い伝播するため、OTS 1.1 に対する互換性は保証されます。

トランザクションのコミットメント

Transaction サービスでは、トランザクションを終了する場合、次の 2 種類のコミットメント制御を使用します。

- 1 フェーズコミットメント
  トランザクションに参加しているリソースが 1 つの場合は、そのリソースに対して即座にコミットを要求します。
- 2 フェーズコミットメント

トランザクションに参加しているリソースが複数の場合は、リソース全体のコミットを実施します。この際にトランザクションの終了処理は次の 2 つのフェーズに分けて行います。

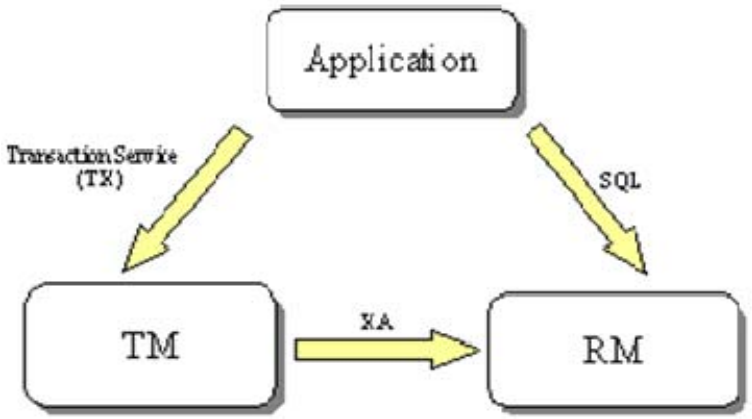
第 1 フェーズ
トランザクションに参加している全リソースに対してプリペアを要求します。リソース側ではこれを要求されると、アクセスしたデータを更新可能であり、トランザクションをコミットできるかどうかを通知します。リソースのうちのいずれかがコミット不可能であると通知してきた場合、自動的にトランザクションはロールバックします。

第 2 フェーズ
第 1 フェーズの結果に基づいて、トランザクションのコミットあるいはロールバックを全リソースに対して要求します。

トランザクションに参加したリソースが単一か複数かを判断して WebOTX が自動的に必要となるコミットメント制御を行いますので利用者は意識する必要がありません。

データベースのサポート

Transaction サービスは、X/Open 分散トランザクション処理にしたがってデータベースをサポートします。
X/Open 分散トランザクション処理では、トランザクションサービスを提供する部分をトランザクションマネージャ、データベースなど永続的なリソースを提供する部分をリソースマネージャと呼びます。トランザクションマネージャとリソースマネージャとのインタフェースは XA インタフェースと呼ばれるインタフェースが使われます。



Transaction サービスは、トランザクションマネージャとして以下の XA インタフェースをサポートするデータベースなどと協調してトランザクションとして動作します。

グローバルトランザクションとローカルトランザクション

Transaction サービスは、X/Open 分散トランザクション処理にしたがって次の 2 つのトランザクションをサポートします。

- グローバルトランザクション
  トランザクションサービスの制御の元にあるトランザクションで、ネットワーク上全体としてコミットまたはロールバックされます。
- ローカルトランザクション
  トランザクションサービスでの制御とは独立に、コミット、ロールバックを行うトランザクションです。ローカルトランザクションを完了させるためには、SQLのCOMMIT文またはROLLBACK文を使用する必要があります。

グローバルトランザクション内でローカルトランザクションを行うと、ネスティッドトランザクションのように一部分だけ全体のトランザクションの結果に独立したデータの更新ができます。ネスティッドトランザクションと異なるのは、コミット制御も独立に行うことができる点です。

トランザクションブランチ

XAインタフェースにおいて基本となる概念にトランザクションブランチがあります。トランザクションブランチとは、各スレッドおよびリソースマネージャ上に生成されるリカバリの最小単位です。例えば、1 つのトランザクシ

ョン内で複数のアプリケーションスレッドが連携して動作する場合、スレッドごとにトランザクションブランチが生成されます。トランザクションブランチは、アプリケーションがトランザクション処理を実行する前にそのスレッド上で開始され、アプリケーションがトランザクション処理を終了後にそのスレッド上で終了します。トランザクションブランチを開始・終了することで、トランザクションマネージャはリソースマネージャに対してリカバリの対象となるトランザクション処理の範囲を示します。

Transaction サービスは、サーバオブジェクトの呼び出しと戻りで、自動的にトランザクションブランチの開始および終了を行います。また、トランザクションの開始、終了時にも自動的にトランザクションブランチの開始および終了を行います。

トランザクション識別子

トランザクションブランチを識別するためにトランザクション識別子が使われます。
トランザクション識別子は、OTS(JTS)と XA インタフェースでは形式が異なりますが、基本的にはフォーマット識別子、グローバルトランザクション識別子、ブランチ識別子の 3 つからなります。フォーマット識別子はトランザクションサービス/トランザクションマネージャを識別するコードです。グローバルトランザクション識別子はグローバルトランザクションを識別するバイト列です。ブランチ識別子はトランザクションブランチを識別するバイト列です。

密結合と疎結合

密結合とは、1 つのグローバルトランザクション内の複数のトランザクションブランチがデータベースの同一エントリを更新することを認めるものです。
逆に、同一エントリの更新を認めない動作を疎結合と呼びます。提供される結合度についてはデータベースに依存します。

最も強力な密結合は、トランザクションブランチが別のプロセスや別のマシンの場合でも、同一エントリの更新を認めます。密結合はトランザクション識別子のブランチ識別子を同一にしてデータベースにアクセスすることで実現します。
Transaction サービスでは、同一のグローバルトランザクション内で同一のオブジェクトに対する複数のメソッド呼び出しは密結合として動作させます。異なるオブジェクトや異なるプロセス、マシンへの呼び出しは疎結合として動作させます。分散トランザクションで複数のマシン、複数のプロセスで同一のデータベースエントリを更新することは、データベース破壊などの危険な状況を起こしやすいので、これを避けるためにこのような制御を採用しています。

静的登録と動的登録

トランザクションマネージャは、一般に、アプリケーションが接続しているデータベースすべてに対してトランザクションブランチを開始します。
これを静的登録と呼びます。この場合、トランザクションマネージャは、どのデータベースへのアクセスが行われるかを判断できないため、実際にはアクセスされないデータベースが存在しても、すべてのデータベースでトランザクションブランチを開始してしまいます。この場合無駄なトランザクションブランチも開始することになります。
そこでリソースマネージャがアプリケーションからデータベースにアクセスがあった場合にのみ、トランザクションブランチを開始する方法があります。これを動的登録と呼びます。動的登録が行われると、本当に必要なデータベースに対してのみトランザクションブランチが開始され、コミットプロトコルに参加するようになるので、無駄がなくなります。
Transaction サービスは、静的登録、動的登録どちらにも対応していますので、データベース側が動的登録をサポートしていれば、効率のよいトランザクションを実行することができます。

データベースの識別方式

Transaction サービスは、データベースをデータベース識別名により一意に識別しています。C++で使用するデータベース識別名は、データベースの種類を示すリソースタイプとオープン文字列、クローズ文字列を代表する名前です。また、Java で使用するデータベース識別名は、接続文字列やユーザ名といった JDBC データソースのプロパティ情報に対応してデータベースに接続する際のコネクション情報に対応する名前です。

ACOS上トランザクションの 2 フェーズコミットメント参加

Transaction サービスでは、ACOS Access Toolkit(AAT)が提供する JDBC ドライバと連携することで、本来は 2 フェーズコミットメントをサポートしていない ACOS 上のトランザクションを WebOTX が管理する 2 フェーズコミットメントトランザクションに参加させることができます。

それを実現するために、Transactionサービスでは、ACOS上のデータベースを管理するために使用する「1 フェーズコミット対応リソース」を実装しており、ここからAAT提供のJDBCドライバを経由してデータベースの更新処理を行います。

Transaction サービスでは、同一グローバルトランザクションへの、1 つの 1 フェーズコミット対応リソースと、任意の数の 2 フェーズコミット対応リソース(従来から Transaction サービスが提供するデータベースアクセス用のリソース)の登録ができるようになっています。

つまりこれによって ACOS 上のデータベースと、オープンサーバ上のデータベース(Oracle など)の同時更新が可能になります。

なお、本来であれば 1 フェーズコミットメントでの動作を前提としているものを 2 フェーズコミットメントトランザクションに参加させているため、いくつかの制限、および運用操作上の注意事項を設けています。それらの詳細については、運用編(運用操作)をご参照ください。

システム構成について

Transaction サービスでは、アプリケーションがネットワーク上のどこにあるかを意識する必要はありません。ネットワーク上の最適な位置にアプリケーションを再配置することができます。従って、Transaction サービスを使うと、1 台のサーバで集中して処理を行う集中型のシステムや複数サーバ・クライアントが協調して処理を行う分散型のシステムなど、さまざまな業務形態に合わせたシステムを構築することができます。

Transaction サービスを使ったトランザクションシステムの構成は基本的に、「サーバトランザクションモデル」「クライアントトランザクションモデル」、およびこれらを組み合わせた「複合形態モデル」に分けられます。それぞれの詳細については「デザイン編」をご参照ください。

6.ネットワーク通信

6.1.ネットワークプロトコル

6.1.1.RMI over IIOP

WebOTX では、EJB の通信基盤として RMI over IIOP の機能を提供しています。

RMI over IIOP は、Java™ の RMI (Remote Method Invocation: リモートメソッド呼び出し)でプロトコルに Internet Inter-ORB Protocol(IIOP)を使用して通信をおこなう機能です。

