WebSphere™ Application Server

WebSphere ビジネス構築のソリューション

バージョン 4.0

SD88-7362-01
(英文原典：SC09-4432-02)

WebSphere™ Application Server

WebSphere ビジネス構築のソリューション

バージョン 4.0

SD88-7362-01
(英文原典：SC09-4432-02)

ご注意

本書の情報およびそれによってサポートされる製品を使用する前に、 209ページの『特記事項』に記載する一般情報をお読みください。

この版は、SD88-7362-00 の改訂版です。

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

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

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

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

(URL は、変更になる場合があります)

原　典： SC09-4432-02
WebSphere™ Application Server
Building Business Solutions with WebSphere
Version 4.0

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2001.3

目次

図

表

本書について

本書では、インターネットおよびワールド・ワイド・ウェブ (WWW) 上でビジネスを行うためのインフラストラクチャーを、WebSphere Application Server™ を使用して作成する方法を説明します。本書は、WebSphere Application Server スタンダード版、アドバンスド版およびエンタープライズ版のアーキテクチャーおよび機能に関する説明を含み、また、各版でサポートする技術についても取り上げています。また、本書では、WebSphere Application Server を使用したビジネス・ソリューションの作成方法を示すサンプル・アプリケーションについても説明します。

本書の対象読者

本書は、終端相互接続の e-business システムの設計と構築を担当するソフトウェア・エンジニア、設計者、および管理者を対象としています。 WebSphere Application Server に関する予備知識は全く必要ありません。本書は、読者が従来のプログラミング概念、オブジェクト指向プログラミングおよびコンポーネントに精通していることを前提としています。

本書の構成

本書の構成は以下のとおりです。

第 1 部: *WebSphere Application Server の理解* では、WebSphere Application Server の 3 つの版と、各版がサポートする技術について説明します。

- 5ページの『第1章 WebSphere Application Server スタンダード版』では、WebSphere Application Server スタンダード版の機能とアーキテクチャーについて説明します。
- 11ページの『第2章 WebSphere Application Server アドバンスド版』では、WebSphere Application Server アドバンスド版の機能とアーキテクチャーについて説明します。
- 19ページの『第3章 JavaServer Pages の使用』では、WebSphere Application Server による Sun JavaServer Pages™ (JSP) 仕様のインプリメンテーションについて説明します。
- 27ページの『第4章 サーブレットの使用』では、WebSphere Application Server によるサーブレットのインプリメンテーションについて説明します。

- 35ページの『第5章 Enterprise Beans の使用』では、Sun Enterprise JavaBeans™ 仕様の WebSphere Application Server のインプリメンテーションについて説明します。
- 75ページの『第6章 Web アプリケーションの開発』では、JSP ページ、Enterprise Beans、およびサーブレットを使用して、ワールド・ワイド・ウェブ (WWW) 上でアクセス可能なビジネス・アプリケーションの作成方法について説明します。
- 85ページの『第7章 WebSphere Application Server エンタープライズ版』では、WebSphere Application Server エンタープライズ版の機能とアーキテクチャーについて説明します。
- 93ページの『第8章 Component Broker の使用』では、IBM® Component Broker について説明します。
- 121ページの『第9章 TXSeries の使用』では、IBM TXSeries™ について説明します。

第 2 部: *WebSphere Application Server の使用* では、WebSphere Application Server を使用して、ビジネス・システムをインプリメントするための各種方法を説明するサンプル・アプリケーションについて述べます。

- 157ページの『第10章 サンプル・アプリケーションの概要』では、サンプル・アプリケーションのシナリオおよびその目的について説明します。
- 161ページの『第11章 サンプル・アプリケーションの設計』では、サンプル・アプリケーションの設計方法を説明します。
- 167ページの『第12章 サンプル・アプリケーションのインプリメント』では、WebSphere Application Server アドバンスド版およびエンタープライズ版のさまざまな機能を使用して、サンプル・アプリケーションをインプリメントする方法について説明します。
- 179ページの『第13章 サンプル・アプリケーションの技術的な詳細』では、サンプル・アプリケーションの各コンポーネントに関する技術的な詳細について説明します。
- 203ページの『第14章 サンプル・アプリケーションの拡張』では、WebSphere の追加製品および追加機能を使用して、サンプル・アプリケーションを拡張するためのいくつかの方法について説明します。

関連情報

本書で説明するトピックの詳細については、以下の資料を参照してください。

- *WebSphere Application Server 概説*
- *Enterprise Beans の作成*
- Component Broker プロダクト資料
- TXSeries Encina® および CICS® プロダクト資料
- IBM VisualAge™ for Java™ プロダクト資料

本書で使用されている規則

WebSphere Application Server エンタープライズ版の資料では、次の表記上の規則とキー入力の規則が使用されています。

表 1. 本書で使用する規則

規則	意味
太字	コマンド名を示します。グラフィカル・ユーザー・インターフェース (GUI) については、メニュー、メニュー項目、ラベル、およびボタンも示します。
モノスペース	コマンド・プロンプトに記述どおりに入力しなければならないテキストと、記述どおりに使用しなければならない値を示します。コマンド、関数、およびリソース定義属性とその値などが含まれます。また、画面上のテキストやコードの例も示します。
イタリック	自分で指定しなければならない可変値を示します (たとえば、 *fileName* の部分にはファイル名を指定する必要があります)。また、強調目的の場合や資料の名称を示す場合もあります。
Ctrl-*x*	制御文字シーケンスを示します (*x* はキーの名前)。たとえば、Ctrl-c は、 Ctrl キーを押さえたままで c キーを押すことを示します。
Return	Return、Enter、または左矢印が書かれたキーを表します。
%	**root** 特権を必要としないコマンドの UNIX コマンド・シェル・プロンプトを表します。
#	**root** 特権を必要とするコマンドの UNIX コマンド・シェル・プロンプトを表します。
C:¥>	Windows NT® のコマンド・プロンプトを表します。

表 1. 本書で使用する規則 (続き)

規則	意味
>	メニューの説明で使用されている場合は、一連のメニュー選択項目を示します。たとえば、「**「ファイル (File)」>「新規作成 (New)」**をクリックします。」は、**「ファイル (File)」**メニューの**「新規作成 (New)」**コマンドをクリックするという意味です。
	ツリー・ビューの説明で使用されている場合は、一連のフォルダーまたはオブジェクトの展開を示します。たとえば、「**「管理ゾーン (Management Zones)」> 「サンプル・セルおよびワークグループ・ゾーン (Sample Cell and Work Group Zone)」> 「構成 (Configuration)」**を展開します。」には、次のような意味があります。 1. 「管理ゾーン (Management Zones)」フォルダーを展開します。 2. 「サンプル・セルおよびワークグループ・ゾーン (Sample Cell and Work Group Zone)」という名前の管理ゾーンを展開します。 3. 「構成 (Configurations)」フォルダーを展開します。 **注:** 画面中のオブジェクトの隣に正符号 (**+**) があると、そのオブジェクトを展開できます。オブジェクトを展開すると、正符号が負符号 (**-**) に置き換えられます。
+	ツリー構造を展開すると、より多くのオブジェクトを表示できます。展開するには、オブジェクトの隣にある正符号 (**+**) をクリックします。
-	ツリー構造の分岐を縮小して、その分岐に含まれるオブジェクトが表示されないようにします。ツリー構造の分岐を縮小するには、分岐の先頭のオブジェクトの隣にある負符号 (**-**) をクリックします。
コマンドの入力	コマンドを「入力する」または「実行する」ように指示される場合には、コマンドをタイプしてから Return を押します。たとえば、「**ls** コマンドを入力する。」という指示があった場合には、 **ls** をコマンド・プロンプトにタイプしてから Return を押します。
[ ]	構文記述中のオプション項目を囲みます。
{ }	構文記述中で必須選択項目のリストを囲みます。
|	構文記述中の中括弧 (**{ }**) で囲まれた選択項目のリスト中の項目を分離します。
...	省略記号。構文記述中の省略記号は、前述の項目を 1 回以上繰り返せることを示します。例の中の省略記号は、簡潔にするためにその情報が例から省略されたことを示します。
IN	関数の説明の中で、データを関数に渡す場合、値が使用されるパラメーターを示します。このパラメーターは、変更されたデータを呼び出し側のルーチンに戻す場合には使用しません。 (作成するコードの中には IN 宣言を含めないでください。)

表 1. 本書で使用する規則 (続き)

規則	意味
OUT	関数の説明の中で、変更されたデータを呼び出し側のルーチンに戻す場合、値が使用されるパラメーターを示します。このパラメーターは、データを関数に渡す場合には使用しません。 (作成するコードの中には OUT 宣言を含めないでください。)
INOUT	関数の説明の中で、値が関数に渡され、関数によって変更された後、呼び出し側のルーチンに戻されるパラメーターを示します。このパラメーターは、IN パラメーターとしてかつ OUT パラメーターとして機能します。 (作成するコードの中には INOUT 宣言を含めないでください。)
$CICS	CICS プロダクトのインストール先の絶対パス名を示します。たとえば、Windows NT の場合は C:¥opt¥cics、 Solaris の場合は /opt/cics です。環境変数 CICS がプロダクトのパス名に設定されている場合は、記載されているとおりに例を使用することができます。そうでない場合、$CICS 部分はすべて CICS のプロダクトのパス名に置き換えてください。
CICS on Open Systems	サポートされているすべての UNIX プラットフォーム用の CICS プロダクトを指します。
TXSeries CICS	CICS for AIX、CICS for Solaris、および CICS for Windows NT プロダクトを指します。
CICS	CICS on Open Systems プロダクトと CICS for Windows NT プロダクトの総称です。複数の CICS on Open Systems プロダクト間の違いを強調する場合は、特定バージョンの CICS on Open Systems プロダクトに言及します。CICS ファミリー中の他の CICS プロダクトは、オペレーティング・システム別に区別されます。たとえば、CICS for OS/2 や、 ESA、MVS、および VSE プラットフォーム用の IBM メインフレーム・ベース CICS という具合です。

第1部 WebSphere Application Server の理解

IBM WebSphere Application Server は、単純な Web サイト・ストアフロントから、企業のコンピューティング・インフラストラクチャーの全面的な変更まで、広範囲にわたる次世代ビジネス・アプリケーションを設計、インプリメント、展開、管理するための包括的なミドルウェア・パッケージを提供します。

WebSphere Application Server は、エレクトロニック・コマース (電子商取引) への展望を開く鍵であり、高性能な Web サイトを構築、管理し、それらを新しい、または、既存の非 Web ビジネス・システムと統合するのに役立つ一連のソフトウェア・プロダクトおよびアーキテクチャーです。 WebSphere Application Server は以下のような企業を対象としています。

- Web 上で強力な地位を確立するため、または、現在の地位を高めるため、最新のテクノロジーを使用する企業
- 分散ビジネス・システムおよびアプリケーションを組織全体にわたって展開する企業
- Web を、非 Web システムおよびアプリケーションに統合する企業

WebSphere Application Server は、以下の環境を実現し、エンタープライズ・クラスのコンポーネント・サーバー・ソリューションの基本エレメントを提供します。

- 設計開発チーム内の役割が自然に明確化される。
- オブジェクト管理をシステムに代行させることにより、設計担当者およびプログラマーの負担を軽減する。
- オペレーティング・システムを利用して、最適なパフォーマンスと最適な規模を実現し、保全性とセキュリティーを提供する。

WebSphere Application Server では、以下を提供することにより、これらの環境を実現します。

- Java 2 プラットフォーム Enterprise 版 (J2EE™) 仕様、共通オブジェクト・リクエスト・ブローカー・アーキテクチャー (CORBA) といった業界標準に基づくオブジェクト管理のためのフレームワークおよびランタイム・インプリメンテーション。
- プログラム・リソースを構成し、フレームワーク・サービスの統合に必要なコードを生成するビジュアル開発ツール

- 各種の永続的なデータ記憶装置をサポートし、既存のデータおよびトランザクション・システムからのオブジェクト生成を可能にするアプリケーション・アダプターおよびコネクター
- オブジェクトをサーバーに関連付け、システムへの更新を調整するのに役立つシステム管理ツール

開発者は、WebSphere プロダクト・ファミリーの統合プログラミング・モデルに基づき、さまざまなプラットフォームおよびアプリケーション・サーバー上で展開可能なシステムを開発することができます。このようなパラダイムによって、コードの再利用が可能になり、コンピューティング・インフラストラクチャーへの投資額は保証されます。

WebSphere の多階層分散設計では、プレゼンテーション、ビジネス・ロジック、データ・リソースがそれぞれ異なる層に分けられます。そのため、さまざまな種類のクライアントを使用して、いつでもどこからでも、エンタープライズ・ビジネス・アプリケーションおよびデータにアクセスすることができます。

WebSphere では、新たに開発されたコード、既存のプロシージャー・アプリケーション、企業内のデータを再利用することができます。WebSphere を使用することにより、開発、導入した EJB アプリケーションを移植できるため、システムにスケーラビリティーを提供し、非 WebSphere 環境との相互運用性を実現します。また、WebSphere Application Server 間の相互運用性により、システムを単純なビジネス・アプリケーションから、高性能のアプリケーション・サーバー上で稼働する複雑な企業規模のシステムへと発展させることが可能です。WebSphere は各種のハードウェアおよびオペレーティング・システム・プラットフォームをサポートしているため、企業は自らのニーズに合ったプラットフォームを選択することができます。

WebSphere Application Server は、Java™ 2 プラットフォーム Enterprise 版仕様に準拠しています。 J2EE は、多階層アプリケーションの開発に伴うコストおよび複雑さを軽減し、組織の要求にあわせて、アプリケーションを容易に展開および拡張することができます。J2EE は、多階層アプリケーション開発用の標準化モデル、および多階層アプリケーションをホスティングするためのプラットフォームを含むアーキテクチャーを定義します。 J2EE は、Java クライアント、アプレット、サーブレット、JavaServer Pages (JSP pages)、および Enterprise Beans をサポートします。J2EE の標準サービスには、HTTP、HTTPS、Java Transaction API (JTA)、 Java データベース・コネクティビティ

ー (JDBC) API、Java メッセージ・サービス (JMS) API、 Java Naming and Directory Interface (JNDI) API および J2EE コネクター・アーキテクチャーなどが含まれます。

本書の第 1 部では、WebSphere Application Server の機能および可能性について概説します。以下の項目について、説明します。

- 5ページの『第1章 WebSphere Application Server スタンダード版』
- 11ページの『第2章 WebSphere Application Server アドバンスド版』
- 19ページの『第3章 JavaServer Pages の使用』
- 27ページの『第4章 サーブレットの使用』
- 35ページの『第5章 Enterprise Beans の使用』
- 75ページの『第6章 Web アプリケーションの開発』
- 85ページの『第7章 WebSphere Application Server エンタープライズ版』
- 93ページの『第8章 Component Broker の使用』
- 121ページの『第9章 TXSeries の使用』

155ページの『第2部 WebSphere Application Server の使用』では、WebSphere Application Server の機能と可能性を、ビジネス・アプリケーションに適用する方法について説明します。

第1章 WebSphere Application Server スタンダード版

WebSphere Application Server スタンダード版 (スタンダード版 Application Server) は、サーバー側のビジネス・アプリケーションの移植性と、Java テクノロジーの高度な性能および管理の容易性を組み合わせることにより、Java ベースの Web アプリケーション開発のための広範なプラットフォームを提供しています。これにより、エンタープライズ・データベースおよびトランザクション・システムとの強力な対話が可能となり、実行時環境、開発環境およびシステム管理環境を提供します。

スタンダード版 Application Server は、アクティブな Web サイトや、サーブレットなどの Web アプリケーションの作成に使用されます。本節では、スタンダード版 Application Server の以下の点について説明します。

- 『スタンダード版 Application Server の機能』
- 6ページの『実行時およびシステム管理アーキテクチャー』
- 9ページの『スタンダード版アプリケーション開発環境』

スタンダード版 Application Server の詳細については、スタンダード版「インフォセンター」を参照してください。下記の WebSphere Application Server ライブラリーで参照することができます。
www.software.ibm.com/webservers/appserv/library.html.

スタンダード版 Application Server の機能

スタンダード版 Application Server は、単一マシンの Web アプリケーション・サーバーです。一般的なスタンダード版 Application Server のインストールでは、Web サイトをサポートします。アプリケーションを作成するには、開発者は、拡張 Hypertext Markup Language (HTML) コンテンツをインプリメントする必要があります。開発者は、Web サイトの作成とテストを行い、アクティブ・サイトにアップデートを発行します。

スタンダード版 Application Server は、HTML タグの拡張セットを提供し、アプリケーション開発ツールは、このタグによるページ作成 (スクリプト記述) をサポートしています。サポートされるコンテンツとページ・スタイルは、以下のとおりです。

- イメージ・クリップ、サウンド・クリップ、およびビデオ・クリップが組み込まれた HTML。

- クライアント・サイドのスクリプト (JavaScript など) が組み込まれた HTML。
- JavaServer Pages (JSP)。拡張セットのマークアップ・タグを持つアクティブな HTML ページです。 JSP タグを使用することにより、プログラマーはデータ駆動型のページ・コンテンツとページ・フローを開発できます。詳細については、19ページの『第3章 JavaServer Pages の使用』を参照してください。
- Sun Microsystem の サーブレット仕様に従い記述されたサーブレット。スタンダード版 Application Server には、サーブレット・エンジンが備えられています。詳細については、27ページの『第4章 サーブレットの使用』を参照してください。
- Sun Microsystem の JavaBeans™ 仕様に従い記述された JavaBeans コンポーネント。
- Sun Microsystem の Enterprise JavaBeans (EJB) 仕様に従い記述された Enterprise Beans。詳細については、35ページの『第5章 Enterprise Beans の使用』を参照してください。
- Extensible Markup Language (XML) ファイル。これは、構成データや状態情報などの管理情報を保管するために使用するファイルです。

WebSphere Application Server がサポートする Web アプリケーションの一般的なアーキテクチャーについての詳細は、75ページの『第6章 Web アプリケーションの開発』を参照してください。

実行時およびシステム管理アーキテクチャー

7ページの図1 に、スタンダード版 Application Server の実行時アーキテクチャーを示します。

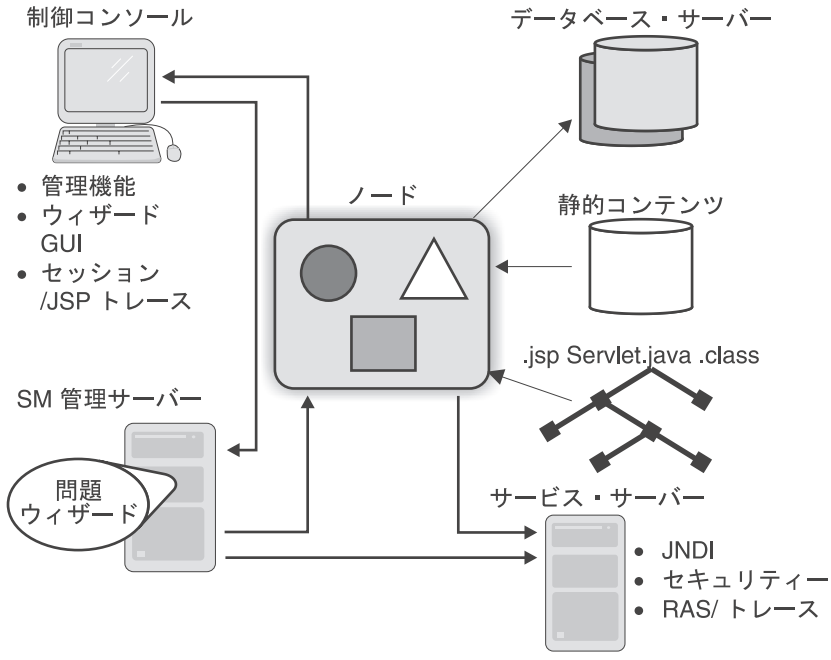

図 1. WebSphere スタンダード版 Application Server 実行時アーキテクチャー

スタンダード版 Application Server ランタイムおよびシステム管理のコンポーネントを以下に示します。

注: ほとんどのランタイムおよびシステム管理機能が、内部にインプリメントされています。プログラマーと管理者に対して、これらの機能が提示されることはほとんどありません。

Web アプリケーション・サーバー

Web アプリケーション・サーバーは、配置済みサーブレット、およびコンパイルされた配置済み JSP ページの実行時環境です。また、サーブレットのコンテナーもインプリメントしています。Web アプリケーション・サーバーは、サービス・サーバーと統合して、管理するリソースのセキュリティーを確保しています。これはマルチスレッド化されており、着信要求を管理するための基本スレッド管理とスケジューリング・ポリシーもインプリメントされています。

Web アプリケーション・サーバーは、Java データベース・コントロール (JDBC™) および XA 準拠データベースと統合された Java トランザクション・

サービス (JTS) のインプリメンテーションを提供します。また、Web アプリケーションとデータベース間の対話を改善する接続マネージャーも提供しています。

Web サーバー

スタンダード版 Application Server は、IBM、Netscape および Apache Web サーバーなど、各種多数の Hypertext Transfer Protocol Daemon (HTTPD) サーバーと統合されます。また、Web サーバーを強化する 2 つのプラグイン拡張機能も提供します。

- **サービス** − サービス・プラグインにより、Web サーバーは、スタンダード版 Application Server ランタイムが使用するものと同じセキュリティー、ネーミング、およびトレース・サービスを使用できます。
- **NCF** − NCF プラグインにより、Web サーバーは、管理者により Web アプリケーション・サーバーに定義されたフィルターに一致する URL を経路指定します。この URL は、サーブレット・インスタンスまたは JSP ページ (自動的にコンパイルされ、サーブレットとして実行される) のいずれかを表します。JSP ページを明示的に管理する必要はありません。NCF プラグインは、Object Request Broker (ORB) を使用して、Web アプリケーション・サーバーを呼び出し、サーブレットおよび JSP ページを実行します。ORB では、システム内のパフォーマンスの最適化をサポートしており、Web アプリケーション・サーバーの呼び出し時におけるユーザー識別など、Web サーバーのコンテキスト情報を伝達します。スタンダード版 Application Server ランタイム、アドバンスド版 Application Server ランタイム、および ORB は、これらのスレッドが Web サーバーにより制御されるスレッドの下にある Web サーバー内で実行されるよう、オプションとして構成することが可能です。

システム管理サーバー

スタンダード版 Application Server システム管理 (SM) サーバーは、サーブレット、 JSP アプリケーションおよびそのデータを Web サイトの管理全般に統合するための、単一サイト・ソリューションを提供します。SM サーバーの機能には、以下があります。

- **ユーザー管理およびグループ管理** − これらの機能は、Web サイト・ユーザーの管理と、オプションでのオペレーティング・システム・ユーザー ID およびグループの管理をサポートします。ユーザー、グループ、およびセキュリティーの管理も、ネイティブの Web サーバーおよびオペレーティング・システム管理ツールによって行うことができます。

- **許可管理機能** − この機能により、管理者は URL、サーブレット、JSP ページ、および Enterprise Bean ホームに対するアクセスを制御することができます。許可管理機能は、オプションとなっています。
- **定義と配置** − これらの機能は、Enterprise Beans と Java アーカイブ (JAR) ファイル、サーブレット・インスタンス、および JSP ページのインストールと配置をサポートします。特定 URL およびディレクトリーにファイルを配置することにより、定義と配置が起動されます。Enterprise Bean のサポートは、明示的に使用可能にする必要があります。デフォルトでは、名前付きサーブレットとそのプロパティーのみが管理されます。
- **操作** − この機能により、管理者はサービス・サーバーの起動と停止、サーブレットの停止、およびデータベース接続とトランザクションの管理を行うことができます。
- **問題判別ウィザード (Apache Web サーバーのみ)** − Apache Web サーバーと統合された Web アプリケーション・サーバーは、問題判別ウィザードを使用して、トレース・ログの読み取り、トレースや操作ログのマージ、問題の検出、および修正アクションの推奨を行うことができます。

サービス・サーバー

サービス・サーバーは、スタンダード版 Application Server の Web サーバー・プラグインや SM サーバーで必要とされる、ユーティリティー機能とアプリケーションをインプリメントします。これには、認証、許可、トレースや監査、ネーム・サーバー、およびトレース・サービスをサポートするセキュリティー・アプリケーションが含まれています。これらのアプリケーションおよびサービスは、個別のサーバー、または Web アプリケーション・サーバー自体で実行するよう構成することができます。

このサーバーの存在とその操作は、システム管理者およびアプリケーション開発者の側では特に意識する必要はありません。

スタンダード版アプリケーション開発環境

WebSphere Studio は、スタンダード版 Application Server 用のアプリケーション開発環境です。これは多目的の Web サイト開発ツールで、個人の Web ページから、e-business アプリケーションのフロントエンドとなる Web サイトまであらゆるサイトの作成に使用することができます。WebSphere Studio は、HTML コンテンツに適したツール・セットを提供し、その他のコンテンツ開発ツールと統合できます。

スタンダード版 Application Server アプリケーション開発環境は、以下をサポートします。

- **静的な HTML およびアクティブな HTML** − 多くの URL は、静的なコンテンツ (ハイパーテキスト・リンク、組み込みピクチャー、サウンドやビデオが含まれた HTML ファイル) にマップしています。HTML ファイルには、クライアントによりアクティブに生成されるものもあり、サーバーによりアクティブに生成される HTML ファイルもあります。クライアントのアクティブな HTML は、ブラウザーで実行されるクライアント側のスクリプト (JavaScript スクリプトやアプレットなど) を含みます。サーバーのアクティブな HTML は、JSP ページ、または明示的にコーディングされたサーブレットである場合があります。
- **サーブレット** − サーブレットを WebSphere Studio で開発し、スタンダード版 Application Server で実行することができます。サーブレットは、そのソース・コードが変更されると、自動的に再ロードされます。名前付きのサーブレット・インスタンスは、その初期化値を使用して SM サーバーによって明示的に構成されます。 WebSphere Application Server を使用してサーブレットをインプリメントする方法の詳細については、27ページの『第4章 サーブレットの使用』を参照してください。
- **JSP ページ** − リレーショナル・データベースへのアクセスは、すべて、ツールにより生成された Bean 経由で行われます。JSP 開発者は、JSP ページ内の Bean マークアップ・タグによって Bean にアクセスします。JSP ファイルは、アプリケーション開発ツールによって作成される場合は、暗黙的にコンパイル、配置、およびインストールされます。 WebSphere Application Server を使用して、JSP ページをインプリメントする方法についての詳細は、19ページの『第3章 JavaServer Pages の使用』を参照してください。
- **データベース・サポート** − スタンダード版 Application Server は、拡張 HTML タグ・セットを定義して、以下のデータベース操作を実行します。
  - テーブルのシグニチャー定義 (たとえば、列名および値)
  - 列の値の URL データ型
  - 名前付きテーブルでの作成、除去、更新、および削除 (CRUD) 操作
  - 基本照会サポート
  - 照会結果からのテーブルのフォーマット

第2章 WebSphere Application Server アドバンスド版

WebSphere Application Server アドバンスド版 (またはアドバンスド版 Application Server) は、スタンダード版 Application Server 上で構築されます。これは、Enterprise JavaBeans (EJB) 仕様 (ワークロード管理などのサービスのサポートを含む) で作成されたアプリケーション用に、複数マシン上でのサーバー機能を導入し、既存のビジネス・システムへの Web アプリケーションとのより緊密な統合を可能にしています。

アドバンスド版 Application Server の詳細については、アドバンスド版「インフォセンター」を参照してください。これは、下記の WebSphere Application Server ライブラリーで参照することができます。
www.software.ibm.com/webservers/appserv/library.html.

アドバンスド版 Application Server の機能

アドバンスド版 Application Server では、スタンダード版 Application Server の機能を拡張し、スケーラブルなトランザクション型の Web アプリケーションを開発することができます。アドバンスド版 Application Server では、単一マシン上で稼働するスタンダード版 Application Server とは異なり、複数マシン上で実行することができます。これは、ビジネス・ロジックをインプリメントし、永続データにアクセスする Enterprise Beans、分散システム管理、および共有名、セキュリティーおよびトランザクション・サービスをサポートします。

スタンダード版 Application Server の機能の詳細は、5ページの『第1章 WebSphere Application Server スタンダード版』を参照してください。

実行時およびシステム管理アーキテクチャー

アドバンスド版 Application Server 実行時環境では、スタンダード版 Application Server 実行時環境の単一マシン・サーバーの他に、クラスター・システムと分散システムがサポートされます。12ページの図2 には、ハイレベルな視点からのアドバンスド版 Application Server システム体系を示します。

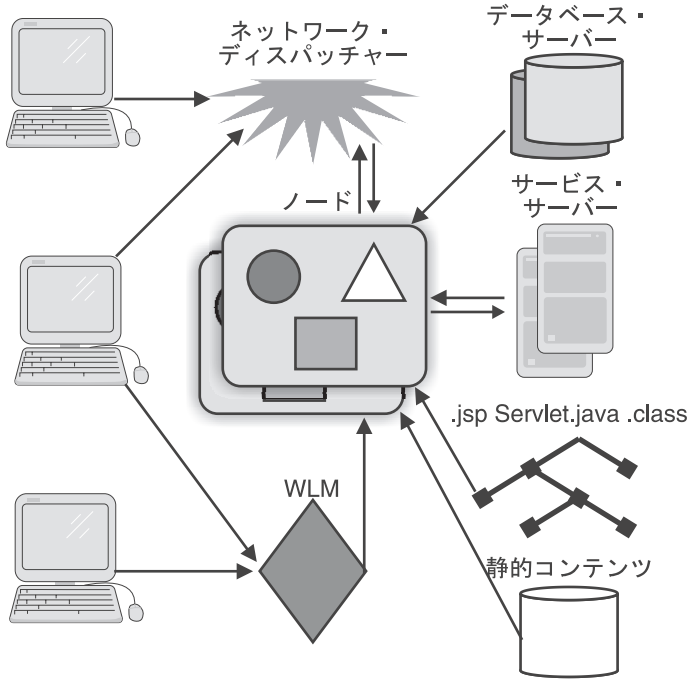

図 2. ハイレベルな視点からのアドバンスド版 Application Server システム体系

アドバンスド版 Application Server システムにおけるエレメントは、以下のとおりです。

トポロジー

アドバンスド版 Application Server システムのトポロジーには、以下の 3 つのエレメントがあります。

ノード

ノードは、アドバンスド版 Application Server ランタイムをインストールする物理システムを表します。ノードでは、複数インスタンスの Web アプリケーション・サーバーを定義してアクティブにできます。各インスタンスは、単一でマルチスレッドのオペレーティング・システム・プロセスです。13ページの図3 に一般的なノードの構成を示します。

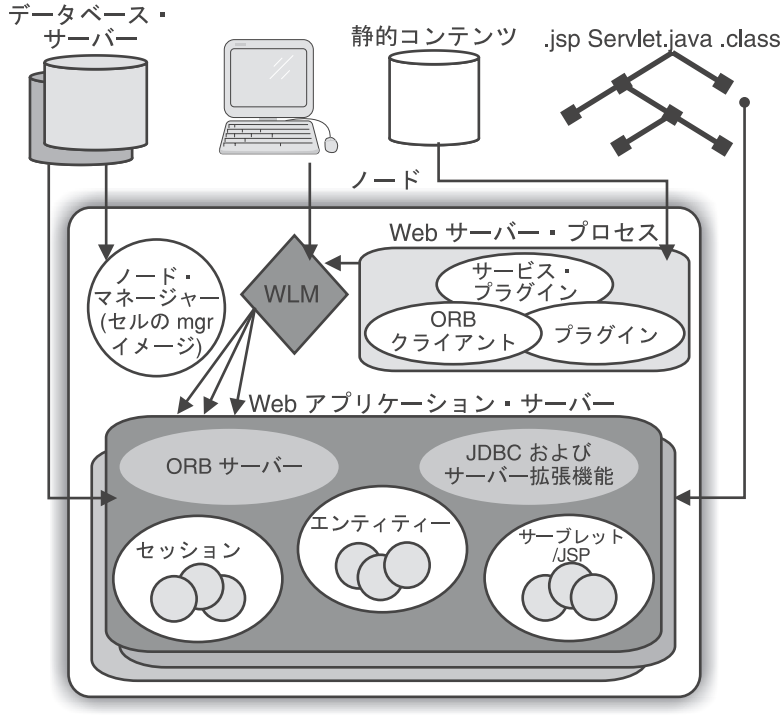

図 3. アドバンスド版 Application Server システムの一般的なノード

サーバー・グループ

サーバー・グループは、Web アプリケーション・サーバー・インスタンスの名前付き集合で、オプションとして異なるホスト上に構成できます。グループ内のすべてのインスタンスは、同じデータベースとファイル・システムに接続されます。異なるサーバー・グループの Web アプリケーション・サーバー・インスタンスを同じノードに構成することができます。

可用性、パフォーマンス、および管理の容易性などの必要性から、管理者はサーバー・グループを定義します。サーバー・グループ内のすべての Web アプリケーション・サーバー・インスタンスは、同じサーブレット、JSP ページおよび Enterprise Beans を実行する複製であり、同じポリシーで構成された同じコンテナーを持ちます。サーバー・グループは、クライアント・アプリケーション、サーバー・グループ内のアプリケーション・ロジック、およびその他のサーバー・グループによって、単一の Web アプリケーション・サーバー・インスタンスと見なされます。

サーバー・グループ内の各 Web アプリケーション・サーバー・インスタンスには名前があります。たとえば、Underwriting ApplicationServerGroup では、マシン pinnacle.uas.com 上に UAS1 というインスタンスと、マシン kaitlin.uas.com 上に UAS2 というインスタンスをとることができます。

セル

セルは、ノードの名前付き集合です。1 つのノードは、1 つのセル内にのみ存在できます。サーバー・グループは、セルをまたぐことはできません。

一般的に、セル内のノードは、同じファイル・システムとデータベースにアクセスすることはありません。ただし、サーバー・グループのサーバーがインストールされるノードはすべて、以下のリソースにアクセス可能でなければなりません。

- HTML、サーブレット、JSP ページ、および Java クラスを保持するファイル・システム。
- サーバー・グループに配置された Enterprise Beans により参照されるデータベースおよびデータベース・サーバー。

サーバー・グループのメンバーシップが変更された場合、セル・マネージャーは、そのすべてのアクティブ・ノードのマネージャーに通知する必要があります。

ワークロードおよび可用性の管理

ワークロード管理は、クライアント要求の分散を最適化します。着信作業要求は、最も効率的に要求の処理を行うことのできるアプリケーション・サーバーおよびその他オブジェクトに分散されます。ワークロード管理はまた、フェイルオーバー (サーバーが利用不能の場合にクライアント要求を他のサーバーにリダイレクトする) も提供します。

複製 (アプリケーション・オブジェクトのレプリカ) およびモデル (複製を作成するためのテンプレート) を使用して、Enterprise Beans やサーブレットなどのオブジェクトの複数のインスタンスや、アプリケーション・サーバー全体を作成することができます。アドバンスド版 Application Server は、Enterprise Beans の複製および EJB サーバーのワークロードを自動的に管理します。サーブレットのリダイレクトにより、クライアントの要求を、ローカルあるいはリモートのサーブレットの複製に発送し、個々のサーブレットの複製にかかる負荷を平衡化します。管理サーバーも、ワークロード管理に加えることができます (フェイルオーバー・サポート用)。

トランザクション、セッション、およびワークロードの平衡化は、障害のバイパスのための基本であり、可用性を向上させます。Web アプリケーション・サーバー・インスタンスに障害が発生した場合、ノード・マネージャーは障害を検出して、そのプロセス中でアクティブなセッションとトランザクションをロールバックし、プロセスを再始動します。また、ノード・マネージャーは、障害が発生したセルの管理プログラムに通知をします。

オブジェクト・トランザクション・モニター (OTM) により、サーバー・グループにワークロードおよび可用性管理をインプリメントします。 OTM は、サーバー・グループの Web アプリケーション・サーバー・インスタンスにおけるトランザクションおよびセッションのバランスをとります。トランザクション型、およびセッション型ではない要求は、ランダムにインスタンスに経路指定されます。

システム管理

システム管理 (SM) ツールは、以下の追加サービスを提供することにより、スタンダード版 Application Server のシステム管理モデルを拡張します。

- **セル・マネージャー** – セル・マネージャーは、セル内の自らのサーバー・グループに常駐し、Enterprise Bean トランザクション、リレーショナル・データベース・パーシスタンス、および OTM を使用することにより、スケーラビリティーと可用性を実現します。セル・マネージャー・インスタンス (セル・マネージャー・アプリケーションを実行するサーバー・プロセス) は、サーバー・グループ内のいずれのノードにも割り当てることができます。ただし、通常、これは少数のノードに統合されます。
- **ノード・マネージャー** – ノード・マネージャーは、ローカル・サーバー・インスタンスおよび構成情報の管理を分担します。ノード・マネージャーは、セル・マネージャーに代わって操作を実行します。セル・マネージャーおよびノード・マネージャーともに、Enterprise Bean アプリケーションとしてインプリメントされます。

ノード・マネージャーおよびセル・マネージャーは、システム・アクティビティーのモニターも行い、可用性を検査します。また、使用可能な場合には、クラスター化したオペレーティング・システム・メンバーシップのサービスを使用することもできます。

システム管理ツールは、Web アプリケーション・サーバーおよびサーバー・グループの定義、構成、および操作もサポートします。

- **単一ポイント管理** – この機能により、SM コンソールからリモート・ノードを管理します。

- **単一論理イメージ** — この機能は、サーバー・グループ・レベルで実行される操作 (開始および停止など) を、グループ内の個々のサーバー・インスタンスでの操作にマップします。これにより、グループ内のそれぞれの複製に同じ操作を何度も行う必要がなくなります。

オブジェクト・サービスおよびサポート・サービス

オブジェクト・サービスおよびサポート・サービスは、企業内のすべてのノードとサーバー・グループ間で共用されます。これには、以下が含まれます。

名前サービス

企業内のアプリケーション・サーバー・クラスターは、グローバルのネーム・スペースを共用します。ネーム・サーバーは、ネーム・スペースをインプリメントし、企業内には複数のネーム・サーバーを置くことができます。

RAS サービス

信頼性、可用性、および保守容易性 (RAS) サービスにより、企業全体に及ぶランタイム、アプリケーション RAS、およびトレース用のログを表示することができます。アプリケーションおよびシステムの RAS レコードは、ノード固有のログに記録され、ノード・マネージャー、セル・マネージャー、および RAS サービスが共同して、RAS 情報およびトレース情報を要求時に集中的に表示することができます。

セキュリティー・サービス

セキュリティー・サービスにより、ユーザー・レジストリー情報へのアクセス、および個々の Web アプリケーション・サーバーに対する許可規則が提供されます。これにより、ローカル認証と許可が可能になります。

トランザクション

アドバンスド版 Application Server は、分散トランザクションおよびクライアント境界指定トランザクションをサポートします。

パーシスタンス

パーシスタンスは、Java Database Connectivity (JDBC) と互換性のある外部のデータベースへの呼び出しによりインプリメントされます。このデータベースは、Web アプリケーション・サーバー・インスタンス間で共用できます。また、アドバンスド版 Application Server は、他のアプリケーションで共用されるデータベースもサポートします。