WebOTX は RMI over IIOP 1.0 に準拠しています。RMI over IIOP は、Object Management Group(OMG)が策定した仕様に基づいています。

- Java To IDL マッピング仕様

  RMI インタフェースから IIOP を介して通信をおこなうための仕様。

- Objects-by-Value 仕様

  Java オブジェクトのような複合データを通信するために CORBA 2.3 で規定された仕様。

また、J2EE で必要となる機能を実現するために以下の仕様をサポートしています。

- Portable Interceptor 仕様

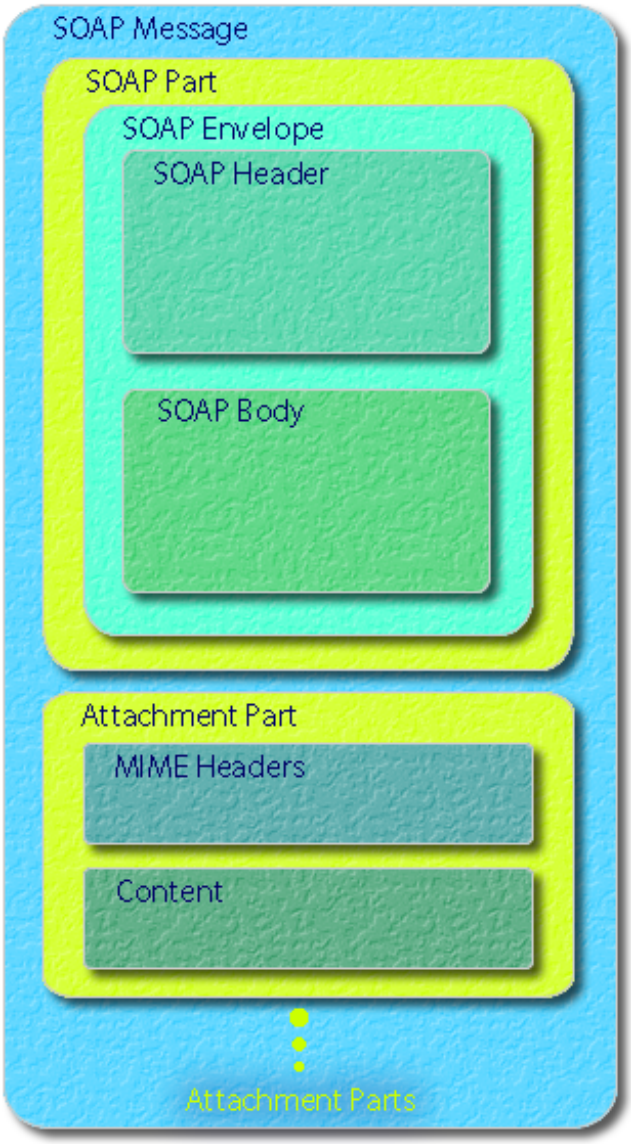
  通信処理の途中で、セキュリティ認証やアクセス可否判定などの処理を割り込み実行するために CORBA 2.5 で規定された仕様。

- Common Secure Interoperability Version 2 (CSIv2) 仕様

  セキュリティ認証情報を IIOP プロトコルで伝播するために CORBA 2.6 で規定された仕様。

6.1.2.SOAP

SOAP は XML 文書を交換するためのプロトコル仕様です。HTTP や SMTP などの既存のプロトコルに乗せて使うことが想定されている上位プロトコルです。図に SOAP メッセージの構造を示します。



- SOAP Message

  HTTP のヘッダなども含めた SOAP メッセージ全体を指します。

- SOAP Part

  XML で書かれるメッセージ本体の領域を指します。

- Attachment Part

  添付データの領域を指します。Attachment Part は、複数の領域をあとに続けて確保することができます。

- SOAP Envelope

  SOAP メッセージの一番外側の要素です。

- SOAP Header

  SOAP 通信や Web サービスのアプリケーションが動作する上で必要な付帯事項を書き込む要素です。

- SOAP Body

  通信したいメインの内容を書き込む要素です。

- MIME Headers

添付データの MIME エンコーディングに関するヘッダを書き込む部分です。

- Content

MIME エンコーディングした添付データを書き込む部分です。

7.運用管理

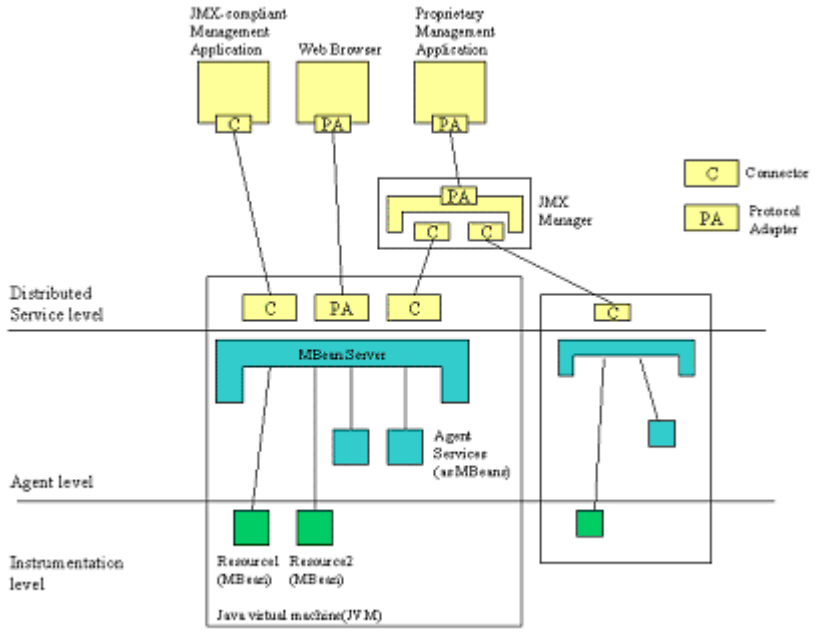
WebOTX の運用管理が提供している技術や特徴的な機能について説明します。

7.1.Java Management Extensions (JMX)

WebOTX の運用管理フレームワークは JMX 1.2 仕様(JSR-3)を基に実装しています。全ての管理対象リソースは model MBean という Java オブジェクトによりモデル化しています。これにより全ての WebOTX システムの運用管理操作およびイベント通知は JMX の API によって行うことができます。本節では JMX で規定されているアーキテクチャについて説明します。

JMX アーキテクチャは次の 3 レベルで構成されており、エージェントを通して運用管理クライアントから管理リソースを制御する仕組みとなっています。

- Instrumentation level
- Agent level
- Distributed services level



7.1.1.Instrumentation level

Instrumentation level は JMX 管理リソースの実装を提供します。JMX 管理リソースは具体的にはサービス、デバイス、ユーザ、アプリケーションを指します。この管理リソースを JMX 準拠の Java Bean である Managed Bean(MBean)によって管理します。MBean は JMX エージェントにより管理されます。

Instrumentation level ではさらに Notification メカニズムを提供します。Notification 機能により管理リソースで発生したさまざまなイベント(状態の変更、コンフィグレーションの変更、重大障害の発生など)をリモートの運用管理クライアントに通知することができます。

MBean

MBean は対象管理リソースのマネージメントインタフェースを提供する Java オブジェクトです。管理リソースごとに MBean は提供されます。

MBeanの要素

MBean のマネージメントインタフェースは次の要素を提供します。

- 属性 (Attribute)

  管理リソースのコンフィグレーション情報を属性として定義します。MBean の属性を取得、設定する

ことによりコンフィグレーション情報の参照、更新が行えます。MBean は属性を取得、設定するためのメソッド(getter メソッド、setter メソッド)を提供します。

- オペレーション（Operation）

  起動、停止など管理リソースに対する操作をオペレーションとして定義します。

- 通知(notification)

  管理リソースから通知するイベントを notification として定義します。運用管理クライアントは notification を受信することにより管理リソースで発生したイベントをリアルタイムで知ることができます。

- コンストラクタ（Constructor）

  コンストラクタのインタフェースを提供することにより、MBean を生成しエージェントに登録を行うことができます。

MBean の属性およびオペレーションは Java の public メソッドとして提供されます。運用管理クライアントは管理対象リソースの運用管理を行うため、JMX エージェントを通じて MBean の属性を取得、設定したり、オペレーションを実行したり、Notification イベントを受信したりします。

7.1.2.Agent level

Agent level はエージェントの実装を提供します。エージェントは直接管理リソースをコントロールし、リモートのマネージメントアプリケーション(運用管理クライアント)からリソースの利用を可能にします。

JMX Agent は MBean server と MBean を扱うためのサービス群で構成されています。Distributed Service level でエージェントにアクセスするためのアダプタやコネクタを利用します。

MBeanServer

MBeanServer はエージェント内で 1 つ作成され、エージェントに登録された全ての MBean を管理します。外部モジュールに対して MBean にアクセスするためのインタフェースを提供します。ただし MBeanServer インタフェースはエージェント内の同一 Java VM にあるモジュールで利用することができますが、リモートプロセスに対してそのインタフェースを公開していません。リモートプロセスに対してのインタフェースについては JMX Remote API で規定されています。

7.1.3.Distributed service level

Distribute Services level は JMX Manager の実装を提供します。Agent を操作するまたは Agent を階層化するためのインタフェースやコンポーネントについて定義します。