これらのサービスの詳細については、56ページの『オブジェクト・サービス』を参照してください。

アドバンスド版アプリケーション開発環境

アドバンスド版 Application Server は、このサーバーに組み込まれている WebSphere Studio、VisualAge for Java、およびその他の Java や HTML の開発ツールを使用して開発することができます。

アプリケーション・モデル

アドバンスド版 Application Server のアプリケーション・モデルは、リレーショナル・データベース・システムに支援された、オブジェクト指向のビジネス・ロジックです。アプリケーションは、Web 駆動型のシンまたはシック・クライアントに統合できます。また、アドバンスド版 Application Server は、アプリケーション・サーバーで稼働する既存の呼び出し手続き型アプリケーションとの統合も制限付きでサポートします。

アプリケーションには、一般的に以下の 4 つのパーツがあります。

- HTML および JSP ページは、ユーザー・インターフェースと、アプリケーションのフォーム・フローを提供します。
  - JSP ページは、Enterprise Beans により状態データにアクセスします。JSP タグは、JavaBeans コンポーネントへのアクセスに使用され、コンポーネントは Enterprise Beans にアクセスします。 (アプリケーション開発ツールは、JSP タグを使用して Enterprise Beans にアクセスするための JavaBeans コンポーネントを生成します。)
  - パフォーマンス向上のため、ステートレスな JSP ページを使用しなければなりません。アドバンスド版 Application Server は、クラスター・サーバーをサポートし、パフォーマンスおよび可用性を向上させていますが、JSP の状態情報をキャッシュすると、この利点が失われてしまいます。その代わり、アドバンスド版 Application Server には、クラスター内でのセッション Bean データへのアクセス、および JSP ページの保管を行うためのタグがあります。

  JSP ページの詳細については、19ページの『第3章 JavaServer Pages の使用』を参照してください。
- Enterprise Beans は、ビジネス・ロジック、トランザクション操作、およびデータベースへのアクセスをインプリメントします。この Bean は、Web アプリケーションと、非 Web コンピューター・システム間の橋渡しの役割を果たします。詳細については、35ページの『第5章 Enterprise Beans の使用』を参照してください。

- サーブレットは、アプリケーションの他のコンポーネント間の作業を調整します。また、Web ページのコンテンツの動的な生成などのタスクも実行することができます。詳細については、27ページの『第4章 サーブレットの使用』を参照してください。
- リレーショナル・データベースを使用して、Enterprise Beans のパーシスタンスおよび照会機能をインプリメントします。一般的に新規のアプリケーションに対しては新規のデータベースが定義されますが、アドバンスド版 Application Server では、新規のアプリケーションで既存のデータベースを使用することもできます。

75ページの『第6章 Web アプリケーションの開発』では、アドバンスド版 Application Server の Web アプリケーションについて、詳細に説明しています。アドバンスド版 Application Server を使用して開発するアプリケーションの例については、155ページの『第2部 WebSphere Application Server の使用』を参照してください。

WebSphere Studio

WebSphere Studio は、アドバンスド版 Application Server に組み込まれています。WebSphere Studio の詳細については、9ページの『スタンダード版アプリケーション開発環境』を参照してください。

VisualAge for Java

VisualAge for Java は、Java プログラム開発の完全なサイクルをサポートする統合開発環境です。VisualAge for Java は、アドバンスド版 Application Server の一部ではありませんが、WebSphere Application Server 環境に強固に統合されています。この統合により、VisualAge の開発者は、VisualAge プログラムを終了することなく、Java プログラムを開発、展開、およびテストすることができます。また、この統合により、開発者は企業環境の複雑さを管理しやすくなるとともに、定型ステップの自動化も可能となります。

VisualAge for Java のビジュアル・プログラミング・フィーチャーを使用して、Java のアプレットおよびアプリケーションを短時間で開発することができます。また、SmartGuides を使用することにより、アプレット、サーブレット、アプリケーション、Java Beans および Enterprise Java Beans (EJB) 仕様に従って構築された Enterprise Beans の作成などをはじめとする、多くの作業を迅速に行えるようになります。また、基礎となるファイル・システムの要求に応じて、既存のコードをインポートしたり、コードをエクスポートすることもできます。

第3章 JavaServer Pages の使用

JSP (JavaServer Pages) は、サーバー側で実行されるコンポーネントであり、開発者は JSP を使用して、スクリプト・コマンドを HTML ページに埋め込み、サーバー・コンポーネントを利用することができます。JSP ページは、HTML、XML (Extensible Markup Language) などの構造化ドキュメントをサーバー内に動的に生成するために使用されます。

JSP ページは、JavaSoft サーブレット・アプリケーション・プログラミング・インターフェース (API) の WebSphere Application Server のインプリメンテーションを使用します。1 つの Java クラスは 1 つの JSP ページに対応し、1 つの JSP をインプリメントします。JSP ページでは JavaBeans コンポーネント・アーキテクチャーを利用して、Java をスクリプト言語として使用します。また、サーブレットをコンパイル・ページ・オブジェクトとして使用します。

本節では、JSP ページに関する以下の項目について説明します。

- 20ページの『JSP ページにおける処理』
- 21ページの『JSP ページの呼び出し』
- 22ページの『HTML 文書における Java コードの使用』
- 23ページの『JSP タグの使用』
- 23ページの『アプリケーションにおける JSP ページの使用』
- 25ページの『JSP ページの長所』

本節では、155ページの『第2部 WebSphere Application Server の使用』で説明する、WebSphere ファミリーのアプリケーション例を理解するために必要なバックグラウンド情報を提供することを目的としています。JSP ページを作成する際の、開発上の実際的な問題については解説しません。JSP ページの開発の詳細については、以下を参照してください。

- IBM VisualAge for Java のプロダクト資料とサンプル
- WebSphere Studio の資料およびサンプル

JSP ページにおける処理

JSP ページは、直接呼び出すことも、サーブレットを使用して呼び出すこともできます。JSP ファイルには .jsp という拡張子が付けられます。呼び出された JSP ページは、構文解析されて一時 Java ソース・ファイルに変換されます。この一時 Java ソース・ファイルがサーブレットにコンパイルされ、このサーブレットから、ブラウザーで読み取りおよび表示が可能な標準の HTML ファイルが出力されます。以上のプロセスを図4 に示します。

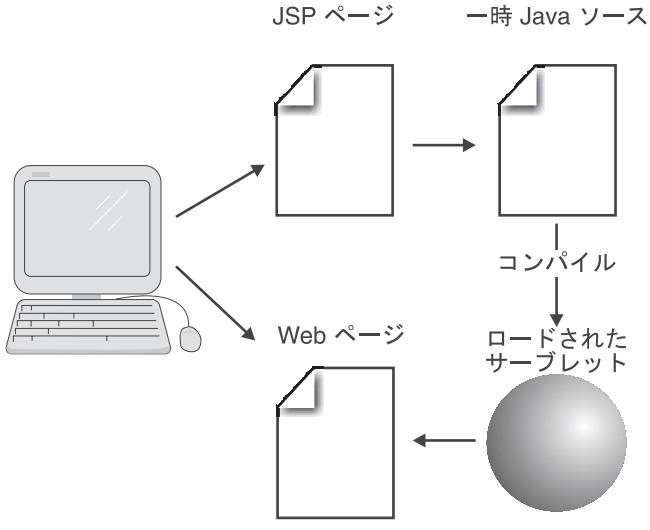

図 4. JSP プログラムのフロー

クライアントからの要求をページに送信し、ページからの応答をクライアントに送信できるプロセッサーによって JSP ページが解釈されます。すべての JSP ページ・プロセッサーでは、要求および応答に使用される HTTP プロトコルが必ずサポートされており、オプションでその他のプロトコルがサポートされている場合もあります。

すべての JSP プロセッサーは、Java で記述されたスクリプト・エレメントをサポートしています。Java で記述された言語を実行するためには、Java 実行時環境が必要です。タグ拡張機構がインプリメントされている場合には、タグ・ハンドラーの実行時には Java 実行時環境も必要になります。

JSP コンパイラーは JSP ファイルを処理し、対応する Java サーブレット・クラスを生成します。生成された Java サーブレット・クラスは、対応する JSP ページに対するすべての要求を処理するため、メモリー内にロードされます。

Java サーブレット・クラスは、サーバーがシャットダウンされるまで、または、ソース・ページが変更されるまで、メモリー内に存在します。

JSP ページに対する要求が発行されると、サーバーはその JSP ページが変更されていないかどうかをチェックします。JSP ページが変更されていなければ、ロードされているサーブレットが使用され、変更されている場合は、JSP ページがサーバーによって自動的にコンパイルされ、サーブレットが再びロードされます。

JSP ページの初回呼び出しで遅延が発生します。これは、JSP ページをサーブレットにコンパイルしてメモリーにロードしなければならないためです。(変更された JSP ページに対する要求が発行された場合も同様)。ただし、以降の呼び出しでは遅延は発生しません。

JSP ページの呼び出し

JSP ページは Web ブラウザーから呼び出すことも、サーブレットを使用して呼び出すこともできます。

図5 は、JSP ページがブラウザーから直接呼び出される時の様子を示しています。JSP ページは URL として、または、HTML ページ内で呼び出されます。JSP ページは、クライアントからの要求を受け取ると、必要な処理を実行するサーバーのコンポーネントに情報を要求します。JSP ページは情報を受け取ると HTML ページに挿入し、その HTML ページがブラウザーに表示されます。

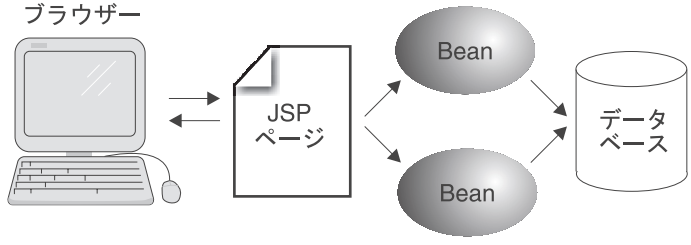

図 5. ブラウザーからの JSP ページの要求

22ページの図6 は、JSP ページがサーブレットから呼び出される時の様子を示しています。ブラウザーからの要求はサーブレットに送信され、サーブレットが JSP ページを呼び出します。サーブレットは、必要な処理を実行するため、サーバーのコンポーネント (この例では Java Beans) と対話します。サーブレットでは、オプションで bean を作成し、保存することもできます。次に、JSP

ページがサーバー・コンポーネントから必要な情報を取り出して HTML ページにマージし、HTML ページがブラウザーに表示されます。

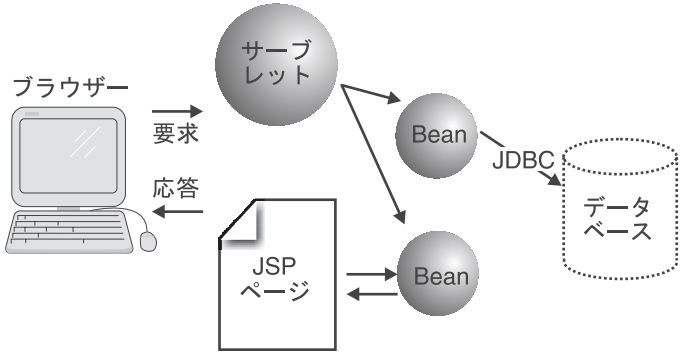

図 6. サーブレットからの JSP ページの要求

JSP ページを呼び出すためのこの方式は、155ページの『第2部 WebSphere Application Server の使用』で説明する WebSphere のサンプル・アプリケーションで使用されています。JSP ページは、コマンド Bean、つまり、ビジネス・ロジックからサーブレットを分離するために使用する Java Bean を介して、サーブレットにより呼び出されます。アプリケーションは、JSP ページを使用して、クライアントに戻す処理結果をフォーマットおよび表示します。

HTML 文書における Java コードの使用

JSP ページでは、Java コードを HTML 文書に組み込む方法として、以下の 2 つを提供しています。

- *スクリプトレット* と呼ばれるタグを使用して Java コードを HTML 文書に直接埋め込む、スクリプトによる方法
- Java コードを Java Beans などのコンポーネント内に含ませるための、コンポーネント中心のアプローチ。プログラミング・ロジックはコンポーネントに含まれます。コンポーネント中心の方法では、タグ (<BEAN> タグ) を使用して、HTML 文書に挿入する情報をコンポーネントに要求します。

通常、スクリプトレットは、簡単ないくつかのコマンドをインプリメントするために使用されます。スクリプトレットは、Beans を使用する意義があまりない場合に使用するようにしてください。ただし、スクリプトレットを多用すると HTML 文書の管理が難しくなることがあります。また、HTML と Java コードが密接に関連するようになるため、開発チーム内で担当を分けることも困難になります。

コンポーネントを使用して Java コードを HTML 文書に組み込むと、ビジネス・ロジックとその表示方法を切り離すことができます。HTML の作成者はタグ・セットを介してのみコンポーネントと対話し、コンポーネントのプログラミングには直接関与しないため、開発チームにおける役割分担が一層明確になります。また、JSP ページでコンポーネントを再利用することも可能になります。

JSP タグの使用

JSP タグ は、JSP ページの機能を HTML または XML で表現するものであり、その機能のプログラム構文をカプセル化します。JSP タグは、以下の機能を備えています。

- JavaBeans コンポーネントの特性、メソッド、要求パラメーター、およびセッション・データへのアクセス
- Bean またはデータ値、要求パラメーター、およびセッション・データに基づく HTML の条件付き組み込み
- エラー・ページおよびページ組み込みの定義
- Java プログラム言語、または JavaScript スクリプト言語で作成されたスクリプトの実行

基本的な JSP タグ・セットおよびタグ拡張機構に関する詳細については、Sun JSP Specification を参照してください。

JSP タグは、WebSphere Studio ツール・セットに含まれている JSP 作成ツールおよび JSP コンパイラーを使用して操作することができます。これらの詳しい使用方法については、WebSphere Studio のマニュアルを参照してください。

アプリケーションにおける JSP ページの使用

JSP ページをサーブレット、HTTP、HTML、XML、アプレット、Enterprise Beans と組み合わせて使用し、以下のアプリケーション・モデルをインプリメントすることができます。

2 階層モデル

このアプリケーション・モデルでは、クライアントの要求を処理するため、JSP ページまたはサーブレットからリソース (データベース、既存のアプリケーションなど) に直接アクセスすることができます。このモデルは、簡単にプログラミングすることができ、JSP ページの作成者が要求およびリソース (特定のまたは複数の) の状態に基づく動的なコンテンツを生成できるという点で優れています。ただし、2 階層モデルで

は、それぞれのクライアントがリソースへの接続を確立するか、クライアント間でリソースへの接続を共有しなければならないので、多数のクライアントを同時に処理することができません。多くの場合、2 階層モデルは CGI (コモン・ゲートウェイ・インターフェース) の代わりに使用されます。

多階層モデル

このアプリケーション・モデルは、3 つ以上の階層から構成されています。中間層に含まれている JSP ページは Enterprise Bean を使用して、バック・エンドのリソースと対話します。EJB サーバーおよび Enterprise Bean が、リソースへのアクセスを管理するための機能を提供します。これにより、パフォーマンスの問題が解決されます。また、EJB サーバーは、トランザクション・サービスおよびセキュリティー・サービスも提供します。WebSphere のサンプル・アプリケーションでは、このアプリケーション・モデルを使用しています。

疎結合アプリケーション

対等関係、あるいは、クライアント / サーバーの依存関係を持つアプリケーションは、疎結合状態にあると見なされます。これらのアプリケーションは、イントラネット、エクストラネット、インターネットのいずれかを使用して通信することができます。疎結合アプリケーションの代表的な例は、取引先企業間のサプライ・チェーン・アプリケーションです。疎結合アプリケーションの場合、依存するもう一方のアプリケーションが変更されても影響を受けません。このような疎結合を実現するため、アプリケーションは、JSP を使用して生成された HTML または XML を使用して、HTTP を介して通信します。

JSP ページにおける XML の使用

JSP ページは、XML 入出力データを処理するのに適しています。静的な XML ページは、XML を静的なテンプレートとして JSP ページ内に記述することによって生成できます。動的な XML ページは、XML を生成する JavaBeans コンポーネント、Enterprise Beans、カスタム・タグのいずれかを使用して生成することができます。XML 入力データは、POST 引き数または QUERY 引き数から受信され、スクリプト・プログラムで操作されるか、JavaBeans、Enterprise Beans、カスタム・タグのいずれかに直接送信されます。

JSP ページには XML の処理に役立つ 2 つの属性があります。XML フラグメントは、コンポーネントに対する入力データのテンプレートとして、または、XML フラグメントによって拡張される出力データのテンプレートとして、JSP ページ内に直接記述することができます。それ

だけでなく、JSP タグ拡張機構を使用して、XML を操作するためのタグまたはディレクティブを作成することもできます。

要求のリダイレクト

クライアントに送信しなければならないデータは、クライアントのプロパティーによって大きく異なります。これらのプロパティーは、エンコードして直接要求に含めたり、クライアントのプロファイルに保存したりすることができます。いずれの場合も、最初の JSP ページで要求に関する詳細情報が判別され、必要に応じて、要求が 2 番目の JSP ページにリダイレクトされます。

このようなプログラミング・モデルは、基礎となるサーブレット API によってサポートされます。ただし、応答ストリームがクライアントに返送されている場合は、HTTP プロトコルによって要求のリダイレクトが妨げられることがあります。その結果、一般的な状態で要求をリダイレクトするのが困難になる場合もあります。このような問題を解決するため、JSP Specification では、出力ストリーム上でバッファリングを提供しています。そのため、出力バッファーがフラッシュされる前にいずれかの時点で JSP コードによって要求をリダイレクトすることができます。また、バッファリングを使用すると、エラー・ページの処理が容易になります。これは、要求をリダイレクトすることによってエラー・ページも処理されるためです。

要求の組み込み

要求が最初の JSP ページで受け取られると、結果の生成が開始されます。JSP ページには他のページのコンテンツを動的に組み込む必要があります。これらのコンテンツは静的なものでも、他の JSP ページ、サーブレット、既存のアプリケーションのいずれかによって動的に生成されたものでもかまいません。このモデルは、主に、表示方法に関係ないコンテンツ (たとえば、後で他のフォーマットに変換される XML など) を生成する場合に使用されます。

JSP ページの長所

JSP ページは、他の Web アプリケーション・アーキテクチャーに比べ、以下の点で優れています。

コンテンツの表示と生成の分離

コンテンツおよびデータの生成はサーバー・コンポーネントによって行われ、JSP ページは、生成されたコンテンツを取り出し、HTML 文書に組み込む役割を果します。

Model-View-Controller(MVC) アーキテクチャーのサポート

JSP ページは、Web アプリケーションで MVC アーキテクチャーをサポートするための優れた機能を備えています。JSP テクノロジーが開発される以前は、サーブレットによって制御ロジック (コントローラー) と動的コンテンツの生成 (ビュー) の両方が行われていました。サーブレットへのこの二重の役割のインプリメントは、出力フォーマットを変更するとサーブレットを再コンパイルしなければならず、サーブレットの保守を容易に行うことができませんでした。しかし、JSP ページを使用してコンテンツを生成することで、出力フォーマットを変更しても、サーブレットに変更を加えて、再コンパイルする必要がなくなりました。このアプリケーション・アーキテクチャーの詳細については、77ページの『Model-View-Controller アーキテクチャーのインプリメント』を参照してください。

開発チーム内における役割の分離

ビジネス・ロジックのコンポーネントへのカプセル化、サーブレットを使用した制御ロジックの処理、JSP ページによる HTML の動的生成により、Web アプリケーション開発チーム内での役割分担が明確化されます。JSP ファイルは HTML の作成者が作成し、サーブレットおよび JavaBeans のコーディングはソフトウェア開発者が担当します。

優れた移植性

JSP ページでは、Java スクリプト言語などの標準のポータブル・コンポーネント、JavaBeans コンポーネント・アーキテクチャー、HTML が使用されています。また、標準の JSP アプリケーション・プログラミング・インターフェース (API) により、アプリケーション・サーバー間で JSP ページを移植することができます。

Java プログラミングの利点を活用

JSP ページでは、強力なシステム、オブジェクト指向のコード、効果的なツール、優れたメモリー管理といった、Java を使用することによる利点がそのまま生かされています。

従来のスキルの使用

JSP ページは従来の Java および HTML プログラミング・スキル・セットをもとに構築することができます。

JavaBeans イントロスペクション

基礎を成す JavaBeans コンポーネント・アーキテクチャーを使用することにより、JSP ページを他のスクリプト言語と区別することができます。単純なタグ付き機能を JavaBeans イントロスペクションの機能と組み合わせることで、生成された HTML にマージするためのデータを簡単に取り出せるようになります。

第4章 サーブレットの使用

本節では、Java 対応の Web サーバー内で実行され、Web サーバー機能へのユーザーのアクセスを可能にする Java アプリケーションであるサーブレットについて説明します。サーブレットは、Web サーバーおよびアプリケーション・サーバーの機能を拡張するために使用されます。サーブレットは、サーブレット・アプリケーション・プログラミング・インターフェース (API) の WebSphere Application Server のインプリメンテーションを使用します。本節では、サーブレットに関する以下の項目について説明します。

- 『サーブレットのプログラミング・モデル』
- 28ページの『サーブレット API』
- 29ページの『サーブレットのライフ・サイクル』
- 30ページの『サーブレットの実行時環境』
- 31ページの『サーブレット・セッションの管理』
- 32ページの『サーブレットの長所』

本章では、155ページの『第2部 WebSphere Application Server の使用』に記載されている WebSphere ファミリーのアプリケーション例について理解するために必要な基礎情報を提供します。サーブレットを作成する際の、開発上の実際的な問題については解説しません。サーブレットの開発に関する詳細については、以下を参照してください。

- IBM VisualAge for Java のプロダクト資料とサンプル
- WebSphere Studio のマニュアルおよびサンプル
- 「*Enterprise Beans の作成*」- Enterprise Beans を使用するサーブレットの記述方法について説明しています。

サーブレットのプログラミング・モデル

サーブレットでは、要求 / 応答プログラミング・モデルをサポートしています。クライアントがアプリケーション・サーバーに要求を送信すると、その要求はアプリケーション・サーバーからサーブレットに送信されます。サーブレットは要求を受け取ると、アプリケーション・サーバーがクライアントに送信する応答を作成します。クライアント (通常は Web ブラウザー) がサーブレットと直接対話することはありません。

サーブレットは Web サーバーと同じプロセス内で実行されます。Web サーバーの役割は、それぞれのサーブレット・インスタンスを初期設定し、起動して、破棄することです。図7 に示すように、Web サーバー・プロセス内では、それぞれのサーブレットは別々のスレッドとして実行されます。サーブレット・インスタンスは 1 つだけであり、複数のクライアントの要求を処理するために複数のスレッドが作成されます。これにより、サーバーのリソースが効率的に使用され、サーブレットがその環境と対話できるようになります。

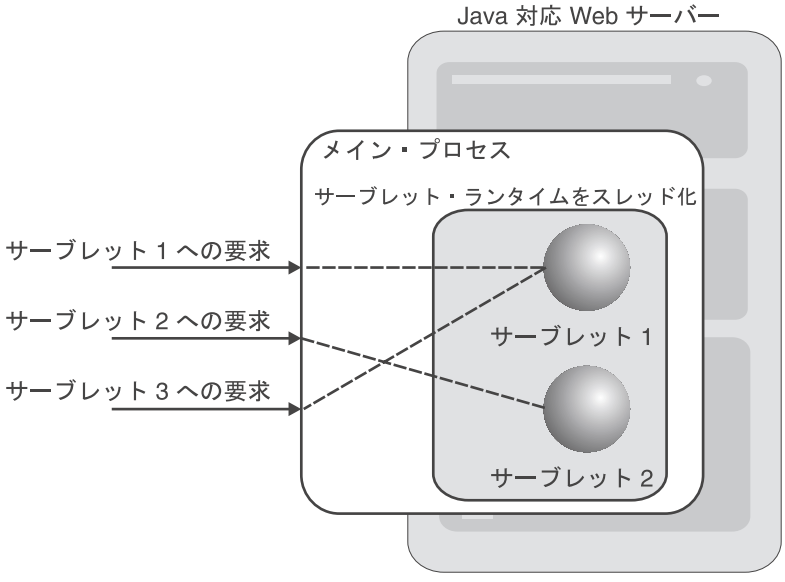

図 7. サーブレットの実行モデル

サーブレットは、対応するサービスが初めて要求された時に動的にロードされるか、または Web サーバーの起動時に自動的にロードされます。

サーブレット API

サーブレットは、Java サーブレット開発キット (JSDK) アプリケーション・プログラミング・インターフェース (API) に基づいて構築されており、この JSDK API によってサーブレットと Java 対応サーバー間の標準インターフェースが定義されます。このため、サーブレットはプラットフォームから独立していて、異なるベンダーの Web サーバー間で移植することができます。Web サーバーは、サーブレットのインターフェースを介してサーブレットと通信します。クライアントからサーバーに要求が発行されると、サーブレットが呼び出され、JSDK API を介して要求が処理されます。

Java クラスは、サーブレット・インターフェースをインプリメントすることによって、サーブレットとなります。そのためには、GenericServlet クラス (プロトコル独立サーブレットの場合) または HttpServlet クラス (特定の HTTP サーブレットの場合) を拡張します。アプレットと同様、サーブレットには main メソッドがありません。このインターフェースによって、メソッドが以下のタスクを実行するように定義されます。

- サーブレットの初期化とロード
- クライアントからの要求に対する応答の構成
- 不要になったサーブレットの破棄およびサーブレットが使用していたリソースの解放

サーブレットを作成する場合、開発者はこれらのメソッドをインプリメントします。開発者は、これらのメソッドを拡張して、サーブレットの操作をカスタマイズすることができます。たとえば、サーブレットが破棄された場合に、データベース接続を解放するようにカスタマイズすることができます。

サーブレットのライフ・サイクル

サーブレットの一般的なライフ・サイクルは以下のとおりです。

1. サーブレットが、アプリケーション・サーバーによって呼び出されてメモリー内にロードされるか、または、アプリケーション・サーバーの起動時に自動的にロードされます。サーブレット・インスタンスは 1 つだけであり、複数のクライアントの要求を処理するために複数のスレッドが作成されます。
2. アプリケーション・サーバーが init メソッドを呼び出し、サーブレットを初期設定します。
3. クライアントからの要求が HTTP サーバーで受信され、サーブレットの実行時環境へ転送されて、サーブレットに渡されます。 HTTP 要求のタイプによって、doGet メソッドと doPost メソッドのいずれかが呼び出されます。
4. サーブレットが要求を処理し、出力ストリームを使用してその結果を HTTP サーバーに戻します。
5. HTTP サーバーからクライアントに結果が送信されます。
6. アプリケーション・サーバーが自らの destroy メソッドを呼び出し、サーブレットをアンロードします。

30ページの図8 に、サーブレットのライフ・サイクルを示します。

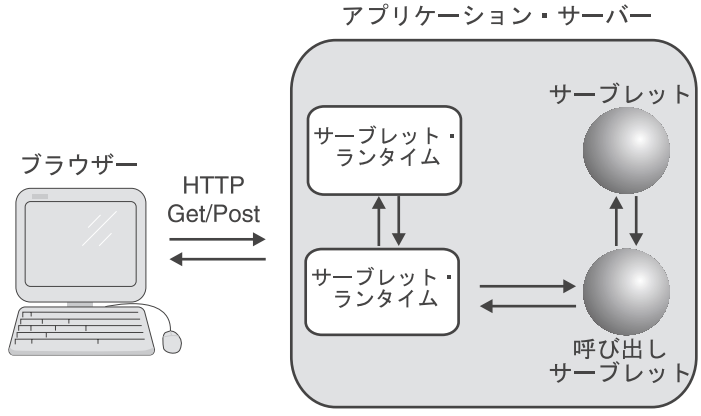

図 8. サーブレットの基本フロー

サーブレットの実行時環境

WebSphere Application Server サーブレットの実行時環境は、サーブレットのパフォーマンスおよび管理を支援するためのコンポーネントを提供します。サーブレットの実行時環境のコンポーネントについて以下で説明します。

サーブレット・コンテナー

サーブレットは、特別なサーブレット固有のコンテナー内で実行されます。このコンテナーは、主に、サーブレットのパフォーマンスを改善することを目的としています。サーブレット・コンテナーでは、トランザクション、パーシスタンス、セキュリティーといったサポートは、サーブレットの実行時環境では必要とされないため提供されません。サーブレット・コンテナーは EJB コンテナーと同様のプログラミング・モデルに従っていますが、サーブレット・コンテナーを Enterprise Beans の展開や実行に使用することはできません。

サーブレット・キュー

サーブレット・キューは、サーブレットの要求を HTTP サーバーからサーブレット・プロセスへディスパッチします。サーバー・グループのすべての複製は、スレッドを使用可能にするため、同一のキューを共用してキューからの要求を処理します。すべての複製が使用中である場合、要求はキュー内に残され、最終的には Web サーバーのスレッドでバックアップされます。

サーブレット・グループ

サーブレット・グループは、サーブレットの複製を実行する EJB サーバー・グループです。サーブレット・グループは、1 つまたは複数のサーブレット・コンテナーを実行し、着信要求をサーブレット・キュー上で listen することができます。サーブレット・グループは、同一のマシン上で実行されるサーブレットの複製に限定されます。

類似性

類似性を使用すると、要求を特定のサーブレット・インスタンスまたはサーブレット・グループに送信することができます。サーブレットの実行時環境では以下の 2 つのタイプの類似性がサポートされています。

- **アプリケーション類似性** − URL をサーブレットのインスタンスにマップすることにより、要求を特定のサーブレット・インスタンスに送信します。アプリケーションの要求を特定のクライアント・インスタンスに戻す必要がある場合 (つまり、アプリケーションがステートフルである場合) には、そのアプリケーションを提供するサーバー・グループを単一インスタンスとして構成しなければなりません。複製は使用することができません。このタイプのサーブレットはスケーラビリティーを備えていません。
- **セッション類似性** − スケーラブルな共用のセッション・インフラストラクチャーを提供することにより、要求をサーブレット・グループに送信します。これにより、クラスター化された HTTP セッションのパフォーマンスを向上させることができます。

サーブレット・セッションの管理

クライアントはセッションを使用してサーブレットに接続します。セッションとは、一定時間維持されるブラウザーからの接続であり、永続的な接続ではありません。

サーブレットにはセッション管理機能が組み込まれています。サーブレット API は、セッションを管理する多くのクラスおよびインターフェースを定義します。すべてのクライアントには HttpSession オブジェクトが関連付けられています。サーブレットは、HttpSession オブジェクトを使用して、クライアントに関する情報を保存したり、取り出したりします。 HttpSession オブジェクトには、単一セッションに関する情報が保存されます。また、Java オブジェクトおよびデータベース接続をセッション・オブジェクトに保存することもできます。

サーブレットは、要求オブジェクトの getSession メソッドを呼び出して、現行の HttpSession オブジェクトを取得します。このメソッドはブール値のパラメ

ーターを受け取ります。ブール値のパラメーターが真である場合、新しいセッションが検出されると、新しいセッションが作成されます。パラメーターが偽である場合には、新しいセッションが検出されると getSession メソッドによってヌルが返されます。

セッションは、以下のいずれかの方法で終了されます。

- 一定時間 (通常は 30 分) が経過した後、サーバーによって自動的に終了される
- サーブレットを使用して手動で終了する

セッションが終了すると、HttpSession オブジェクト、および、HttpSession オブジェクトに含まれているデータ値がシステムから削除されます。セッションのライフ・サイクルを過ぎてもセッション・データが残されるようにするには、セッション・データをデータベースなどの永続的なリソースに保存しなければなりません。このようなことは、セッション・データを保存しておき、サイトの使用状況の傾向を分析する場合などに必要となります。

状態情報の保守についての詳細は、79ページの『Web アプリケーションの状態情報の保持』を参照してください。セッション管理の詳細については、JavaSoft Servlet Specification を参照してください。

サーブレットの長所

サーブレットは、分散アプリケーション開発のためのテクノロジーとして、多くの長所を持っています。これらの長所について、以下で説明します。

移植性 サーブレットは Java を使用して記述されているため、プラットフォーム間で移植することができます。また、サーブレットと Web サーバー間の標準インターフェースはサーブレットの API によって定義されるので、Java 対応アプリケーション・サーバー間で移植することもできます。

永続性 サーブレットはロードされた後もメモリー内に保存されるので、次の要求が発行されるまでシステム・リソース (データベース接続など) を保持することができます。また、サーブレットは、Web サーバーの起動時に自動的にロードされます。

Java プログラム言語の長所を活用
サーブレットでは、強力なシステム、オブジェクト指向のコード、優れたメモリー管理といった、Java プログラム言語の長所が生かされています。

CGI に代わる機能を提供

サーブレットは、Web アプリケーションにおいて CGI に代わる役割を果たします。CGI ではサーブレット・プログラミング・モデルと同様の要求および応答プログラミング・モデルをサポートしていますが、以下の問題があります。

- CGI モデルでは、それぞれのアプリケーション・コンポーネントが独立したプログラムとして実行されます。したがって、CGI プログラムにアクセスする HTTP 要求ごとに新しいプロセスを起動しなければならないため、Web サーバーが処理できる要求の数が制限されます。多数の CGI 要求が発行されると Web サーバーのメモリーおよび処理リソースが消耗します。
- CGI プログラムは独立したプロセスで実行されるため、CGI プログラムは、実行が開始されると、Web サーバーと対話することができません。
- CGI アプリケーションを Web サーバーに直接リンクするプラグイン API は、Web サーバーのベンダーによって提供されます。このような API を使用すると、CGI アプリケーションのパフォーマンスを改善することができますが、Web サーバー間で CGI アプリケーションを移植するのが難しくなります。

パフォーマンスおよび移植性に関するこれらの問題は、Web アプリケーションでサーブレットを使用することによって解消されます。サーブレットは、GCI プログラムに比べ、優れたパフォーマンスを発揮します。 サーブレットのプログラミング・モデルは CGI のプログラミング・モデルとの類似性が高いため、CGI アプリケーションからサーブレットへの移行を容易に行うことができます。サーブレットは Web サーバー・プロセス内で実行されるので、サーバーと通信する際にも障害はありません。それぞれのサーブレットは Web サーバー内でスレッドとして実行されるため、サーバーのリソースをより効率的に使用することができます。また、サーブレットをプラットフォーム間およびアプリケーション・サーバー間で移植することもできます。

クライアントに関する前提条件が少ない

サーブレットを使用することによって Java をサポートするための機能がアプリケーションの中間層に移動します。そのため、サーブレットのクライアントに関する前提条件が緩和されます。たとえば、Java 仮想マシン (JVM) を必要としない HTTP クライアントをサーブレットでサポートすることも可能です。

シン・クライアントのサポート

バック・エンド・アクセス、ビジネス・ロジック、および接続をクライ

アントからサーブレットに移動させることにより、Web アプリケーションの多階層アーキテクチャーに一層適したサポートを提供する、より軽いシン・クライアントを実現します。シン・クライアントの詳細については、「*WebSphere Application Server 概説*」を参照してください。

共通のセキュリティー・モデル

サーブレットは、JSP ページおよび Enterprise Beans と同一のセキュリティー・モデルを共有します。詳細については、 59ページの『セキュリティー・サービス』を参照してください。

Web アプリケーション・セキュリティー

セキュリティーは Web アプリケーションにおける最大の課題であり、一般的には、ファイアウォールを使用して外部アクセスを制御します。サーブレットではファイアウォールでサポートされている HTTP プロトコルを使用します。そのため、ファイアウォールに対する単一ポイントのエントリーを提供し、バック・エンド・リソースをクライアントから見えなくすることができます。詳細については、81ページの『Web アプリケーションでのセキュリティーのインプリメント』を参照してください。

アプレットの機能を拡張

アプレットは、アプレットを作り出したサーバー以外のサーバーとの接続が禁止されている厳しいセキュリティー・モデルで運用されます。そのため、アプレットはサーブレットを使用して他のサーバーのリソースに接続します。たとえば、サーブレットを使用して、アプレットを作り出したサーバー以外のサーバー上のデータベースに接続することがあります。こうすると、多数のアプレットでサーブレットを共用して、リソースをより効率的に使用することができます。アプレット・ベースのクライアントに関する詳細については、「*WebSphere Application Server 概説*」を参照してください。

第5章 Enterprise Beans の使用

Enterprise Beans は、Java プログラム言語で分散アプリケーションを構築するための標準コンポーネント・アーキテクチャーです。 Enterprise Beans は、ホーム環境に常駐し、実行環境で実行される必要があるサーバー・サイド・コンポーネントです。実行環境では、トランザクションのサポート、パーシスタンス、およびリソース管理などの実行時サービスが提供されます。

本節では、WebSphere Application Server の Enterprise JavaBeans (EJB) 環境の概要について説明します。本節で説明するトピックは以下のとおりです。

- 『EJB 環境』
- 37ページの『Enterprise Bean アプリケーションの開発』
- 42ページの『セッション Bean』
- 46ページの『エンティティー Bean』
- 50ページの『EJB 仕様に対する機能拡張』
- 56ページの『オブジェクト・サービス』

本節では、155ページの『第2部 WebSphere Application Server の使用』に記載されている、WebSphere ファミリーのアプリケーション例について理解するために必要な基礎情報を提供します。Enterprise Beans を作成する際の実用上の開発問題には重点を置いていません。Enterprise Bean の開発の詳細については、以下を参照してください。

- *Enterprise Beans の作成*
- IBM VisualAge for Java のプロダクト資料とサンプル

EJB 環境

EJB 仕様には、Enterprise Beans のサポートに必要なサービスが詳細に記述されています。 EJB 仕様では、Enterprise Bean のビジネス・ロジックは、パーシスタンス、トランザクション、セキュリティー、およびその他のミドルウェア関連のサービスが持つ複雑さから解放されています。

クライアント・ビュー

36ページの図9 に、EJB 環境のクライアント・ビューを示します。EJB クライアントは、EJB コンテナーおよびクライアント・コンテナーによって定義され

るインターフェースおよび EJB サーバーによって提供されるサービスを使用して Enterprise Beans と対話します。クライアントは、任意の Enterprise Bean のホーム・インターフェースとリモート・インターフェースを使用して Enterprise Bean と対話します。EJB サーバー環境とコンテナーについては、以下の節で説明します。

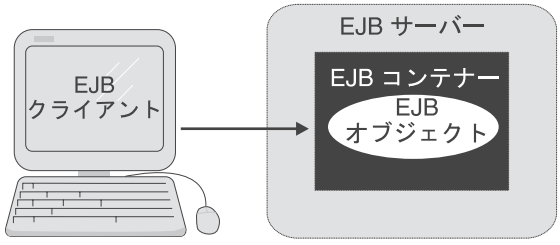

図 9. EJB 環境のクライアント・ビュー

EJB サーバー環境

EJB サーバーはコンテナーを管理し、EJB 仕様で要求されているサービスを提供します。必須のサービスを以下に示します。

- JNDI (Java Naming and Directory Interface)
- オブジェクト・トランザクション・サービス (OTS) 準拠のトランザクション・サービス
- セキュリティー

EJB サーバーには、オプションで、Java データベース・コネクティビティー (JDBC) を介したデータ・ストアへのアクセスのサービスがあります。EJB 仕様で必要となるサービスの詳細については、56ページの『オブジェクト・サービス』を参照してください。

コンテナー

Enterprise Beans はコンテナー に配置されます。EJB 仕様で説明されているように、コンテナーは 1 つまたは複数の Enterprise Beans を管理します。配置により、オブジェクト・サービスの具体的なインプリメンテーションが追加されます。

コンテナーの次の 2 つのインターフェースを介して、必要なオブジェクト・サービスへのアクセスが可能となります。

- ホーム・インターフェース は、コンテナー内の Enterprise Bean の作成、削除、および検索のためのメソッドを公開しています。デプロイメント・ディスクリプター は、ホーム・インターフェースに意味のある名前を付けるための配置ツールによって使用されます。コンテナーはこの名前をネーム・スペースに登録します。クライアントは、JNDI を使用してこの名前にアクセスすることができます。
- リモート・インターフェース (*EJBObject* インターフェース とも呼ばれます) は、Enterprise Bean クラスが提供するビジネス・メソッドを定義します。

クライアントはコンテナーが提供するインターフェースと対話します。図10 は、コンテナーが提供するホーム・インターフェースとリモート・インターフェースを介して、クライアント・アプリケーションが Enterprise Bean とどのように対話するのかを示したものです。

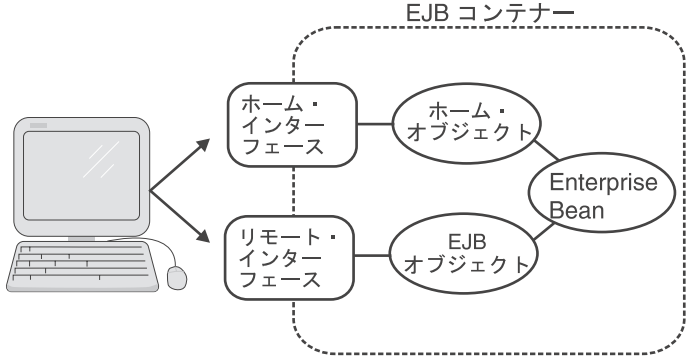

図 10. クライアントと Enterprise Bean との対話

クライアント・アプリケーションは、ホーム・インターフェースを使用して、ホーム・オブジェクト にアクセスします。ホーム・オブジェクトは、それが使用する Enterprise Bean クラスのインスタンスを、作成、除去、および検索することができます。クライアントは、リモート・インターフェースを使用して *EJB オブジェクト* にアクセスします。EJB オブジェクトは、Enterprise Bean のインスタンスのビジネス・メソッドを呼び出すことができます。

Enterprise Bean アプリケーションの開発

本節では、Enterprise Beans を使ったアプリケーション開発の高度な概要を説明します。本節では、以下のトピックについて説明します。

- 38ページの『Enterprise Bean のタイプ』

- 『アプリケーションにおけるセッション Bean とエンティティー Bean の使用』
- 40ページの『開発チームの役割』
- 41ページの『アプリケーション開発プロセス』

Enterprise Bean のタイプ

EJB のアーキテクチャーには、分散アプリケーションの構築用に、セッション Bean とエンティティー Bean という 2 種類の Enterprise Beans があります。各タイプの Bean のアプリケーション内部の機能は異なります。

- セッション *Bean* は、EJB クライアントに代わって作業を行います。セッション Bean のインスタンスの*会話* 状態は永続的ではないため、サーバー障害後は元の状態に復帰しません。通常、セッション Bean のライフ・サイクルは、EJB クライアントのライフ・サイクルと同じです。このタイプの Enterprise Bean の詳細については、42ページの『セッション Bean』を参照してください。
- エンティティー *Bean* は、アプリケーションにおける永続的エンティティーを表します。エンティティー Bean は、しばしば、永続データを表し、永続データの操作に使用されます。永続データは、オブジェクト指向データベースやリレーショナル・データベースなどのデータ・ソースに保管することができます。永続データは、アプリケーションを起動したり、トランザクションを実行することにより作成することもできます。エンティティー Bean によって表される永続データは、エンティティー Bean のコンテナーまたはエンティティー Bean それ自体のいずれかによって管理されます (前者をコンテナー管理のパーシスタンス、後者を Bean 管理のパーシスタンスといいます)。このタイプの Enterprise Bean の詳細については、46ページの『エンティティー Bean』を参照してください。

セッション Bean とエンティティー Bean の最大の相違点は、セッション Bean には Enterprise Bean を一意に識別する基本キー・クラスがない点です。セッション Bean は固有ではないため、基本キーは不要です。したがって、セッション Bean を必要に応じて作成し、クライアントと関係付けたり、削除することができます。これとは対照的に、エンティティー Bean は、固有のものとして識別可能な永続データを表します。

アプリケーションにおけるセッション Bean とエンティティー Bean の使用

エンティティー Bean は永続データを表し、クライアント間で共有する必要はありません。エンティティー Bean では、特定のクライアントに関連する情報を保持することはできません。セッション Bean ではデータに直接アクセスす

ることはできませんが、そのクライアントに関する情報を保持することができます。したがって、ほとんどのアプリケーションでは両方のタイプの Bean を使用する必要があります。

図11 は、セッション Bean とエンティティー Bean がアプリケーション内でどのように機能するかを示したものです。この例では、クライアント・アプリケーションは EJB サーバーにログオンしています。クライアント・アプリケーションで、従業員のリストを表示し、従業員のデータを変更することができます。

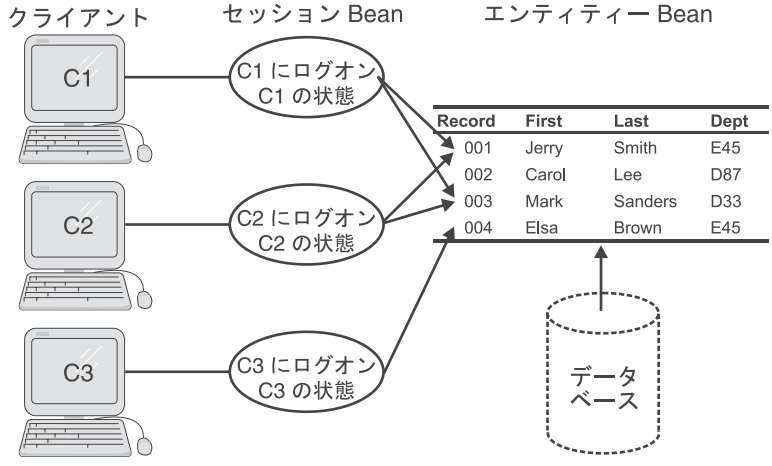

図 11. セッション Bean とエンティティー Bean を使用したアプリケーションの構築

EJB クライアントは、サーバーに接続して、後に使用するために、接続情報を保管する必要があります。セッション Bean を使用すると、これらのタスクを実行することができます。

従業員のリストを表示するには、EJB クライアントはデータベースから従業員のデータを取り出す必要があります。エンティティー Bean を使用すると、このタスクを実行することができます。各エンティティー Bean は、データベース内の 1 つの行を表します。ログイン・セッションを表すセッション Bean は、エンティティー Bean を介して従業員データにアクセスすることができます。

従業員情報は、トランザクションを介して変更することができます。セッション Bean では、データベースに保管されているデータを表すエンティティー Bean のトランザクション・コンテキストを管理することができます。

開発チームの役割

ビジネス・ロジックの開発を他のタスクから切り離すには、アプリケーションの開発チームのメンバーに、異なる役割を割り当てる必要があります。 EJB 仕様は、アプリケーション・プロセスと配置プロセスに、6 つの役割を定義しています。ソフトウェア開発担当者の役割は、1 つに限定されているわけではありません。ソフトウェア開発担当者は、プロジェクトの状況に応じて、数多くの役割を担当することができます。それぞれの役割については、以下のとおりです。

Enterprise Bean 提供者

提供者は、アプリケーションのビジネス・ロジックを理解しており、Java プログラム言語でビジネス・ロジックをインプリメントする方法を知っている必要があります。また、Enterprise Beans のインターフェースとセマンティクスについても理解している必要があります。提供者は、Enterprise Bean を永続的にするかしないかを決定します。Enterprise Bean を永続的にする場合は、提供者は、永続状態のフィールドを識別し、Bean 管理のパーシスタンスを使用するのか、コンテナー管理のパーシスタンスを使用するのかを決定します。さらに提供者は、トランザクションの有効範囲内で Enterprise Bean が関係する場合の、Enterprise Bean の振る舞いを指定します。提供者には、Enterprise Beans と、それに関係するすべてのファイルを、Java アーカイブ (JAR) ファイルにパッケージ化する責任があります。パッケージ後、この JAR ファイルは配置担当者に渡されます。

EJB コンテナー提供者

コンテナー提供者は、Enterprise Bean コンテナーを作成します。Enterprise Bean コンテナーは、Enterprise Beans に、クライアント・インターフェース、パーシスタンス、スケーラビリティー、セキュリティー、およびトランザクション・サポートを提供するものです。コンテナー提供者は、ビジネス・ロジックを基本サービスにリンクします。

EJB サーバー提供者

サーバー提供者は、オペレーティング・システムとミドルウェアのサービスをコンテナーに入れます。現時点ではコンテナーとサーバーとの間のインターフェースを定義した仕様がないため、ほとんどの場合、サーバーとコンテナーは同じベンダーが提供します。

配置担当者

配置担当者は、Enterprise Bean のクラスを EJB サーバーにインストールします。配置担当者は Enterprise Beans と EJB サーバー環境を理解

して、EJB サーバーのツールを使って Enterprise Bean の要件を構成する必要があります。また配置担当者は、JNDI を介して Enterprise Beans をアクセス可能にします。

アプリケーション組み立て担当者

アプリケーション組み立て担当者は、アプレット、サーブレット、およびネイティブの CORBA アプリケーションなど、Enterprise Beans を使用するアプリケーションを作成します。アプリケーション組み立て担当者は、既存の Enterprise Beans から新しい Enterprise Beans を作成することもできます。

システム管理者

システム管理者は、サーバーおよびコンテナー提供者が提供するモニター・ツールと管理ツールを使用して、システムが適切に動作することを保証します。

アプリケーション開発プロセス

Enterprise Beans を使用したアプリケーション開発は、以下の 5 つのプロセスから構成されます。

1. 提供者は、Enterprise Beans のセット内にアプリケーションのビジネス・ロジックをコーディングし、次に Bean とその Bean のデプロイメント・ディスクリプターを JAR ファイルにパッケージ化します。
2. コンテナー提供者は、Enterprise Beans が使用するコンテナーを作成します。
3. サーバー提供者は、アプリケーションが要求する必要不可欠なサービスを提供するよう、EJB サーバーをセットアップします。
4. Bean は EJB サーバーに配置されます。配置担当者は、コンテナーが内部的に使用する追加クラスを生成します。通常、クラスの生成には、コンテナー・ソフトウェアが提供するツールを使用します。
5. アプリケーション組み立て担当者が、EJB クライアントをビルドします。

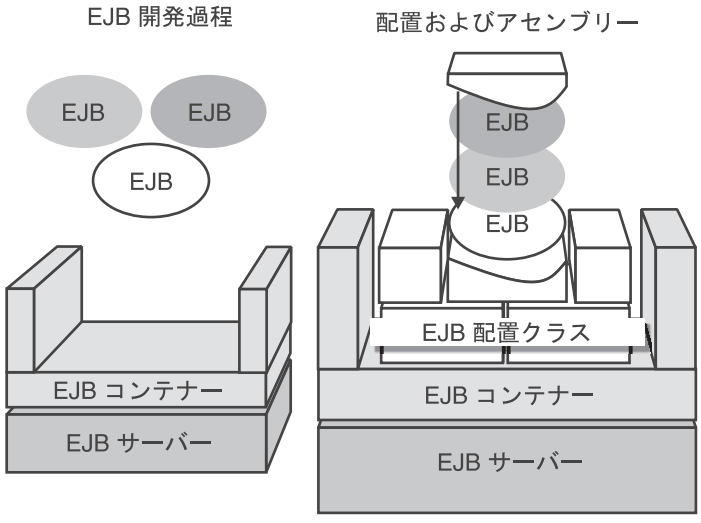

図 12. Enterprise Bean アプリケーション開発プロセス

セッション Bean

セッション Bean は、単一のユーザー・セッションの期間中のみ存在する、一時的なオブジェクトです。セッション Bean は、ユーザーのセッション、タスク、あるいは一時オブジェクトに関連付けられているデータとメソッドをカプセル化します。

セッション Bean は、EJB クライアントに代わって作業単位を実行します。セッション Bean はクライアント間では共用されないため、クライアント固有のセッション情報を保持することができます。セッション Bean は、EJB 仕様に準拠するコンテナーでは必須のサポートです。

ステートレス・セッション Bean とステートフル・セッション Bean

セッション Bean のインスタンス内のデータは、一時的です。これが失われても、実際に損害を受けることはありません。セッション Bean の設計では、そのデータの寿命を短くするか、長くするかを決定します。

- セッション Bean がメソッドに関する特定のデータを保守する必要がある場合は、このセッション Bean はステートフル・セッション Bean と呼ばれます。セッション Bean がメソッドを終了した後もデータを保持している場合、セッション Bean は会話状態である、といいます。特定のクライアントがステートフル・セッション Bean のインスタンスを一度使用したら、その

クライアントは、そのインスタンスの特定の状態が必要とされる限り、そのインスタンスを使い続ける必要があります。

- セッション Bean がメソッドに関する特定のデータを保守する必要がない場合は、このセッション Bean はステートレス・セッション Bean と呼ばれます。ステートレス・セッション Bean の場合、クライアントは任意のインスタンスを使用して、そのセッション Bean の任意のメソッドを呼び出すことができます。

コンテナーは、ステートフル・セッション Bean のインスタンスをメモリーから除去し、それを永続記憶域に保管することにより、そのセッション Bean を管理することができます。これを*非活性化* といいます。セッション Bean が再度呼び出されると、コンテナーは新しいインスタンスを作成し、非活性化時に保管されたデータを使ってそのインスタンスを初期設定します。これを*活動化* といいます。

コンテナーはステートレス・セッション Bean に対しては非活性化を使用しません。メモリー・リソースが少なくなると、ステートレス・セッション Bean を破棄することができます。ステートレス・セッション Bean のすべてのインスタンスは同じであるため、コンテナーは使用可能な任意のインスタンスを使って、クライアント要求を満たすことができます。

セッション Bean のデータが永続的に保管されることはありません。セッション Bean クラスには、永続データ・ソースにデータを保管したり、そこからデータを取り出すためのメソッドは含まれません。ただし、オプションとして、セッション Bean はトランザクション・セーフ・モードでデータにアクセスし、それを更新することができます。

セッション Bean コンポーネント

すべてのセッション Bean には、Bean クラス、ホーム・インターフェース、およびリモート・インターフェースが含まれる必要があります。

Bean クラス

Bean クラスは、Enterprise Bean に関連付けられているデータをカプセル化します。また、このデータにアクセスするためのビジネス・メソッドを含んでいます。Bean クラスには、コンテナーが Enterprise Bean のインスタンスのライフ・サイクルを管理するために使用するメソッドも含まれています。他の Enterprise Beans などのクライアントとユーザー・アプリケーションは、このクラスのオブジェクトに直接アクセスすることはありません。その代わりに、ホーム・インターフェースとリモート・インターフェースに関連付けられているコンテナー生成クラスを使用して Enterprise Bean を操作します。

Bean クラスには、ビジネス・メソッドを含めることができますが、必ず含めなければならないということではありません。これらのメソッドは public でなければなりません。ビジネス・メソッドの名前は、EJB アーキテクチャーで使用されている他の名前と競合することはできません。(たとえば、計算を実行するメソッドを宣言する必要がある場合、そのメソッドに ejbCalculate という名前を付けないでください。この名前には、calculate の方がふさわしいでしょう。)

ビジネス・メソッドの各パラメーターは、許可済みの Java RMI 型でなければなりません。throws 文節には、特定のアプリケーション例外を含むことができます。

Bean クラスに Enterprise Bean のリモート・インターフェースをインプリメントすることはお勧めしません。

ホーム・インターフェース

ホーム・インターフェースには、Enterprise Bean のインターフェースの作成と除去のためにクライアントにより使用されるメソッドが含まれます。このインターフェースは、Enterprise Bean の配置時にコンテナーによって、一般に EJB ホーム・クラスとして知られているクラスにインプリメントされます。

セッション Bean のホーム・インターフェースは、Enterprise Bean のインスタンスの作成と除去、およびインスタンスに関するメタデータの取得を行うためにクライアントが使用するメソッドを定義します。ホーム・インターフェースは開発者によって定義され、コンテナーによって配置時に作成された EJB ホーム・クラスにインプリメントされます。コンテナーによって、JNDI (Java Naming and Directory Interface) を介してクライアントからホーム・インターフェースにアクセスすることができます。

ホーム・インターフェースのメソッドは、Java RMI 規則に準拠する必要があります。

ホーム・インターフェースは、1 つまたは複数の create メソッドをインプリメントする必要があります。create メソッドの戻りの型は、Enterprise Bean のリモート・インターフェース型です。

リモート・インターフェース

セッション Bean のリモート・インターフェースは、Enterprise Bean クラスで使用可能なビジネス・メソッドへのアクセスを提供します。また、Enterprise Bean のインスタンスの除去、および Enterprise Bean のホーム・インターフェースとハンドルの取得を行うメソッドも提供します。リモート・インターフェ

ースは開発者によって定義され、配置時にコンテナーにより作成された EJB オブジェクトにインプリメントされます。

クライアントは、ホーム・インターフェースを使用して Enterprise Bean にアクセスした後、リモート・インターフェースを使用して、その Enterprise Bean クラスに定義されているビジネス・メソッドを呼び出します。このインターフェースは、Enterprise Bean の配置時にコンテナーによって、一般に EJB オブジェクト・クラスとして知られているクラスにインプリメントされます。

リモート・インターフェースには、EJB クライアントの開発者が使用するメソッドが含まれています。 各メソッドには、全く同じパラメーターを持つ Bean クラスの一致検索メソッドがなければなりません。リモート・インターフェースのメソッドは、Java RMI 規則に準拠する必要があります。戻りの型は有効な Java RMI 型でなければならず、メソッドの throws 節には java.rmi.RemoteException が含まれている必要があります。

セッション Bean のライフ・サイクル

セッション Bean のライフ・サイクルの間、セッション Bean のインスタンスは以下の状態を経ます。

1. *作成状態*。セッション Bean のライフ・サイクルは、Bean のホーム・インターフェースで定義されている create メソッドをクライアントが呼び出した時点で始まります。コンテナーがセッション・コンテキストを設定し、新規のセッション Bean のインスタンスを作成します。
2. *作動可能状態*。セッション Bean のインスタンスが作成されると、クライアントはその Bean のリモート・インターフェースに定義されているビジネス・メソッドを呼び出すことができます。
3. *プール状態*。ステートフル・セッション Bean のインスタンスが不要になると、コンテナーはそのインスタンスを非活性化します。クライアントが非活性化されたセッション Bean のインスタンスのメソッドを呼び出すと、コンテナーはそのインスタンスを活動化し、それを作動可能状態に戻します。

   ステートレス・セッション Bean のインスタンスは、非活性化も活動化もされません。これらのインスタンスは、除去されるまで常に準備状態で存在します。
4. *除去可能状態*。セッション Bean のライフ・サイクルは、クライアントまたはコンテナーがその Bean のホーム・インターフェースまたはリモート・インターフェースで定義されている remove メソッドを呼び出した時点で終了します。

セッション Bean のライフ・サイクルの詳細については、「*Enterprise Beans の作成*」を参照してください。

エンティティー Bean

エンティティー Bean は、データベースに保存することができる永続データを表し、トランザクションまたはアプリケーションによって作成されます。

Bean 管理のパーシスタンス (BMP)

Bean 管理のパーシスタンス (BMP) では、エンティティー Bean は永続データの保管と取り出しを管理します。エンティティー Bean の開発者は、エンティティー Bean のビジネス・ロジックをエンコードし、永続記憶域に対して明示的にデータベース呼び出しを行うか、その他のタイプのアクセスを行う必要があります。コンテナーが ejbFind、ejbLoad、ejbStore などのライフ・サイクル・メソッドを使用して Enterprise Bean を呼び出した場合、開発者はその Bean の状態を保管し、復元する必要があります。

BMP を持つエンティティー Bean を使用する利点は、Bean を個々の EJB クライアントのデータ処理要件を満たすように調整することができる点です。たとえば、BMP を持つエンティティー Bean を、特定のタイプのデータ・ストレージへのアクセスを最適化するように作成することができます。ただし、このカスタム化では、BMP を持つエンティティー Bean の再利用と保守はより難しくなります。

EJB コンテナーがサポートしていないデータ・ストアに対しては、BMP を持つエンティティー Bean をお勧めします。たとえば、ファイルへの永続データの保管は、一部のタイプのコンテナーでのみサポートしています。(ほとんどのコンテナーは永続データをデータベースに保管します。) BMP を持つエンティティー Bean を使用した場合、コンテナーがファイルへのデータ保管をサポートしていなくても、それが可能になります。

コンテナー管理のパーシスタンス (CMP)

コンテナー管理のパーシスタンス (CMP) では、コンテナーは、エンティティー Bean と永続データ・ソースとの間の対話を処理します。エンティティー Bean の開発者は、CMP を持つエンティティー Bean のデータベース接続を明示的にコーディングする必要はありません。開発者はどのフィールドを永続的にするかを指定するだけでよいのです。データベース呼び出しを行う際の詳細は、コンテナーにより処理されます。コンテナーはエンティティー Bean のフィールドをデータベースまたは既存のアプリケーションにマップします。さらにコンテナーは、接続の共用プールを使用しデータをキャッシュすることによ

って、データベース・アクセスを効率的に管理します。 VisualAge for Java も、CMP を持つエンティティー Bean のフィールドを、データベース列に関連付けることができます。

CMP を持つすべてのエンティティー Bean には以下のコンポーネントがあります。

- Bean クラス
- ホーム・インターフェース
- リモート・インターフェース
- 基本キー・クラス
- finder メソッド

これらのコンポーネントについて以下に説明します。

Bean クラス

Bean クラスは、エンティティー Bean に関連付けられているデータへのアクセスとその操作に使用されるビジネス・メソッドを定義し、インプリメントします。また Bean クラスは、エンティティー Bean のインスタンスの作成に使用されるメソッドも定義し、インプリメントします。さらに、インスタンスのライフ・サイクルにおける重要なイベントをエンティティー Bean のインスタンスに通知するためにコンテナーが使用するメソッド (コールバック・メソッド) をインプリメントします。Bean クラスは *name*Bean と呼ばれます。ここで、*name* は、エンティティー Bean に割り当てられた名前です。

ホーム・インターフェース

ホーム・インターフェースは、以下のタスクを実行するために EJB クライアントが使用するメソッドを定義します。

- Bean の新しいインスタンスを作成する。
- Bean のインスタンスを検索する。
- Bean のインスタンスを除去する。
- インスタンスに関する情報を入手する。

ホーム・インターフェースは、コンテナーの配置ツールによって生成される EJB ホーム・クラスによってインプリメントされます。コンテナーは、Java Native Directory Interface (JNDI) のアプリケーション・プログラミング・インターフェース (API) を使用する EJB クライアントからアクセスできるネーム・スペースに、ホーム・インターフェースを登録します。ホーム・インターフェースは、*name*Home と呼ばれます。ここで、*name* は、Enterprise Bean に割り当てられた名前です。

注: すべてのホーム・インターフェースは、Enterprise Bean のインプリメンテーションに関連付けられている必要があります。抽象ホーム・インターフェースまたは Bean に関連付けられていないホーム・インターフェースは、サポートされません。

リモート・インターフェース

リモート・インターフェースは、Bean クラスで使用可能なビジネス・メソッドへのアクセスを提供します。リモート・インターフェースは、*name* と呼ばれます。ここで、*name* は、Enterprise Bean に割り当てられた名前です。

コンテナーは、ホーム・インターフェース、ハンドル、および Enterprise Bean インスタンスの基本キーを戻し、インターフェースのインスタンスを比較し、Enterprise Bean のインスタンスを除去するためのメソッドをインプリメントします。

基本キー・クラス

すべてのエンティティー Bean は、コンテナー内に固有の ID を持っています。Bean の ID は、オブジェクトのホーム・インターフェースの名前と、その基本キー の組み合わせを使用して定義されます。クライアントは基本キーを使用して、エンティティー Bean のインスタンスの作成または検索を行います。基本キーは、エンティティー Bean のインスタンスの作成時に割り当てられます。同じ ID を持つ 2 つの Enterprise Bean のインスタンスは同一と見なされます。

プリミティブな Java データ・タイプ (整数型、long 型、ストリング型など) の単一フィールドからなる単純な基本キーは、展開時に割り当てることができます。複数フィールド、あるいは、複雑な Java データ・タイプからなる複合基本キーは、基本キー・クラス でカプセル化する必要があります。このクラスはパブリックで、シリアライズ可能でなければなりません。基本キー・クラスのインスタンス変数はパブリックで、その変数名は、Bean クラスで定義されている変数名のサブセットと一致する必要があります。

規則上、基本キー・クラスは、*name*Key と呼ばれています。ここで、*name* は、Enterprise Bean に割り当てられた名前です。

finder メソッド

finder メソッドは、基本キー値以外の検索基準を使用して、データベースからインスタンスを取り出します。(たとえば、これらのメソッドを使用して、カスタマー番号により検索を行うことができます。) findByPrimaryKey メソッド以外の finder メソッドごとに、固有の照会ストリングを定義する必要があります。

必要に応じて、EJB サーバー (Component Broker) 環境の場合は finder ヘルパー・クラスを、EJB サーバー (アドバンスド版 Application Server) の場合は finder ヘルパー・インターフェースを作成することができます。コンテナーはそれを使用して、finder メソッドを使ったデータベースの照会に必要なコードを生成します。

エンティティー Bean のライフ・サイクル

エンティティー Bean がコンテナーに配置された後、クライアントはそのエンティティー Bean のインスタンスを必要に応じて作成したり使用したりすることができます。エンティティー Bean のインスタンスは、ライフ・サイクルの間、以下の状態を経ます。

1. *作成状態* − エンティティー Bean のインスタンスのライフ・サイクルは、コンテナーがそのインスタンスを作成し、そのセッション・コンテキストを設定した時点で始まります。
2. *プール状態* − エンティティー Bean のインスタンスが作成されると、指定されたエンティティー Bean クラスの使用可能なインスタンスのプールに配置されます。プール内のエンティティー Bean クラスのすべてのインスタンスは同一です。インスタンスがこのようなプール状態にある間は、コンテナーはこれを使用して Bean の finder メソッドを呼び出すことができます。
3. *作動可能状態* − クライアントが特定のエンティティー Bean のインスタンスを要求すると、コンテナーはプールからインスタンスを 1 つ取り出し、クライアントが初期設定した EJB オブジェクトにそれを関連付けます。次に示す 2 つのイベントが発生すると、エンティティー Bean のインスタンスが作動可能状態になります。
   - クライアントが、Bean のホーム・インターフェースで定義されている create メソッドを呼び出して、エンティティー Bean クラスの新しい固有エンティティー (およびデータ・ソース内の新しいレコード) を作成したとき。
   - クライアントが finder メソッドを呼び出して、データ・ソース内の既存のレコードに関連付けられているエンティティー Bean クラスの既存のインスタンスを操作したとき。

   Enterprise Bean のインスタンスが作動可能状態にある場合は、コンテナーはそのインスタンス内のデータをデータ・ソース内の対応するデータに同期させることができます。さらに、クライアントは Bean のインスタンスのビジネス・メソッドを呼び出すことができます。

   作動可能状態にあるエンティティー Bean のインスタンスが不要になると、その Bean はプール状態になります。

4. *除去可能状態* − エンティティー Bean のインスタンスのライフ・サイクルは、プール状態にあるそのエンティティー Bean のインスタンスのコンテキストを、コンテナーが除去した時点で終了します。

注: エンティティー Bean のインスタンスを除去しても、データ・ソースに保管されているデータは除去されません。

エンティティー Bean のライフ・サイクルの詳細については、「*Enterprise Beans の作成*」を参照してください。

EJB 仕様に対する機能拡張

WebSphere Application Server では、アプリケーション設計者が機能を追加することができるように、EJB 仕様を機能拡張しています。 EJB 仕様に対する機能拡張には、アクセス Bean、継承、および関連などがあります。

アクセス Bean

アクセス Bean は、1 つまたは複数の Enterprise Beans を包み込むラッパーとして機能する JavaBeans のコンポーネントです。アクセス Bean は、プレゼンテーション担当の開発者と、ビジネス・ロジック担当の開発者の間を橋渡しします。アクセス Bean は、Enterprise Bean のホーム・インターフェースとリモート・インターフェースを EJB クライアントから隠ぺいすることにより、クライアント・アプリケーションでの Enterprise Beans の使用を容易にします。

アクセス Bean を使用すると、Enterprise Beans を使用する EJB クライアントの開発が、ローカルの JavaBeans コンポーネントを使用する場合と同様に容易になります。EJB クライアントはアクセス Bean ラッパーとのみ対話し、直接 Enterprise Bean を使用することはありません。これにより、ネーミング・サービスと Enterprise Bean のホーム・インターフェースおよびリモート・インターフェースに明示的にリモート呼び出しを行う必要がなくなります。アクセス Bean は、Enterprise Bean の属性のローカル・コピーを保持することにより、パフォーマンスを向上させることもできます。アクセス Bean は Java コードのみで作成されているため、アプリケーションの移植性には影響しません。

アクセス Bean は主にサーブレットと JavaServer Pages (JSP) プログラムをサポートすることを目的としたものです。ただし、アクセス Bean は、サーバー側の Enterprise Bean にアクセスするアプリケーションであればどのアプリケーションでも使用することができます。

VisualAge for Java は、以下の 3 つのタイプのアクセス Bean をサポートしています。

- **ラップされた Bean** − セッション Bean のインスタンスのラッパーとして機能します。
- **エンティティー Bean 用の CopyHelper** − エンティティー Bean のインスタンスのラッパーとして機能し、そのインスタンスの属性のローカル・コピーを保持します。 EJB クライアントは、リモート・エンティティー Bean から直接属性のコピーを取り出す代わりに、それにアクセスします。
- **複数のエンティティー Bean 用の行セット** − 1 つ以上のエンティティー Bean のインスタンスのラッパーとして機能し、それらインスタンスの属性のローカル・コピーを保管します。行セットを使用した場合、EJB クライアントは、複数のエンティティー Bean を個々に初期化したりそれぞれに接続しなくても、それらを使用することができます。

アクセス Bean は、アドバンスド版 Application Server のランタイムを介してのみ使用することができ、VisualAge for Java でのみ作成することができます。 VisualAge for Java は、クライアント JAR ファイルの作成時に、選択された Enterprise Beans に関連付けられているアクセス Bean をエクスポートします。各アクセス Bean クラスは、manifest ファイルにおいては Java Bean として識別されます。

アクセス Bean の開発に関する詳細については、VisualAge for Java の資料を参照してください。

継承

WebSphere Application Server の Enterprise Bean アーキテクチャーは、ポリモアフィズムをサポートし、Enterprise Beans の再利用をより簡単に行うことができる Enterprise Beans の継承を認識します。継承を使用すると、Enterprise Bean のインターフェースを継承する新しい Bean を作成するためのベースとして、Enterprise Bean を使用することができます。継承は VisualAge for Java でのみインプリメントされます。

本節では、継承がどのように WebSphere Enterprise Bean モデルに適合するかについてのバックグラウンド情報について説明します。継承を使用するための手引きを意図しているわけではありません。詳細については、VisualAge for Java の資料を参照してください。

継承が Enterprise Bean に与える影響

Enterprise Beans は、その親のインターフェースを直接継承しません。親の Enterprise Bean のインターフェースは、52ページの表2 に示すように、子のインターフェースを作成するためのテンプレートとして機能します。この例では、親の Enterprise Bean は Parent、子の Enterprise Bean は Child という名

前が付けられています。親の基本キー・クラスは、その名前を子に残すインターフェースにしか過ぎません。

表 2. 親と子のインターフェース

インターフェース	Parent Bean インターフェースの名前	Child Bean インターフェースの名前
リモート・インターフェース	Parent	Child
ホーム・インターフェース	ParentHome	ChildHome
Bean クラス	ParentBean	ChildBean
基本キー・クラス	ParentKey	ParentKey

注: 継承を使用しない Enterprise Beans または継承階層のルートにある Enterprise Beans には、追加のインプリメンテーション要件はありません。

継承は、Enterprise Bean の以下のコンポーネントに影響を与えます。

リモート・インターフェース

子の Enterprise Bean のリモート・インターフェースは、親 Bean のリモート・インターフェースを拡張します。そのため、親のインスタンスを使用する予定の場所で、確実に、子のインスタンスを使用することができます。

ホーム・インターフェース

子の Enterprise Bean のホーム・インターフェースは、親 Bean のホーム・インターフェースを拡張しません。メソッドのインプリメンテーションは、以下に示すように、親と子の Bean で異なります。

- 親のホーム・インターフェースの create メソッドは、親の Enterprise Bean のインスタンスのみ作成します。
- 子のホーム・インターフェースの create メソッドは、子の Enterprise Bean のインスタンス (親のように扱われます) のみ作成します。
- 親のホーム・インターフェースの remove メソッドは、親の Enterprise Bean のインスタンスのみ除去します。
- 子のホーム・インターフェースの remove メソッドは、親と子の両方の Enterprise Bean のインスタンスを除去します。
- 親のホーム・インターフェースのカスタムの finder メソッドは、親のインスタンスを含む列挙内容を戻します。
- 子のホーム・インターフェースのカスタムの finder メソッドは、親と子の両方のインスタンスを含む列挙内容を戻します。

Bean クラス

継承階層のルートである Bean インプリメンテーションは、子のエンティティー Bean のインスタンスのタイプを決定する識別フィールドを定義します。インスタンスの型を決定するためのリモート・インターフェースのメソッドは、その識別に対しても定義する必要があります。

子の Bean のインプリメンテーション・クラスは、親 Bean のリモート・インターフェースで定義されているメソッドをすべて含むプロキシー・クラスから拡張します。これらのメソッドのどれを呼び出しても、それは親 Bean のインスタンスにリダイレクトされます。

基本キー・クラス

エンティティー Bean は、その基本キーに対して Java キー・クラスを定義します。継承階層にあるすべてのエンティティー Bean では、その基本キーに対して同じキー・クラスを使用する必要があります。

エンティティー Bean における継承

継承により、CMP を持つエンティティー Bean に、以下の機能が追加されます。

- 識別の関係は、エンティティー Bean 間で指定することができます。
- 基本キー検索では、ターゲット・クラスとそのサブクラスが戻されます。
- 関係 finder では、ターゲット・クラスとそのサブクラスのインスタンスを含む結果が戻されます。

たとえば、銀行の預金口座を表すエンティティー Bean である Account を作成したとします。この Account Bean は、SavingsAccount、CheckingAccount および CorporateAccount Enterprise Beans などの関連するエンティティー Bean のグループの親として振る舞うことができます。 Account Bean の子は、Account Bean として識別されます。すべての Account Bean を検索すると、Account Bean とそこから派生したエンティティー Bean すべてが戻されます。

継承を使用する、BMP を持つエンティティー Bean は、1 つの例外を除き、CMP を持つエンティティー Bean と同じプログラミング・モデルに従います。 BMP finder メソッドは、それが定義されているホームによって管理される Bean のタイプを含む列挙のみ戻すことができます。

セッション Bean における継承

継承を使用する、セッション Bean は、継承を使用するエンティティー Bean と同じプログラミング・モデルに従います。子のセッション Bean は、その親と同様にそれ自体の状態を管理する必要があります。ステートレス・セッション Bean を継承するすべてのセッション Bean は、ステートレスでなければな

りません。ステートフル・セッション Bean を継承するすべてのセッション Bean は、ステートフルでなければなりません。

VisualAge for Java における継承のサポート

VisualAge for Java の EJB ツールを使用して、Enterprise Beans 間の継承を定義することができます。これらのツールは、Enterprise Beans の階層をデータベース・テーブルにマップすることができます。これらのツールにより、選択したデータベースのマッピング・スキーマに従って、子の Bean のインスタンスを抽出するためのパーシスター・コードと finder コードを生成することができます。(親ホームに検索操作を行った結果、その子の Bean のインスタンスが戻される可能性があることを意味します。)

親と子の Enterprise Bean クラスの間の関係は、子クラスとして識別される Enterprise Beans に対するメタデータで定義されます。この定義をするには、親と子のクラスの Bean モデルを関係付ける一般化オブジェクトを生成します。

VisualAge for Java は、作成およびインポートされた Enterprise Beans 間の継承関係を認識します。JAR ファイル内のデプロイメント・ディスクリプターには、親クラスの情報が含まれます。他のソースのエンティティー Bean は、デプロイメント・ディスクリプターによって指定されたインターフェースとクラスからこの情報を取り出します。

永続的ストレージにマップされる継承階層の場合、ルート・クラスは階層全体に対してテーブル作成ストリングを定義します。このストリングは、すべてのサブクラスのマップ先の単一テーブルか、階層全体のマップ先のルート・テーブルとリーフ・テーブルのいずれかを定義します。どのクラスに対してもテーブルの作成を要求できるため、子クラスはその親のクラスのテーブル作成ストリングを戻します。

子クラスには、マップ・ブラウザーによって提供される最小のパーシスタンス・マップが必要です。新しい子クラスのパーシスターに対して照会を生成する場合には、これが必要になります。少なくとも、マップはどの継承戦略を使用するのかを指定する必要があります。マップは、オプションで、識別フィールドに値を提供します。

関連

WebSphere Application Server の Enterprise Bean アーキテクチャーでは、関連は EJB 仕様の拡張機能として提供されます。関連は、データベースの検索を改善するために CMP を持つエンティティー Bean 間の関係を定義します。関連は VisualAge for Java でのみインプリメントされます。

VisualAge for Java は、関連用のマッピング・サポートおよび配置サポートを提供します。 VisualAge for Java による関連のインプリメンテーションは、WebSphere Application Server アドバンスド版の EJB サーバーにのみ配置することができます。WebSphere Application Server エンタープライズ版のユーザーは、関連の関係を持つ Enterprise Beans を修正し、VisualAge Component Development Toolkit のオブジェクト・ビルダーを使用してそれをエンタープライズ・サーバーに配置することができます。

本節では、関連がどのように WebSphere Enterprise Bean モデルに適合するかについて説明します。詳細については、VisualAge for Java の資料を参照してください。