これらのコンポーネントの機能について以下に説明します。ただし JMX 1.2 では Distribute Services level について具体的な仕様を規定していません。

- コネクタを通してエージェントにアクセスするためのインタフェースを提供します。
- リソースを運用管理するためのさまざまな形態のクライアントを提供します。
- 複数の JMX エージェントの運用管理を提供します。
- セキュリティ(ユーザ認証、暗号化)を提供します。

コネクタ・アダプタ

リモートの運用管理クライアントが、管理リソースをコントロールするには MBean に対してアクションするためのインタフェースと通信プロトコルおよびリモート通信におけるセキュリティが必要です。このインタフェースと通信プロトコルおよびセキュリティを提供するのがコネクタ・アダプタです。JMX で規定されたクライアントとエージェントとの間で利用するのがコネクタで、既存プロトコル(HTTP を利用した Web ブラウザや SMTP を利用した運用管理アプリケーション)を使用するアプリケーションからエージェントにアクセスを行うために利用するのがアダプタです。アダプタは既存プロトコルから JMX のプロトコルにプロトコル変換を行う機能を有しています。

コネクタについては JMX Remote API で仕様が規定されています。

運用管理クライアント

運用管理クライアントは大きく 2 種類に分かれます。

- JMX 準拠の運用管理クライアント

  JMX(厳密には JMX Remote API)で規定している API を用いてエージェントにアクセスするクライアントです。

- 既存プロトコルを利用している運用管理クライアント

  HTTP や SMTP など既存で定義されているプロトコルを用いてエージェントにアクセスするクライアントです。

WebOTX では運用管理を行うため次のクライアントを提供しています。これ以外にも運用管理 API を利用して独自にカスタマイズした運用管理クライアントを作成することが可能です。

- 運用管理ツール

  JMX 準拠のインタフェースでエージェントにアクセスする GUI ベースのクライアントです。WebOTX 運用環境をインストールしたマシンから利用可能です。

- 運用管理コンソール

  HTTP サーバを通して HTTP プロトコルでエージェントにアクセスする Web ベースの GUI クライアントです。Web ブラウザから利用可能です。

- 運用管理コマンド

  JMX 準拠のインタフェースでエージェントにアクセスするコマンドベースのクライアントです。シェルスクリプトやバッチファイルから利用できます。

7.2.JMX Remote API

WebOTX における JMX Remote API の実装について説明します。

JMX 仕様ではリモートの運用管理クライアントからどのようにエージェントにアクセスするのかはアダプタ・コネクタを用いてアクセスするというガイドラインは示していましたが、詳細な仕様を規定していません。JMX Remote API では JMX 準拠の運用管理クライアントが利用するコネクタの仕様について規定しています。WebOTX では JMX Remote API で規定されたコネクタとそのコネクタを利用した JMX 準拠の運用管理クライアントを提供しています。

7.2.1.コネクタ

リモートの運用管理クライアントから MBeanServer にアクセスするために JMX Remote API ではいくつかのプロトコルを利用したコネクタインタフェース(MBeanServerConnection インタフェース)について規定しています。コネクタインタフェースについてはプロトコルに依存せず、MBeanServer のインタフェースと同様なものを提供しています。

コネクタの種類

JMX Remote API としては、クライアント、サーバ間の通信プロトコルにより、標準で次の 2 つのコネクタについて規定しています。

- RMI コネクタ

  リモート呼び出しを行う場合、Java で一般的に用いられている Remote Method Invocation (RMI)で通信を行います。

- Generic コネクタ

  クライアント、サーバ間で JMX Remote API で独自に規定したメッセージをやりとりすることにより通

信を行うコネクタです。実際に通信に利用するプロトコルについては任意ですが、JMX Remote API では TCP ベースの JMX Messaging Protocol (JMXMP)について実装を提供しています。

WebOTX では以下のコネクタを提供しています。

- JMXMP コネクタ

7.3.J2EE Management

WebOTX における J2EE Management の実装について説明します。

J2EE Management 仕様は J2EE において、Java エンタープライズ・マネージメントやモニタリング・サービスを実現するための共通フレームワークとして、JMX を基盤機能として規定されています。J2EE Management 仕様の目的は、J2EE が提供する情報にアクセスするため、より良い定義モデルを供給する J2EE アーキテクチャの、基本的な管理可能の概観を抽象化することにあります。加えてこの仕様では、プラットフォームリソースのモニタリングやコントロールに参加する J2EE コンポーネントに操作が共通の標準 API を定義しています。J2EE Management 仕様で規定している基本機能は次のとおりです。

- ディスカバリ – 管理システムの管理オブジェクトを発見したり操作したりする機能
- イベント – 管理オブジェクトで発生した重要なイベントの通知を受け取る機能
- 状態管理 – 管理オブジェクトの実行状態を監視しコントロールする機能
- パフォーマンス – 管理オブジェクトの基本性能統計をモニタリングする機能

なお J2EE Management 仕様は上記以外に既存マネージメントプロトコル(SNMP/CIM)とのマッピングについても規定しているが、これについては WebOTX では未サポートであるため説明は省略します。

7.3.1.管理オブジェクト

J2EE Management 仕様では、管理リソース単位で管理オブジェクト(Managed Object 以下 MO と略します)を定義しています。管理オブジェクトはリソースのコンフィグレーションの取得や設定、状態管理、パフォーマンス情報の取得をするためのメソッドを提供します。管理オブジェクトは JMX 仕様で規定されている MBean で実装されています。J2EE Management 仕様では J2EE アプリケーションサーバを構成する全ての管理リソースについて具体的に管理オブジェクトのインタフェースを規定しています。これに従い WebOTX も全ての管理リソースを管理オブジェクトとして JMX の model MBean の形式で提供しています。

また J2EE Management 仕様では、次に示す管理オブジェクトとしての振る舞いを規定しています。MO は MBean であるので属性、オペレーション、コンストラクタ、notification をインタフェースとして提供します。

7.3.2.状態管理

J2EEMangement では状態のあるリソースの MO には状態管理を行い、運用管理クライアントに対してその状態を取得できるインタフェースを提供するように規定されています。さらに状態に変更があった場合、その変更をイベントとして通知することも規定されています。状態管理を行うためのインタフェースとして StateManageable インタフェースが用意されており、MO はそのインタフェースを実装するようになっています。WebOTX で提供する MBean も状態管理を提供するものは全てこのインタフェースを実装しています。

StateManageable インタフェースの概要を以下に示します。

- 状態を表す属性
- 開始した時刻を示す属性
- 開始メソッド
- 停止メソッド

7.3.3.パフォーマンスモニタリング

J2EE Management では管理リソースのパフォーマンス情報を提供するため、リソース毎に提供すべきパフォーマンス情報について規定しています。それぞれの MO に対して提供すべきパフォーマンス情報を定義するインタフェースを規定しています。例えば EJB の MO については EJBStats というインタフェースを規定し、そのインタフェースを実装した MO を提供しています。WebOTX では以下のパフォーマンスデータを提供しています。

- EJBStats

  EJB に関するパフォーマンスデータ

- EntityBeanStats

  Entity Bean に関するパフォーマンスデータ

- JavaMailStats

  JavaMail に関するパフォーマンスデータ

- JCAStats

  JCA リソースアダプタに関するパフォーマンスデータ

- JDBCStats

  JDBC に関するパフォーマンスデータ

- JMSStats

  JMS に関するパフォーマンスデータ

- JTAStats

  JTA に関するパフォーマンスデータ

- JVMStats

  JVM に関するパフォーマンスデータ

- MessageDrivenBeanStats

  メッセージドリブン Bean に関するパフォーマンスデータ

- ServletStats

  Servlet に関するパフォーマンスデータ

- SessionBeanStats

  Session Bean に関するパフォーマンスデータ

- StatefulSessionBeanStats

  ステートフル Session Bean に関するパフォーマンスデータ

- StatelessSessionBeanStats

  ステートレス Session Bean に関するパフォーマンスデータ

- URLSts

  URL に関するパフォーマンスデータ

7.3.4.イベント

J2EEMangement ではイベント通知の実装については JMX の Notification モデルで実装することを規定しています。WebOTX でも JMX および JMX Remote API の Notification モデルをサポートします。以下に WebOTX が標準的に提供するイベントの概要について示します。

- MO(MBean)が登録された、削除された
- MO(MBean)の属性値の値が更新された
- WARNING レベル以上の問題が発生した
- サービスのアライブチェックで生存確認に失敗した

- 統計情報のモニタリングで閾値を設定し、その閾値を越えた

7.4.ドメイン管理

WebOTX ではシステムをドメインという構成単位で管理を行います。まずはドメインの構成について説明します。

7.4.1.ドメインとは

WebOTX の基本的な構成単位をドメインと呼びます。このドメインは JMX 仕様で定義されているドメインに対応します。ドメインは WebOTX で管理するリソースやサービス群を論理的にグルーピングしています。ドメインは独立して運用され、コンフィグレーション情報など構成情報はドメイン単位で独立したディレクトリに管理されます。それぞれの業務システムの要件にあわせてドメインを構成することができます。例えば 1 つのホストで商用環境と評価環境を構築したい場合、そのホスト上に商用のドメインと評価用のドメインを構成させることができます。また稼動、待機構成のクラスタ構成を行いたい場合は、複数のホストで 1 つのドメインを定義できます。

7.4.2.ドメインのタイプ

WebOTX では大きく 2 つのドメインにタイプ分けしています。

管理ドメイン

WebOTXをインストールした時点で、ホスト単位で1つ作成されます。管理ドメインはそのホスト上で動作するユーザドメインの管理を行います。管理ドメインで以下の操作を行うことができます。

- ユーザドメインの作成、削除
- ユーザドメインの起動、停止
- そのホストで動作しているユーザドメインの一覧取得
- ユーザドメイン間の依存関係(主に起動順序)の設定

ユーザドメイン

システム構成によってユーザが任意に作成するドメイン。ドメイン作成時に、そのドメイン上で管理するリソースやサービスを任意に指定（WebOTX の Edition により指定できるリソースやサービスに制限されます）することができます。ユーザドメイン上で実際の業務システムを構成するアプリケーションやサービスが動作します。

7.5.WebOTX Model MBean

model MBean は JMX 仕様で規定されている MBean であり、WebOTX ではこの ModelMBean を独自に機能強化しています。WebOTX Model MBean の機能と特長について説明します。

7.5.1.ディスクリプタ

model MBean ではさまざまなポリシについての定義を行うためディスクリプタを規定しています。ディスクリプタはポリシについての設定を定めるフィールドとその値を保持しています。ポリシは各フィールドに設定された値によって決定されます。ディスクリプタは ModelMBeanInfo の `getDescripter()`メソッドにより取得することができます。

7.5.2.ポリシ

model MBean はリソース管理を容易に行えるようにするためさまざまなポリシが規定されています。以下に model MBean で規定されているポリシについて説明します。

Notificationポリシ

model MBean は提供する属性に対して更新(SetAttribute/SetAttributes メソッドが実行された)があった場合、属性変更通知(jmx.attribute.change)を通知する実装がなされています。

Notificationロギングポリシー

MBean で発生した全ての Notification 通知をログファイルに記録する仕組みを実装しています。ログ採取の有無やログファイル名を設定します。

Persistenceポリシ

MBean のコンフィグレーション情報などを永続的に扱うために、model MBean ではファイルなど外部リソースに属性の値を格納するインタフェースをサポートしています。永続化をサポートしない場合、MBean の属性値は一時的なものになり、システムの再起動などを行った場合、設定された値はリセットされます。WebOTX では永続化が必要については MBean 毎に xml ファイルとして永続化する実装を行っています。

また、永続化するタイミングについて以下に示すポリシを PercistPolicy フィールドに設定します。これらは MBean の属性の特性に応じて設定します。

- Never:

  永続化をサポートしない。属性は一時的である。

- OnTimer:

  persistPeriod で指定した間隔で、その間に属性値の更新が合った場合、外部リソースに保存します。

- OnUpdate:

  属性値の更新を行ったタイミングで外部リソースに保存します。

- NoMoreOftenThan:

  属性値の更新を行ったタイミングで、なおかつ前回更新時より persistPeriod 経過している場合、外部リソースに保存します。これは頻繁に更新されるデータに対して更新回数を抑えられる特長を持ちます。

Cacheポリシ

MBean では属性もしくはオペレーションの戻り値に対して、キャッシュポリシーとともにデータのキャッシュ値および既定値を保持しています。運用管理クライアントから、属性値の取得要求があった場合、キャッシュポリシーに従い保持しているキャッシュ値が有効であると判断した場合、リソースに対してデータ取得要求を行わず、キャッシュ値を返却する仕組みを実装しています。リソースとの直接のやりとりが行われないため、実行時のアプリケーションリソースと性能の管理活動のインパクトを最小限にすることができます。

MBean の属性単位で、秒単位で表現される currencyTimeLimit フィールド(キャッシュ値の有効期限)と lastUpdatedTimeStamp フィールド(前回更新時刻)が設定でき、運用管理クライアントから属性取得要求があった時刻が lastUpdateTimeStamp+currencyTimeLimit を過ぎている場合、属性値は無効と判断され、過ぎていない場合有効と判断されます。有効と判断された場合は即座にキャッシュ値を返却し、無効と判断された場合は getMethod フィールドで設定された getter メソッドを呼び出してリソースに対して属性値取得要求を行います。getMethod フィールドで getter メソッドが定義されていない場合、常に default フィールドで設定された既定値が返されます。

Protocol Mapポリシ

MBean のデフォルト動作と API はほとんどのアプリケーションの管理ニーズを満たすことになっています。しかしながら model MBean は複雑なリソース管理のシナリオも提供します。model MBean API はアプリケーションの MBean の属性と MIB や CIM オブジェクトのような既存マネージメントデータモデルとのマッピングを ProtocolMap フィールドの定義に従って行うことができます。

例えば MIB generator は JMX エージェントと相互に作用し、SNMP 管理システムによってロードされた MIB ファイルを作成します。作成した MIB ファイルは JMX エージェントによって知られている resource 情報を表現します。

なお WebOTX で提供する MBean については ProtocolMap フィールドを設定していません。

Export ポリシ

JMX エージェントがマルチ環境で動作する場合、各 JMX エージェントは適切なディレクトリかルックアップサービスで存在および有効性を公示することが運用管理を容易にする。JMX では MBean が現在どの JMX エージェントに登録されているかを、他の JMX エージェントからも位置を確定できるようにするしくみを提供している。このようにディレクトリおよびルックアップサービスにより MBean の位置を公示する場合は、Export フィールドを定義するようになっています。

なお WebOTX で提供する MBean については Export フィールドを設定していません。

Visibility ポリシ

運用管理を行うにあたって、運用管理クライアントを GUI で提供することは必要であるが、多種多様な MBean の属性やオペレーションをどのように見せるかについてポリシを規定しています。

例えば、エンタープライズマネージャは全ての情報を見て、より高いレベルの管理オブジェクトと対話したいと思うかもしれません。ドメインマネージャは、一般にアプリケーションの内容だけを見たいと思うでしょう。JMX ではこのようにユーザの関心度に対応したクライアントビューを見せる仕組みを提供しています。

Visibility フィールドは MBean、属性あるいはオペレーション単位に 1～4 の整数で設定します。最大は 1 で、ユーザの関心度が最も高いことを示しています。最小は 4 であまり重要性のない、特別の場合だけ必要な情報であることを意味しています。なお JMX 仕様はこれら 1-4 のレベルを厳密に定義していません。MBean 作成者に定義を委ねています。WebOTX の運用管理ツールでは通常ユーザでは Visibility が 1 のものしか表示しません。

Presentation ポリシ

PresentationString フィールドは任意のディスクリプタに定義することができる文字列です。この文字列はコンソールに関するヒントを提供するための、XML にフォーマットされた文字列であると規定されています。しかしながら、JMX 1.2 ではプレゼンテーションフィールドの標準のセットはまだ定義されておらず、WebOTX でも PresentationString については何も設定していません。

7.5.3.Notification

WebOTX Model MBean ではシステムの運用管理を行なうためのさまざまな Notification を提供します。これらの Notification は統合運用管理ツールがリアルタイムに通知を受け取り、システム稼動に影響がある問題が発生したことを即座に確認することができます。

イベントNotification

運用管理エージェントが WARNING 以上の問題を検出すると、通常その内容は OS のイベントログやシスログに出力しますが、それに加えイベント Notification を発行します。

アライブチェックNotification

WebOTX 上で動作するサービスやサーバ AP プロセスなど状態管理を行なっているリソースに対してアライブチェックを行なうことができます。アライブチェックは一定時間ごとに行なわれ、生存確認が行なえなくなった場合アライブチェック Notification を発行します。

モニタリングNotification

MBean 上で統計情報として提供している属性に関して閾値を設定しモニタリングを行なうことができます。モニタリングは一定時間ごとに行なわれ、閾値を越えた場合、モニタリング Notification を発行します。

7.6.管理対象リソース・サービス

WebOTX で MBean として提供する管理対象リソース・サービスについて説明します。MBean の持つ ObjectName の J2EEType プロパティでその識別を行います。

- J2EEDomain

  ドメインに対応している MBean です。各ドメインで必ず 1 つ作成されます。

- J2EEServer

  ドメイン内のエージェントに対応している MBean です。各ドメインで必ず 1 つ作成されます。

- J2EEApplication

  ドメイン内で配備されているアプリケーション(ear)に対応している MBean です。配備されているアプリケーションに応じて作成されます。

- AppClientModule

  ドメイン内で配備されたアプリケーションクライアントモジュール(jar)に対応している MBean です。配備されているクライアントモジュールに応じて作成されます。

- EJBModule

  ドメイン内で配備された EJB モジュール(jar)に対応している MBean です。配備されている EJB モジュールに応じて作成されます。

- WebModule

  ドメイン内で配備された Web モジュール(war)に対応している MBean です。配備されている Web モジュールに応じて作成されます。

- ResourceAdapterModule

  ドメイン内で配備されたリソースアダプタモジュール(rar)に対応している MBean です。配備されているリソースアダプタモジュールに応じて作成されます。

- EntityBean

  ドメイン内で配備された Entity Bean に対応している MBean です。配備されている Entity Bean に応じて作成されます。

- StatefulSessionBean

  ドメイン内で配備されたステートフル Session Bean に対応している MBean です。配備されているステートフル Session Bean に応じて作成されます。

- StatelessSessionBean

  ドメイン内で配備されたステートレス Session Bean に対応している MBean です。配備されているステートレス Session Bean に応じて作成されます。