関連は CMP を持つエンティティー Bean にのみ適用されます。 VisualAge for Java の EJB ツールを使用すると、関連の統一モデリング言語 (UML) の定義に従って、Bean 間の関連を定義することができます。これらのツールを使用すると、関連を作成、編集、および削除することができ、参加しているエンティティー Bean 間の整合性を維持することができます。単一値フォワード関係 (一対一関係) と多値フォワード関係 (一対多関係) の両方を定義することができます。

関連の端点ごとに、ナビゲート可能性と多様性が収集されます。マップ・ブラウザーを使用すると、関連を他のエンティティー Bean のキー・フィールドにマップすることができます。配置ツールは、後方関連マッピングのための finder を生成します。

キー・フィールドには、エンティティー Bean 間の単一関係をサポートするデータが含まれます。最初、関係は (VisualAge for Java で定義される) 同じグループ内のエンティティー Bean のみ参照します。サブクラスは、エンティティー Bean 間の単一値フォワード関係のキー・フィールドとアクセサーを継承します。ただし、多値フォワード関係の finder は、ターゲット・クラスのサブクラスではなく、そのホームにのみ生成されます。

単一値の関連の役割をクラスに対して定義すると、適切なパブリックなキー・フィールドが Bean クラスに追加されます。これらの定義の妥当性検査は、配置コードの生成時に行われます。定義済みの関係をサポートする適切な構成が Bean にない場合は、コード生成は停止します。

オブジェクト・サービス

WebSphere Application Server は、EJB 環境に対して以下のオブジェクト・サービスをサポートしています。

- 『ネーミング・サービスとディレクトリー・サービス』
- 59ページの『セキュリティー・サービス』
- 63ページの『パーシスタンス・サービス』
- 67ページの『トランザクション・サービス』

Java および共通オブジェクト・リクエスト・ブローカー・アーキテクチャー (CORBA) の両方のオブジェクト・サービスをサポートしています。 Java のオブジェクト・サービスは、Java Naming and Directory Interface (JNDI)、Java Transaction Service (JTS)、および Java データベース・コネクティビティー (JDBC) などの Java の標準インターフェースのアドバンスド版 Application Server インプリメンテーションに適用されます。 CORBA のオブジェクト・サービスは、エンタープライズ版 Application Server のインプリメンテーションに適用されます。インターオペラビリティーの問題については、155ページの『第2部 WebSphere Application Server の使用』を参照してください。オブジェクト・サービスは、アドバンスド版 Application Server とエンタープライズ版 Application Server の両方に適用されます。

ネーミング・サービスとディレクトリー・サービス

ネーミング・サービス を使用すると、Enterprise Beans を名前で検索することができます。多くのネーミング・サービスは、*ディレクトリー・サービス* で拡張されています。ディレクトリー・サービスを使用するとオブジェクトに属性を持たせることができます。ディレクトリー・サービスを使用すると、Enterprise Bean を検索する以外にも、Enterprise Beans の属性がわかり、その属性で Enterprise Bean を検索することができます。

WebSphere Application Server は、以下の 3 つのネーミング・サービスとディレクトリー・サービスをサポートしています。

- Java Naming and Directory Interface (JNDI) は、Java アプリケーションのためのディレクトリーおよびネーミング機能を備えています。 JNDI は、特定のディレクトリー・サービスのインプリメンテーションには依存しない、ディレクトリーへの一般的なアクセス方法を提供します。 JNDI の Service Provider Interface (SPI) を使用すると、JNDI を使用するアプリケーションからさまざまなディレクトリーにアクセスすることができます。
- 共通オブジェクト・リクエスト・ブローカー・アーキテクチャー (CORBA) の CosNaming ネーミング・サービスを使用すると、オブジェクト・リクエ

スト・ブローカー (ORB) ベースのシステムのクライアントは、リモート・オブジェクトを見つけることができます。アドバンスド版 Application Server では、基本のネーミング・サポートを必要とする状況で、CosNaming ネーミング・サービスの永続的な Java インプリメンテーションを使用することができます。

- Component Broker ネーミング・サービスは、ORB ベースのシステムに堅固なネーミング・サービスを提供します。Component Broker ネーミング・サービスについては 95ページの『実行時アーキテクチャー』で説明します。

ネーミング・サービスのコンポーネント

WebSphere Application Server のネーミング・サービスには、以下の 3 つのコンポーネントがあります。

- ロケーション・サービス・デーモン
- 永続的ネーム・サーバー
- EJB サーバー

図13 は、ネーミング・サービスのコンポーネントをシステム・レベルで示したものです。

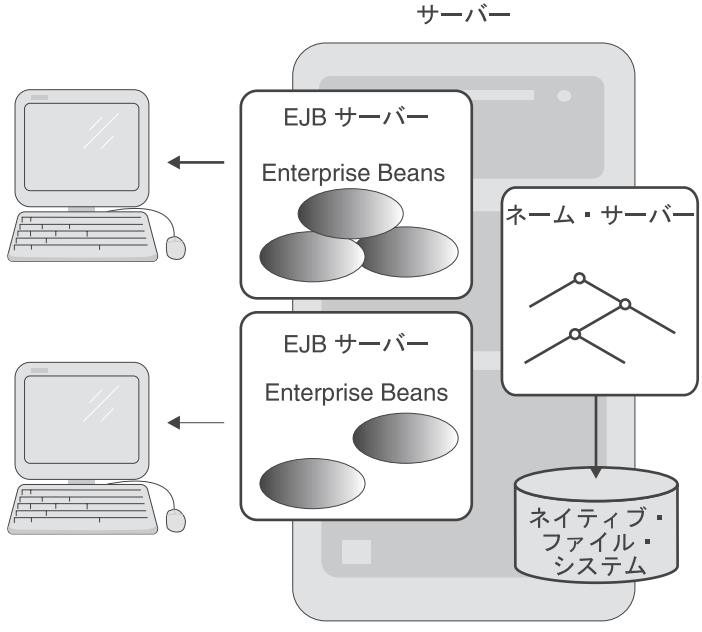

図 13. WebSphere Application Server のオブジェクト・サービス・システム

ロケーション・サービス・デーモン: ロケーション・サービス・デーモンは、ネットワーク全体で使用されているネーム・スペースを追跡します。ロケーション・サービス・デーモンが実行するタスクは、EJB クライアントが使用するネーミング・サービスによって異なります。

- JNDI を使用する EJB クライアントは、セッションの初期コンテキストを要求します。次に EJB クライアントは、オブジェクト要求を待っているロケーション・サーバー・デーモンに暗黙的に接続します。
- CosNaming ネーミング・サービスを使用する EJB クライアントは、オブジェクト要求を発行します。このオブジェクト要求はロケーション・サーバー・デーモンに送られ、ロケーション・サーバー・デーモンはそのオブジェクトのインプリメンテーションが実行されているホストを取り出します。Enterprise Bean を実行しているホストを指す新しい相互利用可能オブジェクト要求 (IOR) がクライアント ORB に戻されます。

永続的ネーム・サーバー: EJB サーバーは、永続的ネーム・サーバーに Enterprise Beans を登録します。各永続的ネーム・サーバーは、ネットワーク内で実行されているネーム・スペースの数には関係なく、独立したネーム・スペースを保守します。

永続的ネーム・サーバーは、ネイティブのファイル・システムを使用して、永続的ストレージで EJB のネーム・スペースを保守します。永続的ネーム・サーバーは、システムの開始時に、ロケーション・サーバー・デーモンに接続し、メモリー内のネーム・スペースのコピーをロードします。キャッシュされていたネーミング情報を使用して、オブジェクトのリストなど、永続的ストレージにあるネーム・スペースのデータに影響を与えない操作を実行します。新規のオブジェクト名の作成など、ネーム・スペースのデータに永続的に影響を与える操作は、メモリー内およびディスク上のコピーに対してトランザクション的に実行されます。

EJB サーバー: Enterprise Beans は、EJB サーバーに配置されます。EJB サーバーは開始時に、すべての EJB ホーム・オブジェクトをネーム・スペースに登録します。この操作を実行するために、ロケーション・サーバー・デーモンのホスト名と開始時のポート番号が EJB サーバーに与えられます。EJB サーバーはネーム・スペースのルートから初期コンテキストを取り出し、JNDI 名を持つ EJB ホーム・オブジェクトをネーム・スペースにバインドします。

セキュリティー・サービス

WebSphere Application Server は統一されたセキュリティー・モデルを提供しています。 1 つのポリシーで Web ページ、サーブレット、および Enterprise Beans のセキュリティーを制御することができます。本節では、以下のトピックについて説明します。

- 『セキュリティー・サーバー』
- 『セキュリティー・コラボレーター』
- 60ページの『セキュリティー・プラグイン』
- 60ページの『認証サービス』
- 61ページの『許可サービス』
- 61ページの『委任ポリシー』
- 62ページの『セキュリティー・サービスの使用』

セキュリティー・アプリケーションは、セキュリティー・サーバー、セキュリティー・コラボレーター、およびセキュリティー・プラグインで構成されます。

セキュリティー・サーバー

セキュリティー・サーバーで、セキュリティー・ポリシーとセキュリティー・サービスを集中制御することができます。セキュリティー・サーバーは、Web サーバーと EJB サーバーに、以下のものを提供します。

- 認証、許可、および委任の各ポリシー
- 軽量の第三者認証 (LTPA) モデルを使用した場合のトークン・サービスを含む、認証ならびに許可サービス

セキュリティー・サーバーは、システム管理機能 (SMF) と結合されます。セキュリティー・サーバーのすべてのインスタンスは、そのセキュリティー・ポリシーを SMF を介して取得します。

セキュリティー・コラボレーター

セキュリティー・コラボレーターは、セキュリティー・サーバーと共に働き、Enterprise Beans のリモート・メソッドの呼び出しのたびに、以下のサービスを実行します。

- 許可の検査
- セキュリティー・トレース情報のログ記録
- 委任ポリシーの実施

セキュリティー・コラボレーターは、Lightweight Directory Access Protocol (LDAP) と Windows NT や UNIX などのオペレーティング・システムに基づくユーザー・レジストリーをサポートしています。セキュリティー・コラボレーターは、アドバンスド版およびスタンダード版 Application Servers の単一ユーザー・レジストリーもサポートしています。

セキュリティー・プラグイン

セキュリティー・プラグインは、Web サーバーに常駐し、HTML ページ、サーブレット、それに JSP ページなどのリソースへのアクセスを保護します。セキュリティー・プラグインは、セキュリティー・サーバーに、認証サービスと許可サービスを求めます。

認証サービス

認証スキームの基本的な前提事項は、クライアントとサーバーは互いを信用しない、というものです。 WebSphere Application Server では、認証は、(ユーザー ID やパスワードなどの) 信任証、証明書、あるいはトークンの妥当性検査に基づきます。信任証は、ユーザー・レジストリーと照合して検査されます。認証が Secure Sockets Layer (SSL) の相互接続を介してユーザーによって提供されたクライアント証明書に基づいている場合は、証明書検査リストが使用されます。

WebSphere Application Server は、第三者認証スキームをサポートしています。クライアント・プリンシパルとサーバー・プリンシパルは、両者から信用されている第三者、たとえば 認証トークン・サーバーに認証されます。これにより、ユーザー・レジストリーを集中管理することができます。

プリンシパルには、システム・リソースへのアクセスを制御するために使用する多くの属性があります。これらの属性は、個々のユーザー ID またはグループを介して管理することができます。可能な場合、グループに対するアクセス制御を管理してください。個々のユーザーに対するアクセス制御を管理することは、一度に多くの属性を変更する必要がある場合には面倒になります。

ユーザーと、Web サーバーまたは EJB サーバー間の認証を行うためのポリシーは、次のメカニズムによって指定することができます。

- チャレンジ・メカニズム は、サーバーがユーザーから認証データを受け取る方法を指定します。チャレンジ・メカニズムは、ユーザー ID とパスワード、または証明書に基づきます。チャレンジ・メカニズムの使用はオプションです。

- *認証メカニズム* は、認証データをユーザー・レジストリーと照合して妥当性を検査します。認証メカニズムには、ネットワーク・ユーザー・レジストリー認証、LTPA、およびオペレーティング・システム・ベースの認証が含まれます。

認証ポリシーの一部として、一組の*制約* を指定することができます。これには、安全なチャネル (たとえば、SSL 接続が必要なもの) を使用する、あるいは信用されているクライアントのグループにアクセスを制限する、などが含まれます。

許可サービス

WebSphere Application Server は、セキュリティーに機能ベースのモデルを使用しています。個々のリソースをアプリケーションに集め、メソッドをメソッド・グループに集めます。各ユーザーは、ユーザーによる実行が許可されているアプリケーション内のメソッドを識別する (アプリケーションとメソッド・グループの) ペアのセットを持っています。各 (アプリケーションとメソッド・グループの) ペアは、許可と呼ばれます。 WebSphere Application Server の管理者は、許可を管理します。ユーザーが操作の実行を試行したときに、セキュリティーのランタイムによって、アクセスを認める許可が決定されます。

Enterprise Bean の許可ポリシーは、以下の方法のいずれか 1 つで管理することができます。

- EJB アプリケーションの許可ポリシーは、アプリケーションに含まれる Home オブジェクトと Home オブジェクトに含まれる Enterprise Beans に適用されます。これにより、Home オブジェクトのメソッドが Enterprise Bean のメソッドと異なるメソッド・グループに割り当てられる場合を除き、Enterprise Bean のインスタンスとは関係なく、許可ポリシーが Home オブジェクト・インスタンスに適用されます。
- Enterprise Beans の許可ポリシーは、Home オブジェクトのインスタンスには適用されません。したがって、許可ポリシーを単独で定義することができます。

委任ポリシー

通常、メソッドは、その操作を発行したプロセスのプリンシパルの下で実行されます。ただし、メソッドを別のプリンシパルの下で実行する必要がある場合があります (たとえば、アクセスを許可されていないリソースをクライアントが使用する場合)。メソッドは、クライアントの識別、システム・サーバーあるいはサーバー・グループの識別、あるいは企業のユーザー・レジストリーで指定された識別で実行することができます。企業のユーザー・レジストリーがオ

ペレーティング・システムのレジストリーではない場合、この識別がオペレーティング・システムの識別にマップされる必要はありません。

Enterprise Bean のメソッドのデフォルトの委任ポリシーは、そのデプロイメント・ディスクリプターから生じます。デフォルトの委任ポリシーは、メソッドまたは Bean に対して委任を指定することができます。管理者は、WebSphere Administrative Console を使用して委任ポリシーを変更することができます。

セキュリティー・サービスの使用

クライアントが Enterprise Bean またはその Home オブジェクトのメソッドを実行する場合、EJB サーバーはプリンシパル (クライアント) がそのメソッドの実行を許可されているかどうかを判別します。

図14 は、Account Enterprise Bean の getBalance メソッドにアクセスするプリンシパル Teller の例を示したものです。

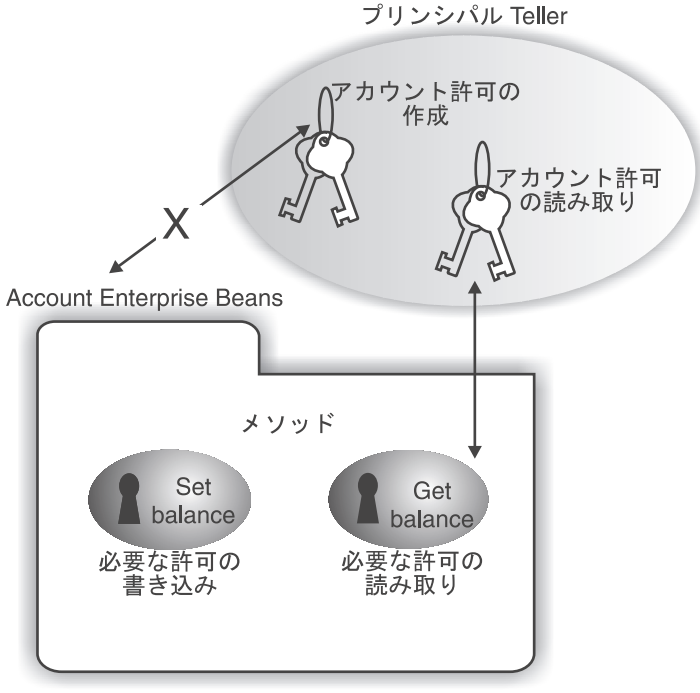

図 14. EJB メソッドの許可ベースの保護

アクセスが許可されているかどうかは、次のステップで判別されます。

1. EJB サーバーは、委任ポリシーを使用して、検査する識別 (この場合はクライアントの識別) を判別します。

2. EJB サーバーは、クライアントのプリンシパルを識別します。プリンシパルを判別できない場合は、要求は拒否されます。この例の場合、Teller はプリンシパルとして識別されています。
3. メソッドに対応する許可セットが解決されます。この例では、getBalance メソッドを使用するために必要な許可は、AccountRead 許可です。
4. Teller が AccountRead 許可を持っている場合は、getBalance メソッドが呼び出されます。

パーシスタンス・サービス

本節では、WebSphere Application Server のパーシスタンス・サービスの基本となるコンポーネントとインターフェースについて説明します。これには、以下が含まれます。

- 『コネクター Bean』
- 64ページの『アダプター Bean』
- 64ページの『パーシスター』
- 65ページの『共通コネクター・フレームワーク (CCF)』
- 67ページの『パーシスタンスのインプリメント』

コネクター Bean

コネクター *Bean* は、アプリケーション、コンポーネント、およびデータベースへのアクセスを単純化する JavaBeans です。コネクター Bean を使用すると、サーブレットと JSP ページにパーシスタンスをインプリメントすることができます。コネクター Bean には、次の 2 つのタイプがあります。

- **データ・コネクター Bean** − リレーショナル・データベースへのアクセスをサポートします。
- **手続き型コネクター Bean** − 既存の手続き型システムと単一の呼び出しおよびその戻りとの対話をカプセル化します。

WebSphere Application Server、VisualAge for Java および WebSphere Studio は、コネクター Bean の以下のインプリメンテーションをサポートおよび生成します。

- JDBC
- SQLJ
- 顧客情報管理システム (CICS)
- ECI、EPI および EXCI
- MQSeries
- Enterprise Beans へのアクセス

- Internet Management Specification (IMS) のトランザクション

サーブレットはコネクター Bean を使用して、JDBC と SQLJ を介して関係データ・ソースにアクセスします。また、サーブレットは、コネクター Bean を使用して、CICS のトランザクションなどの外部アプリケーションとのインターフェースも呼び出します。サーブレットは、Enterprise Beans を呼び出して、暗黙的にパーシスタンス・サービスを使用することもできます。サーブレットの詳細については、27ページの『第4章 サーブレットの使用』を参照してください。

JSP ページは、UseBean タグを使用してコネクター Bean にアクセスします。サーブレットと JSP ページについては、19ページの『第3章 JavaServer Pages の使用』を参照してください。

アダプター Bean

アダプター *Bean* は、リレーショナル・データベース管理スキーマをサポートする JavaBeans です。アダプター Bean を使用すると、サーブレットと JSP ページにパーシスタンスをインプリメントすることができます。アダプター Bean は、Java データベース・コネクティビティー (JDBC) および IBM 共通コネクター・フレームワーク (CCF) などのパーシスタンス・インターフェースを直接使用します。

WebSphere Studio および VisualAge for Java は、アダプター Bean を生成するためのツールを提供しています。これらのツールによるアダプター Bean のインプリメンテーションにより、JDBC、SQL、行セット、およびその他のデータベース照会言語を簡単に使えるようになります。このインプリメンテーションは、WebSphere のデータベース接続マネージャーとも対話します。

アダプター Bean には、サーブレットから、または JSP ページの UseBean タグを使用して直接アクセスすることができます。

パーシスター

WebSphere Application Server の Enterprise Bean のプログラミング・モデルは、ヘルパー・クラス内で JDBC とその他のパーシスタンス API を直接使用することを推奨しています。このヘルパー・クラスは、パーシスター と呼ばれます。パーシスター・クラスは、VisualAge for Java などの EJB 開発ツールによって生成されます。

パーシスターは、永続データを直接保持している Enterprise Beans によってインプリメントされます。BMP を持つエンティティー Bean は、パーシスター内部で (JDBC などの) Java パーシスタンス・サービスを呼び出すことによっ

て、永続データの作成、取り出し、および変更を行う機能をインプリメントします。CMP を持つエンティティー Bean は、コンテナーによって作成されるヘルパー・クラスと、永続状態データを保持するための配置ツールに依存しているため、CMP を持つエンティティー Bean では、パーシスターをインプリメントする必要はありません。セッション Bean は永続的ではありませんが、エンティティー Bean を使用するか、パーシスター内の他の Java パーシスタンス・サービスを呼び出すことによって永続状態データを保持することができます。

Enterprise Beans の詳細については、『Enterprise JavaBeans』を参照してください。

共通コネクター・フレームワーク (CCF)

CCF は、コネクター を開発するためのフレームワークを提供します。コネクターは、クライアント / サーバー・プロトコルのクライアント側をインプリメントするソフトウェア・パッケージまたはライブラリーです。コネクターの例として、JECI for CICS、JBAPI for SAP、および JDBC があります。

これらのパッケージはすべてサーバー・システムとの接続を作成します。各接続は、要求の送信先のサーバー・インスタンス (たとえば CICS 領域やデータベースのテーブル) を定義します。さらに、コネクターとサーバーは、セキュリティーの委任やトランザクションへの関与などのユーザー情報に接続を関連付けます。

66ページの図15 は、CCF クライアント・インターフェースと CCF インフラストラクチャー・インターフェースの 2 つのインターフェース・グループに分割された CCF アーキテクチャーを示したものです。

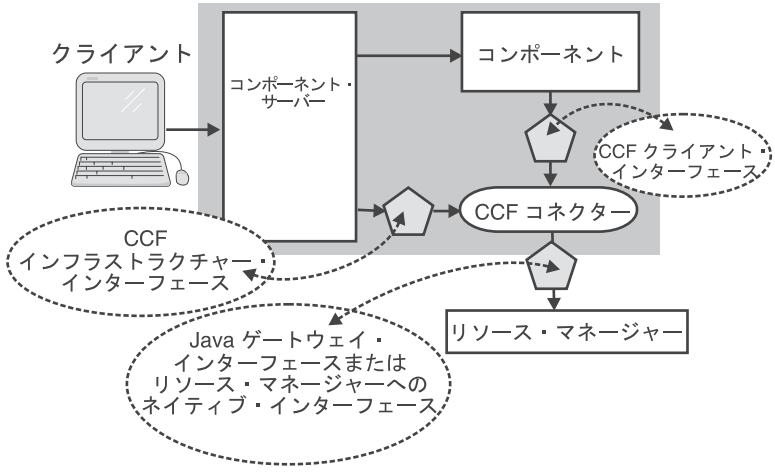

図 15. CCF アーキテクチャー

コネクターは CCF クライアント・インターフェースを使用して、リソース・マネージャーと対話します。

CCF インフラストラクチャー・インターフェースには、以下の 2 つのインターフェースがあります。

- **サービス品質 (QOS)** − このインターフェースは、コンポーネント・サーバーによってインプリメントされます。QOS は、CCF コネクターのサービス品質面を、コンポーネント・サーバーのサービスの実際のインプリメンテーションに適合させます。QOS は、セキュリティー情報の取り出しや、現行トランザクションへの関与などのタスクに使用することができます。
- **状態管理 (SM)** − このインターフェースは、CCF コネクターによってインプリメントされます。SM は、コンポーネント・サーバーの QOS インプリメンテーションが CCF コネクター状態を制御するために使用します。SM によって制御される状態には、物理接続と、(現行トランザクションに関しては) 接続されているリソース・マネージャーのトランザクション状態があります。

内部的には、CCF コネクターは、Java ゲートウェイ・インターフェースやネイティブ・インターフェースを介してアクセスできる、メーカー独自のコネクターを使用します。

VisualAge for Java のエンタープライズ・アクセス・ビルダー (EAB) ツールで 2 つの Java Beans を定義することにより、CCF コネクターを容易に使用することができます。

- **コマンド Bean** (またはコマンド) は、アプリケーション・システムとの単一の対話をカプセル化します。コマンド Bean を使用すると、CCF コネクターに対する呼び出しを手動でスクリプトする必要がなくなります。CCF のクライアント・インターフェースは EAB コマンドと対話するため、EAB コマンドで CCF ベースのコネクターを使用することができます。
- **ナビゲーター Bean** (またはナビゲーター) は、アプリケーション・システムとの連続した対話をインプリメントします。ナビゲーター Bean をコマンド Bean や他のナビゲーター Bean と組み合わせると、一連の複雑な対話を作成することができます。

EAB、コマンド Bean、ナビゲーター Bean の詳細については、VisualAge for Java の資料を参照してください。

パーシスタンスのインプリメント

WebSphere Application Server のプログラミング・モデルは、Java の汎用クラスの使用をサポートしています。開発者は任意のクラス・ライブラリーを使用して、サーブレット、JavaBean、セッション Bean、あるいは BMP を持つエンティティー Bean 内にパーシスタンスをインプリメントすることができます。コンポーネントの再利用と移動を容易に実行するためには、以下のパーシスタンスのインプリメントのためのガイドラインに従ってください。

- CCF をカプセル化するアクセス Bean を使用します。CCF を直接使用しないでください。このタイプの Bean の使用についての詳細は、50ページの『アクセス Bean』を参照してください。
- JDBC 呼び出しを、セッション Bean と BMP を持つエンティティー Bean の予測子にカプセル化することができます。 JDBC 呼び出しを、サーブレットおよび JSP ページのための JavaBeans コンポーネントにカプセル化することができます。
- セッション Bean と BMP を持つエンティティー Bean は、既存システムの接続をアクセス Bean に委任する予測子を介して、既存システムにアクセスすることができます。

トランザクション・サービス

トランザクションのサポートは、EJB アーキテクチャーの極めて重要なコンポーネントです。WebSphere Application Server は、Enterprise Beans にトランザクション処理サービスを提供しています。本節では、以下のトランザクション関連のトピックについて説明します。

- 68ページの『Enterprise Beans のためのトランザクション管理』
- 69ページの『トランザクション属性』

- 70ページの『トランザクション分離レベル属性』
- 70ページの『ロック』
- 70ページの『OTS および EJB トランザクション・モデル』
- 73ページの『セッションの同期化のインプリメント』

Enterprise Beans のためのトランザクション管理

EJB アーキテクチャーは、Enterprise Beans のためのトランザクション処理を管理するための方法として、以下の 2 つを提供しています。

- *Bean 管理のトランザクション* では、Enterprise Bean がトランザクションを制御します。
- *コンテナー管理のトランザクション* では、コンテナーがその Enterprise Beans に代わってトランザクションを制御します。

セッション Bean は、コンテナー管理のトランザクションか Bean 管理のトランザクションのどちらかを使用することができます。エンティティー Bean はコンテナー管理のトランザクションを使用する必要があります。トランザクション処理は、EJB サーバーが扱います。サーバー提供者は、2 フェーズ・コミット、トランザクション・コンテキストの伝達、および分散 2 フェーズ・コミットなどのトランザクション・サービスを使用することができます。ネストされたトランザクションは、サポートされていません。

EJB クライアントもトランザクションを管理することができます。EJB 環境におけるトランザクション管理の詳細については、「*Enterprise Beans の作成*」を参照してください。

Bean によるトランザクションの管理: 必要に応じて、セッション Bean がトランザクションに直接関与することがあります。セッション Bean がトランザクションのアクティブな関係者であることを示すためには、そのデプロイメント・ディスクリプターのトランザクション属性を TX_BEAN_MANAGED に設定します。セッション Bean の開発者は、javax.transaction.UserTransaction インターフェースを使用して、トランザクションを明示的に分離する必要があります。 (このインターフェースもクライアント管理されるトランザクションに使用することができます。) セッション Bean のデプロイメント・ディスクリプター内のトランザクション分離レベルも設定してください。

コンテナーによるトランザクションの管理: EJB API では、Enterprise Bean および EJB アプリケーションの開発者が、トランザクションを使用するための特別なコードを作成する必要はありません。コンテナーがトランザクションを管理するため、Enterprise Bean および EJB のアプリケーション開発者がアプリケーションのビジネス・ロジックに煩わされずに済みます。

EJB クライアントが Enterprise Bean のメソッドを呼び出すと、コンテナーがそのメソッド呼び出しを代行受信してそのトランザクションを管理します。コンテナーがトランザクションの分離を管理する方法は、そのメソッドまたは Enterprise Bean のどちらかのトランザクション属性で指定されます。

コンテナー管理のトランザクションを使用可能にするには、Enterprise Bean のデプロイメント・ディスクリプターのトランザクション属性を TX_BEAN_MANAGED 以外の任意の値に設定します。トランザクション属性の値は、アプリケーションの要件によって異なります。また、トランザクションの分離レベルも設定する必要があります。詳細については、70ページの『トランザクション分離レベル属性』を参照してください。

トランザクション属性

トランザクション属性は、コンテナーが Enterprise Bean のメソッドを呼び出す際のトランザクションの形態を定義します。この属性は、Bean 全体に設定することも、Bean の個々のメソッドに設定することもできます。この属性の値は、以下のとおりです。

TX_BEAN_MANAGED

トランザクションの境界を明示的に管理するためのメソッドを Bean が呼び出すことができることをコンテナーに通知します。(詳細については、68ページの『Bean によるトランザクションの管理』を参照してください。)

TX_MANDATORY

コンテナーは、クライアントに関連付けられたトランザクション・コンテキスト内で常に Bean メソッドを呼び出します。

TX_NOT_SUPPORTED

コンテナーは、トランザクション・コンテキストなしで Bean メソッドを呼び出します。

TX_REQUIRES_NEW

メソッドが既存のトランザクション・コンテキスト内で呼び出されるかどうかに関係なく、コンテナーは常に新しいトランザクション・コンテキスト内で Bean メソッドを呼び出します。

TX_REQUIRED

コンテナーは、トランザクション・コンテキスト内で Bean メソッドを呼び出します。メソッドがトランザクション・コンテキスト外で呼び出された場合は、コンテナーは新しいトランザクション・コンテキストを作成し、そのコンテキスト内からその Bean メソッドを呼び出します。

TX_SUPPORTS

クライアントがトランザクション・コンテキスト内で Bean メソッドを呼び出した場合は、コンテナーは、トランザクション・コンテキスト内で Bean メソッドを呼び出します。クライアントがトランザクション・コンテキストなしで Bean メソッドを呼び出した場合は、コンテナーは、トランザクション・コンテキストなしで Bean メソッドを呼び出します。

トランザクション分離レベル属性

トランザクション分離レベルは、あるトランザクションを別のトランザクションから分離する方法を決定します。この分離は読み取り専用です。トランザクション分離レベルは、Bean 全体に対して設定することができます。トランザクション内での最初のメソッド呼び出しでは、Bean の分離レベルが使用されます。それ以降の呼び出しでは異なる分離レベルは無視されます。

コンテナーは、以下のようにトランザクション分離レベル属性を使用します。

- **Bean 管理のパーシスタンス (BMP) を持つセッション Bean およびエンティティー Bean** − Bean が使用するデータベース接続ごとに、コンテナーは各トランザクションの開始時にトランザクション分離レベルを設定します。
- **コンテナー管理のパーシスタンス (CMP) を持つエンティティー Bean** − コンテナーは、指定された分離レベルを実現するデータベース・アクセスを生成します。

ロック

Enterprise Beans に対して書き込みロックを定義することはできません。その対処方法としては、メソッドを読み取り専用として定義するか、読み取り専用でないメソッドに対してはデータベースの行に対する書き込みロックを獲得します。

OTS および EJB トランザクション・モデル

EJB のトランザクション・モデルは、Object Transaction Service (OTS) のトランザクション・モデルに類似しています。 OTS のキーとなるコンポーネントは、ほぼ直接 EJB のトランザクション・サービスにマップすることができます。これらのトランザクション・モデルの動作を理解すると、EJB 環境におけるトランザクション動作の理解に役立ちます。

EJB トランザクション・モデルは、Java トランザクション API (JTA) および Java トランザクション・サービス (JTS) を使用します。JTA は、トランザクション・マネージャーと、アプリケーション、リソース・マネージャー、およびトランザクションに関与するアプリケーション・サーバーとの間のインター

フェースを指定します。JTS とは、OTS をバインディングする Java プログラム言語です。JTS は、サーバー間でトランザクションを伝達するための標準の IIOP プロトコルを提供します。

OTS トランザクション・モデルを 図16 に示します。波線で囲まれた部分はトランザクションを表します。この部分には、トランザクションに関与するオブジェクトがすべて含まれています。コミットとロールバックは、このグループ内のすべてのリソース・オブジェクトに適用されます。

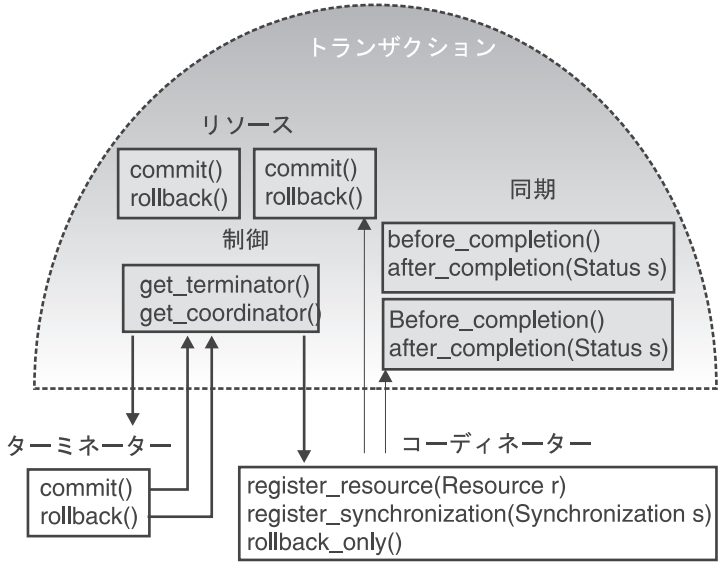

図 16. OTS トランザクション・モデル

制御オブジェクト

トランザクションを表します。Enterprise Bean は、制御オブジェクトに直接関係することはありません。その代わりに、コンテナーが制御オブジェクトを使用して、Bean の代わりにトランザクションを管理します。

ターミネーター・オブジェクト

メソッドが戻ったときにコンテナーがトランザクションを終了する必要がある場合、コンテナーがこのオブジェクトを使用してトランザクションをコミットまたはロールバックします。

リソース・オブジェクト

2 フェーズ・コミット・プロトコルをインプリメントします。たとえば、リソース・オブジェクトがデータベースとの接続を表す場合、トラ

ンザクションをコミットすると、そのデータベースは更新されます。トランザクションをロールバックすると、トランザクションの開始以降この接続を介して行われたデータベースのすべての変更は元に戻されます。コミットまたはロールバックが完了した後、データベース内の対応する行がアンロックされます。ロック・レベルは、デプロイメント・ディスクリプターで指定されています。リソース・オブジェクトは、トランザクション全体をコミットするかあるいはロールバックするかについて決定します。

同期オブジェクト

完了済みトランザクションがコミットされたのかあるいはロールバックされたのかについて通知されます。リソース・オブジェクトと異なり、同期オブジェクトは 2 フェーズ・コミット・プロトコルには関与せず、トランザクションでは受動的な役割を果たします。 (セッション Bean が、特殊なインターフェースをインプリメントすることにより、この役割を果たします。)

コーディネーター・オブジェクト

トランザクションに関与するリソース・オブジェクトと同期オブジェクトを登録します。Enterprise Bean はこのオブジェクトに直接アクセスしません。Enterprise Bean と共に使用することを目的としたトランザクション対応オブジェクトは、現行トランザクションのコーディネーターへの参照を透過的に得ることにより、自己登録します。

OTS はトランザクション・オブジェクトと*リカバリー可能* オブジェクトを区別します。この区別は Enterprise Beans に関連しています。

- トランザクション・オブジェクトは、トランザクションに関連付けられます。トランザクション・オブジェクトは、コミット・メソッドおよびロールバック・メソッドを持たないため、トランザクションがこのオブジェクトを直接操作することはできません。トランザクション・オブジェクトは、そのオブジェクトの現行トランザクションに関連付けられているリカバリー可能オブジェクト (またはリソース) のマネージャーとして機能します。
- リカバリー可能オブジェクトはコミット・メソッドとロールバック・メソッドを持っているため、トランザクションはそのオブジェクトの状態または振る舞いを直接操作することができます。

トランザクション・オブジェクトの例として、コンテナー管理のトランザクションを使用する Enterprise Bean があります。コンテナーは、Enterprise Bean に代わってトランザクションを保守します。Enterprise Bean によって割り当てられたリカバリー可能オブジェクトは、コンテナーの支援を受けて、現行トランザクションに透過的に配置されます。 Enterprise Bean は、コミット・メソ

ッドあるいはロールバック・メソッドを持たないため、トランザクションは Enterprise Bean を直接操作することはできません。 (リカバリー可能リソースである Enterprise Bean を作成するには、追加作業が必要です。Enterprise Beans がトランザクションの結果に直接影響を与える内部状態を持つことはほとんどありません。)

Bean は、コンテナーがコミットまたはロールバックを行おうとする前に、トランザクションのロールバックを表明できます。その場合、Bean は、セッションの同期インターフェースを介してトランザクションの結果の通知を受け取ります。

セッションの同期化のインプリメント

セッション Bean は、コンテナーのコールバックの形で、トランザクションの同期の通知を Bean に提供するインターフェースを、オプションでインプリメントすることができます。セッション Bean はこれらの通知を使用して、トランザクション内でキャッシュされたデータベース・データを管理します。

afterBegin 通知は、セッション・インスタンスに、新しいトランザクションが開始されたことを知らせます。この時点で、このインスタンスはすでにトランザクションに入っており、そのトランザクションの有効範囲内で、必要なデータベース作業を行うことができます。

beforeCompletion 通知は、セッション・インスタンスのクライアントはその現行トランザクションに対する作業を完了したが、セッション・インスタンスがまだそのリソースをコミットしていない場合に発行されます。この時点で、セッション・インスタンスは、キャッシュしたデータベースの更新をすべて書き込む必要があります。このインスタンスは、そのセッション・コンテキストの *setRollbackOnly* メソッドを呼び出して、そのトランザクションをロールバックすることができます。

afterCompletion 通知は、現行トランザクションが完了したことを知らせます。完了状態が真 の場合は、トランザクションがコミットされたことを示します。完了状態が偽 の場合は、ロールバックされたことを示します。

アドバンスド版 Application Server では、エンティティー Bean は、セッション同期インターフェースをインプリメントすることができ、トランザクションが開始または完了すると通知を受けることができます。このタイプの セッション同期は、テスト時に処理されるトランザクション数を見つけ出すための、155ページの『第2部 WebSphere Application Server の使用』の例で使用されています。エンティティー Bean が afterBegin メソッドを呼び出すたびに、関連情報がコンソールに表示されます。

注: エンティティー Bean へのセッション同期のインプリメントは、すべてのコンテナー・ベンダーでサポートされているわけではありません。

第6章 Web アプリケーションの開発

JSP ページ、Enterprise Beans、およびサーブレットを使用して、WebSphere Application Server のすべての版において、Web アプリケーションをインプリメントすることができます。本節では、これらのコンポーネントを使用した Web アプリケーションの開発に関する事項について説明します。本節で説明するトピックは以下のとおりです。