- MessageDrivenBean

  ドメイン内で配備されたメッセージドリブン Bean に対応している MBean です。配備されているメッセージドリブン Bean に応じて作成されます。

- Servlet

  ドメイン内で配備された Servlet に対応している MBean です。配備されている Servlet に応じて作成されます。

- ResourceAdapter

  ドメイン内で配備されたリソースアダプタに対応している MBean です。配備されているリソースアダプタに応じて作成されます。

- JavaMailResource

  ドメイン内の JavaMail リソースに対応している MBean です。J2EEServer 単位で 1 つ作成されます。

- JCAResource

  ドメイン内の JCA リソースに対応している MBean です。J2EEServer 単位で 1 つ作成されます。

- JCAConnectionFactory

  ドメイン内で登録された JCA コネクションファクトリに対応している MBean です。登録されている JCA

コネクションファクトリに応じて作成されます。

- JCAManagedConnectionFactory

  ドメイン内で登録された JCA マネージドコネクションファクトリに対応している MBean です。登録されている JCA マネージドコネクションファクトリに応じて作成されます。

- JDBCDataSource

  ドメイン内で登録された JDBC データソース対応している MBean です。登録されている JDBC データソースに応じて作成されます。

- JMSResource

  ドメイン内の JMS リソースに対応している MBean です。J2EEServer 単位で 1 つ作成されます。

- JNDIResource

  ドメイン内の JNDI リソースに対応している MBean です。J2EEServer 単位で 1 つ作成されます。

- JTAResource

  ドメイン内の JTA リソースに対応している MBean です。J2EEServer 単位で 1 つ作成されます。

- RMI_IIOPResource

  ドメイン内の JTA リソースに対応している MBean です。J2EEServer 単位で 1 つ作成されます。

- URLResource

  ドメイン内の URL リソースに対応している MBean です。J2EEServer 単位で 1 つ作成されます。

- JVM

  ドメイン内の JVM に対応している MBean です。Java VM 単位で 1 つ作成されます。

7.7.ライフサイクルサービス

WebOTX ではドメイン内で動作する各種サービスをライフサイクル管理しています。ドメイン毎に動作させるライフサイクルサービスを選択することができます。

7.7.1.提供するライフサイクルサービス

WebOTX では次のライフサイクルサービスを提供します。ただし Edition により利用できないサービスがあります。

サービス名称	Web Edition	Standard-J Edition	Standard Edition	Enterprise Edition
HTTP サーバ	○	○	○	○
Web コンテナ	○	○	○	○
JNDI サービス	○	○	○	○
EJB コンテナ	×	○	○（注）	○（注）
JMS サービス	×	○	○	○
Object Broker サービス	○	○	○	○
Transaction サービス	○	○	○	○
TP モニタサービス	×	×	○	○
Adm リスナサービス	×	×	○	○

（注）
Standard Edition および Enterprise Edition の EJB コンテナは WebOTX エージェントの Java VM とは独立した Java VM で動作します。

7.7.2.サービスの運用

各サービスの運用について説明します。サービスに関する設定はライフサイクルサービスの MBean によって行うことができます。

- サービスの追加・削除

  サービスの追加・削除はドメイン作成時に選択します。ドメイン作成時にそのドメインで動作させたいサービスを選択します。

- サービスの起動・停止

  サービスの起動・停止は運用管理コマンドで行うことができます。サービス毎に起動・停止させることが可能です。

- サービスの自動起動

  追加したサービスはデフォルトではドメイン起動時に自動起動します。ドメイン起動時にサービスを自動起動させたくない場合は、コマンドにて設定を変更することが可能です。

- サービスの依存関係

  サービス間で依存関係を持つものがあります。基本的には WebOTX で依存関係の制御を行います。依存関係のあるサービスが起動していない場合は、サービスの起動を行うことができません。また依存関係のあるサービスを停止しようとする場合も、サービスが起動していれば停止することはできません。

7.8.システム管理ツール

WebOTX ではアプリケーション実行環境を運用操作するための次のツールを提供しています。これらのツール群を利用することでリモートの運用コンソールより複数の実行環境(Web サーバ、AP サーバ)の統合的な運用ができます。

7.8.1.運用管理ツール

マルチドメインで構成された WebOTX システムを一元的に運用管理するための GUI ツールです。コンフィグレーション、アプリケーションの配備、状態監視、障害監視など全ての運用操作がこの統合運用管理ツールで行なうことができます。

7.8.2.運用管理コンソール

Web ブラウザを利用して管理するツールです。Web ブラウザさえ動作する環境があれば、リモートから簡単に管理が行えます。

7.8.3.運用管理コマンド

WebOTX ではオペレータを介することなく自動で運用を行うためのバッチファイルやシェルスクリプトで使用できるコマンドを提供しています。これにより自動でアプリケーションの展開、サービスの開始、停止、設定の変更などを行うことができます。

7.8.4.運用管理API

WebOTX では JMX Remote API インタフェースを提供しています。これにより該当システムに特化したコマンドや運用管理ツールを JMX の API を利用して開発することができます。

7.9.ユーザ管理

WebOTX では、WebOTX を管理する運用ユーザ、Web アプリケーションを利用するユーザの管理に LDAP サーバを利用することができます。

7.9.1.LDAPサーバ

LDAP とは、「Lightweight Directory Access Protocol」の略で、つまりディレクトリサービスにアクセスするための軽量なプロトコルということになります。ディレクトリサービスとは、ネットワークを利用するユーザ名やマシン名などの様々な情報を管理するためのサービスのことで、ユーザ名などのキーとなる値から様々な情報を検索することが可能です。つまり LDAP サーバとは、LDAP と呼ばれるプロトコルを用いて参照することができるディレクトリサービスとなります。

WebOTX では、V6.5 より以下の LDAP サーバを利用することができます。

・EnterpriseDirectoryServer V5.0(Windows,Linux) V4.1(HP-UX)

・OpenLDAP V2.0

LDAP サーバに WebOTX の運用に使用するユーザ、Web アプリケーションで利用するユーザを登録しておくことによって、WebOTX は LDAP サーバからユーザ情報を取得することができます。

7.10.サーバ管理

WebOTX AS では、WebOTX AS が動作するサーバの管理を行う機能を提供しています。

7.10.1.ワーキングドメインコーディネータ

クライアントからの一時的なアクセス増が予想されるような場合、最大限の負荷を想定してサーバの増設を行うとコストがかかってしまいます。この問題を解決するために、限られた資源を有効活用し、高負荷時にも安定した業務アプリケーションの運用を可能とする機能を Working Domain Coordinator は提供します。

Working Domain Coordinator は、負荷分散装置の利用を前提として構築された複数のサーバを効率よく利用するために、業務アプリケーションの負荷に応じて、処理を行うサーバの数を調節します。WebOTX AS 上で動作する業務アプリケーションそれぞれの負荷を監視し、負荷の低い業務アプリケーションのドメインを負荷の高い業務アプリケーションのドメインに切り替え、負荷分散装置に対して振り分け先の設定を行います。

負荷監視機能

Working Domain Coordinator は、通常時、一定間隔で負荷監視対象の業務アプリケーションが動作するプロセスグループのキュー滞留数を監視します。定期監視中、キュー滞留数上限値を超える負荷を検出した場合、高負荷時の監視設定に切り替え、負荷状態の継続性を判定するため、監視を続けます。高負荷時の監視では、高負荷時監視間隔で、設定されたサンプリング回数分、キュー滞留数の採取を行います。サンプリングの結果、そのうちの何割がキュー滞留数上限値を超えているかを算定し、この割合が負荷超過率を超えていた場合、継続的な高負荷状態であると判断し、その高負荷業務アプリケーションと処理に対して余力のある低負荷業務アプリケーションの切り替えを行います。

業務アプリケーションの稼動時間や優先度に応じて、監視条件の設定値を調整することで、より正確に高負荷を検出することが可能となります。

また、定期監視で得られた負荷情報は、高負荷業務アプリケーションと低負荷業務アプリケーションの切り替え時の基準値として利用されます。

ビジネスロジックの切り替え機能

負荷監視機能により、継続的な高負荷状態を検出すると、高負荷業務アプリケーションを処理するサーバを増やすための切り替え処理を行います。切り替え処理は、業務アプリケーションに対する処理に余力のあるサーバをひとつ選び、そのサーバ内で稼動するドメインを停止させ、継続的な高負荷状態にある業務アプリケーションのドメイン起動を行います。低負荷業務アプリケーションは、負荷監視機能の定期監視で得られた負荷情報をもとに選択されます。また、各業務アプリケーションに設定された優先度に応じて切り替え対象の選択を調整することができます。以下の項目を業務アプリケーション毎に設定することが可能です。

・優先度
　業務アプリケーション毎の優先度を設定することができます。複数の異なる優先度の業務アプリケーションが動作していた場合、最も優先度が低い業務アプリケーションが停止の対象となります。また、設定が可能な最高値の優先度を設定された業務アプリケーションは、停止の対象とはなりません。

・稼動サーバ数上限値
　特定の業務アプリケーションが稼動するサーバ数の上限値を設定することができます。高負荷状態と

判断された業務アプリケーションが稼動しているサーバ数がこの稼動サーバ数の上限値に達している場合は、切り替え処理は行われません。

・稼動サーバ数下限値
　特定の業務アプリケーションが稼動するサーバ数の下限値を設定することができます。余裕があり停止可能と判断された業務アプリケーションが稼動しているサーバ数がこの稼動サーバ数の下限値の場合は、そのサーバは切り替え対象のサーバとして選択されません。

高負荷業務アプリケーションが検出されても、切り替えの対象となる低負荷の業務アプリケーションが存在しない場合は、業務アプリケーションの切り替えは行われません。

負荷分散制御装置(LB)の制御機能

Working Domain Coordinator は、高負荷業務アプリケーションと低負荷業務アプリケーションの切り替えを行う際、起動および、停止するドメインの LB への振り分け先の制御を行う機能を提供します。多種多様な LB に対応するため、LB の振り分け先設定の制御タイプを3タイプ設けています。以下に各制御モードを説明します。

(1)　LB 制御あり

LB の振り分け先制御を Working Domain Coordinator が自動で行うモードです。Working Domain Coordinator がインストールされたサーバから LB 制御コマンドを発行することで、LB の振り分け先を制御することができる場合、このモードを利用することができます。このモードでは、Working Domain Coordinator は、切り替え処理中、停止対象のドメインに設定された LB の振り分け先削除用コマンドを発行します。コマンド発行後、そのドメインで動作中のプロセスグループの状態を確認し、クライアントからのリクエストを処理中であれば、処理の終了を待ち合わせます。処理中のリクエストがなくなったことを確認後、ドメインを停止します。また、ドメインの起動時は、ドメインの起動処理が完了したことを確認した後、そのドメインに設定された LB の振り分け先追加用コマンドを発行します。

(2)　LB 制御なし

LB の振り分け先設定を LB のヘルスチェック機能で制御するモードです。このモードでは、Working Domain Coordinator は、高負荷時の切り替え処理で対象のサーバを選択すると、停止対象ドメインの停止処理を行います。このとき、LB は自身のヘルスチェック機能により、停止したドメインの閉塞を検出し、そのドメインを振り分け先から削除します。次に、Working Domain Coordinator は起動対象のドメインの起動を行います。LB は、ドメイン停止時と同様、自身のヘルスチェック機能により、起動したドメインの開放を検出し、そのドメインを振り分け先に追加します。

(3)　LB 制御依頼

LB が Web 管理ツールのみしかサポートしていないなどの理由により、コマンドを発行することで振り分け先を制御することができない場合、ドメイン停止処理を一時停止し、その間に運用管理者が振り分け先を制御するモードです。このモードでは、Working Domain Coordinator は、高負荷時の切り替え処理でサーバを選択すると、停止対象ドメインの振り分け先からの削除依頼メッセージをイベントログ(Unix では syslog)に出力します。メッセージの ID は、OTX22040012 です。運用担当者は、このメッセージに記載されたドメインを LB の振り分け先から削除し、Working Domain Coordinator に対してドメインの停止処理を再開する操作を行います。ドメインの停止処理の再開の通知を受けた Working Domain Coordinator は、そのドメインの停止処理を行います。次に、Working Domain Coordinator は起動対象のドメインの起動を行います。そのドメインの起動が完了すると、起動したドメインの振り分け先への追加依頼メッセージをイベントログ(Unix では syslog)に出力します。メッセージの ID は、OTX22040011 です。運用担当者は、このメッセージに記載されたドメインを LB の振り分け先に追加します。

制御対象サーバのメンテナンス支援機能

Working Domain Coordinator により制御されるサーバにメンテナンスの必要が生じた場合、そのサーバに対してのみ、負荷分散制御の対象から一時的に除外する機能を提供します。制御対象から除外されたサーバには、Working Domain Coordinator の負荷の監視や高負荷業務アプリケーションと低負荷業務アプリケーションの切り替え処理などの制御は行われません。

なお、制御対象サーバを負荷分散制御の対象から除外した状態で、Working Domain Coordinator の再起動を行うと、負荷の監視は再開されます。

BIG-IP連携機能

Working Domain Coordinator は、BIG-IP の iControl API に対応しています。この API を利用して、BIG-IP のプールに対し、自動的にドメインのアドレス登録や登録解除を行う機能を提供します。これによって、簡単な設定を行うだけで、エラーが発生することなく、ドメイン上で動作させる業務の切り替えを行うことができます。

7.11.データベースサーバ管理

WebOTX AS では、データベースサーバの管理を行う機能を提供しています。

7.11.1.データベースコントローラ

データベースコントローラは、JMX の運用管理インタフェースを介して、WebOTX AS のドメインからデータベースの起動や停止、状態監視を簡単に行うための機能を提供します。

データベースコントローラは、WebOTX Developer のテストサーバで、Java SE6 を利用する環境では、自動的に JavaDB(Apache Derby)のデータベースと接続を行うための JDBC データソースの定義を作成しますので、インストールと同時に利用可能となります。

データベースの起動停止機能

データベースコントローラは、「起動コマンド」、「停止コマンド」を使い、データベースの起動や停止を行います。「起動コマンド」、「停止コマンド」はそれぞれ複数の登録が可能です。また、「コマンド発行の待ち合わせ時間(ミリ秒)」を任意に設定することで、処理に時間を要するようなコマンドにも幅広く対応します。さらに、「ドメイン起動時の連動起動使用有無」を設定することで、ドメインの起動や停止と合わせて、データベースの起動や停止を行うことができます。

データベースの状態監視機能

データベースコントローラは、制御対象のデータベースの JDBC データソースを JNDI サーバより lookup し、SQL 命令を発行することで、状態監視を実現します。
通常、状態監視のための SQL 命令は、JDBC データソースの「データベースサーバの状態監視コマンド」を使用します。しかし、JDBC データソースの「データベースサーバの状態監視コマンド」が設定されていない場合に、データベースコントローラで登録する「データベースの状態監視コマンド」を使用することができます。

データベースの状態監視は、状態監視モードが設定され、制御対象のデータベースの JDBC データソースが登録されている必要があります。データベースの状態監視を行うための条件が満たされていない場合は、「起動コマンド」、「停止コマンド」の発行時の状態を保持します。

また、データベースの状態監視は、管理エージェント内で実施されます。この時、JDBC データソースがプロセスグループとの共有設定になっている場合、JDBC データソースの初期プールサイズの設定次第では、管理エージェント内で実施する状態監視時に不要なリソース消費を起こしてしまう恐れがあります。この状態を回避するために、データベースコントローラでは、JDBC データソースの「useStaticPool」プロパティに false を設定し、「初期プールサイズ」に 0 を設定します。こうすることで、管理エージェント内での状態監視は常に1つの JDBC コネクションを使いまわすことになり、不要なリソース消費を回避しています。

8.アプリケーション配備

8.1.デプロイヤーの役割

デプロイヤー（アプリケーションを配備する人）は、特定の運用環境のエキスパートであり、その環境にJ2EEアプリケーションや他の種別のアプリケーションを配備する作業を担当します。配備タスクを実行する際には、配備ツールや運用管理コマンドを使用します。また、実行環境が Standard/Enterprise Editionであれば、さらに TP システムに特化した WebOTX 運用管理ツールなども使えます。デプロイヤーは、Web や EJB コンポーネント、あるいは他のアプリケーションを実行環境である各コンテナにインストールします。その時の操作で、アプリケーション開発者とアプリケーション・アセンブラによって宣言された外部依存性を全て解決するように構成します。

8.1.1.アプリケーション・アセンブラ

デプロイヤーは、アプリケーション・アセンブラとの関係も理解しておく必要があります。この役割は、アプリケーション・コンポーネント開発者や作成したコンポーネントの集合を利用して、それらを完全な J2EEアプリケーションへとアセンブル(組み立て)します。通常、特定業務向けのソリューション提供を専門としています。アセンブラ担当者の役割は、デプロイヤーが配備プロセス中に解決しなければならないアプリケーションの外部依存性を記述する、アセンブリ情報を提供することです。

8.1.2.システム管理者

システム管理者は、企業のコンピューティング環境やネットワーク基盤の構成と管理を担当します。また、配備された J2EE アプリケーションの実行時のヘルスチェックも担当します。 WebOTX のシステムでは通常、WebOTX 統合運用管理ツールを使って実行時の監視と管理を行います。

8.2.配備ユニット

配備ユニットは、独立して配備することが可能なコンテンツの集まりで構成されます。配備ユニットは、その内容によって、次の 8 タイプに分けられます。

- J2EE アプリケーション
- J2EE モジュール(Enterprise JavaBeans、Web、リソースアダプタ、アプリケーション・クライアント)
- CORBA アプリケーション
- 共有コンポーネント
- SIP アプリケーション

J2EE アプリケーションは、1 つ以上の J2EE モジュール、SIP アプリケーションと、1 つの XML ベースのJ2EE アプリケーション配備記述子からなります。アプリケーション配備記述子には、そのアプリケーション内に含まれるモジュールリストと、カスタマイズ方法が定義されています。J2EE アプリケーションは、ファイル拡張子「.ear」(Enterprise ARchive)を持つ、Java アーカイブファイルとしてパッケージ化されます。このファイル形式の目的は、移植可能であることが保証されているアプリケーション配備ユニットを提供することです。

CORBA アプリケーション、共有コンポーネントは Standard/Enterprise Edition へ、SIP アプリケーションは WebOTX SIP Application Server へ配備することができます。

8.2.1.J2EEモジュール

J2EE モジュールは、Web や EJB コンテナタイプの配備記述子を含んだ 1 つ以上のコンポーネントの集まりです。1 つの記述子は J2EE 標準です。さらに他にも WebOTX 固有の配備記述子を含みます。