- 『Web アプリケーション・プログラミング・モデル』
- 77ページの『Web アプリケーションにおける JSP ページ、サーブレット、および Enterprise Beans の使用』

Web アプリケーションのコンポーネントの詳細については、以下を参照してください。

- 19ページの『第3章 JavaServer Pages の使用』
- 27ページの『第4章 サーブレットの使用』
- 35ページの『第5章 Enterprise Beans の使用』

Web アプリケーション・プログラミング・モデル

Web アプリケーション・プログラミング・モデルは、多階層アーキテクチャーに基づいて構築されています。エンタープライズ・アプリケーションは別々のコンピューター上で実行される複数のコンポーネントから構成されており、それらのコンポーネントによって一連の物理層が形成されています。アプリケーション・コンポーネントは、実行する機能ごとに論理層に割り当てられます。物理層と論理層は直接対応している必要はないため、たとえば、別々の論理層に割り当てられているアプリケーション・コンポーネントを、同一の物理層で稼働させることができます。Web アプリケーション・プログラミング・モデルは、高機能 Web アプリケーションを備えているシン・クライアントとエンタープライズ・サーバーのサポートを目的として開発されました。多階層アーキテクチャーおよびクライアント・トポロジーの詳細については、「*WebSphere Application Server 概説*」を参照してください。

76ページの図17 は多階層アプリケーションの例を示しています。ブラウザーに表示されるコンテンツは Web サーバーによって提供されます。Web サーバー

は、コンテンツを提供するため、データベースおよびトランザクション処理モニターを管理するリソース・マネージャーと通信します。

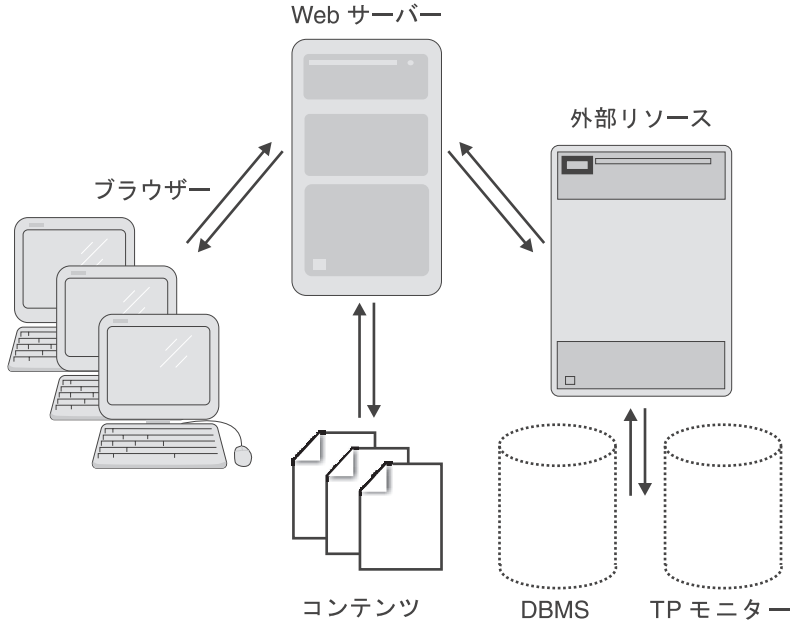

図 17. Web アプリケーション・コンポーネント

第 1 階層

第 1 階層には、クライアント (この例では Web ブラウザー) が含まれています。クライアントは、HTTP (Hypertext Transfer Protocol) や IIOP (Internet Inter-ORB Protocol) などの業界標準のプロトコルを使用して Web サーバーと通信します。

第 2 階層

第 2 階層 (中間層) は、クライアントとエンタープライズ・サーバーのリソースおよびデータとの間にあります。アプリケーション・サーバー (WebSphere Application Server など) は、中間層に位置しています。図の例の場合、中間層には Web サーバーが含まれており、この Web サーバーが静的コンテンツや動的コンテンツから構成される Web ページを調整し、集めて組み立て、クライアントに送信します。第 2 階層には、第 3 階層へのアクセスを制御するリソース・マネージャーも含まれます。

この層は、Web アプリケーション・プログラミング・モデルの中心となる層です。CGI (コモン・ゲートウェイ・インターフェース) ベースのプログラミング・モデルを依然として使用している Web アプリケーションも多数ありますが、Java ベースのプログラミング・モデルが主流となりつつあります。WebSphere Application Server のサンプル・アプリケーションでは、中間層のロジックはサーブレットおよび JSP ページを使用して、Java によってインプリメントされます。Enterprise Beans および Component Broker のビジネス・オブジェクトは、アプリケーションのビジネス・ロジックをインプリメントします。Java Beans は、サーブレットとビジネス・ロジックとの間のインターフェースとして機能します。

第 3 階層

第 3 階層は、データベースおよび外部トランザクション処理モニター (CICS など) など、組織全体で使用するリソースを含んでいます。第 3 階層のリソースは第 2 階層によって管理されます。

Web アプリケーションにおける JSP ページ、サーブレット、および Enterprise Beans の使用

JSP ページとサーブレットは、相補的なテクノロジーです。本節では、Web アプリケーションでこれらを共に使用する方法に関して以下の項目について説明します。

- 『Model-View-Controller アーキテクチャーのインプリメント』
- 79ページの『Web アプリケーションの状態情報の保持』
- 81ページの『Web アプリケーションでのセキュリティーのインプリメント』

Model-View-Controller アーキテクチャーのインプリメント

Model-View-Controller (MVC) アーキテクチャーではアプリケーションを以下のパーツに分けています。

- **モデル** − アプリケーションが内部的にどのように動作するかを示すもの (アプリケーションのビジネス・ロジックなど) です。
- **ビュー** − クライアントがモデルの状態 (クライアント・ユーザー・インターフェースなど) を確認するための方法です。
- **コントローラー** − クライアントがアプリケーションの状態を変更したり、アプリケーションへの入力データを提供したりするための方法です。

MVC アーキテクチャーを Web アプリケーションに適用するために、さまざまな WebSphere コンポーネント・アーキテクチャーを以下のように使用することができます。

- ビジネス・ロジックをインプリメントする JavaBeans コンポーネントおよび Enterprise Beans をモデルとして使用する。
- 動的なコンテンツを表示するため、JSP ページをビューとして使用する。
- サーブレットをコントローラーとして使用する。サーブレットは、タスク (動的なコンテンツの生成など) を処理するため、他のエンティティーとの調整を行い、HTTP 接続を処理します。JavaBeans コンポーネントを、コントローラーとモデルの間のインターフェースとして使用できます。

サーブレットは MVC アーキテクチャーのビュー・コンポーネントとして使用することもできますが、あまりお勧めしません。サーブレットで動的なコンテンツを生成すると、出力フォーマットを変更する場合に、コンテンツも変更され、再コンパイルしなければならなくなるからです。その結果、開発者と HTML の作成者との役割分担を明確にするのが難しくなります。さらに、サーブレットの中には実行時に Web サーバーを停止してから再起動し、最新のサーブレットをロードしなければならないものもあります (反対に、WebSphere Application Server のサーブレットの実行時環境では、サーブレットに対して変更が加えられているかどうか必ずチェックされ、自動的に最新のサーブレットがロードされます)。

このような問題は、JSP ページを使用して動的なコンテンツを表示することによって解消できます。JSP ページはサーブレットから独立して動作します。出力フォーマットへの変更は JSP ファイル内で行われるので、サーブレットを再コンパイルする必要はありません。また、JSP ページを使用してコンテンツを表示することにより、HTML の作成者 (JSP ページの保守担当者) と開発者 (ビジネス・ロジックをインプリメントするサーブレットおよびコンポーネントの保守担当者) との役割分担が明確化されます。

155ページの『第2部 WebSphere Application Server の使用』で説明するサンプル・アプリケーションでは、MVC アーキテクチャーを使用しています。

- モデルは、Enterprise Beans および Component Broker の管理下のオブジェクトとしてインプリメントされます。これらのビジネス・コンポーネントは、タスク処理を実行し、口座、顧客、トランザクション・レコードなどの永続的エンティティーを表します。
- ビューは、Web サイトおよび JSP ページによってインプリメントされます。これらは、アプリケーションがクライアントにデータを表示する方法を制御します。

- コントローラーは、サーブレットによってインプリメントされます。サーブレットは、クライアントの要求を Web サイト (ビュー) から受信して、これらをビジネス・コンポーネント (モデル) に渡して処理します。JavaBeans コンポーネントは、コントローラーのサーブレットとモデルのコンポーネント間のコントローラーとして機能します。

Web アプリケーションの状態情報の保持

HTTP はステートレスなプロトコルです。 HTTP ではクライアントから要求が発行されるたびに新しい接続が確立され、要求が発行されてから次の要求が発行されるまでは状態情報が保持されることはありません。つまり、サーバーは、連続する要求が同一のクライアントから発行されたものかどうか識別することができません。

しかし、多くの Web アプリケーションでは、次の要求が発行されるまで情報を保持することが重要な要件となります。したがって、HTTP を使用する Web アプリケーションに状態情報を追加するため、以下のような方法が採用されています。

- Web サーバーによる認証
- 非表示フィールド
- Cookies
- サーブレットによるセッション管理

これらの方法はすべて、セッション (一定時間維持されるブラウザーからの接続) という概念に基づいています。

Web サーバーによる認証

ほとんどの Web サーバーは、ユーザー ID とパスワードを使用してログインしたクライアントにのみリソースへのアクセスを許可するユーザー認証機能を備えています。ユーザー ID はクライアント・セッションをトラックするために使用されます。クライアントがログインすると、ブラウザーによってユーザー ID が保存され、すべての要求と共に送信されます。

この方法には以下のような長所があります。

- インプリメントが容易である。
- ほとんどの Web サーバーで自動的に実行される。
- さまざまなマシンからのクライアント要求を処理できる。

一方、以下のような 2 つの短所もあります。

- ユーザーは Web サイトを見るたびにログインしなければならない。機密情報へのアクセス時にはログインの手順を踏まなければならないことを想定し、そのことに理解を示しているユーザーであっても、公開情報に対してログインを行わなければならないのは過剰制限であると言えます。
- 同一のクライアントの複数のセッションを処理することができない。

非表示フィールド

その名が示すとおり、非表示フィールドはクライアントのブラウザーに表示されない HTML フォーム内のフィールドです。非表示フィールドは、HTML フォームがサブミットされるとサーバーに送信されます。

この方法には以下のような長所があります。

- すべてのブラウザーでサポートされている。
- 特別なサーバーのセットアップは不要である。
- ユーザーはログインする必要がない。

非表示フィールドの短所は、動的に生成されたフォームに対してのみ有効であるという点です。静的な Web ページに対してこの方法を使用することはできません。また、非表示フィールドは、電子メールで送受信されるページおよびブックマーク付きページで使用することはできず、ブラウザーがシャットダウンされている状態で使用することもできません。

Cookies

Cookie は、Web サーバーとブラウザー間で送受信されるデータです。ブラウザーは Web サーバーから Cookie を受信してローカルに保存します。ブラウザーは、サーバー上のページにアクセスする時に、以前に受信した Cookie をサーバーに送信し、サーバーはそれをもとにクライアントを識別します。このような方法をとることによって、セッションを簡単にトラックできるようになります。

Cookies の短所は、ブラウザーで Cookies が拒否されるようにクライアント側で設定でき、このために Cookies によるセッションのトラックの信頼性が失われてしまうことです。

サーブレットによるセッション管理

サーブレットにはセッション管理機能が組み込まれています。サーブレット API は、セッションを管理する多くのクラスおよびインターフェースを定義します。また、サーブレットは、保持している Cookies を使用してセッションの追跡を行います。セッション管理の詳細については、31ページの『サーブレット・セッションの管理』を参照してください。

Web アプリケーションでのセキュリティーのインプリメント

サーブレット、JSP ページ、および Enterprise Beans は、同じセキュリティー・モデルを使用しています。詳細については、 59ページの『セキュリティー・サービス』を参照してください。

インターネットを介して他のアプリケーションと通信する、サーブレットと JSP ページについては、セキュリティーを強化する必要があります。インターネットは双方向通信チャネルであり、企業はインターネットを利用して、多数のユーザーに情報とサービスを提供することができます。また、ユーザーも、インターネットを利用して企業の情報にアクセスすることができます。インターネットを利用してユーザーにサービスを提供する企業は、セキュリティー対策を講じる必要があります。

Web サーバーを保護するためには、Web サーバーを実行するコンピューターを保護することと、Web サーバー自体を保護することが必要です。サーバーを保護するためには、さまざまなレベルのアクセスを設定できるユーザー管理システムをセットアップして、匿名アクセスが禁止されるようにする必要があります。

クライアントのコンピューターを保護するためには、ユーザーにどのソフトウェアの実行許可を与えるか管理しなければなりません。特に重要なソフトウェアは、WWW 上の情報へアクセスする時に使用されるブラウザーです。ブラウザーにはセキュリティーについて問題があることもありますが、クライアントで最新バージョンのブラウザーを使用するようにすれば、そのような問題を解決できる場合もあります。

Web サーバーと受信側の間でデータを転送する場合、情報を保護するには暗号化を行います。サーブレットおよび JSP ページでは、以下の暗号化方式が採用されています。

Secure Sockets Layer (SSL)

SSL では、インターネットを介して送信される情報が自動的に暗号化され、データを読む前に暗号化解除されます。 SSL は未加工の TCP/IP データ・ストリームとアプリケーションの間に存在します。標準の TCP/IP プロトコルでは 2 台のコンピューター間で匿名のデータ・ストリームが送信されますが、SSL を使用することにより、認証、データ保全性などの機能が付加されます。

SSL を使用することによって、ユーザーも、サーバー開発者も共に複雑な暗号化作業を行わなくても済むようになります。SSL では、ディジタル証明書およびディジタル署名付きメッセージによって、クライアント側とサーバー側の両

方で認証を行うことができます。また、接続を暗号化するのではなく、認証することで、悪用されないように保護することもできます。

SSL プロトコルは、man-in-the-middle (中間にいる人による) アタックやリプレイ・アタックを防ぐために開発されました。man-in-the-middle アタックとは、それぞれの通信者には相手と通信しているように思いこませ、すべての通信内容を代わりに受信してしまうことです。リプレイ・アタックとは、通信者間で送受信されるメッセージを保存して、再生 (リプレイ) することです。

ディジタル証明書

ディジタル証明書は、秘密鍵と公開鍵から構成されています。秘密鍵はデータ・ブロックに署名を付けるために使用されます。公開鍵は署名を検証するために使用されます。

図18 に、ディジタル証明書の仕組みを示します。エンティティー A と B がインターネットを介して通信を行っています。エンティティー A がエンティティー B にメッセージを送信します。エンティティー B は秘密鍵を使用してそのメッセージに署名を付け、エンティティー A に戻します。エンティティー A は元のメッセージとエンティティー B の公開鍵でメッセージのコピーを比較して、メッセージの署名が本物であるかどうか確認します。第三者 (エンティティー C) によってメッセージが代わりに受信され、変更が加えられた場合、そのメッセージはエンティティー B の公開鍵と一致しなくなるため、エンティティー A はそのメッセージを拒否することができます。

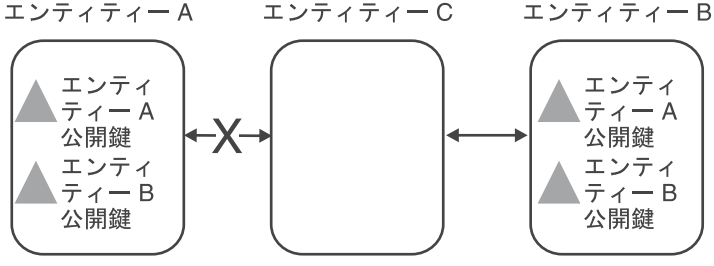

図 18. ディジタル証明書による認証

クライアント証明書

クライアント証明書の目的は、個々のユーザーの ID を確認することです。クライアント証明書は、個々のユーザー名を特定のキーにバインドする役割を果たします。クライアント証明書を使用すれば、ユーザー ID とパスワードを覚えておく必要がなくなります。また、匿名性が排除されるので、Web サイトを

見た人に関する情報を収集するために使用することもできます。クライアント証明書は認証局から購入することができます。

サーバー証明書

サーバー証明書は、SSL をインプリメントする Web サーバーによって発行されます。ブラウザーが SSL を使用して Web サーバーに接続すると、Web サーバーはサーバー証明書として自らの公開鍵をそのブラウザーに送信します。サーバー証明書は、サーバーの ID を認証し、サーバーの公開鍵を配布するために使用されます。この公開鍵を使用して、クライアントはサーバーに送信する初期設定情報を暗号化します。

サーバー証明書は認証局から購入することができます。

第7章 WebSphere Application Server エンタープライズ版

WebSphere Application Server エンタープライズ版 (またはエンタープライズ版 Application Server) は、WebSphere Application Server アドバンスド版を拡張して、企業レベルでの分散トランザクション、および分散オブジェクト・アプリケーションを処理します。これは、TXSeries トランザクション・アプリケーション環境を、Component Broker による完全に分散したオブジェクトとビジネス・プロセスの統合機能を組み合わせます。また、Enterprise Beans 作成用の開発ツールも含まれます。

本節では、エンタープライズ版 Application Server の以下の点について説明します。

- 『エンタープライズ版 Application Server の機能』
- 88ページの『実行時およびシステム管理アーキテクチャー』
- 89ページの『エンタープライズ版アプリケーション開発環境およびツール』

エンタープライズ版 Application Server の機能

エンタープライズ版 Application Server には、アドバンスド版 Application Server の全機能が含まれています。また、以下の製品も含まれています。

Component Broker

Component Broker は、分散オブジェクト・コンピューティング用のエンタープライズ・ソリューションです。これにより、分散コンポーネント・ベースのソリューション開発および配置用のスケーラブルで管理が容易なランタイムが提供されます。Component Broker は、オブジェクト管理グループ (OMG) の Common Object Request Broker Architecture (CORBA) のオープンな標準規格をインプリメントします。 Java および C++ プログラム言語で作成されたコンポーネントをサポートします。

Component Broker は、オブジェクト指向アプリケーション用のアプリケーション・サーバーを提供します。これは、オブジェクトの照会やトランザクションとの統合、通知サービス、ビジネス・ルール、複雑な構成、および継承や関連モデルなどの高度なオブジェクト・サービスを必要とする環境です。また、手続き型のアプリケーションとデータベースと EJB アプリケーションとの統合もサポートします。このアプローチは、ビジネス・プロセスの増分型のリエンジニアリングをサポートします。

Component Broker は、複合照会および複合オブジェクト構成を使用し、トランザクション比率が高い、異質なバックエンド・リソース・マネージャー環境にあるアプリケーションのパフォーマンスを最適化することができます。

Component Broker では、アドバンスド版 Application Server とは別個に Enterprise JavaBeans (EJB) 仕様をインプリメントしています。Component Broker EJB サーバーは、アドバンスド版 Application Server の代わりとして使用することができます。また、2 つの EJB サーバーを同時に使用することもできます。

TXSeries

TXSeries は、企業間における統合を可能にし、高いレベルでのアプリケーションのスケーラビリティー、可用性、整合性、持続性、およびセキュリティーを提供します。これは、分散トランザクション・アプリケーションを作成するための以下の 2 つの一般的なミドルウェア・パッケージから構成されています。

顧客情報管理システム (CICS)

CICS は、IBM の汎用オンライン・トランザクション処理ソフトウェアです。CICS は、デスクトップから最大級のメインフレームに及ぶ広範なオペレーティング・システム上で稼働するアプリケーション・サーバーです。TXSeries CICS は、AIX、Solaris、および Windows NT 上で稼働しますが、OS/390®、OS/400、OS/2®、および VMS で稼働するバージョンの CICS もあります。CICS は、セキュリティー、データ保全性、およびリソースのスケジューリングを扱います。 CICS は、オンライン・トランザクション処理アプリケーションによって必要とされる基本的なビジネス・ソフトウェア・サービスを統合します。CICS は、IBM VisualAge for Java により提供される Enterprise Beans をサポートします。これは、既存の CICS スキル・セットの活用を希望するお客さま向けの WebSphere Application Server プログラミング・モデルを自然な形で発展させたものです。

Encina

Encina は、オープン分散システムの開発と管理に使用するソフトウェア製品のファミリーです。これは、以下の製品から構成されます。

Encina モニター

トランザクション処理アプリケーションの開発、実行、および管理を行うための、トランザクション処理モニターです。

リカバリー可能キューイング・サービス (RQS)

RQS により、アプリケーションがトランザクション作業をキューに入れ、後で処理できるようにします。

構造化ファイル・サーバー (SFS)

SFS は、レコード単位のファイル・システムで、トランザクションの整合性、ログによるリカバリー、および広範なスケーラビリティーを提供します。

対等通信 (PPC) サービス

PPC サービスにより、Encina トランザクション処理システムが、システム・ネットワーク体系 (SNA) LU (論理装置) 6.2 通信インターフェースを持つシステム (一般にメインフレーム) と相互通信をできるようにします。

Encina++

Encina++ は、Encina 用のオブジェクト指向アプリケーション・プログラミング・インターフェース (API) です。CORBA および分散コンピューティング環境 (DCE) の両方がサポートされます。Encina++ サーバーは、C++ プログラム言語で、クライアントは Java または C++ プログラム言語で作成できます。

Encina ツールキット

Encina ツールキットは、モジュール、ライブラリー、およびプログラムの集合で、大規模な分散型クライアント / サーバーのシステム開発に必要な機能を提供します。 C プログラム言語のトランザクション拡張機能である Transactional-C が含まれています。

DCE-Encina Lightweight Client (DE-Light)

DE-Light は、DCE クライアントとして稼働していないシステムに対して DCE および Encina の機能を拡張します。Java および C プログラム言語で作成されたクライアントがサポートされ、Encina アプリケーションにアクセスするためのゲートウェイ・サーバーを提供します。

エンタープライズ版 Application Server は、WebSphere アプリケーション、Component Broker アプリケーション、および CICS アプリケーション上で Encina アプリケーションと Enterprise Beans を使用するためのアプリケーション開発ツールを提供します。Encina は、C、Java、および C++ プログラム言語で、パフォーマンスの高いトランザクション処理アプリケーションの開発を目的とするお客様をサポートします。

実行時およびシステム管理アーキテクチャー

エンタープライズ版 Application Server 実行時環境は、アドバンスド版 Application Server 実行時環境の機能を拡張します。この実行時環境は、重要なビジネス・アプリケーションの実行時および開発環境を柔軟に運用するよう設計されています。CORBA および EJB 仕様などの業界標準がサポートおよびインプリメントされ、セキュアなトランザクション環境下でのインターオペラビリティーを実現しています。

図19 に、このインターオペラビリティーが機能する様子を示します。

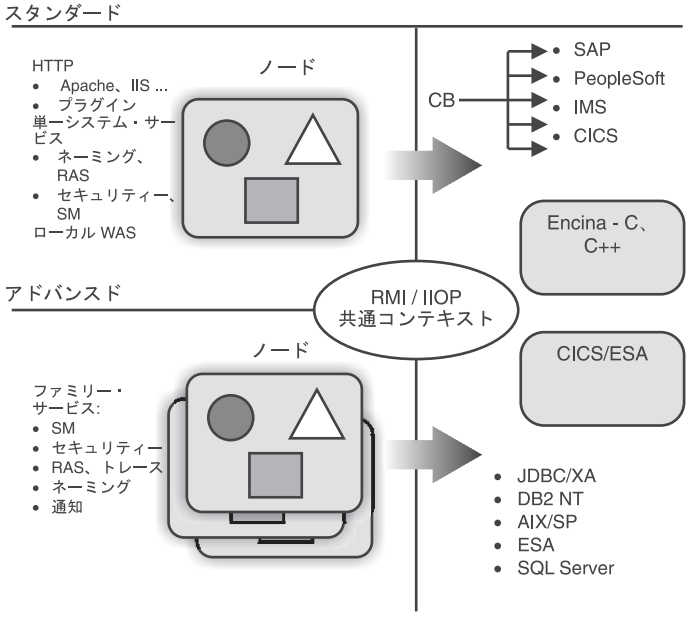

図 19. エンタープライズ版 Application Server のインターオペラビリティー

- **共通プロトコル** − Remote Method Invocation (RMI) プロトコルおよび Internet Inter-ORB Protocol (IIOP) によって、WebSphere ファミリーのインターオペラビリティーのためのセキュリティー、トランザクション、およびネーミングのための共通のコンテキストが提供されています。
- **リソース・マネージャー・アクセス** − リソース・マネージャーへのアクセスは、Component Broker の共通コネクター・フレームワーク (CCF) およびアプリケーション・アダプター・フレームワークにより実現されます。
- **サーバー** − エンタープライズ版 Application Server は、Component Broker サーバー・グループ、Component Broker/390 システム、または TXSeries

CICS、Encina あるいは Encina++ システムのインターネット・フロントエンド・サポートを提供する 1 つ以上のアドバンスド版 Application Server から構成されています。アドバンスド版 Application Server は、Web アプリケーションを直接サポートするサーブレット、JavaServer Pages (JSP)、および Enterprise Beans のランタイムを提供します。EJB サーバーは、共用エンティティーおよびビジネス・プロセスをインプリメントし、セッション Bean を代替ランタイムのインプリメンテーションにマップします。Component Broker も、Web アプリケーションを直接にサポートする Enterprise Beans のホストにすることができます。

- **システム管理** − エンタープライズ版 Application Server システム管理 (SM) ソリューションは、アドバンスド版 Application Server システム管理の機能に基づいて作成されています。これにより、Tivoli® Suites システム管理との統合が可能となっています。

  エンタープライズ版 Application Server システム管理モデルは、ビジネス・アプリケーションを構成するすべてのコンポーネントを管理・統合します。これは、個々のコンポーネントを管理できても、アプリケーション全体は管理しないアドバンスド版 Application Server およびスタンダード版 Application Server とは異なります。

エンタープライズ版アプリケーション開発環境およびツール

エンタープライズ版 Application Server には、お客さま指向であり提供者を意識したビジネスのあらゆる局面のアプリケーションを作成するためのツール・セットが含まれています。強力な Web サイトの作成、Web ビジネスに対応していないコンピューター・リソースを結合するための分散トランザクション・アプリケーションの作成、Web システムおよび非 Web システムの統合、またはこれらすべてを望む場合にも、エンタープライズ版 Application Server はその手助けとなります。

エンタープライズ版 Application Server には、アドバンスド版 Application Server およびスタンダード版 Application Server で提供されるすべてのアプリケーション開発ツールが含まれています。また、以下のアプリケーション開発ツールも提供されています。

WebSphere Studio

WebSphere Studio は、エンタープライズ版 Application Server に組み込まれています。WebSphere Studio の詳細については、9ページの『スタンダード版アプリケーション開発環境』を参照してください。

IBM VisualAge for Java エンタープライズ版

VisualAge for Java は、Enterprise Beans の設計、インプリメントおよびテストのための IBM のプラットフォームです。以下がサポートされます。

- セッションおよびエンティティー Enterprise Beans
- Rational Rose 間とのモデル・インポートおよびエクスポート
- 単体テストおよびデバッグ
- Enterprise Bean の継承性および関連
- リレーショナル・データベース・マッピング
- 既存のデータベースからの Enterprise Beans のボトムアップ・プロダクション
- 値による独立オブジェクトのサポート
- Bean 管理パーシスタンスのプログラミング・モデル、およびデータベース呼び出しを行うセッション Bean
- オブジェクト・アダプター Bean の自動生成
- JSP ページの自動生成
- Enterprise Bean Java アーカイブ (JAR) ファイルのエクスポート
- スタンダード版 Application Server の配置済み JAR ファイルのエクスポート
- Component Broker での Enterprise Beans 配置のための制御情報とマップ・ステートメント

VisualAge for Java エンタープライズ版は、エンタープライズ版 Application Server に組み込まれています。

IBM Enterprise Access Builder (EAB)

EAB は、EJB アプリケーションと非 EJB アプリケーションの統合をサポートします。 EAB は、既存のインターフェース定義をインポートし、単一呼び出しを処理して外部アプリケーションとの対話を戻す JavaBeans コンポーネントであるコマンド・アダプター Bean を作成します。既存のアプリケーションを使用する場合、単一呼び出しと応答対話以外にも要求されることがあります。EAB プログラミング・モデルは、複数のコマンド・アダプター Bean を特殊な可搬性のない Enterprise Bean (EJB アプリケーションと他のタイプのアプリケーションとの接続に使用) に結合することもできます。

IBM TeamConnection®

TeamConnection は、分散環境におけるチーム開発をサポートします。各種レベルでのソース・コード・アクセス、分散コンパイル、およびバージョン管理が可能です。

VisualAge コンポーネント開発ツールキット

VisualAge コンポーネント開発ツールキットは、Object Builder、分散トレース機能およびデバッガーから構成されます。これらのツールには、以下の機能があります。

- Component Broker への Enterprise Beans の配置
- キャッシュおよびオブジェクト指向 SQL (OOSQL)
- EAB パーツと Application Adaptor との統合
- 分散トレースおよびデバッグ
- Rational Rose の統合
- Component Broker/390 のサポート
- C++ による EJB インターフェースのインプリメント

VisualAge for C++ プロフェッショナル版

VisualAge for C++ は、完全な C++ 開発環境です。この開発環境は、高性能な高速計算アプリケーションには特に有用です。そのオープン・クラス・ライブラリーは、AIX および Windows NT システム上で、堅固なアプリケーションを作成するための、高機能のクラス・ライブラリーとフレームワークを提供します。

IBM DB2®

IBM DB2 は、リソース・マネージャーとして、TXSeries および Component Broker とともに使用することができる分散リレーショナル・データベースです。アドバンスド版 Application Server に含まれる EJB 管理サーバーは、DB2 を使用することができます。また、Component Broker に含まれる EJB 管理サーバーは、DB2 を使用する必要があります。また、アドバンスド版およびエンタープライズ版 の両方の Application Server において、コンテナー管理のパーシスタンス (CMP) のエンティティー Bean に関連付けられた永続データを保管するために、DB2 を使用することもできます。

MQSeries

MQSeries® は、IBM の主要なメッセージングおよびキューイング・サービスです。MQSeries は、企業のビジネス・プロセスを統合するためのオープンなアー

キテクチャーを提供します。MQSeries のアプリケーションは、データをメッセージとして送受信することにより、異なるプラットフォーム間での情報交換を行います。基盤となる MQSeries ソフトウェアは、ネットワーク・インターフェースの問題を解決し、確実にメッセージを送達するとともに、通信プロトコルの処理を行います。そのため、プログラマーは根底にあるネットワークの複雑さに煩わされることなく、主要なビジネス要件の処理に自らのスキルを使用することができます。

MQSeries は、正式には、WebSphere プラットフォームの一部ではありませんが、WebSphere Application Server とともに使用するために、特にライセンス交付を受けた MQSeries のバージョンは、エンタープライズ版 Application Server に組み込まれています。MQSeries を使用することにより、多階層の WebSphere アプリケーションのスケーラビリティー、パフォーマンス、および移植性を向上させることができます。MQSeries を、WebSphere Application Server 上で実行する Java アプリケーションと統合することにより、多様なプラットフォーム上のバックエンド・リソースおよび既存のシステムへのアクセスが可能になります。

第8章 Component Broker の使用

IBM Component Broker は、分散オブジェクトを使用するアプリケーションを開発、配置、および管理するための統合環境です。 Component Broker では、共通オブジェクト・リクエスト・ブローカー・アーキテクチャー (CORBA) をベースにしたアプリケーションを作成するための、共通の高度なプログラミング・モデルとフレームワークが提供されます。

本節では、以下のトピックについて説明します。

- 『Component Broker のインプリメンテーション』
- 94ページの『Component Broker の実行時環境』
- 112ページの『Component Broker のアプリケーション開発環境』
- 117ページの『Component Broker のシステム管理』

本節では、155ページの『第2部 WebSphere Application Server の使用』で説明する、WebSphere ファミリーのアプリケーション例を理解するために必要なバックグラウンド情報について説明します。本節は、Component Broker でエンタープライズ・アプリケーションを開発する場合の実用上の問題については詳述しません。

Component Broker の詳細については、製品の資料を参照してください。

Component Broker のインプリメンテーション

Component Broker のインプリメンテーションは、CORBA に基づいており、Enterprise JavaBeans (EJB) の仕様に記述されているプログラミング・モデルもサポートしています。Component Broker のインプリメンテーションは、次の 3 つの部分で構成されます。

- 分散オブジェクト・アプリケーションをサポートする実行時環境。
- ソフトウェア開発者がビジネス・オブジェクトを作成し、複数のアプリケーションにおいてそれらを結合できるようにする開発環境。Component Broker のアプリケーション開発環境は、Component Broker のアプリケーション・サーバーで実行されるビジネス・オブジェクトの作成のために最適化されています。
- 大規模な分散コンピューティング環境を管理するためのシステム管理ツール。

図20 は、Component Broker の各部分がどのように一体化して機能するかを示したものです。

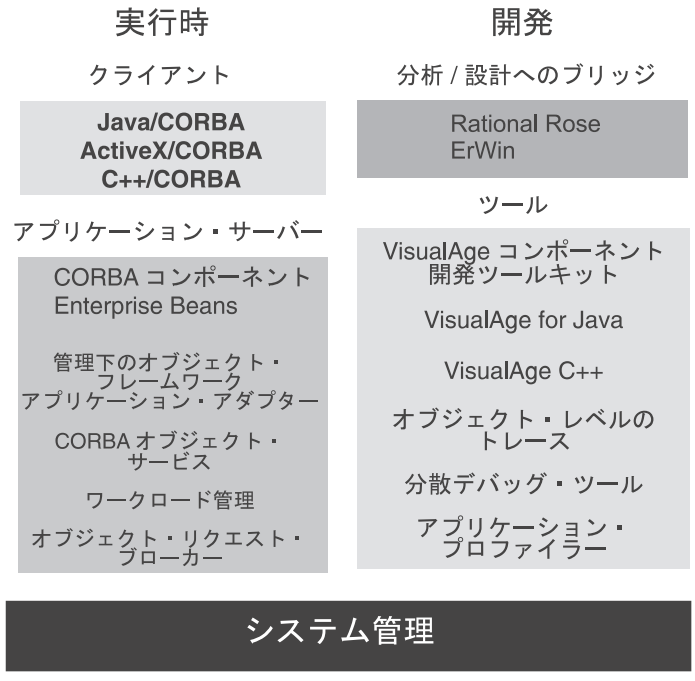

図 20. Component Broker の構成

Component Broker のインプリメンテーションでは、下位レベルのプラットフォーム固有のインターフェースの詳細は隠ぺいされるため、開発者はアプリケーションの設計、コーディング、およびテストに集中することができます。プラットフォームのエキスパートが、これらのアプリケーションをサポートするために、セキュリティー、保全性、およびワークロード管理をサポートするのに最適な実行時環境を選択して適用します。

Component Broker の実行時環境

Component Broker の実行時環境では、多階層分散オブジェクト・アプリケーションに必要なサービスが提供されます。 Component Broker の実行時環境は、ActiveX クライアントと、Java および C++ プログラム言語で作成されたクライアントをサポートしています。さらに、ビジネス・オブジェクトを管理するための堅固なアプリケーション・サーバーを提供し、パーシスタンスをサポートするための、データベースとトランザクション・マネージャーとのインターフェース・セットを組み込んでいます。

実行時アーキテクチャー

Component Broker の実行時アーキテクチャーのキー属性は、柔軟性と選択範囲の広さです。アプリケーションにおいては、基本となるビジネス・オブジェクトのロジックに影響を与えることなく、さまざまなタイプのクライアントと補助データ・ストアを使用することができます。

図21 は、Component Broker の実行時アーキテクチャーの上位からの構造を示したものです。

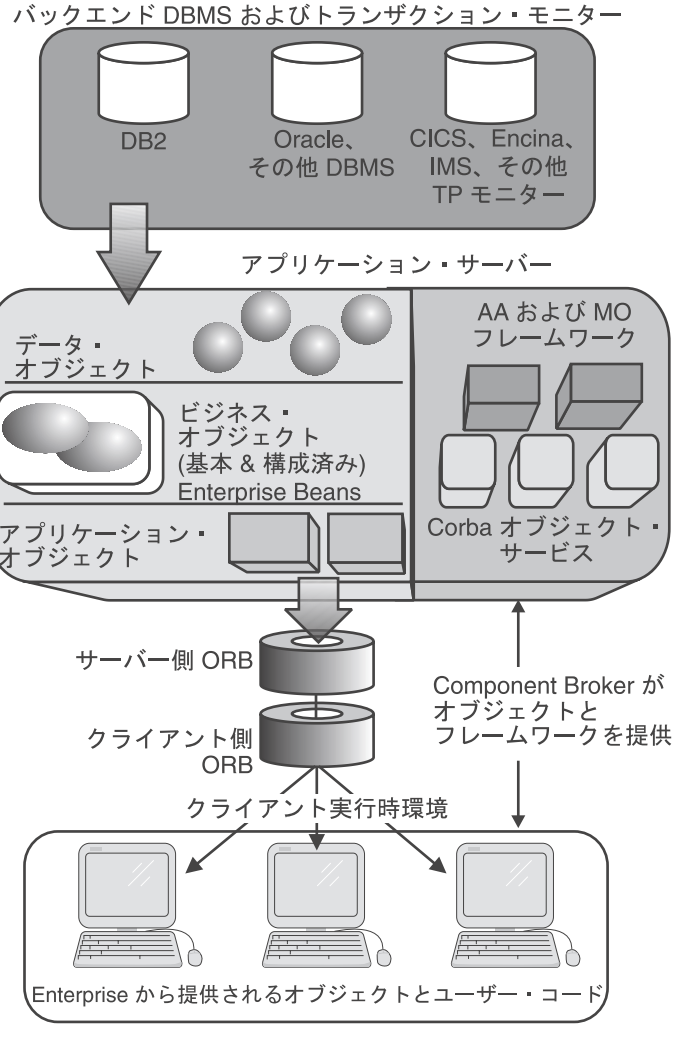

図 21. Component Broker の実行時アーキテクチャーの上位からの構造

Component Broker の実行時アーキテクチャーは、以下のもので構成されています。

- **クライアントの実行時環境** − クライアントの実行時環境により、クライアントは、アプリケーション・サーバーに接続することができます。
- **オブジェクト・リクエスト・ブローカー** − CORBA 準拠のオブジェクト・リクエスト・ブローカー (ORB) を使用することにより、クライアントは、アプリケーション・サーバーと通信することができます。 ORB の一部は通信チャネルの各端点に常駐します。
- **アプリケーション・サーバー** − Component Broker の実行時環境の最も重要な部分は、オブジェクト指向アプリケーション・サーバーです。アプリケーション・サーバーは、Component Broker のコンポーネントをビジネス・オブジェクトと統合します。アプリケーション・サーバーには、CORBA コンポーネントや Enterprise Beans などのビジネス・オブジェクトが含まれます。またアプリケーション・サーバーは、ビジネス・オブジェクトをサポートするサービスもインプリメントしています。 Component Broker は、以下の 3 つのタイプのオブジェクトをサポートしています。
  - アプリケーション・オブジェクト は、クライアントに上位アプリケーション・インターフェースを提供します。アプリケーション・オブジェクトはクライアントごとに固有で、通常、セッション期間中のみ存在します。アプリケーション・オブジェクトは、ビジネス・プロセスをインプリメントする場合にしばしば使用されます。
  - ビジネス・オブジェクト (または、エンティティー・オブジェクト) は、企業情報システムの個々のコンポーネント (銀行システムのアカウントなど) を表しています。ビジネス・オブジェクトは、基本キーと呼ばれる固有の ID を持ち、常に永続的です。
  - データ・オブジェクト は、ビジネス・オブジェクトの状態を指定します。ビジネス・オブジェクトの状態は、クライアント・セッション間で保持される必要があります。またデータ・オブジェクトは、バックエンドのリソース・マネージャーとのインターフェースも構成します。
- **管理下のオブジェクト・フレームワーク** − Component Broker オブジェクトは、サーバー環境内でのオブジェクト管理を受け持つ管理下のオブジェクト・フレームワーク (MOFW) と、特定の補助データ・ストアをサポートするアプリケーション・アダプター (AA) をインプリメントします。

これら上位 Component Broker サービスの背後には、サーバー・フレームワークが使用する CORBA オブジェクト・サービスがあります。サーバー・フレームワークは、システム・サービスと CORBA サービスとの下位インターフェー

スをアプリケーション開発者から隠ぺいするため、プラットフォームを超えた移植可能なアプリケーション・オブジェクトの作成に役立ちます。

バックエンドのデータベース管理システム (DBMS) とトランザクション・マネージャーは、実際には、Component Broker のインプリメンテーションの一部ではありません。しかし、これらが組み込まれているのは、Component Broker が、バックエンドおよび既存のリソースとの優れた設計のインターフェースを提供し、それらのリソースの再使用が操作上可能であるためです。

クライアントのサポート

Component Broker の設計目的は、広範囲のクライアントをサポートすることです。クライアントは、Windows NT などのオペレーティング・システムや Web ブラウザーで実行することができます。ActiveX をベースにしたアプリケーション、CORBA に準拠した C++ アプリケーション、Java アプレット、および Java アプリケーションはすべて、クライアントの実行時環境でサポートされます。

図22 は、クライアントのプログラミング・モデルを示したものです。

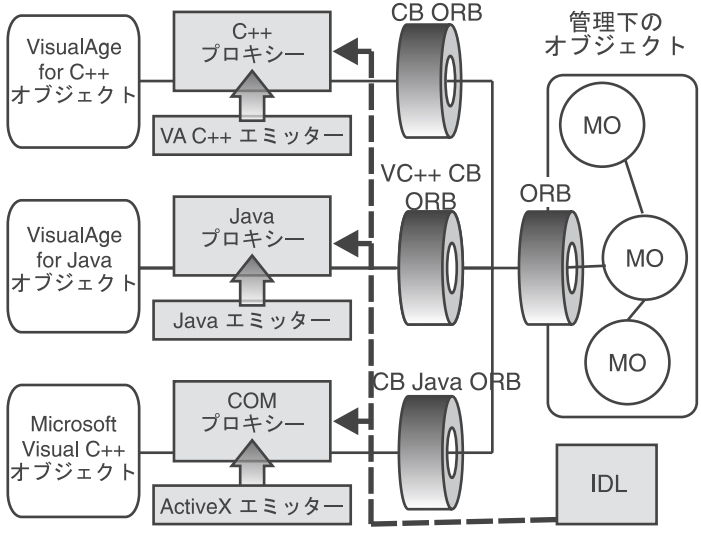

図 22. Component Broker のクライアント・プログラミング・モデル

クライアント・アプリケーションの開発には、IBM VisualAge for Java、IBM VisualAge for C++、Microsoft VisualBasic®、および Microsoft Visual C++® などのツールを使用します。クライアント・アプリケーションは、プロキシー・

オブジェクトを使用してサーバーと通信します。各サーバー・オブジェクトには、それぞれ対応するクライアント・プロキシー・オブジェクトがあり、このプロキシー・オブジェクトがクライアントのユーザー・インターフェースおよびアプリケーション・コードと対話します。プロキシー・オブジェクトは、ビジネス・ロジックのインプリメントはせず、対応するサーバー・オブジェクトへの実行要求の転送を行います。

クライアントのプロキシー・オブジェクトは、言語固有のエミッターにより作成されます。C++、Java、および ActiveX のエミッターは、言語に依存しないインターフェース定義言語 (IDL) ファイルから言語固有のバインディングを生成します。 ActiveX クライアントの場合は、Component Broker オブジェクト用のラッパーとして機能する特殊な COM オブジェクトが作成されます。COM オブジェクトは、COM アプリケーション・プログラミング・インターフェース (API) 呼び出しを、CORBA プロキシー・オブジェクト呼び出しに変換します。このプロキシー・オブジェクトは、開発プロセス中に自動的に作成されます。クライアント / サーバー通信の処理に必要なコードは、Web サーバーからダウンロードすることができます。あるいは、Component Broker のシステム管理ツールを使用してインストールすることができます。クライアント・バインディングは、VisualAge Component Development Toolkit を使用して作成されます。

プロキシー・オブジェクトは、ORB を介してクライアント要求を送信します。ORB は、Internet Inter-ORB Protocol (IIOP) を介してサーバー上の Component Broker ORB と通信し、Component Broker ORB はその要求を管理下のオブジェクトに送信します。ORB による、クライアントのプロキシー・オブジェクトの処理方法の詳細については、99ページの『Component Broker ORB』を参照してください。

管理下のオブジェクトは、各種クライアントに対するサービス提供機能を持つ、再利用可能なコンポーネントです。Component Broker のアプリケーション・サーバー上の管理下のオブジェクトは、要求が実際にどのように送達されても影響を受けません。管理オブジェクトは、サービスの提供先のクライアントのインプリメンテーション言語、プラットフォーム、あるいはオペレーティング・システムを認識する必要はありません。

Component Broker のプログラミング・モデルの詳細については、*Component Broker*「プログラミング・ガイド」を参照してください。

Component Broker ORB

Component Broker は、クライアントとサーバーによる要求と応答の交換を可能にする (ORB) を提供しています。ORB は、多階層アプリケーションにおいて分散オブジェクトを使用するための標準的な方法を提供しています。ORB を使用することにより、開発者はロケーションを意識する必要のないプログラミング・モデルをインプリメントすることができます。アプリケーション・ソフトウェアにロケーション情報を組み込んだり、さまざまな低水準の転送プロトコルを扱う必要はありません。

インターフェース定義言語

Component Broker は、CORBA Interface Definition Language (IDL) をサポートしています。 IDL を使用すると、クライアントとサーバーは、それらの通信の基本となる、プラットフォームにも言語にも依存しない標準を持つことができます。

IDL はオブジェクトの外部インターフェースを公開し、システムのアーキテクチャーを構成するオブジェクトを記述するために使用されます。IDL コンパイラーは、特定のプログラム言語とのインターフェースを使用するために必要となるコードを生成します。IDL のインターフェースは型に相当するもので、グループ化してモジュールにすることができます。また IDL は、複数のインターフェース継承もサポートしています。

Component Broker のオブジェクト・ビルダー・ツールを使用すると、IDL ファイルを自動的に生成することができます。サーバー側のオブジェクトをインプリメントする場合は、アプリケーションのビジネス・ロジックを IDL で生成されたコードに追加する必要があります。クライアント側の開発者は、Component Broker のツールを使用して、開発者のネイティブのプログラム言語環境で提供されているオペレーションを呼び出すだけで済みます。

プロキシー・オブジェクト

サーバー上のオブジェクトは、クライアントからはあたかもクライアントのプログラムに常駐しているかのように見えます。これは、プロキシー・オブジェクトを使用することにより実現されます。プロキシー・オブジェクトは、クライアントと ORB の両方と通信します。プロキシー・オブジェクトは、それが対応するサーバー・オブジェクトと同じインターフェースを持っていますが、そのサーバー・オブジェクトのメソッドを直接インプリメントしているわけではありません。その代わりにプロキシー・オブジェクトは、ORB を介してサーバーに送られる形式にメソッド呼び出しを変換します。これによりサーバーは、ターゲット・オブジェクトを見つけ出し、実際のメソッドのインプリメンテーションを実行します。

オブジェクト・アダプター

オブジェクト・アダプター (OA) は、ORB が受信した各要求を解析し、それを要求の宛先であるサーバー・オブジェクトのインプリメンテーションにディスパッチします。オブジェクト・アダプターは、ORB の通信フレームワークと、サーバーに常駐するオブジェクトとの間の調整役として機能します。

サーバー側の ORB は、クライアント要求を受信すると、その要求内の情報を含む要求オブジェクトを作成します。次にその要求オブジェクトが要求キューに挿入されます。 OA は処理を行うためにそのキューから各要求オブジェクトを除去します。

OA は、最初にターゲット・オブジェクトを検索します。Component Broker においては、コンテナー・オブジェクトとホーム・オブジェクトを使用すると便利です。次に、OA はクライアントのメソッド要求をアンマーシャルし、サーバー・オブジェクトに適切なメソッド呼び出しをディスパッチします。そのメソッドのインプリメンテーションからの結果がクライアントに戻されます。

相互利用可能なオブジェクト・リファレンス

相互利用可能なオブジェクト・リファレンス (IOR) は、オブジェクトによるネットワーク境界を超えた通信を可能にするための、分散オブジェクトのポインターです。IOR は、以下のようにフォーマット設定されています。

- **タイプ ID** − IOR と IOR が表すオブジェクトのタイプを関連付けます。
- **タグ付きプロファイル** − サーバーと接続してターゲット・オブジェクトを検索するためのプロトコルです。

IOR の表記は、どこで使用するかによって異なります。たとえば、IOR が ORB 間で送信される場合は、ワイヤー・レベルのメッセージ形式で表され、プロキシー・オブジェクトに保管されている場合は、オブジェクト形式で表されます。

クライアントは、IOR をストリングに変換し、それをファイルに保存して、終了することができます。クライアントが再度活動化されると、その IOR をファイルから読み出し、オブジェクト・リファレンスに再変換することができます。

管理下のオブジェクト・フレームワーク

Component Broker のサーバー・オブジェクトは、MOFW から取得されます。*管理下のオブジェクト* は、ビジネス・オブジェクトのランタイム・ラッパーです。管理オブジェクトは、ビジネス・オブジェクトによるメソッドの実行方法を制御し、サーバーの動作を操作します。

管理下のオブジェクトにより、ビジネス・アブストラクションのサーバー側のインプリメンテーションが扱いやすくなります。このアブストラクションは、インターフェース定義言語 (IDL) を使用して表されます。 IDL により、クライアント用のプロキシー・オブジェクトと、サーバー用に完全にインプリメントされた管理下のオブジェクトが作成されます。プロキシー・オブジェクトと管理下のオブジェクトはどちらも、MOFW および CORBA のインターフェースを継承します。これらのインターフェースを継承するオブジェクトには、リモートでアクセスすることができます。またこのようなオブジェクトは、Component Broker のサービスに加えることができます。

管理下のオブジェクトは、セキュリティー、トランザクションの保全性、活動化および分離の各ポリシーを含むオブジェクト・サービスを提供するコンテナー として構成されます。Component Broker サーバーのコンテナーの詳細については、『コンテナー』を参照してください。

コンテナーは、上記の 3 つのインスタンスすべてが適切に初期設定され、相互に関連付けられるようにします。これらのインスタンスが一体となって管理下のオブジェクトになります。この管理下のオブジェクト (あるいは管理下のオブジェクト・アセンブリー) は、クライアント側のプロキシー・オブジェクトからの要求を処理するサーバー側のオブジェクトです。

オブジェクト・ビルダー (これは VisualAge Component Development Toolkit の一部です) は、管理下のオブジェクトに作成されるほとんどすべてのコードを生成します。さらに、オブジェクト・ビルダーは、インプリメントの必要があるメソッド、特に構築中のビジネス・ソリューションに関係するメソッドを明確に識別します。

サーバーのランタイム・コンポーネント

Component Broker サーバー上のコンテナー、管理下のオブジェクト、およびその他のランタイム・コンポーネント間の対話は、コンポーネントごとに割り当てられた役割と責任によって決まります。本節では、主要なランタイム・エレメントの機能について説明します。

コンテナー: コンテナー は、管理下のオブジェクトを保管するための場所を提供し、ビジネス・オブジェクトとアプリケーション・アダプターのインプリメンテーション間の基本的な対話を記述します。コンテナーを使用すると、管理下のオブジェクト・フレームワークから継承したインターフェースを介して、基本となる振る舞いをビジネス・オブジェクトに追加することができます。

コンテナーはメモリー内に常駐するオブジェクトを追跡し、オブジェクトのサービスを管理します。コンテナーは、オブジェクトの活動化および非活性化を行い、バックエンドのデータ・ストアへのリソース・マネージャーの接続機構を介してトランザクション対話を駆動し、必要なセキュリティー・レベルを確立します。

コンテナーのサービスは、クライアントおよびビジネス・オブジェクトの開発者のプログラミング・モデルには含まれていないため、開発者はコンテナーを意識する必要はありません。Component Broker のプログラミング・モデルのこのような側面により、ドメインのエキスパートはビジネス・ソリューションに集中することができ、システム・エキスパートはサービスの品質に集中することができる、という責任の分離が可能になります。

システム機能をオブジェクトに追加したり、統合化し、調整した 1 つの方法でオブジェクト・サービスをパッケージ化するだけでなく、コンテナーは管理機能に境界も設けます。これらの機能には、トランザクションの振る舞い、状態管理、およびオブジェクト分離のためのポリシーが含まれます。

任意のポリシー・セットを管理するようにコンテナーを構成することができます。この構成により、管理者は、特定のオブジェクト・セットで決定されるポリシーを制御することができます。たとえば、1 つのオブジェクトをすべてのクライアントで共用するため、そのオブジェクトは、1 つのサーバー内に一度しか存在しない、という要件をコンテナー・ポリシーはインプリメントすることができます。異なるセキュリティー・ポリシーを特定のクラスごとに確立するために、1 つのリソース・マネージャーを複数のコンテナーに構成することができます。

mixin オブジェクト: *mixin* オブジェクト は、実行時にコンテナーによって提供され、コンテナーに関連付けられているポリシーをインプリメントします。 mixin オブジェクトは、オブジェクト・サービスをビジネス・オブジェクトに統合します。mixin オブジェクトは管理下のオブジェクトによって動的に呼び出され、コンテナーのポリシーと管理下のオブジェクトの振る舞いを混合します。

ホーム: ホーム は、オブジェクトのコレクションです。ホームは、Component Broker プログラミング・モデルの一部でもあり、EJB 仕様の一部でもあります。ホームは、これら言語に依存しない、Java 指向テクノロジーの統合のキーとなるものです。

ホームは、管理下のオブジェクトを作り出すために使用されます。ホームは、コンテナーと共に機能します。コンテナーは、ビジネス・オブジェクト・クラ

スを継承する特定の管理下のオブジェクト・サブクラスを認識します。このコンテナーは、管理下のオブジェクトに関連付けられているデータ・オブジェクトのインプリメンテーション・クラスも認識します。ホームは、管理下のオブジェクト・サブクラス、データ・オブジェクト・インプリメンテーション・クラス、および適切な mixin オブジェクトの各インスタンスを作成します。

クライアントは、ホームのファクトリー・インターフェースを使用して、ホームにオブジェクトを作成することができます。完全に初期化されたコピー・ヘルパー・オブジェクト、あるいは、作成されるオブジェクト用の基本キー・ストリングを提供することができます。

多くの管理下のオブジェクトを同じホームに作成することができます。ただし、これらの管理下のオブジェクトはすべて同じクラスのインスタンスになります。したがって、ホームとは、特定のタイプの製品オブジェクト専用に設計された工場のようなものです。たとえば、アカウント・クラスのオブジェクトに必要なホームと、顧客クラスのオブジェクトに必要なホームは異なります。ホームは、自らが作成した管理下のオブジェクトであればすべて探し出すことができます。

ホームはコンテナーに構成されます。ホームは特定のクラスのオブジェクトのコレクションを表すため、コンテナーとのこの関連付けにより、希望するオブジェクト・サービスをクラスに提供することができます。

104ページの図23 は、ホームとコンテナーの関係を示したものです。

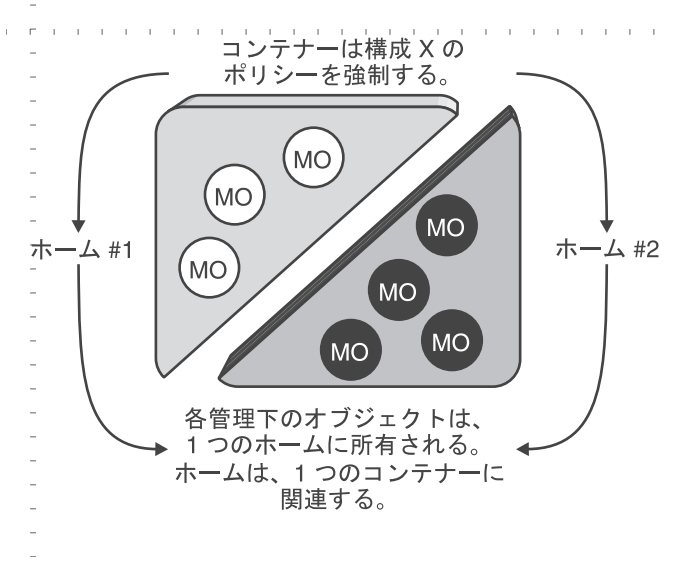

図 23. アプリケーション・アダプターのコンテナーとホーム

1 つのコンテナーには多くのホームがあり、各ホームは、特定のタイプのオブジェクトに対応します。ホームは、新しいタイプのオブジェクトをサポートするために、コンテナーに追加することができます。

同じコンテナーに関連付けられているホームは、DB2 や CICS など、同じバックエンドのデータ・ストレージあるいは手続き型環境を使用します。これらのホームの管理下のオブジェクトは、コンテナー・レベルでインプリメントされている同様のポリシー・セットにも従います。

ホームはまた、ホーム内のオブジェクトの集合であるコレクションをサポートします。ホームは、コレクションの内の個々のオブジェクトにアクセスするためのイテレーター・オブジェクトを提供します。イテレーターを使用すると、ホーム内の他のオブジェクトにもアクセスすることができます。

アプリケーション・アダプター

アプリケーション・アダプターは、コンテナーに構成されるリソース・マネージャー接続機構です。アプリケーション・アダプターを使用すると、コンテナー内のオブジェクトは、バックエンドの特定のリソース・マネージャーにアクセスすることができます。 1 つのアプリケーション・アダプターで、1 種類のデータ・ストアにアクセスすることができます。複数のアプリケーション・アダプターを構成することにより、リソース・マネージャーをコンテナーに追加することができます。

たとえば、ビジネス・オブジェクトは、状態の一部を DB2 データベースから、そして別の状態を既存の CICS のトランザクションから取得する必要がある場合があります。これらのリソース・マネージャーをサポートするために、コンテナーは適切なアプリケーション・アダプターを構成します。これにより、複数のデータ・オブジェクト (それぞれは特定のバックエンド・リソースに関連付けられている状態を表しています) に、1 つのビジネス・オブジェクトを関連付けることができます。アプリケーション・アダプターを使用すると、複数のバックエンドのデータ・ストアからオブジェクトを柔軟に構成することができ、より高レベルの操作上の再利用が実現します。

Component Broker は、次のアプリケーション・アダプターを提供しています。

- DB2 アプリケーション・アダプター
- Oracle アプリケーション・アダプター
- Informix アプリケーション・アダプター
- IMS™、SAP、および CICS へのアクセスを可能にする手続き型アプリケーション・アダプター
- MQSeries アプリケーション・アダプター

アプリケーション・アダプターは共通のフレームワークに構築されますが、リソース・マネージャーが異なれば、アプリケーション・アダプターのインプリメントの方法も Component Broker プラットフォームごとに異なります。ソフトウェア・ベンダーは、アプリケーション・アダプターのフレームワークを使用して、リソース・マネージャーにインプリメンテーションを提供することができます。

アプリケーション・アダプターの使用についての詳細は、以下の Component Broker の資料を参照してください。

- *手続き型アプリケーション・アダプター開発ガイド*
- *データベース・アプリケーション・アダプター開発ガイド*
- *MQSeries アプリケーション・アダプター開発ガイド*

合成ビジネス・オブジェクト

オブジェクト・テクノロジーの最も強力な機能の 1 つに、合成、すなわち既存のオブジェクトから新しいオブジェクトを作成する機能があります。 MOFW は、合成されたオブジェクトを、他のオブジェクトの場合と同じ方法で処理します。

106ページの図24 は、合成の簡単な例を示したものです。

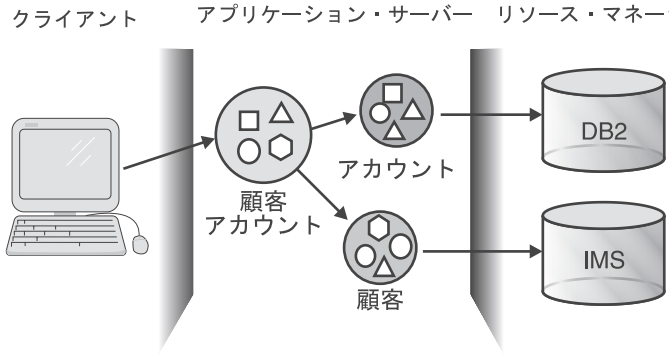

図 24. 合成 (既存のオブジェクトから新しいオブジェクトを形成)

2 つのビジネス・オブジェクトが組み合わされて 1 つの複合オブジェクトになります。(わかりやすくするためにこの図では、データ・オブジェクトを省略しています。) このタイプの合成は、基本オブジェクトが一緒に使用されることが多いというクライアントのアクセス・パターンで有効です。このようなタイプのクライアントの場合、情報にアクセスし、処理を実行するためのメカニズムとしては、1 つの統一されたインターフェースの方がさらに便利です。通常、そのようなクライアントは合成が使用されていることにさえ気付きません。そのようなクライアントは、合成オブジェクトを、分散システムの他のオブジェクトと同様なものとして見ています。各ビジネス・オブジェクトに個別にアクセスする必要のある他のクライアントは、その個別アクセスを継続して行うことができます。

アプリケーション・オブジェクト

アプリケーションでは、プロセスやタスクの管理、あるいはフローの制御を何度も行う必要があります。アプリケーション・オブジェクト により、アプリケーションの作業フローが指示され、論理的なビジネス・プロセスが実行されます。アプリケーション・オブジェクトは、一般に合成ビジネス・オブジェクトと基本ビジネス・オブジェクトと共に使用され、ビジネス・モデルをインプリメントします。 これらのオブジェクトは、一般に 1 つのクライアント・セッションに存在するため、セッション・オブジェクトとしても知られています。

相互利用可能オブジェクト

Component Broker サーバー環境は、Java および C++ プログラム言語で作成されたビジネス・オブジェクト・ロジックをサポートしています。ただし、このロジックを取り巻くフレームワークとインフラストラクチャーは、C++ プログラム言語で作成されています。Component Broker は、言語間オブジェクト・モデル (IOM) として知られる機能を提供しています。この機能を使用すること

により、C++ および Java プログラム間で、相互に直接対話することができます。IOM は MOFW に統合されています。

Enterprise Beans と管理下のオブジェクト

Component Broker は、Enterprise Beans のための実行時環境を提供します。Enterprise Beans は、MOFW に適合します。ビジネス・オブジェクトのクラス内でのロジックのプログラミングは、Enterprise Bean が代行します。VisualAge Component Development Toolkit を使用すると、Enterprise Beans を設計し、それを配置することができます。

コンテナー管理のパーシスタンスを持つエンティティー Bean は、CORBA コンポーネントの場合と同じ方法でデータ・オブジェクトを使用します。Enterprise Bean はそれ自体のパーシスタンスを処理するため、Bean 管理のパーシスタンスを持つエンティティー Bean は、データ・オブジェクトを使用しません。セッション Bean もデータ・オブジェクトを使用しません。

オブジェクト・サービス

Component Broker には、分散オブジェクトを扱うために必要な共通機能のインプリメンテーションを提供する豊富なオブジェクト・サービスのセットが付属しています。CORBA のサービスは、サポートされているすべてのプラットフォームに対して Component Broker でインプリメントされています。詳細については、*Component Broker*「*上級プログラミング・ガイド*」を参照してください。

ネーミング・サービス

Component Broker のネーミング・サービスを使用すると、CORBA ベースのアプリケーションでオブジェクトの登録と検出を行うことができます。オブジェクトの登録と検出は、システム・ネーム・ツリー として知られる特殊なディレクトリー内のエントリーにマップされる、人間が読むことができる名前を使用することによって実現します。ネーミング・サービスは、CORBA 仕様に完全準拠しています。

Component Broker の各ネットワーク (すなわち、同じシステム・マネージャーによって管理されるホスト) には、独自のネーム・ツリーがあります。このネーム・ツリーは、管理対象の各サーバー・ホスト上で稼働するネーム・サーバーによってインプリメントされます。このネーム・サーバーは、ホスト用のネーミング・サービスを提供しますが、セルおよび作業グループのネーミング・サービスを提供する場合もあります。ネーム・サーバーは、Component Broker によって、多数のシステム・オブジェクトを収容するために使用されます。これらのオブジェクトには、システムのネーム・スペース内で使用されるネーミ

ング・コンテキスト、ファクトリー・ファインダー、ロケーション・オブジェクト、およびイベント・チャネルなどがあります。

ネーミング・サービスは、すべてのオブジェクトへの参照を保管しているわけではありません。その代わりに、ホームなどの上位のオブジェクトへの参照を保管しています。ホーム内の特定のオブジェクトを検出するには、Component Broker インフラストラクチャーの他の部分が提供するメカニズムを使用します。

ネーミング・サービスのインプリメンテーションは、プラットフォームによって異なります。すべてのネーミング・サービス要求は CORBA インターフェースを介して行われるので、インプリメンテーションが異なっても、Component Broker アプリケーションに対しては透過的です。

- OS/390 プラットフォームでは、CORBA ネーミング・サービスは、Lightweight Directory Access Protocol (LDAP) にマップされます。基本となるデータ・ストアは DB2 です。
- ワークステーション・プラットフォームでは、分散コンピューティング環境 (DCE) のセル・ディレクトリー・サービス (CDS) が使用されます。 CDS サーバーは、クリアリングハウス と呼ばれるデータベースを管理します。クリアリングハウスには、DCE セル内のネットワーク・リソースの名前と属性 (ロケーションを含む) が含まれています。マスター・クリアリングハウスを、ネットワーク内のサーバーに複製することができます。その場合、すべてのレプリカは読み取り専用となります。

セキュリティー・サービス

Component Broker のセキュリティー・サービスは、情報とリソースへのアクセスを制御します。 Component Broker は、クライアントとサーバーの双方向認証とメッセージ伝送のための保護をサポートしています。また、ログ記録と他の関連するセキュリティー機能も提供しています。

セキュリティー・サービスのワークステーション・インプリメンテーションは、DCE のセキュリティー・サービスに基づきます。 Kerberos セキュリティーが必要な場合は、OS/390 で DCE を使用することができます。OS/390 では、ユーザー ID とパスワードもサポートされています。

Component Broker は、OS/390 などのシステムにおいて既存のセキュリティー・サービスをサポートしています。サーバーで提供されているセキュリティー・アーキテクチャーを統合して、既存のプラットフォームのセキュリティー・ソフトウェアにマップする必要があります。既存のセキュリティー・ポリシーと完全に統合するために、認証検査、ログ記録、および管理はすべて

OS/390 のシステム許可機能 (SAF) インターフェースを使用します。信頼性が高く、確立されたインフラストラクチャーへの投資の再利用は、ビジネス・ロジックのコードの再利用と同様に重要になり得ます。

ライフ・サイクル・サービス

Component Broker のライフ・サイクル・サービスは、分散環境におけるオブジェクトの作成、コピー、移動、および削除のための操作を提供します。

Component Broker は、サーバーのネットワーク内部におけるオブジェクトの配置をより細かく制御することにより、ライフ・サイクル・サービスのための OMG 標準を拡張しています。オブジェクトの配置を制御するには、オブジェクトごとに*ロケーション* を指定します。ロケーションは、地理的意味、構造的意味、あるいは時間的意味としても解釈できる、*接近性* の概念を具体化したものです。ロケーションは、システム管理機能を使用して定義、構成され、会社内の組織変更のような煩わしさからクライアント・コードを解放します。

外部化サービス

オブジェクトは、環境間、すなわち同一のホスト・マシン上のプロセス間、またはオブジェクト・リクエスト・ブローカー (ORB) を介したクライアントとサーバー間のいずれかで交換されることがしばしばあります。外部化サービスは、オブジェクトが非オブジェクト形式でその状態を保管し、復元することを可能にするメカニズムを提供しています。このサービスにより、オブジェクトの状態は、オブジェクトそれ自体の存在とは関係なく存在することができます。元のオブジェクトまたはそのオブジェクトが存在している ORB プロセスが継続して存在しているかどうかには関係なく、その状態をいつまでも保持することができます。

Component Broker のフレームワークは、外部化サービスを透過的に使用します。ほとんどの場合、アプリケーションはサービスを直接使用する必要はありません。

識別サービス

識別サービスを使用すると、分散ネットワークに常駐するオブジェクトの識別をすることができます。Component Broker は、オブジェクトをそのコンテナー、サーバー、ホスト、およびドメイン内に位置付ける相対情報から、そのオブジェクトの識別を取り出します。この情報は Component Broker の管理下のオブジェクト・フレームワーク内で使用され、各オブジェクトを一意に識別します。

照会サービス

照会サービスを使用すると、指定した条件のセットを満たすオブジェクトを検索することができます。照会サービスはオブジェクト指向の SQL と共に機能し、表示、結合、展開などを使用して各種コレクションを処理します。

オブジェクト・トランザクション・サービス

高性能で、信頼性の高いトランザクション・サポートは、大規模な商用アプリケーション・システムには不可欠です。 Component Broker の場合、オブジェクトはそのトランザクション機能を、管理下のオブジェクトおよびアプリケーション・アダプター・フレームワークを介して受け取ります。これらの機能を使用して、トランザクションを暗黙的に管理することができます。また、オブジェクト・トランザクション・サービスを使用することにより、トランザクションを明示的に管理することもできます。オブジェクト・トランザクション・サービスを使用すると、プログラマーは分散環境において標準のオブジェクト指向インターフェースを使用してトランザクションをインプリメントすることができます。 Component Broker では、オブジェクト・トランザクション・サービスの使用により、データが常に矛盾なく更新されることが保証されます。

トランザクション・モニターやデータベース・システムなどのバックエンドのリソース・マネージャーは、Component Broker のトランザクション環境に統合されます。 Component Broker の調整機能で、DB2 などのリソース・マネージャーを扱う場合には、2 フェーズ・コミット・プロトコルを使用します。OS/390 環境では、Component Broker は、リソース・リカバリー・サービス (RRS) を使って、必要なトランザクションの統合を提供します。 Component Broker は、X/Open の分散トランザクション処理仕様に完全準拠しています。

プログラミング・モデル

Component Broker のプログラミング・モデルを使用して、クライアント・プログラマーは管理下のオブジェクトの作成と検出および管理下のオブジェクトのメソッドの呼び出しを行うことができます。本節では、プログラミング・モデル、実行時環境、およびヘルパー・クラスについて説明します。

Component Broker のプログラミング・モデルの詳細については、*Component Broker*「プログラミング・ガイド」を参照してください。

プログラミング・モデルと実行時環境

111ページの図25 は、Component Broker のプログラミング・モデルが実行時環境の機能をどのように使用するかを示したものです。実行時環境は、以下のステップで実行されます。

1. ネーミング・サービスを使用して、管理下のオブジェクトが常駐するホームを検出する。通常、クライアントはホームと対話して特定のオブジェクトを作成し、配置する。
2. オブジェクトの相互利用可能なオブジェクト・リファレンス (IOR) をクライアントに戻し、クライアントはそれを使用して適切な管理下のオブジェクトを見つける。
3. クライアントに対してローカルなプロキシー・オブジェクトにその IOR を保管する。次にクライアントはそのプロキシー・オブジェクトを使用して、リモートの管理下のオブジェクトと、それがあたかもローカル・プロセスであるかのように通信する。

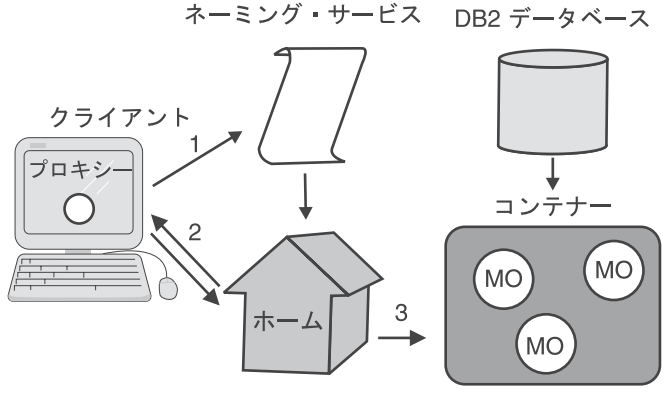

図 25. ランタイム・コンポーネント間のコラボレーション

注: クライアントは、ネーミング・サービスに対してオブジェクトのホームを要求する場合、まず、ネーミング・サービスへアクセスする必要があります。ネーミング・サービスへアクセスすることは、オブジェクト・リクエスト・ブローカー (ORB) のブートストラップ・プロシージャーの一部です。 ORB が初期設定されると、自動的にクライアントはネーミング・サービスにアクセスすることができます。

クライアントは、オブジェクトをリモート・システムのメモリーに格納する必要があるかどうかを知る必要はありません。クライアントにとって、オブジェクトは単にそこに存在するものです。このような単一レベルのストアという概念が、Component Broker のプログラミング・モデルには組み込まれています。

ヘルパー・クラス

Component Broker のプログラミング・モデルでは、リモート・オブジェクトの検出と作成に使用される、次の 2 つのヘルパー・クラス が指定されます。

- **キー・クラス** − オブジェクトを検索します。キー・クラスを使用すると、クライアントは、検索操作でホームが使用する 基本キー値の属性をローカルに設定することができます。キー・クラスは、コンパイル時検査も提供しています。
- **コピー・クラス** − オブジェクトを作成します。コピー・クラスを使用すると、クライアントは、リモート・オブジェクトが必要とする必須の属性値を設定し、コピーのコピーをホームに送ることができます。オブジェクトの初期設定は、同一の言語およびプロセスで実行されます。オブジェクトを作成すると、ORB を介して 1 つの通信のみを要求します。

これらのクラスのいずれも、オブジェクト・ビルダーの開発過程で作成されます。

Component Broker のアプリケーション開発環境

Component Broker のアプリケーション開発環境では、CORBA をベースとするアプリケーションを設計、コーディング、デバッグ、および配置するためのフレームワークが提供されています。他のベンダーによる、Component Broker のツール・スイートあるいはツールを使用して、アプリケーションを開発することができます。

この開発環境では、一般的な解析、設計、およびリエンジニアリングのためのツールとのブリッジも提供されます。開発環境は、IBM VisualAge プロダクト・ファミリーと統合されていますが、他のソフトウェア開発ツールと共に使用することができます。Component Broker のアプリケーション開発環境は、Component Broker のアプリケーション・サーバーに Enterprise Beans を配置することもサポートしています。

Component Broker のアプリケーション開発ツールの使用についての詳細は、*Component Broker*「アプリケーション開発ツール・ガイド」を参照してください。

VisualAge Component Development Toolkit

Component Broker の開発環境で最も重要なのは、VisualAge Component Development Toolkit です。このツールキットを使用することにより、開発者は、オブジェクト指向の分散アプリケーション作成することができます。このツールキットは、ビジネス・オブジェクトの構成とテストのプロセスを自動化するツール・セットを提供しています。

VisualAge Component Development Toolkit は、実行時環境でサポートされているすべてのプラットフォーム上でインプリメントされており、複数のプラットフォームでの開発に使用することができます。このツールキットには、以下のものが含まれています。

- オブジェクト・ビルダー・ツール。CORBA ベースのアプリケーションの開発に使用されます。このツールは、Component Broker のランタイム・プログラミング・モデルと管理下のオブジェクト・フレームワークをサポートしており、管理下のオブジェクト・フレームワークが必要とするサポート・コードを生成することもできます。
- IDL コンパイラー。リモート・オブジェクト通信のためのクライアント側およびサーバー側のコードを生成します。
- モデリング、分析、および設計のためのツールとのブリッジ
- IBM VisualAge および Microsoft Visual ソフトウェア開発製品など、その他の開発環境のサポート
- CICS 接続
- IBM TeamConnection

オブジェクト・ビルダー

オブジェクト・ビルダーは、VisualAge Component Development Toolkit における主要な開発ツールです。オブジェクト・ビルダーは、サーバー・オブジェクトの構築をサポートするために設計されたものです。開発者は、ビジネス・ロジックを、フレームワーク内の該当する場所に埋め込みます。オブジェクト・ビルダーにより、コード、makefile、およびアプリケーションのテストに必要なアプリケーション構成情報が生成されます。オブジェクト・ビルダーは、大規模な開発プロジェクトをサポートしています。またオブジェクト・ビルダーを使用して、Enterprise Beans を配置し、Enterprise Beans と CORBA コンポーネントの両方を使用するアプリケーションを開発することもできます。

オブジェクト・ビルダーは、サーバー・オブジェクトの振る舞いと属性を定義し、サーバー・オブジェクトが Component Broker の実行時環境で動作するために必要なコードを生成します。オブジェクト・ビルダーは、IDL と C++ を使用する CORBA プログラミング・モデルをサポートしています。テスト用のユニット、およびサーバーのセットアップ・スクリプトを備えたクライアント / サーバーのフル・パッケージを含む、完全な実動可能アプリケーションを生成することができます。

オブジェクト・ビルダーは、Rational Rose のオブジェクト・モデル、ERWin および Data Definition Language (DDL) モデル、CICS 接続プロシージャー・マップ、および IDL ファイルを入力として受け入れます。 オブジェクト・ビ

ルダーは、Java、C++、および ActiveX のクライアントに必要な IDL とバインディング・ファイルを生成します。オブジェクト・ビルダーをエンタープライズ・アクセス・ビルダー (EAB) と併用すると、オブジェクト・ビルダーによりサーバー側のコードを Java および C++ で生成することができます。オブジェクト・ビルダーは、たとえばビジネス・ロジックをインプリメントするためにコードをビジネス・オブジェクトに手動で追加するためのインターフェースを提供してます。オブジェクト・ビルダーは、XML (Extensible Markup Language) ファイル、リレーショナル・データベース DDL ファイル、およびシステム管理 DDL ファイルを出力します。