J2EE モジュールは、次の 4 タイプがあります。

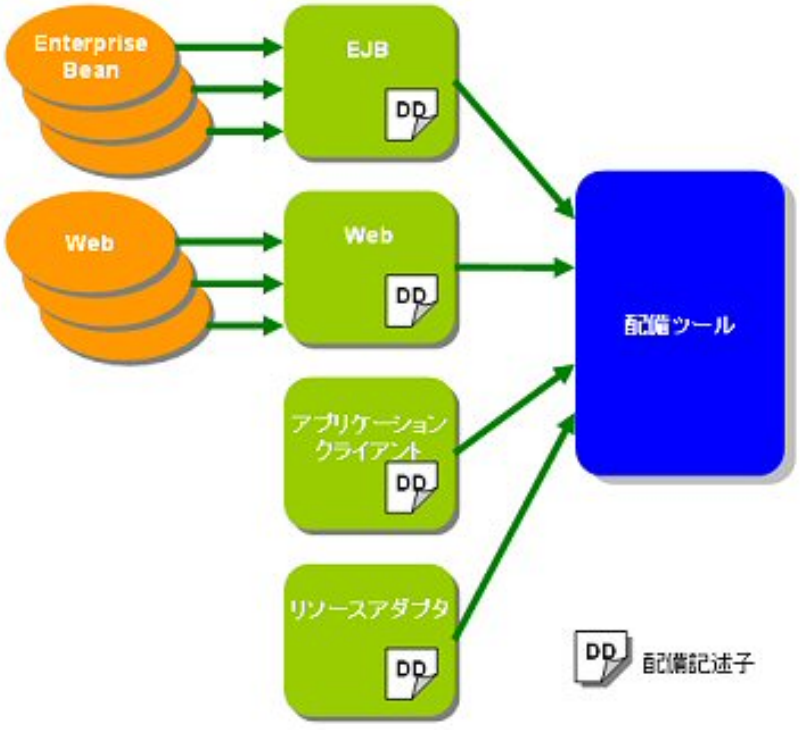
- Enterprise JavaBeans

Enterprise Bean 用のクラスファイルと EJB 配備記述子が含まれています。 EJB モジュールは、ファイル拡張子「.jar」を持つ JAR ファイルとしてパッケージ化されます。配備記述子は、Enterprise JavaBeans 2.1 仕様書で定義されます。

- Web
  JSP ファイル、サーブレット用のクラスファイル、GIF ファイルと HTML ファイルおよび、Web 配備記述子が含まれています。 Web モジュールは、ファイル拡張子「.war」(Web ARchive) を持つ JAR ファイルとしてパッケージ化されます。配備記述子は、Java Servlet 2.4 仕様書で定義されます。
- リソースアダプタ
  Java インタフェース、クラス、ネイティブライブラリ、その他のファイルおよび、リソースアダプタ配備記述子が含まれています。リソースアダプタは、特定の EIS に対するコネクタとしてベンダから供給されます。リソースアダプタモジュールは、ファイル拡張子「.rar」(Resource adapter ARchive) を持つ JAR ファイルとしてパッケージ化されます。配備記述子は、J2EE コネクタアーキテクチャ 1.5 仕様書で定義されます。
- アプリケーション・クライアント
  クラスファイルとアプリケーション・クライアントの配備記述子が含まれています。このモジュールは、ファイル拡張子「.jar」を持つ JAR ファイルとしてパッケージ化されます。配備記述子は、J2EE 1.4 仕様書で定義されます。

それらのパッケージ内には、モジュールを配備した後に外部参照クラスがローディングできるように適切に同梱しなければなりません。配備記述子の中に定義した情報は宣言的なものなので、ソースコードの修正を必要とせずに実行時の振る舞いを変更できます。実行環境では、コンテナがこの情報と振舞い方を指示した定義に従って読み込み、動作していきます。

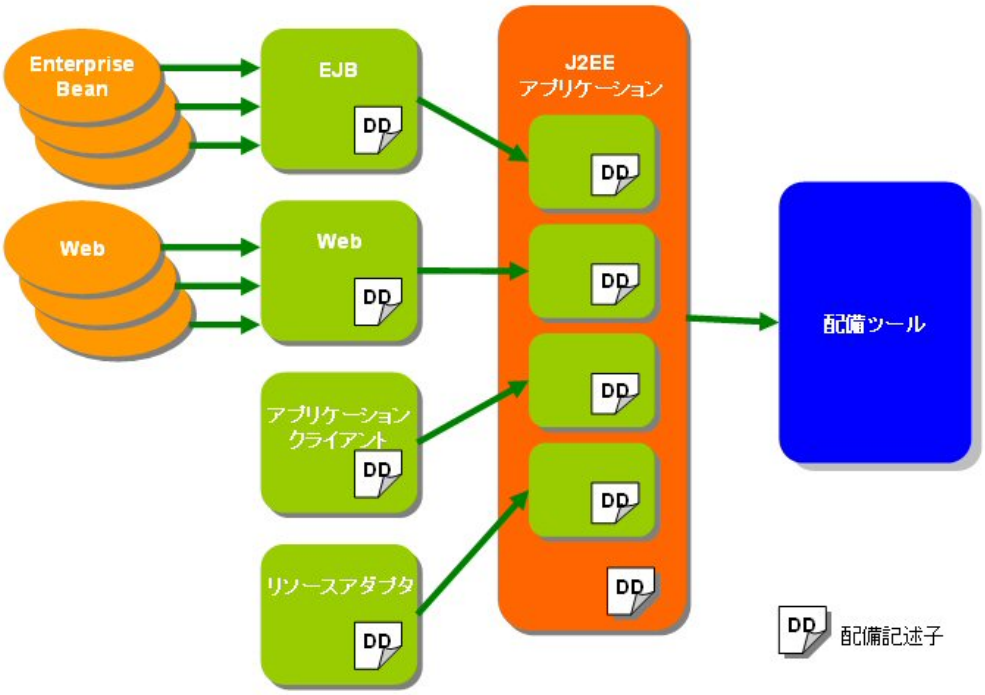
WebOTX V5 以前は、単体モジュールとして配備可能な形式はリソースアダプタだけでした。その他のタイプは、次に説明する J2EE アプリケーション EAR ファイルにまとめる必要がありました。WebOTX V6 からは、各モジュールを単独でコンテナに配備できます。



8.2.2.J2EEアプリケーション

J2EE アプリケーションは、アプリケーション配備記述子で結ばれた、1 つ以上の J2EE モジュール、SIP アプリケーションの集まりです。コンポーネントは、複数のモジュールを組み合わせて 1 つに組み立てることができます。このようにして作成したアプリケーション形式を EAR ファイルと呼びます。

全ての J2EE モジュールは、独立して配備可能なユニットであることに注意する必要があります。これによりコンポーネント開発者は、アプリケーション全体を実装することなく、機能ユニットで作成できるようになります。



5 タイプの配備記述子の詳細に関しては、「運用編」で説明されます。

8.2.3.CORBAアプリケーション

OMG が標準化を行っている CORBA という仕様に準拠したアプリケーションです。Java だけではなく、C/C++や COBOL で実装することもできます。CORBA アプリケーションのファイル名の拡張子は「.cpk」です。CORBA アプリケーションは Standard Edition と Enterprise Edition でサポートしています。

8.2.4.共有コンポーネント

プロセスグループ間で共有するライブラリです。共有コンポーネントのファイル名の拡張子は「.spk」です。共有コンポーネントは Standard Edition と Enterprise Edition でサポートしています。

8.2.5.SIPアプリケーション

SIP と呼ばれる通信制御プロトコルに特化したサーブレットを含むアプリケーションです。また Web アプリケーションのコンテンツを含むこともできます。SIP アプリケーションのファイル名の拡張子は「.sar」です。配備記述子は、SIP Servet API 1.0 仕様書で定義されます。SIP アプリケーションは WebOTX SIP Application Server でサポートしています。

8.3.配備記述子

配備記述子は、XML ベースのテキストファイルで、その要素はユニットをアセンブルして特定の環境に配備する方法を表しています。通常、配備記述子は配備ツールによって自動的に生成されるため、ファイル自身を直接管理する必要はありません。配備記述子の要素には、EJB や Web のコードに直接含まれていないコンポーネントの動作に関する情報が含まれています。その目的は、デプロイヤーにアプリケーションの配備方法を伝えることです。

配備記述子は、次の 2 つの情報を指定します。

- 構造情報

  JAR ファイル内の異なるコンポーネントおよび、それらの相互関係と外部依存関係が記述されています。

- アセンブリ情報

  JAR ファイルのコンテンツを配備可能なユニットにどのように構成できるかが記述されています。アセンブリ情報はオプションです。

8.4.配備ツールの動作

WebOTX 配備ツールは、ある J2EE アプリケーションの配備操作が行われると、その背後でいくつかの配備準備を実行します。アプリケーションの配備処理では、Enterprise Bean、Web コンポーネント、アプリケーション・クライアントの 3 つのタイプで内容が異なります。

Enterprise Bean を EJB コンテナに配備する時には、配備ツールは次のような作業を実行します。

1. WebOTX 固有の実行時情報をアプリケーション内に格納する。
2. 指定の EJB コンテナ・ホストにアプリケーションファイルを転送し、配備要求する。

   配備要求を受け付けた配備サービスは、EJB コンテナにインストールするための事前準備として、次を実行します。
3. Enterprise Bean のホームとリモートインタフェースの実装クラスを生成する。

   V6.3 の EJB 動作の既定値、RMI-IIOP のダイナミック・プロキシによる通信モードの場合は、これで配備サービスでの EJB コンポーネントの加工は完了です。
   従来のスタブ、スケルトンを介した RMI-IIOP 通信モードで配備した場合は、さらに次の(4)、(5)の処理が行われます。
4. その実装クラスからスタブとスケルトンを生成してコンパイルする。
5. クライアントアプリケーションで必要となるスタブを Web サーバにコードベースとして転送する。