オブジェクト・ビルダーは、アプリケーション・オブジェクトと合成ビジネス・オブジェクト (すなわち他のビジネス・オブジェクトを組み合わせたビジネス・オブジェクト) あるいはバックエンド・システムへのデータ・オブジェクトのマッピングなどの共通のタスクをサポートします。オブジェクト・ビルダーは、ビジネス・オブジェクトのデバッグと配置のためのツールも提供しています。

オブジェクト・ビルダーを使用すると、基本コンポーネントからビジネス・オブジェクトを合成してアプリケーションを組み立てることができます。結合 (コンポーネントをより複雑なオブジェクトに結合すること) および分離 (継承によって関連付けられているコンポーネントを特定の基準を使って選択すること) により、ビジネス・オブジェクトを合成することができます。基本オブジェクトおよび結合オブジェクトは、アプリケーションで再利用することができます。

オブジェクト・ビルダーは、フレームワーク完了によるプログラミングもサポートしています。この目的は、プラットフォームを超えた、ビジネス・オブジェクトのソース・コード・レベルでの移植性を実現することです。ビジネス・ロジックのインプリメントは、開発者が行います。それ以外のコードとクライアント・バインディングは、自動的に生成されます。

オブジェクト・ビルダーとソフトウェア構成管理ツールとの間のインターフェースは業界標準に基づくため、サード・パーティーのツールを Component Broker のアプリケーション開発環境に統合することができます。オブジェクト・ビルダーは、Rational Rose などのオブジェクト指向の分析ならびに設計のためのツールから入力された設計を受け入れます。たとえば、システム設計を Rational Rose で作成し、それをオブジェクト・ビルダーにインポートし、次にオブジェクト・ビルダーを使用して最終的なオブジェクトとプログラム・ロジックを追加することができます。

モデリング、分析、および設計のためのツールとのブリッジ

VisualAge Component Development Toolkit には、システム設計ツールとモデリング・ツールとのブリッジが組み込まれています。 たとえば、Rational Rose で作成されたシステム設計をオブジェクト・ビルダーにインポートし、アプリケーションのインプリメントのためのテンプレートとして利用することができます。

- Component Broker は、システム設計および分析のためのツールである Rational Rose にロードすることができる管理下のオブジェクト・フレームワークのモデルを提供しています。したがって、これによって得られるビジネス・オブジェクト設計をオブジェクト・ビルダーにインポートすることができます。
- Logitech の ERWin を使用すると、リレーショナル・データベース管理システムのテーブルを、Component Broker アプリケーションにインポートすることができます。

その他の開発環境のサポート

VisualAge Component Development Toolkit は、IBM VisualAge プロダクト・ファミリー、Microsoft Visual C++ および Microsoft Visual Basic などの開発環境で使用することができます。これらのツールは、アプリケーションを設計する際にオブジェクト・ビルダーの代わりとなるものです。たとえば、VisualAge C++、VisualAge for Java、あるいはその他の開発環境を使用して、クライアントのユーザー・インターフェースを構成することができます。

オブジェクト・ビルダーで 2 つの環境に必要なコードを生成することにより、Microsoft ActiveX コンポーネントと共通利用されるコンポーネントの構築が容易になります。 Microsoft COM オブジェクトは、Component Broker のビジネス・オブジェクトのラッパーとして生成することができます。

VisualAge Component Development Toolkit は、クライアント・コンポーネントのアーキテクチャーもサポートしています。たとえば、クライアント側の JavaBean コンポーネントは、CORBA Internet Inter-ORB protocol (IIOP) 通信バスを介して、Component Broker のビジネス・オブジェクトと統合することができます。

CICS/IMS 接続

CICS/IMS 接続 (CICON) ツールを使用すると、CICS または IMS 手続き型トランザクション・プログラムを、Component Broker で再利用することができます。 CICON は、IBM VisualAge for Java プロダクトに基づきます。CICON

を使用すると、オブジェクト・ビルダーにインポートして他の管理下のオブジェクトと統合することのできる手続きアダプター・オブジェクトを作成することができます。

IBM TeamConnection

IBM TeamConnection は、Component Broker で使用されるチーム・プログラミング環境です。分散オブジェクト・アプリケーションを構築している開発チームに、プログラミング・ライブラリーとツールを提供します。これらの機能は、コンポーネントの共用リポジトリーの中心に位置するものです。開発の作業を、複数のプロジェクトとサブプロジェクトに分割することができます。ソース・コードは、統制をとりながらチーム内で共用することができます。共用することにより、大規模なチーム開発におけるソフトウェアの保全性と生産性が向上します。

IBM TeamConnection は、アプリケーションの配置もサポートしています。共用リポジトリーは、多階層環境におけるスケーラブルなソフトウェア配布の戦略をインプリメントするための出発点になります。ソフトウェアの配置は、アプリケーション・ソリューションのすべてのコンポーネントをパッケージ化し、それらを Component Broker のシステム管理に渡すことから始まります。

Enterprise Bean のサポートと配置

Component Broker は、Enterprise Beans を完全にサポートします。Enterprise Beans は、VisualAge for Java などの統合開発ツールを使用して開発し、Component Broker EJB サーバーで展開することができます。Component Broker Enterprise Beans の展開用に、次のようなツールを提供しています。

- **jetace** − 1 つ以上の Enterprise Beans 用の EJB JAR ファイルの作成および更新を行うことができます。
- オブジェクト・ビルダー − Enterprise Beans を展開するための推奨ツールです。
- **cbejb** − オブジェクト・ビルダーと連携し、EJB サーバーが Enterprise Bean を管理するのに必要なファイルを作成およびコンパイルします。このプロセスの出力は、サーバー側およびクライアント側の、一連の JAR ファイルおよびライブラリー・ファイルです。
- **CBDeployJar** − Enterprise Beans の展開を自動化します。**CBDeployJar** ツールは、EJB 仕様の 1.0 版または 1.1 版と互換性のある JAR ファイルを展開するために使用することができます。
- **CBDeployEar** − J2EE Enterprise Archive (EAR) ファイルに保管されている JAR ファイルから、Enterprise Beans を展開するために使用します。

CBDeployEar ツールは、EAR ファイルから JAR ファイルを抽出し、次に、抽出された JAR ファイル上の **CBDeployJar** ツールを実行します。

- **appbind** − Enterprise Bean の展開者は、アプリケーション固有のネーミング・コンテキストを作成し、それを、選択したファクトリー・ファインダーに関連付けることができます。これにより、このファクトリー・ファインダーを使用して、EJB ホームの検索操作を解決することができます。このツールは、AIX、Windows NT、および Solaris プラットフォーム上でのみ使用することができ、これらのプラットフォームのいずれかにインストールされたサーバーに適用することができます。
- **ejbbind** − Enterprise Bean の Java Naming and Directory Interface (JNDI) ホーム名 (デプロイメント・ディスクリプターに存在します) を EJB サーバー (CB) 中のファクトリーにバインドします。このツールは、AIX、Windows NT、および Solaris プラットフォーム上で稼働するサーバーには使用すべきではありません。

Component Broker 環境における Enterprise Beans の開発と展開についての詳細は、以下の資料を参照してください。

- 「*Enterprise Beans の作成*」
- Component Broker「*アプリケーション開発ツール・ガイド*」

Component Broker のシステム管理

Component Broker のシステム管理ツールは、大規模なサーバー環境を管理するために使用されます。これらのツールは、Component Broker アプリケーション・サーバーのインストール、モニター、および実行をサポートします。
Component Broker のランタイムに、システム管理ツールを使用して、必要なサービス、ロードの必要があるプログラムなどを決定することにより、サーバーのワークロードを管理します。システム管理ツールは、ランタイム・リポジトリーの管理にも使用されます。ビジネス・オブジェクトを登録し、アプリケーションとアプリケーション・ファミリーにグループ化し、管理クライアントに配布することができます。

Component Broker 環境を管理する方法についての詳細は、*Component Broker*「*システム管理ガイド*」を参照してください。

システム管理モデル

Component Broker のシステム管理により、Component Broker ネットワークのあらゆる側面を構成、配置、モニターおよび制御することができます。
Component Broker のシステム管理には、オブジェクト指向のシステム管理モデ

ルが使用されます。すべてのシステム・エンティティーは、オブジェクトで表されます。これらのシステム・オブジェクトは、企業を表すネットワーク、そのネットワークの Component Broker による管理定義、およびそのネットワークと対話するコンポーネントに編成されます。

図26 は、Component Broker で管理されるネットワークの主要部分を示したものです。

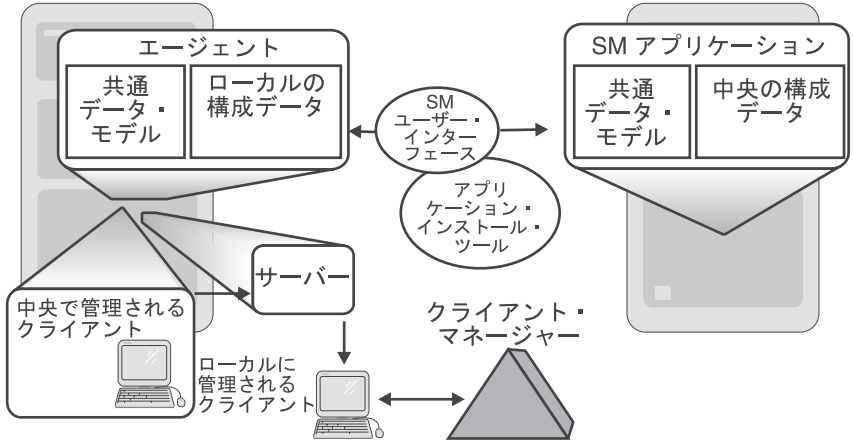

図 26. Component Broker のシステム管理ネットワークのトポロジー

このネットワークは、わかりやすくするために単純化されています。このネットワークは、3 台の System/390 ホスト・マシンで構成されています。1 台のホストは、実際にはシステム・コンプレックス (Sysplex) として構成されたシステムのクラスターです。OS/390 のシステム管理ツールは、Sysplex の構成と操作制御を 1 つのシステム・イメージとして提供する Component Broker アプリケーションです。

注: 実動レベルの管理ネットワークは、通常、図26 に示されているネットワークよりも大規模なものですが、おそらく、1 台のマシン上にネットワーク全体をインプリメントすることができます。

このネットワークは、以下のように構成されています。

- システム管理 (SM) アプリケーションは、中央管理制御ポイントです。システム管理アプリケーションは構成データを中央で保管し、共通データ・モデル (CDM) のコピーが含まれます。システム管理アプリケーションは、1 台のホスト上でそれ自体のプロセスで実行します。 Component Broker ネットワークには 1 つのシステム管理アプリケーションしかありません。 (ただ

し、ネットワークは、緊急時に備えて 2 番目のマシンにオプションとしてシステム管理アプリケーションを構成することができます。)

- Component Broker サーバーは、118ページの図26 の左下に示されている 2 番目のホストで実行されます。このホストもクライアント・アプリケーションを実行しています。

  エージェント は、システム管理アプリケーションが、制御されている Component Broker のサーバーと通信できるようにします。エージェントには、共通データ・モデルのコピーと、そのホストに関連付けられている構成データが含まれます。エージェントには、Component Broker システムのモニターと制御のためのインターフェース・モジュールも含まれます。エージェントは、それが制御するサーバーとは別の独自のプロセスを持っています。
- クライアント・アプリケーションは、118ページの図26 の右下に示されている 3 番目のホストで実行されています。クライアントは、中央管理あるいはローカル管理することができます。ローカル管理されるクライアントの例は、この図の右下隅に示されています。クライアント・マネージャーのユーザー・インターフェースを使用すると、クライアントを制御する属性を設定することができます。

アプリケーション・インストール・ツール は、アプリケーションをホストにロードします。アプリケーション・インストール・ツールを使用すると、構成データをシステム管理アプリケーションまたはエージェントに追加することもできます。

118ページの図26 にはわかるように示していませんが、Component Broker のシステム管理にはログ記録機能が組み込まれています。すべてのコンポーネントは、エラー、活動、およびトレースの各ログ・ファイルを作成します。ログ・ファイルは、システム管理アプリケーション、エージェント、あるいはログ・ブラウザーのユーザー・インターフェースを使って表示することができます。

共通データ・モデル

共通データ・モデル は、Component Broker の構成データの構造の記述またはテンプレートです。共通データ・モデルは、ツリーを構成するオブジェクトのフォルダーを含む階層モデルです。すべてのオブジェクトには、そのフォルダー内に固有の名前が付けられています。したがって、すべてのオブジェクトには、固有の完全修飾パス名があります。

共通データ・モデルは、3 つの概念的な部分 (すなわちワールド) で構成されています。

- モデル・ワールド は、高レベルの用語でシステムのトポロジーを定義します。このデータは、システム管理アプリケーションに保管され、他のデータとは無関係にバックアップおよび復元することができます。
- インストール・ワールド は、低レベルの用語でシステムのトポロジーを定義します。このデータはシステム管理アプリケーションとエージェントの両方に保管されますが、主にエージェントで使用されます。インストール・ワールドのデータは、オブジェクト・ビルダーなどのツールによって生成されるアプリケーション・パッケージを介して提供されます。(管理者は、システム管理アプリケーション内のインストール・データの構造を編集して、改訂された構造をエージェントに送信することができます。)
- イメージ・ワールド は、ローカル・エージェント内の個々のサーバーの定義を提供します。このデータのほとんどは、サーバーからアクセスできるよう共通のデータ・ストアにマップされます。イメージ・ワールドのデータは、モデル・ワールドとインストール・ワールドに保管されているデータから、システム管理コードによって作成されます。

一部のオブジェクト (コンテナーなど) は、モデル・ワールドとイメージ・ワールドの両方に存在します。これらのオブジェクトは、名前で一致検索されます。これらのオブジェクトでは、管理者が属性を提供すると、アプリケーション・パッケージにより対応するイメージ・オブジェクトの関係が提供されます。

第9章 TXSeries の使用

IBM TXSeries は、サーバーを調整および統合して、ネットワーク上の高性能アプリケーションやデータ・ソースを管理する、拡張トランザクション処理ソリューションです。これにより、オンライン・トランザクション処理に必要な信頼性、可用性、およびデータ保全性をすべて備えた分散クライアント / サーバー環境を構築することができます。

本節では、TXSeries 製品ファミリーの以下のメンバーについて説明します。

- Encina について説明します。これは、オープン分散システムの開発と管理、およびトランザクション処理の実行に使用するソフトウェア製品のファミリーです。詳細については、『TXSeries Encina』を参照してください。
- 顧客情報管理システム (CICS) について説明します。これは、IBM の汎用オンライン・トランザクション処理ソフトウェアです。詳細については、140ページの『TXSeries CICS』を参照してください。

本節では、155ページの『第2部 WebSphere Application Server の使用』で説明する WebSphere ファミリーのアプリケーション例の背景となる情報を提供します。ここでは、TXSeries を使用して、エンタープライズ・トランザクション処理アプリケーションを開発する場合の実用上の問題については特に触れません。

- TXSeries に関する一般情報については、「*概説および機能*」を参照してください。
- Encina に関する詳細については、Encina 製品の資料を参照してください。
- CICS に関する詳細については、CICS 製品の資料を参照してください。

TXSeries Encina

Encina は、オープン分散システムの開発と管理に使用するソフトウェア製品のファミリーです。Encina は、オープン・グループ分散コンピューティング環境 (DCE) の基盤となるテクノロジーを使用して、大規模な分散システムの複雑さに対処し、データの保全性を管理するためのインフラストラクチャーを提供します。また、Encina では、多くの分散システム・プログラミングの局面が単純化されているので、アプリケーション開発者は、その背景にある多くの詳細を気にすることなく、プログラムのビジネス・ロジックに専念することができます。

本節では、以下の Encina ファミリーのメンバーについて紹介します。

- 『Encina モニター』
- 127ページの『リカバリー可能キューイング・サービス (RQS)』
- 128ページの『構造化ファイル・サーバー (SFS)』
- 130ページの『対等通信 (PPC) サービス』
- 132ページの『Encina++』
- 136ページの『Encina ツールキット』
- 137ページの『DCE-Encina Lightweight クライアント (DE-Light)』
- 138ページの『WebSphere アドバンスド版と Encina とのインターオペラビリティー』

Encina に関する詳細については、Encina 製品の資料を参照してください。

Encina モニター

Encina モニター (あるいは単にモニター) は、トランザクション処理アプリケーションの開発、実行、および管理手段を提供するトランザクション処理 (TP) モニターです。Encina モニターは、リソース・マネージャーとともに、特定のサーバーを介して一定の方法で特定のデータにアクセスできるユーザーやクライアントを制御する環境を提供するので、大量のデータの整合性を保つことができます。モニターは、スケーラブルであり、IBM メインフレームなどの既存のコンピューター・リソースと相互に機能するオープン・モジュラー・システムを提供します。

Encina モニター機能

モニターは、TP システムに対して、以下の 3 つの機能領域を提供します。

- **実行時環境**モニターの実行時環境は、TP アプリケーションとリソース・マネージャーを調整し、ロード・バランシングや診断情報の収集などの実行時管理タスクを実行します。さらに、この環境は、後続の実行での呼び出しスケジュールやユーザー、トランザクション、データ依存のルーティング、およびクライアント / サーバーのバインディングに関する情報の検索など、実行環境とのその他の対話にも対処します。
- **システム管理機能**モニター・システム管理インターフェースを使用して、モニター・システムの構成、開始、制御、および終了を行います。モニター・の管理は、モニターの管理インターフェースおよび構成インターフェースを介して行います。
- **アプリケーション開発環境** モニター・アプリケーションは、モニター・アプリケーション・プログラミング・インターフェース (API) とその他の Encina インターフェース (Transactional-C (Tran-C) など) を一緒に使用して

開発します。モニターは、DCE RPC やセキュリティーとの対話などのいくつかのタスクをアプリケーションの代わりに実行し、プログラマーの作業を軽減します。サーバーの開発は、Tran-C か、モニター API が提供するその他のトランザクション・インターフェースを使用して行うことができます。また、モニター API は、分散トランザクション処理環境においてクライアントおよびサーバー・アプリケーション・プログラムを管理するための機能を提供します。

モニターの実行時環境

モニターは、クライアント / サーバー・アプリケーションが実行できる環境を提供します。この実行時環境は、モニター・セル か、単一の管理単位です。1 つのセルはノード の集合から構成されています。ノードは、それぞれアプリケーションおよびモニター・ソフトウェアが実行するマシンです。モニター・セルを構成するノードは、DCE セル内のノードのサブセットです。本書において、「セル」という用語は、特に断らない限り、モニター・セルのことを示します。1 つの DCE セルには、複数のモニター・セルが存在する場合もあります。セキュリティーやネーム・サービスなど、いくつかのサービスは DCE セルによって提供されるため、すべてのノードがこれらの DCE サービスにアクセスできなければなりません。

124ページの図27 に示すモニター・セルの物理体系は、ローカル・エリア・ネットワークによって接続された複数のノードのクラスターです。類似するクラスターが、1 つ以上の広域ネットワークに接続している可能性もあります。

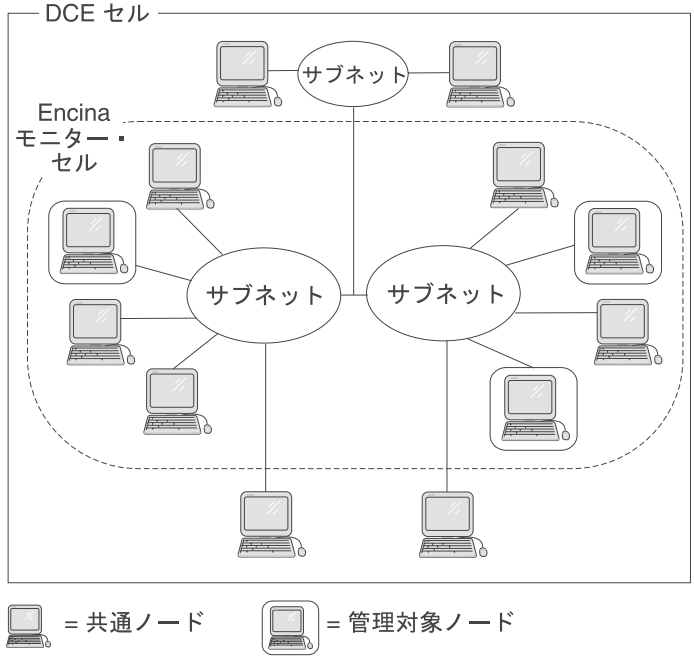

図 27. モニター・セルの物理体系

1 つの モニター・セルには、以下のコンポーネントが含まれています。

セル・マネージャー

セル内のノード・マネージャーとデータ・リポジトリーをモニターし、制御するモニターの一部。セル・マネージャーは、ノード・マネージャー、クライアント、および DCE サービスと通信します。セル・マネージャーの最も重要なタスクは、システムの構成および管理に必要なデータの管理です。

ノード・マネージャー

セル・マネージャーに代わって単一の管理対象ノード内のすべてのアプリケーション・サーバーを制御する Encina の一部。ノード・マネージャーは、単一のノードで実行されるアプリケーション・サーバーの開始とモニターを行います。セル・マネージャーおよびノード・マネージャーは、システムを常にモニターし、障害の発生したアプリケーション・サーバーを検出して報告し、再始動します。

アプリケーション・サーバー

アプリケーション・サーバーは、通常は 1 つ以上のリソース・マネージャーと対話してクライアント要求を処理するコンポーネントです。こ

れらは、管理対象ノードで実行され、アプリケーションのクライアントが必要とするサービスをエクスポートします。

1 つのアプリケーション・サーバーは、クライアントから受信したサービス要求を処理する、処理エージェント (PA) という 1 つ以上のプロセスから構成されています。モニターは、さまざまな処理エージェントに対してクライアント要求を自動的に分散させます。

アプリケーション・クライアント
アプリケーションの一部。ユーザーは、これを使用してモニター・システムと対話し、アクティブなアプリケーション・サーバーによってエクスポートされるサービスを要求します。クライアントに対話式のユーザー・インターフェースを組み込むことも、別の方法で要求を生成することもできます。クライアントは、管理対象ノードで実行する必要のない唯一のコンポーネントです。

リソース・マネージャー
アプリケーション・データなどの共用リソースを管理するコンポーネント。アプリケーション・サーバーは、リソース・マネージャーと通信します。モニター・リソース・マネージャーには、構造化ファイル・サーバー (SFS) およびリカバリー可能キューイング・サービス (RQS) が含まれるので注意してください。

モニターは、セルとモニター・コンポーネントの構成データおよび構成済みの各診断クラスの診断データを含め、そのセルに関するさまざまなデータを管理します。

モニターの実行時環境および Encina アプリケーションの管理についての詳細は、「*Encina 管理ガイド 第 1 巻: 基本管理*」を参照してください。

モニターと DCE の対話

モニターでは、DCE の上に構築されたトランザクション処理環境および Encina ツールキットが用意されています。基盤となる DCE コンポーネントへの Encina の拡張により、必要なアプリケーションの実行環境および関連する管理機能が提供されます。Encina ツールキットは、トランザクションの保全に必要なセマンティクスを提供します。

DCE はインターフェース・モジュラーの集合であり、分散システムを構築するための基本ビルディング・ブロックを提供します。DCE は以下を提供します。

- リモート・プロシージャー呼び出し (RPC) − クライアントとサーバー間の通信のためのプログラミング・パラダイムを提供します。モニターのトランザクション RPC (TRPC) メカニズムは、DCE RPC の上に構築されています。
- セル・ディレクトリー・サービス (CDS) − これにより、DCE セル内のリソースが、一貫性のある方法で識別されます。
- DCE セキュリティー・サービス − モニター・アプリケーションのセキュアな操作のために必要となる認証および許可制御を提供します。

モニターのアプリケーション開発環境

モニター開発環境は、プログラマーがモニター用のアプリケーション・プログラムを開発するためのアプリケーション開発ツール、言語、およびライブラリーから構成されています。開発環境は、主に以下の 3 つのコンポーネントから構成されます。

- Transactional-C (Tran-C) プログラム言語は、C プログラム言語の一連の拡張であり、トランザクション・アプリケーションの開発を簡素化します。Tran-C 実行時システムは、プログラムで使用する Tran-C 関数をサポートするための必要な機能を自動的に呼び出し、トランザクション、および、関連する下位レベルのデータ構造とデータ構成の範囲と状態をモニターします。
- Encina トランザクション RPC (TRPC) は、クライアント / サーバーの対話を定義したり実行したりするために使用します。インターフェースは、DCE IDL への拡張版であるトランザクション・インターフェース定義言語 (TIDL) を使用して記述されます。 DCE RPC 上に構築される TRPC 実行時環境は、Encina モニター・クライアントとアプリケーション・サーバーとの間の通信メカニズムとしてサービスを提供します。
- Encina モニター API は、クライアントおよびアプリケーション・サーバーをモニター・システムに登録できるようにする機能など、分散トランザクション処理アプリケーションの開発をサポートする機能の集合です。

追加のプログラミングおよび診断ツールは、Windows NT および Windows 95/98 システムのプラットフォーム上のみで使用可能です。

- Encina サーバー・ウィザードを使用することにより、Encina および Encina++ サーバーを作成することができます。Encina サーバー・ウィザードは、サーバー用の標準初期化コードの多くを生成し、そのコードをプロジェクトに編成し、サーバーを構築するために必要な、適切な Encina およびシステム・ライブラリーを関連付けます。
- Encina COM ウィザードは、(DLL ファイル形式の) Encina COM コンポーネントを、Encina TIDL インターフェースから作成するために使用されます。これにより、DLL ファイルは、任意の言語で作成されたクライアントに

組み込むことができるようになり、その結果、そのクライアントは、DLL ファイルで定義されたインターフェースをエクスポートする、任意の Encina サーバーにアクセスできるようになります。

- WinTrace ツールは、分散クライアント / サーバー・アプリケーションをデバッグする際の開発者の作業を支援します。この Encina 固有のツールは、アプリケーションの出力ファイルと Encina のトレース・ファイルをフォーマット設定および表示したり、エラー・コードやトレース ID を変換する際に使用します。また、プロセスの実行中に、出力を表示するために使用する Encina Trace Listener サーバーを始動するためにも使用できます。このツールの使用についての詳細は、ツールのオンライン・ヘルプを参照してください。

Encina Monitor 環境で実行されるプログラムの作成についての詳細は、以下の資料を参照してください。

- *Encina アプリケーションの作成*
- *Windows 環境での Encina アプリケーションの作成*
- *Encina Transactional プログラミング・ガイド*
- *Encina モニター・プログラミング・ガイド*

リカバリー可能キューイング・サービス (RQS)

リカバリー可能キューイング・サービス (RQS) によって、アプリケーションは、トランザクション作業を後で処理するようキューイングすることができます。アプリケーションは、キュー内のタスクに関するデータを保管することができます。このデータは、後で別のプログラムによって処理することができます。アプリケーションは、キューイングされた作業がトランザクションで完了されたことを確認してから、そのトランザクションをコミットすることができます。

キュー は、アプリケーション間で情報を受け渡すために使用できる線形のデータ構造です。アプリケーションは、キューの最後にエレメントをエンキュー (追加) したり、キューの先頭から先入れ先出し (FIFO) 法でエレメントをデキュー (除去) したりします (RQS には、エレメントにアクセスするための別の方法も用意されています)。

各キューは、単一の RQS サーバーによって保守されます。そのキューとの対話は、すべてサーバーが処理します。RQS サーバーに複数のキューが存在している場合があります。たとえば、小売業での請求書発行、配送、および在庫管理の各タスクに関連するデータを保管するための 1 つ以上のキューが存在する場合があります。

アプリケーションは、キューを使用して、エレメントの形式でデータを保管します。エレメント は、アプリケーションに固有のレコード単位のデータです。エレメントのフィールドには、関連する情報の一部が保管されます。たとえば、請求書発行エレメントには、顧客名、顧客の口座番号、および現行の口座残高を保管するフィールドがあります。

通常、ビジネス・トランザクションは、トランザクションの作業における各ステップが正常に完了していないと、トランザクション全体が終了できないように構造化されています。RQS サーバーは、クライアントがトランザクションのサブタスクを引き受け、サブタスクをデータで表し、そのデータをエンキューできるようにします。

アプリケーションは、別のアプリケーションによる後続の処理用に、別のキューにエレメントを再度キューイングすることができます。*再キューイング* は、エレメントをあるキューから別のキューに移動するプロセスです。

デキュー要求の処理時にさまざまなキューから選択されたアプリケーションは、キュー・セット (キューの集合) を使用して、選択プロセスを単純化することができます。1 つのキューは、複数のキュー・セットに属することができます。あるキュー・セットに属するキューには、そのキュー・セットの一部としてアクセスすることも、個別にアクセスすることもできます。

ロック によって、RQS 内のエレメントおよびキューの整合性が保証されます。RQS は、操作期間またはトランザクションの期間のロックをサポートしています。RQS が提供するロック・モデルは、同じエレメントまたはキューにアクセスする複数のアプリケーションを実行する場合に高い並行性をサポートしています。

RQS プログラムの作成についての詳細は、「*Encina RQS プログラミング・ガイド*」を参照してください。また、RQS サーバーの管理についての詳細は、「*Encina 管理ガイド 第 2 巻: サーバー管理*」を参照してください。

構造化ファイル・サーバー (SFS)

Encina 構造化ファイル・サーバー (SFS) は、レコード単位のファイル・システムです。SFS は、レコードから構成されている*構造化ファイル* を使用します。*レコード* は関連する情報をグループ化したものです。レコードには、レコードの情報の特定の部分を保持する*フィールド* があります。フィールドのサイズと数は事前に定義されています。これらのフィールドには、事前に定義されているさまざまなデータ・タイプを指定することができます。レコードのフィ

ールド・レイアウトは、ファイルの作成時に定義されます。たとえば、各レコードには、名前、従業員番号、および給与のフィールドを使用して、従業員に関する情報を入れることができます。

SFS はデータ処理機能と管理機能の両方を提供します。データ処理機能は、データのアクセスおよび変更 (読み込み、挿入、更新、削除、ロック、アンロックなど) を行うために使用する標準的な操作を提供します。管理機能によって、プログラムは、SFS ファイルとボリュームの作成、照会、および変更、ファイルのコピー、ファイルの削除などを行うことができます。

SFS ファイル内のデータは、すべて SFS サーバーによって管理されます。このデータにアクセスする必要があるプログラムは、そのサーバーに要求を送信しなければなりません。これによって、要求されたデータが検索されるか、指定の操作が実行されます。

SFS には、以下のように多くの利点があります。

- **トランザクションの保護**。SFS は、ファイル内に保管されたデータへのトランザクション・アクセスを提供します。このため、SFS によって管理されるファイルは、サーバーの問題、ネットワーク障害、およびメディアの障害から完全に回復することができます。
- **分散コンピューティングおよびオープン・システムのサポート**。 SFS は、複数のプラットフォームに渡る構造化データへのアクセスを要求するために、一貫性のあるメカニズムを提供します。SFS が使用するクライアント / サーバー・モデルによって、アプリケーションをネットワーク上に簡単かつ透過的に分散させることができます。
- **既存のアプリケーションの移植が簡単**。SFS では、論理 (物理ではなく) データ・モデルが提供されるため、既存の構造化ファイルまたはデータベース・アプリケーションの移植を単純化することができます。
- **ISAM の互換性**。Encina トランザクション索引順次アクセス方式 (T-ISAM) ライブラリーは、SFS 内に保管されたデータにアクセスするための X/Open ISAM 対応の方法を提供します。
- **COBOL レコード・インターフェース**。SFS 外部ファイル・ハンドラー (EXTFH) は、SFS での Micro Focus COBOL の使用をサポートしています。既存の COBOL アプリケーションは、標準的な COBOL 入出力ステートメントを使用して、SFS ファイルにアクセスすることができます。ネイティブ COBOL 入出力呼び出しは、SFS ファイルに透過的にマップされます。

- **データベース・システムとの互換性**。Encina トランザクション Manager-XA (TM-XA) サービスによって、SFS アプリケーションは、X/Open XA インターフェースをサポートするデータベース・アプリケーションとの対話が可能になります。
- **RQS との互換性**。SFS の多くは、RQS と互換性があります。たとえば、SFS が使用するフィールド・タイプには、RQS に対応するタイプがあります。このため、アプリケーションは、SFS と RQS の両方を簡単に使用することができます。

SFS プログラムの作成についての詳細は、「*Encina SFS プログラミング・ガイド*」を参照してください。また、SFS サーバーの管理についての詳細は、「*Encina 管理ガイド 第 2 巻: サーバー管理*」を参照してください。

対等通信 (PPC) サービス

PPC サービスによって、Encina トランザクション処理システムは、システム・ネットワーク体系 (SNA) LU (論理装置) 6.2 通信インターフェースを持つシステム (通常はメインフレーム) と相互に機能することができます。 PPC サービスにより、SNA および Encina トランザクション処理システムを使用しているメインフレーム・システム間の、統合およびマイグレーションが可能になります。

PPC サービスによって両方向の通信が可能になるので、いずれかの側が通信を開始すると、アプリケーションとデータの両方をメインフレームと Encina との間で共用することができます。 SNA ネットワーク上で実行されているアプリケーションは、PPC ゲートウェイを介する会話を、Encina モニター内のアプリケーションに割り振ることができます。

PPC サービスは、DCE 環境における実行のためのスレッド・セーフなルーチンを提供するさまざまなインターフェースをサポートしています。

- 分散プログラム・リンク (DPL)。DPL によって、Encina PPC エグゼクティブ・アプリケーションは、クライアント / サーバーの関係において、CICS アプリケーションおよびその他の Encina DPL アプリケーションと一緒に機能することができます。DPL アプリケーションは、Encina トランザクション・リモート・プロシージャー呼び出し (TRPC) に似た方法で、その他のアプリケーションと通信します。
- X/Open 共通プログラミング・インターフェース・コミュニケーション (CPI-C) および IBM システム・アプリケーション体系 (SAA) CPI-C の両方。CPI-C インターフェースは、PPC エグゼクティブ・アプリケーションとリモート・アプリケーションとの間の対等通信をインプリメントするためのより一般的な方法を提供します。

- トランザクションの境界用の SAA 共通プログラミング・インターフェース・リソース・リカバリー (CPI-RR) インターフェース。
- トランザクション境界用の X/Open TX インターフェース。

これらのインターフェースは、相互に排他的ではありません。使用するインターフェースは、置かれている環境およびアプリケーションの要件によって異なります。

ゲートウェイ・サーバーを介した通信の PPC サービス・モデルを、図28 に示します。

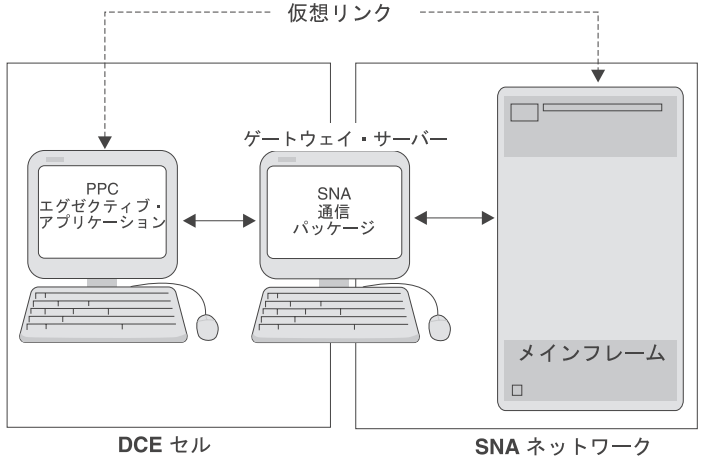

図 28. PPC 通信モデル

ゲートウェイ・サーバーは、DCE セルおよび SNA ネットワークの一部であるマシン上で実行されます。Gateway/SNA は、メインフレーム上の SNA LU 6.2 アプリケーションと DCE ノード上の PPC エグゼクティブ・アプリケーションとの間に仮想リンクを確立します。PPC エグゼクティブ・アプリケーションは、Encina/DCE 環境に完全に統合されます。つまり、アプリケーションは、Encina と DCE を使用して Encina/DCE 環境の他のアプリケーションと通信しながら、SNA を使用してメインフレームと通信することができます。

PPC アプリケーションを構成する 2 つの製品は、以下のとおりです。

- **PPC エグゼクティブ** − 対等通信および 2 フェーズ・コミット・トランザクション・セマンティクスをサポートするライブラリーです。プログラマーは、PPC エグゼクティブが提供する機能を使用して、SNA LU 6.2 プロトコ

ルをサポートする他のアプリケーションと通信するプログラム (*PPC エグゼクティブ・アプリケーション*) を作成します。

- **PPC ゲートウェイ・サーバー** − DCE セル内の Encina アプリケーションと SNA ネットワーク内の LU 6.2 アプリケーションとの間の通信を提供するコミュニケーション・マネージャーです。ゲートウェイを介した会話の割り振りは両方向です。このため、その他のネットワーク上のアプリケーションから、Encina システムのメインフレームまたはマシンにあるデータにトランザクションでアクセスできるようになります。ゲートウェイ・サーバーは、SNA 要件 (ログ記録やリカバリーなど) を PPC エグゼクティブ・アプリケーションから分離します。

PPC プログラムの作成についての詳細は、「*Encina PPC サービス・プログラミング・ガイド*」を参照してください。また、PPC ゲートウェイ・サーバーの管理についての詳細は、「*Encina 管理ガイド 第 2 巻: サーバー管理*」を参照してください。

Encina++

Encina++ は、Encina のためのオブジェクト指向のアプリケーション・プログラミング・インターフェース (API) です。これは、多くの Encina コンポーネントにアクセスする複数のクラスから構成されています。

Encina++ インターフェース

Encina++ は、DCE、CORBA、および DCE と CORBA の両方の混合に基づいたオブジェクト指向アプリケーションの開発をサポートしています。後者のタイプのアプリケーションは、通常、混合アプリケーションといいます。
Encina++ を構成するコンポーネントの中には、Encina++/DCE でしか使用できないものや、Encina++/CORBA でしか使用できないものもあれば、どちらか一方または両方で使用できるものもあります。

共通の Encina++ インターフェース: 以下の Encina++ インターフェースは、どのタイプの Encina++ アプリケーションでも使用することができます。

Encina++

Encina++ インターフェースは、クライアント / サーバー・アプリケーションの作成と管理を可能にし、基礎となる環境にサポートを提供する C++ クラスとメンバー関数を定義します。

Transactional-C++ (Tran-C++)

Tran-C++ インターフェースは、C++ の構成とマクロをはじめ、分散トランザクション処理のためのクラスとメンバー関数を定義します。この

インターフェースは、Encina Transactional-C インターフェースに代わるオブジェクト指向のインターフェースを提供します。

オブジェクト・トランザクション・サービス (OTS)
オブジェクト管理グループ (OMG) OTS インターフェースも、分散トランザクション処理のための C++ のクラスとメンバー関数を定義します。このインターフェースは、*OMG document 94.8.4* で説明しているように、OMG オブジェクト・トランザクション・サービス 仕様をインプリメントします。

共通の Encina++ インターフェースを使用したプログラムの作成についての詳細は、「*Encina オブジェクト指向プログラミング・ガイド*」を参照してください。