※ スタブのコードベース機能は、V6.22 まで制限事項になっていましたが、パッチ（6.22.05）を適用することによって、V6.2 での制限が解除されました。

配備サービスで EJB コンポーネントの加工が完了すると、EJB コンテナは、その起動時に次を実行します。

6. Enterprise Bean のホーム・リファレンス、Bean の環境エントリ、リソース参照などを JNDI 名前サーバに登録する。
7. コンテナが永続化を管理する Entity Bean が存在する場合は、永続化のために使用するデータベース・テーブルを作成する。

Web を Web コンテナに配備する時には、配備ツールは次のような作業を実行します。

1. WebOTX 固有の実行時情報をアプリケーション内に格納する。
2. 指定の Web コンテナ・ホストに生成したファイルを転送し、配備要求する。

   配備要求を受け付けた Web コンテナは、その起動の初期化で次を実行します。
3. 配備命令に含まれるコンテンツ・ルートに対して Web コンポーネントを配置する。
4. アプリケーションのセキュリティ環境を設定する。
5. 環境エントリ、リソース参照、EJB 参照を JNDI 名前サーバに登録する。
6. Web アプリケーションの環境を設定する。

7. Web 配備記述子の指定にしたがって JSP ページをプリコンパイルする。

アプリケーション・クライアントは、単にスタブを含めてローカル・ディスクに出力されます。必要に応じてスタンドアロン型アプリケーションに添付して配布することになります。

8.5.リモートからアクセス可能な配備

デプロイヤーは通常、複数のアプリケーションを複数のコンテナ上に配備する必要があります。これをより簡単に行うためには、配備ツールがコンテナに対するクライアントとしてリモートからアクセスできる必要があります。

WebOTX 配備ツールは、ローカル、リモートを問わず EJB コンテナと Web コンテナにコンポーネントの配備ができます。コンテナに対しては、配備に関する標準プロトコルである JMXMP を使用してコンテナの運用管理バックエンドと通信します。

8.6.EJB コンポーネント動作時の TP システム最適化設定

配備ツールで設定できるEJBコンポーネントの実行時情報は、大半がEJB 2.1 仕様に明記される範囲に限られます。これは、コンポーネントの可搬性を尊重するためであり、アプリケーションサーバに特化した最適化設定は、仕様範囲外となります。

EJB コンポーネントの配備先が、Standard/Enterprise Edition である場合、EJB コンテナの基盤システムが WebOTX アプリケーションサーバ上となります。この運用環境内のアプリケーションでは、ミッションクリティカルなサーバとして堅牢性、ハイパフォーマンス、安定性が満たされます。

WebOTX アプリケーションサーバに付属する運用環境には、運用管理ツールが提供され、これを使うことでアプリケーションをさらに高信頼、OS ネイティブな詳細設定が可能になります。

9.アプリケーション開発

9.1.WebOTX開発環境とは

WebOTX V6 は高い生産性と保守性を持つ統合開発環境を提供します。デファクトスタンダードな統合開発環境 Eclipse をベースに、NEC で独自に開発した J2EE 1.4 対応アプリケーション開発機能として Web アプリケーション開発環境や、Web サービス開発環境、EJB 開発環境、OLF/TP アダプタ開発環境を提供します。ソース編集機能や CVS による版管理、JUnit によるテスト環境、WebOTX のテスト用サーバなどを兼ね備えており生産性を向上させることができます。

9.2.Webサービスアプリケーションの開発

既存の EJB-JAR ファイルや WAR ファイルや JAR ファイルを、ウィザードに答えていくだけで簡単に Web サービス化することができます。

開発手順

1. 既存の EJB-JAR ファイルや WAR ファイルや JAR ファイルや Eclipse プロジェクトを、Web サービス作成ウィザードに指定します。
2. Web サービスの設定をウィザードに答えていくと、Web サービスプロジェクトができます。
3. アーカイブ機能を使用して、Web サービス化された EJB-JAR や WAR ファイルを作成できます。
4. テスト用サーバに配備・テスト・デバッグします。

9.3.Webアプリケーションの開発

J2EE 1.4 に対応した Web アプリケーション（JSP、Servlet、Struts）を作成することができます。また、次のソースの雛形作成機能を持っています。

- Servlet
- JSP
- HTML
- Filter
- Listener
- Struts

web.xml 編集エディタを使用して、Web アプリケーションの配備記述子を編集することもできます。

開発手順

1. Web プロジェクトを作成します。プロジェクト作成時に、コンパイルに必要なライブラリが含まれます。
2. ウィザード（Servlet・JSP・HTML・Filter・Listener・Struts）を使用して、アプリケーションの雛形を作成します。
3. アーカイブ機能を使用して、WAR ファイルを作成します。
4. テスト用サーバに配備・テスト・デバッグします。

9.4.EJBアプリケーションの開発

J2EE 1.4 に対応した EJB 2.1 アプリケーションを開発することができます。ソースの雛形生成機能により、EJB 2.1 に対応した次の雛形ソースを作成することができます。

- ステートレス Session Bean
- ステートフル Session Bean
- コンテナ管理による永続性の Entity Bean (CMP)
- Bean 管理による永続性の Entity Bean (BMP)
- メッセージドリブン Bean

雛形ソースファイルを作成すると同時に、配備記述子（ejb-jar.xml）の雛形も自動的に作成します。

開発手順

1. EJBWeb プロジェクトを作成します。プロジェクト作成時に、コンパイルに必要なライブラリが含まれます。
2. ウィザードを使用して、EJB アプリケーションの雛形を作成します。
3. アーカイブ機能を使用して、EJB-JAR ファイルを作成します。
4. テスト用サーバに配備・テスト・デバッグします。

9.5.テスト用サーバ

WebOTX 開発環境には、Standard-J Edition 相当のテスト用サーバを含んでいます。そのため、WebOTX 開発環境を購入するだけで、J2EE 1.4 に対応した Web サービスアプリケーションや、Web アプリケーション（JSP・Servlet・Struts）や、EJB アプリケーションを、開発・テスト・デバッグすることができます。また、Developer’s Studio からテスト用サーバに対して、シームレスに配備・動作させることができます。

9.6.Developer’s StudioのベースであるEclipseとは

Eclipse とは、オープンソースの統合開発環境(IDE)です。多くの機能をプラグインと呼ばれる方法で実装しており、機能拡張が容易になっています。初めから Java の開発環境が同梱されていて、Java ソース用のエディタや、型階層ビュー、デバッグ用のビューなどが含まれています。商用の IDE に劣らない、さまざまな機能が盛り込まれています。Java だけの開発環境というわけではなく、C、C++、COBOL といったプログラミング言語の開発環境としても開発が進められています。

Eclipse は以下の特長を持っています。

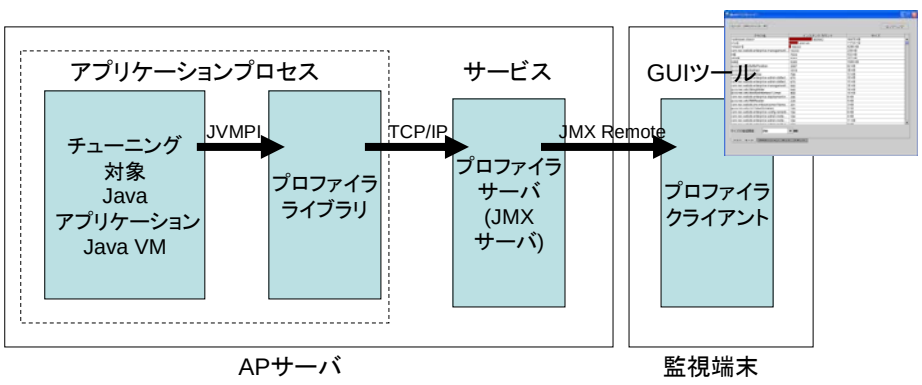
- CVS と連携して、チーム開発をサポート
- JUnit と連携して、効率的なテストを支援
- Ant と連携して、開発作業の自動化を支援
- ソースコードの自動補完、編集時のリアルタイムエラー検出
- ソースコードのリファクタリング
- ソースレベルのデバッグ機能

9.7.プロファイラ

WebOTX では、Java アプリケーションのチューニングを行なうためのプロファイリングツールを提供しています。このプロファイリングツールを使うことにより、CPU プロファイリングおよびヒーププロファイリング結果を GUI 画面上で確認することができ、性能ネックとなっている箇所、メモリをたくさん消費している箇所などの解析が容易に行なえます。

プロファイラの構成

WebOTX プロファイラは以下の構成となっており、JMX Remote プロトコルを通して、リモートの監視端末からプロファイリング結果を参照することが可能です。



CPUプロファイラ

CPU の関連情報を表示します。

- インボケーションツリー
  メソッド呼出関係によって生成されたツリー構造を表示します。
- ホットスポット
  各メソッドが使用している CPU の割合を表示します。
- メソッドグラフ
  ある時刻において、あるメソッドとそのメソッドが呼出すメソッドとの関係を静的に表示します。

ヒーププロファイラ

ヒープの関連情報を表示します。

- クラスモニタ
  各クラスのインスタンス数と、これらのインスタンスが使用しているメモリサイズを表示します。
- アロケーションホットスポット
  メソッドのメモリ配分情報を表示します。