Encina++/DCE インターフェース: DCE (混合アプリケーションを含む) 用にしかサポートされていない Encina++ プログラミング・インターフェースは、専用のサービスを提供する次の 2 つのタイプの Encina に対してオブジェクト指向アクセスを提供します。

リカバリー可能キューイング・サービス C++ インターフェース (RQS++)
RQS++ インターフェースは、RQS サーバーでデータを必要に応じてエンキューおよびデキューするための C++ のクラスおよび関数を定義します。

構造化ファイル・サーバー C++ インターフェース (SFS++)
SFS++ インターフェースは、トランザクションの保全性を保守しながら SFS サーバーでレコード単位のファイルに保管されたデータを操作するための C++ のクラスおよび関数を定義します。

Encina データ定義言語 (DDL) コンパイラーは、RQS++ アプリケーションおよび SFS++ アプリケーションが使用するクラスを生成するために使用します。これらのインターフェースに関する詳細については、「*Encina RQS++ および SFS++ プログラミング・ガイド*」を参照してください。

Encina++/CORBA インターフェース: Encina++ は、CORBA アプリケーションおよび混合 DCE-CORBA アプリケーション用にしかサポートされていない 2 つのプログラミング・インターフェースを提供します。

オブジェクト並行性制御サービス (OCCS)
OCCS インターフェースは、複数のクライアントが共用リソースへのアクセスを調整できるようにする C++ クラスおよび機能を定義しま

す。このインターフェースは、*OMG TC Document 94.5.8* で説明しているように、OMG *オブジェクト並行性制御サービス Proposal* をインプリメントします。

Java OTS クライアント・インターフェース

Java OTS クライアント・インターフェースは、Java クライアント・アプリケーションによる分散トランザクションの開始および制御を可能にする Java クラスおよび機能を定義します。このインターフェースは、*OMG document 94.8.4* で説明しているように、OMG *オブジェクト・トランザクション・サービス* 仕様をインプリメントします。

Encina++/CORBA インターフェースを使用したプログラムの作成についての詳細は、「*Encina オブジェクト指向プログラミング・ガイド*」を参照してください。

Encina++ プログラミング・モデル

Encina++ クラスは、クライアント / オブジェクト・プログラミング・モデルをサポートしています。このモデルでは、サーバーの代わりにクライアントがオブジェクトにアクセスします。サーバーは、1 つ以上のインターフェース (クラス) および各クラスの 1 つ以上のインスタンス (オブジェクト) をエクスポートします。システムで使用できるオブジェクトがどのようにサーバーにマッピングされるかをアプリケーション開発者が知らなくても、クライアント・アプリケーションからサーバーによってエクスポートされたオブジェクトにアクセスすることができます。

クライアントは、サーバーによってエクスポートされたオブジェクトにバインドすることができます。オブジェクトが認識されている場合には、クライアントは個々のオブジェクトにバインドすることができます。あるいは、オブジェクトが認識されていない場合または特定のクラスのすべてのオブジェクトが同じ機能を提供する場合には、クラスにバインドすることができます。通常、アプリケーション開発者がオブジェクトの名前を指定します。作成されたオブジェクトには universal unique identifier (UUID) が付いていますが、オブジェクトに名前を付けることによって、クライアントは UUID ではなく名前でオブジェクトにバインドすることができます。

Encina++ では、インターフェース定義言語 (IDL) を使用して、オブジェクトとのインターフェースをリモート・プロシージャーの形式で指定します。リモート・プロシージャーは、クライアントとサーバー・アプリケーションとの間の通信に使用されます。インターフェース・コンパイラーは、各インターフェースのクライアント・スタブ・クラスとサーバー・スタブ・クラスを含むファ

イルを生成します。これらのスタブ・クラスによって、同じインターフェースがクライアントとサーバーとで多少異なって表示されます。

クライアントおよびサーバーのサポート

Encina++ は、オブジェクト指向の分散トランザクション処理アプリケーションに対して、以下の機能を提供します。

- クライアントおよびサーバーの初期化
- 透過的かつ明示的なバインド
- オブジェクトの登録およびバインド
- XA 対応データベースの統合
- トランザクション・スレッドおよび非トランザクション・スレッド
- 統合された例外処理

Encina++ を使用すると、さまざまなタイプのクライアント・アプリケーションおよびサーバー・アプリケーションを C++ で開発できるだけでなく、Encina++ サーバーにアクセスする Java クライアントも開発することができます。

Encina++ インターフェースは、Encina モニターによってエクスポートされる機能をサポートするよう設計されています。このインターフェースを使用すると、モニター・アプリケーション・サーバーおよびクライアントを C++ で作成することができます。モニター・アプリケーション・サーバーおよびクライアントは、DCE、CORBA、あるいはその両方を使用することができます。Encina モニターに関する詳細は、「*Encina モニター・プログラミング・ガイド*」および「*概説および計画ガイド*」を参照してください。

Encina++ インターフェースは、Encina モニターの制御のもとで実行されない C++ クライアント・アプリケーションとサーバー・アプリケーションの開発もサポートしています。これらのアプリケーションは、DCE、CORBA あるいはその両方を使用することができます。

Encina++ は、IONA Technologies, Ltd. の OrbixWeb オブジェクト・リクエスト・ブローカーで使用するための Java クラスのセットを提供します。OrbixWeb によって、分散アプリケーションを Java 言語で構築することができます。Encina++ Java インターフェースによって、OTS の Java 言語のインプリメントを使用して、トランザクションの開始と制御を行う Java クライアント・アプリケーションを作成することができます。Encina++ Java クライアントは、Encina++/CORBA サーバーによってエクスポートされたオブジェクトにバインドすることができます。

Encina ツールキット

Encina ツールキットは、大規模な分散クライアント / サーバー・システムの開発に必要な機能を提供するモジュール、ライブラリー、およびプログラムの集合です。ツールキットのモジュールには、ログ記録とリカバリー・サービス、トランザクション・サービス、およびトランザクション・リモート・プロシージャー呼び出し (TRPC、DCE RPC テクノロジーの拡張) が含まれます。これらのモジュールによって、分散トランザクションの保全性が透過的に保証されます。ツールキットは、C プログラム言語のトランザクションの拡張である Transactional-C (Tran-C) も提供します。

Encina ツールキットの下位のモジュールには、以下のものがあります。

- 分散トランザクション・サービス (TRAN)。トランザクションを調整します。
- ロック・サービス (LOCK)。データへのアクセスが競合しないようにします。
- ログ・サービス (LOG) およびリカバリー・サービス (REC)。トランザクションの代わりに、データに対する変更がデータ全体に対して実行されるようにしたり、実行されないようにすることができます。
- ボリューム・サービス (VOL)。これによって、アプリケーションは、ボリューム という論理単位でストレージを指定することができます。ボリュームは、単一または複数の物理ディスク区画から構成することができます。また、1 つのディスク全体や複数の物理ディスク全体を 1 つのボリュームにすることもできます。ボリューム・サービスは、ログ・サービスが使用するストレージを保守し、リカバリー可能なアプリケーションを再始動できるように、リカバリー・サービスが必要とするデータを順に保存します。
- トランザクション Manager-XA サービス (TM-XA)。トランザクション管理側の X/Open XA インターフェースをインプリメントします。TM-XA は、リレーショナル・データベース・マネージャーを使用して、分散トランザクションを調整します。
- Transarc/Encina DCE ユーティリティー・ライブラリー (TRDCE)。クライアント・プログラムおよびサーバー・プログラムを構成するためのユーティリティーを提供します。

アプリケーション・プログラムは、通常は、下位のモジュールを直接は使用しません。ただし、必要であればアクセスすることができます。たとえば、Tran-C (TRAN ではなく) は、トランザクションを作成する場合に使用します。Tran-C 自体は TRAN を呼び出しますが、アプリケーションは必ずしも TRAN を直接呼び出す必要はありません。

Encina ツールキットの使用についての詳細は、「*Encina ツールキット・プログラミング・ガイド*」を参照してください。

DCE-Encina Lightweight クライアント (DE-Light)

DCE Encina Lightweight Client™ (DE-Light) は、アプリケーション・プログラミング・インターフェース (API) とゲートウェイ・サーバーのセットで、DCE クライアントとして実行されていないシステムに対して DCE や TXSeries Encina の機能を拡張します。

DE-Light を使用することにより、標準の DCE または Encina クライアントに比べて、クライアントの作成作業が省力化でき、使用する管理リソースや生成されるネットワーク・トラフィックが少なくなります。それにもかかわらず、DE-Light のクライアントは、ロード・バランシング、スケーラビリティー、およびサーバーの複製など、以前は、完全な DCE や Encina クライアントしか利用できなかった利点を利用することができます。さらに、DE-Light では、DCE をサポートしていないけれども、Java をサポートしているシステムから DCE や Encina にアクセスすることができます。

DE-Light は、以下のコンポーネントから構成されています。

- Java API － スタンドアロンの Java アプリケーション用の Java クライアントの開発に使用されます。DE-Light の Java クライアントは、TCP/IP や Hypertext Transfer Protocol (HTTP) を経由して、ゲートウェイと通信します。
- C API － Microsoft Windows 環境用のクライアントの開発に使用されます。DE-Light C のクライアントは、TCP/IP を使用して、既知のエンドポイントでゲートウェイと通信します。
- ゲートウェイ・サーバー － DE-Light クライアントと、DCE または Encina との間の通信を可能にします。

DE-Light の Java クライアントは、以下のいずれかで構成することができます。

- Web サーバー上にある Hypertext Markup Language (HTML) 文書に組み込まれた Java アプレット。このタイプのクライアントでは (138ページの図29 を参照)、ユーザーは、Java 対応の Web ブラウザーを介して Web サーバーに接続し、ブラウザーは、DE-Light Java クライアント・アプレットを、HTML 文書とともに自動的にダウンロードします。DE-Light Java クライアント・アプレットは、適切な DE-Light ゲートウェイに接続します。ゲートウェイは、DCE クライアント・マシン上にある単独のサーバーです。こうし

て、Web クライアントは、DE-Light ゲートウェイを介して、DCE または Encina サーバーと通信できるようになります。

- スタンドアロン Java アプリケーション

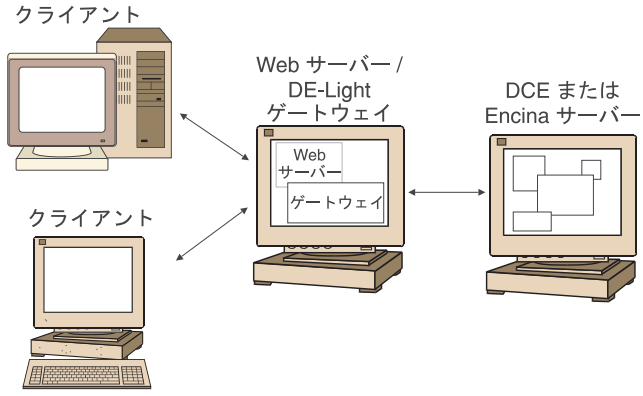

図 29. DE-Light Java クライアント

DE-light プログラムの作成方法についての詳細は、「*Encina DE-Light プログラミング・ガイド*」を参照してください。また、DE-light ゲートウェイを管理する方法についての詳細は、「*Encina 管理ガイド 第 2 巻: サーバー管理*」を参照してください。

WebSphere アドバンスド版と Encina とのインターオペラビリティー

Java プログラム言語で作成されたアドバンスド版 Application Server アプリケーションは、WebSphere アドバンスド版と Encina とのインターオペラビリティー機能を使用して、Encina サーバーおよび Encina++ サーバーと通信することができます。この機能は、Java ベースのサンプル・アプリケーションが Encina++ サーバーと通信できるようにするサンプル・アプリケーションで使用されています。

Java アプリケーションは、ブリッジ・サーバーを使用して Encina アプリケーションまたは Encina++ アプリケーションと通信します。ブリッジ・サーバーは、そのメソッドが Encina アプリケーションまたは Encina++ アプリケーションの TIDL インターフェースに対応している CORBA インターフェースをエクスポートします。Java アプリケーションは IIOP を使用してブリッジ・サーバーにアクセスし、ブリッジ・サーバーは Encina モニター・アプリケーション・サーバーへの TRPC を行います。Java クライアントから Encina への呼び出しは一方向のみです。

注: CORBA インターフェースは、カプセル化されています。つまり、それらを直接呼び出すことはできません。CORBA クライアントは、サポートされていません。

図30 は、ブリッジ・サーバーを使用する分散システムの例です。

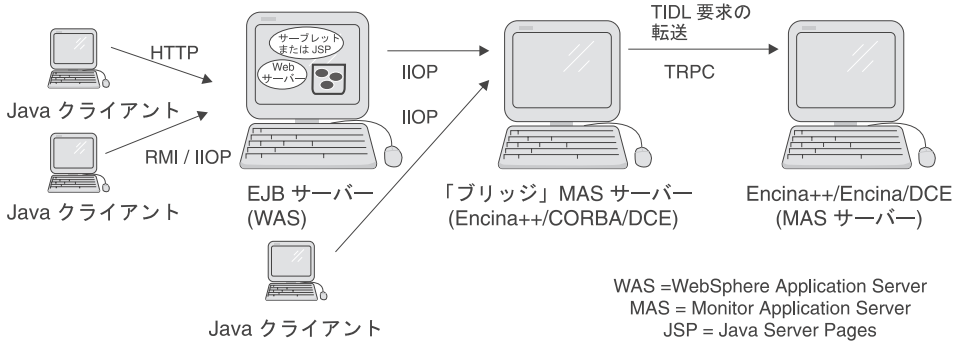

図 30. Java アプリケーションと Encina/Encina++ サーバーとのインターオペラビリティー

wstidl ツールは、Encina TIDL ファイルを入力として取り込み、必要な CORBA IDL インターフェース、Java クラス、サーバーの主要機能、および接続の確立に必要な他のファイルを生成します。生成された Bean または Enterprise Bean は、アドバンスド版 Application Server のアプリケーションの一部として使用することができます。

クライアント側の接続レイヤーは、Bean クラスに組み込まれています。ブリッジ・サーバーでは、接続レイヤーは、CORBA インターフェース、および TIDL パラメーター、TIDL データ・タイプ、および Encina の例外を CORBA の類似のエレメントにマップする追加の特殊ファイルから構成されています。

ブリッジ・サーバーの使用には、以下のような長所があります。

- ブリッジ・サーバーは IBM C++ ORB を使用する。このため、サード・パーティーの ORB に依存する必要はありません。
- ブリッジ・サーバーは、クライアントで発生したトランザクションをサーバーに伝達し、オリジナルの TIDL ファイルに定義されている適切な動作を保存する。

ブリッジ・サーバーの作成方法についての詳細は、「*Encina オブジェクト指向プログラミング・ガイド*」を参照してください。

TXSeries CICS

CICS は、IBM の汎用オンライン・トランザクション処理ソフトウェアです。CICS は、デスクトップから大規模メインフレームにわたる範囲のオペレーティング・システムで実行されるアプリケーション・サーバーです。 TXSeries CICS は WebSphere エンタープライズ版の一部であり、AIX、Solaris、HP-UX、Windows 2000、および Windows NT 上で稼働しますが、その他のバージョンの CICS は OS/390、OS/400、OS/2、VMS などのプラットフォームで稼働します。CICS は、セキュリティー、データ保全性、およびリソースのスケジューリングを扱います。 CICS は、オンライン・トランザクション処理アプリケーションによって必要とされる基本的なビジネス・ソフトウェア・サービスを統合します。

本節では、CICS の高水準の概要について説明します。詳細については、TXSeries「*概説および計画ガイド*」、および CICS の資料を参照してください。

CICS の基本概念

本節では、CICS システムのコンポーネントについて説明します。CICS システムの基本概念についての詳細は、「*概説および計画ガイド*」、「*CICS 管理ガイド (オープン・システム)*」、「*CICS 管理ガイド (Windows システム)*」を参照してください。

領域

CICS のインスタンスを CICS *領域* といいます。UNIX システムまたは Windows システムでは、この領域が複数のプロセスから構成されています。CICS 領域は、単一の管理単位として管理するためのトランザクション処理サービスを提供します。これらのサービスは、通常はユーザー・アプリケーションのビジネス・ロジックをサポートしており、CICS クライアント・アプリケーションおよび 3270 端末ユーザーによって要求されるトランザクションとして実行されます。

CICS 領域は、リソースの獲得と解放を行うためにアプリケーション・プログラムとオペレーティング・システムとが折衝する必要がないようにします。処理中の各タスクは、作業を行うプログラムにはプロセッサー・サイクルとプロセッサー・メモリーを使用し、タスクとユーザー・データ用にメモリーを使用します。タスクは、スクラッチパッド・タイプの計算を実行するためにメモリー領域を少し必要とします。また、データ通信チャネル、データ・ファイル、およびデータベースが必要になる場合もあります。領域は、必要なリソースをすべてオペレーティング・システムから取得します。次に、リソースを必要と

するタスクにリソースを割り振ります。タスクがその処理を完了するか、あるいは特別にリソースを解放すると、領域がリソースを再び獲得します。

CICS 領域は、複数のマシンやプラットフォームに分散している場合でも、リソース・マネージャーに対する同時アクセスを制御し、データ更新の保全性を維持します。たとえば、CICS 領域は、更新、ログの変更、および回復が未確定な更新の同期を取ります。また、セキュリティー・サービスを使用して、許可されたユーザーだけしかデータのアクセスや更新を行えないようにすることもできます。

図31 には、CICS 領域を構成しているコンポーネントが示されています。

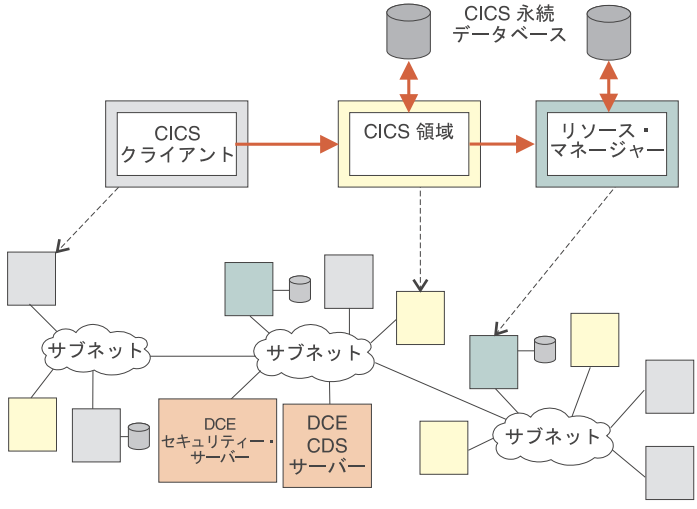

図 31. CICS 領域

一般に、複数の CICS 領域が同じシステムで実行されます。これらの領域は、ある領域を会計処理用に、ある領域は在庫管理用にというように、互いに独立させることができます。あるいは、領域をすべて密接に結合させることもできます。CICS は、領域内 (システム間ともいう) 通信のための機能を多数用意しています。(詳細については、149ページの『CICS システム間通信』を参照してください。)

テーブル

CICS は、テーブルを中心としたシステムです。つまり、CICS 領域の操作は、複数のテーブルによって定義され、制御されます。これらのテーブルは、使用するファイル、使用する端末、使用できるトランザクション (およびこれらの

トランザクションが関連付けられているプログラム) などを CICS に通知します。これらのテーブルの内容は、リソース定義によって制御されます。CICS 管理には多くのリソース定義が関係しています。すべてのトランザクション、プログラム、ファイル、キュー、端末、リモート・システムなどが定義されていなければなりません。

トランザクション

各ユーザーが CICS 領域と対話する場合、1 つ以上のトランザクションが関係します。CICS の用語では、トランザクション とは、ユーザーに提供される基本的な 1 つの操作を意味します。たとえば、銀行取引アプリケーションには、照会トランザクション、借方トランザクション、振替トランザクションなどがあります。この用語は、他の文脈で使用された場合 (Encina での用語として使用された場合) の意味とは多少異なるので注意してください。CICS では、作業最小単位として実行され、耐障害性がありリカバリー可能なアクションのグループを*作業論理単位* (*LUW*) といいます。

アプリケーションを構成するトランザクションは、CICS アプリケーション開発者が作成します。管理者は、領域のトランザクション定義でユーザーに提供するトランザクションを指定します。ユーザーは、通常、トランザクションの名前を 4 文字で指定して、トランザクションを選択します。これにより、CICS は、そのトランザクションを実行するようスケジュールします。トランザクションは、アプリケーション・サーバー というプロセスで実行されます。トランザクションが複数のプログラムでインプリメントされている場合、これらのプログラムは同じプロセスでも別個のプロセスでも実行することができます。CICS 領域は、プログラムの進行をモニターして、データ、通信、およびその他のリソースの要求を提供します。トランザクションが完了すると、CICS 領域はすべてのデータ変更をコミットし、プログラムを終了して、他のトランザクションで使用できるようにリソースを解放します。

ユーザーは、トランザクションを複数の CICS 領域に分散させて、これらの領域にワークロードを分散させることができます。また、トランザクション要求をローカルで (要求を受け取る CICS 領域で) 提供するか、特定のリモート領域に経路指定してそこで実行されるようにするか、あるいはトランザクションが実行できる任意の領域に動的に経路指定するかを、事前に定義することができます。

CICS には、多くの事前定義されたトランザクションが提供されています。これらのトランザクションによって、ユーザーはサインオンおよびサインオフを行うことができます。また、ユーティリティー機能、管理機能、およびデバッグ機能も提供されています。

タスク管理サービス

CICS タスク管理 サービスは、タスクの作成、処理、および終了を制御します。可能な限り、CICS 領域は、最も重要な作業に対して最善の応答時間を割り当てます。通常、複数のタスクが処理時間について競合するため、CICS 領域が優先順位を決定します。複数のタスクが同時に実行されるような場合には、すべてのタスクで同じリソースが使用されなければなりません。領域は、タスクの相対的な優先順位、およびアプリケーション・サーバーや他のシステム・リソースの可用性に応じて、タスクをスケジュールし、ディスパッチします。これによって、タスクの処理率と処理順序が制御されるため、競合またはシステムの過負荷が起こる可能性が低くなります。 CICS 領域は、タスクの存続時間中に何度も優先順位を計算します。たとえば、ファイルまたはユーザーからの入力が必要になるまで、処理を継続できるタスクがあるとします。CICS 領域は、このタスクを中断してその入力を待ち、待機中の新しいタスクを開始するか、中断されている別のタスクの作業を再開することができます。

プログラム管理サービス

CICS プログラム管理 サービスは、タスクの作業を実行するアプリケーション・プログラムにタスクを関連付けるために使用します。多くのタスクで同じアプリケーション・プログラムを使用する場合でも、プログラム制御によってコードのコピーは 1 つしかメモリーにロードされません。各タスクは、このコードをスレッド化して実行するので、多くのユーザーが同じアプリケーション・プログラムの物理コピーを使用して複数のタスクを同時に実行することができます。

1 つのタスクにユーザーとの対話が複数回必要な場合には (入力対象のデータの場合など)、このタスクは、通常、次のプログラムの開始前に終了するように、順番に実行される複数のプログラムとしてインプリメントされます。この技法を、*疑似会話型プログラミング* といいます。各プログラムが終了すると、ユーザーがデータを入力するための画面が表示されます。CICS 領域は、その入力を処理するために次に実行するプログラムを記憶していますが、終了したタスクおよびプログラムが使用していたメモリーは解放します。そのプログラムが他のタスクによって使用されていなければ、領域は、プログラムを記憶していたメモリーも再利用します。このため、CICS 領域がユーザーからのデータ入力を待機している間は、システム・リソースは、他のタスクによって使用されるように解放されます。ただし、ユーザーはプログラムが終了していることに気付くことなく、あたかも会話中であるかのように、システムと続けて通信することができます。

時間管理サービス

時間管理サービスによって、プログラムは、時間に依存する操作の範囲を開始したり、制御したりすることができます。たとえば、1 日のある時間にトランザクション (タスク) を開始して、特定の時間が経過した時点でシグナルを送信したりします。これらのサービスによって、イベントに日付スタンプおよびタイム・スタンプを付けて、ディスクにログを記録して会計処理に使用したり、データ保全性を維持するために使用することができます。また、CICS 領域の自動化の設定も可能になります。

セキュリティー・サービス

CICS 領域は、無許可のログオンに対するセキュリティーを提供します。これにより、特定のユーザーによって個々のリソース (プログラム、ファイルなど) が使用されないように保護することができます。セキュリティー管理サービスは、CICS 内部セキュリティー、外部セキュリティー管理プログラム、分散コンピューティング環境 (DCE) セキュリティー・サービス、またはこれらの組み合わせによって行われる検査に必要なデータを提供します。

CICS 領域は、DCE セキュリティー・サービスを使用して、トランザクション処理サービスを使用するユーザーの認証および許可を一元管理することができます。 CICS は、DCE セキュリティーに代わるより基本的な独自のセキュリティー・サービスを内部的に用意しています。また、システム・セキュリティーのすべての局面を管理するために、特定の目的に使用する外部セキュリティー・パッケージとのインターフェースも提供します。

リカバリー管理サービス

CICS 領域は、ビジネス・システムとデータが常に一貫性のある状態になるようにします。アプリケーションまたはシステムに障害が発生した場合には、領域は、必要に応じて領域自体を自動的に再始動して、データへの変更など、障害発生時に処理されていた未完了の作業を回復します。タスクのデータ変更をコミットできない場合には、領域は、システムが最後に一貫した状態であった時点まで変更内容を動的にバックアウト します。

ユーザー・インターフェース

CICS 端末管理 は、アプリケーションがどのタイプの端末とも通信できるような、1 つの標準的な方法を提供します。CICS 領域は、ユーザーのデバイスを照会し、アプリケーションの出力に最適な特性を判別します。領域は、モデルを使用して、選択した特性に影響を与えることも、端末定義を使用してデバイスに特定の特性を適用することもできます。

CICS アプリケーションとの一般的なユーザー・インターフェースは、IBM 3270 端末 (すなわち実際の 3270 端末か、端末エミュレーター・パッケージ)

を介するものです。ユーザーに 3270 インターフェースを提供しないその他のユーザー・インターフェースは、多くの場合、3270 データ・ストリームを使用して CICS 領域と通信しています。これにより、新しいクライアントが既存の CICS アプリケーションを使用することができます。

CICS アプリケーション・プログラミング・インターフェース

CICS は、アプリケーション・プログラミング・インターフェース (API) を豊富に用意しています。これによって、開発者はトランザクション・アプリケーション・プログラムを作成することができます。

CICS 領域アプリケーション・プログラミング

CICS API は、CICS 領域で操作を実行するための複数のコマンドから構成されています。これらのコマンドは、高水準プログラム言語 (COBOL、C、C++、PL/I、Java など) で作成されたアプリケーション・プログラムに組み込みます。開発者は、EXEC CICS という句に続けて CICS コマンドを指定するだけです。例を以下に示します。

```
EXEC CICS READ FILE('ORDER') INTO(RECORD)
```

プログラム・ソース・ファイルは、プリコンパイラーによって処理されてから、プログラム言語のコンパイラー (COBOL コンパイラー、C コンパイラーなど) によって処理されます。

CICS API コマンドを使用して、以下のタイプの機能を実行することができます。

- ファイル制御 － ファイルの読み取り、書き込み、更新。
- ストレージ管理 － メモリーの割り振りおよび解放。
- プログラム制御 － CICS プログラム間での制御の引き渡し。
- 一時記憶域制御 － 一時記憶キューでの読み取りおよび書き込み。
- 一時データ制御 － 一時データ・キューでの読み取りおよび書き込み。
- インターバル制御機能 － タイマーの使用。
- ジャーナル管理 － 監査証跡、変更記録などについてのジャーナルの作成。
- 基本マッピング・サポートおよび端末サポート － 3270 端末からのデータの送受信。
- 拡張プログラム間通信機能 － SNA LU 6.2 通信を使用した通信。
- タスク制御 － CICS 内部ディスパッチャーの制御。
- 同期点および異常終了サポート － 作業論理単位の処理。

セキュリティー、認証、バッチ・データ交換、モニター、診断など、この他にも多くコマンドが用意されています。

CICS は、これらのコマンドをカプセル化する Java API を提供します。ソフトウェア開発者は、プログラム内の組み込み EXEC CICS コマンドの代替として、この API のクラスを使用することができます。

CICS アプリケーションのプログラミングについての詳細は、下記の資料を参照してください。

- *CICS アプリケーション・プログラミング・ガイド*
- *CICS IIOP および Java プログラミング・ガイド*

CICS クライアント・アプリケーション・プログラミング

CICS は、2 つの API を提供します。これによって、クライアント・マシンにある CICS 以外のアプリケーションは、接続されている CICS 領域の機能を使用することができます。これらの API は、すべての IBM CICS クライアントに共通です。

- 外部呼び出しインターフェース (ECI) によって、クライアント・マシンで実行している CICS 以外のアプリケーションは、CICS 領域で実行している CICS サーバー・プログラムを呼び出すことができます。クライアント・プログラムは、サーバー・プログラムをサブルーチンとして同期または非同期で呼び出すことができます。
- 外部表示インターフェース (EPI) によって、クライアント・アプリケーションは、CICS 領域で実行される既存の 3270 CICS アプリケーションを開始して会話することができます。 CICS アプリケーションは、3270 端末と会話するかのように、クライアント・アプリケーションとの間で 3270 データ・ストリームを送受信します。クライアント・アプリケーションは、これらのデータ・ストリームを取得して、通常は、グラフィカル・ユーザー・インターフェースなどの表示製品で表示します。既存のアプリケーション自体を変更する必要はありません。

CICS クライアントで実行する ECI および EPI アプリケーション・プログラムは、COBOL、C、または C++ で作成することができます。オペレーティング・システム固有の呼び出しを行わないプログラムは、CICS クライアント製品間で移植することができます。クライアント・アプリケーション・プログラムは、ECI と EPI の両方を使用することができます。 ECI および EPI を使用するには、プログラムが適切な ECI ライブラリーおよび EPI ライブラリーにリンクされていなければなりません。

CICS インターネット・アプリケーション・プログラミング

CICS を使用するインターネット・アプリケーションの開発は、インターネット用の通常のプログラミングを拡張することで行います。この開発では、Web サーバーの標準インターフェースを使用します。最も一般に使用されるインターフェースはコモン・ゲートウェイ・インターフェース (CGI) で、これは、Web サーバーからプログラムを呼び出し、そのプログラムからの出力をすべて Web サーバーに戻す方法です。Web サーバーによって呼び出されたプログラムを使用することによって、Web ブラウザーから CICS トランザクションを実行し、ユーザー入力に応答してハイパーテキスト・マークアップ言語 (HTML) ページを動的に作成して、構造化照会言語 (SQL) の照会をデータベース・マネージャーに発行することができます。

CICS ゲートウェイを使用するインターネット・アプリケーションには、通常、以下が含まれます。

- HTML フォーム。データを入力するための Web ページです。ユーザーの入力を、フォームに名前がエンコードされている CICS Internet Gateway に送信します。このフォームは、標準の Web 開発ツールおよびプロセスを使用して開発することができます。
- CICS Internet Gateway。HTML と 3270 データ・ストリームとの間で変換を行うために呼び出される CGI スクリプトです。REXX のようなインタープリター言語または C 言語のようなコンパイル言語を使用して、ユーザー独自の CGI スクリプトを開発することもできます。
- CICS 領域で呼び出されるプログラム。このプログラムは、標準の CICS アプリケーション開発機能を使用して開発することができます。

フォーム以外にも、環境変数およびコマンド行引き数を使用して、データを CICS Internet Gateway に渡すことができます。

リレーショナル・データベースのサポート

CICS は、複数のリレーショナル・データベースの使用をサポートしています。これらのデータベースを使用して、CICS アプリケーションが使用する情報を保管することがでます。また、組み込み構造化照会言語 (SQL) ステートメントを使用することもできます。CICS は XOpen X/A 仕様をサポートしています。つまり、この仕様をサポートしているデータベースであれば、必要に応じて完全な 2 フェーズ・コミット・プロセスを使用して、CICS LUW に参加することができます。

CICS は、以下のリレーショナル・データベース管理システムへのサポートを提供します。

- IBM ユニバーサル・データベース (UDB) (DB2)

- Oracle
- Informix
- Sybase
- Microsoft SQL Server

キュー・サービス

キューは順次記憶機能で、単一の CICS 領域内または相互接続された CICS 領域のシステム内にグローバル・リソースとして存在します。つまり、ファイルやデータベースと同様、キューは特定のタスクに関連付けられていません。タスクは、キューの読み込み、書き込み、または削除を行うことができます。また、キューに関連付けられたポインターは、すべてのタスク間で共用されます。

CICS では、以下の 2 つのタイプのキューが提供されます。

- CICS 一時データ・キュー・サービスは、一般的なキューイング機能を提供します。後続の内部処理または外部処理で使用できるように、データをキューイング (保管) することができます。アプリケーションは以下を行うことができます。
  - 一時データ・キューにデータを書き込む。
  - 一時データ・キューからデータを読み取る。
  - 区画内の一時データ・キューを削除する。
- CICS 一時記憶域キュー・サービスを使用すると、アプリケーション・プログラマーは、データを一時記憶域キューに保管することができます。これらのキューは、主記憶装置か、直接アクセス記憶デバイスにある補助記憶装置のいずれかに配置することができます。一時記憶域キューに保管されるデータを一時データといいます。一時記憶域キューは効果的ですが、必ず CICS 内で作成して処理しなければなりません。ただし、一時記憶域キューは、アプリケーションによって動的に定義することができます。アプリケーションがそれらを使用する前に、CICS に定義しておく必要はありません。アプリケーションは以下を行うことができます。
  - 一時記憶域キューにデータを書き込む。
  - 一時記憶域キューのデータを更新する。
  - 一時記憶域キューからデータを読み取る。
  - 一時記憶域キューを削除する。

これらの名前は永続的でないことを表していますが、CICS キューは永続記憶域です。主記憶装置にある一時記憶域キューを除き、CICS キューは、コールド・スタートで明示的に廃棄されない限り、CICS の実行中は存続します。永

続キューは、Encina 構造化ファイル・サーバー (SFS) か DB2 のいずれかの CICS ファイル管理プログラムによって保管されます。

ユーザー出口

ユーザー出口 とは、CICS モジュール内において、CICS がユーザー作成プログラム (ユーザー出口プログラム) に制御権を渡す位置です。CICS には、ユーザーの出口プログラムが処理を終了すると、制御が戻されます。ユーザーは、ユーザー出口を使用して、独自の要件に従って、CICS システムの機能を拡張したりカスタマイズしたりすることができます。ユーザー出口によって、ユーザーは、CICS システムの操作を効果的に制御することができます

CICS システム間通信

複数システム環境において、CICS 領域は、他の領域と通信して、ローカル領域のユーザーにリモート・システムのサービスを提供し、ローカル領域のサービスをリモート・システムのユーザーに提供することができます。データとアプリケーションは、共に共用することができます。

CICS 相互通信機能

CICS 相互通信機能は、分散システムの操作を単純化します。一般に、このサポートによって標準の CICS 機能 (ファイルやキューへの読み書きなど) が拡張されて、アプリケーションまたはユーザーが、リソースの位置が分からなくてもリモート・システムにあるリソースを使用できるようになります。CICS 相互通信機能を以下に示します。

分散プログラミング・リンク (DPL)

DPL は、EXEC CICS LINK コマンドを拡張して、CICS アプリケーション・プログラムが異なる CICS システムに常駐するプログラムにリンクできるようにします。これを行うと、最初のプログラム (DPL 要求を開始するプログラム) が次のプログラムに制御を渡します。概念上は、DPL には、その他の WebSphere コンポーネントによって使用されるリモート・プロシージャー呼び出し (RPC) およびリモート・メソッド呼び出し (RMI) に類似する点が多くあります。

機能シップ

機能シップによって、アプリケーション・プログラムは、別の CICS システムに属するファイル、一時データ・キュー、および一時記憶域キューにアクセスすることができます。

トランザクション経路指定

トランザクション経路指定によって、リモート・システムでトランザク

ションを実行することができます。トランザクションは、ローカル・システムで実行されたかのように、端末に情報を表示することができます。

非同期処理

非同期処理は、EXEC CICS START コマンドを拡張して、アプリケーションが、別の CICS システムで実行されるトランザクションを開始できるようにします。標準的な EXEC CICS START 呼び出しと同様に、START コマンドで要求されたトランザクションは、START コマンドを送出するアプリケーションから独立して実行されます。

分散トランザクション処理 (DTP)

DTP は、異なるシステムで実行されている 2 つのアプリケーションがお互いに情報を交換できるようにするために、追加の EXEC CICS コマンドを使用します。これらの EXEC CICS コマンドは、論理装置 (LU) 6.2 のマップ済み会話 verb にマップされます。この verb は IBM システム・ネットワーク体系 (SNA) で定義されています。 DTP を使用して、拡張プログラム間通信機能 (APPC) プロトコルを使用する CICS 以外のアプリケーションと通信することができます。

TXSeries 相互通信は SNA LU 6.2 プロトコルに基づいており、多くの場合 APPC と呼ばれます。CICS 領域および Encina 対等通信 (PPC) アプリケーションは、IBM のメインフレーム・ベースの CICS ワークステーションや APPC ワークステーションなどの APPC をサポートしているシステムと SNA を介して通信することができます。

CICS および Encina モニターは、SNA によって定義される 3 つの同期レベルを、SNA ネットワークと TCP/IP ネットワークの両方ですべてサポートしています。TXSeries では、CICS ローカル SNA サポートおよび Encina PPC ゲートウェイ・サーバーを使用することができます。これらの方式は、他の CICS 領域に対する CICS 相互通信機能をすべてサポートしており、また、CICS 以外の領域 (Encina など) に対しては DTP がサポートされています。また、CICS は、IBM CICS クライアントと通信するのに、ローカル SNA サポートを使用することができます。

CICS の相互通信機能についての詳細は、「*CICS 相互通信ガイド*」を参照してください。また、PPC ゲートウェイ・サーバーについての詳細は、130ページの『対等通信 (PPC) サービス』を参照してください。

ユーザーとの通信

ユーザーは、クライアントを介して CICS 領域と通信します。クライアントは、通常は、サーバーと通信し、ユーザーとそのアプリケーション・プログラ

ムとのインターフェースを提供するための製品です。クライアントは、ラップトップ・コンピューターやオープン・システムのワークステーションなどのプラットフォームの範囲で実行されます。

ユーザーは、以下を介して CICS と通信することができます。

- IBM CICS クライアント。これによって、インターネットや Lotus Notes から CICS 領域にアクセスすることもできます。
- 3270 エミュレーション機能を持つ Telnet クライアント。
- CICS 領域に直接接続されたローカル 3270 端末。

ユーザーと通信するために、別のデバイスを CICS クライアントに接続することができます。たとえば、自動預金支払機 (ATM) をクライアントに接続して、その CICS へのユーザー・インターフェースを提供することができます。同様に、クライアントに接続したプリンターを使用して、CICS から出力することができます。

図32 に、CICS クライアントによって使用されるさまざまな通信方式を示します。

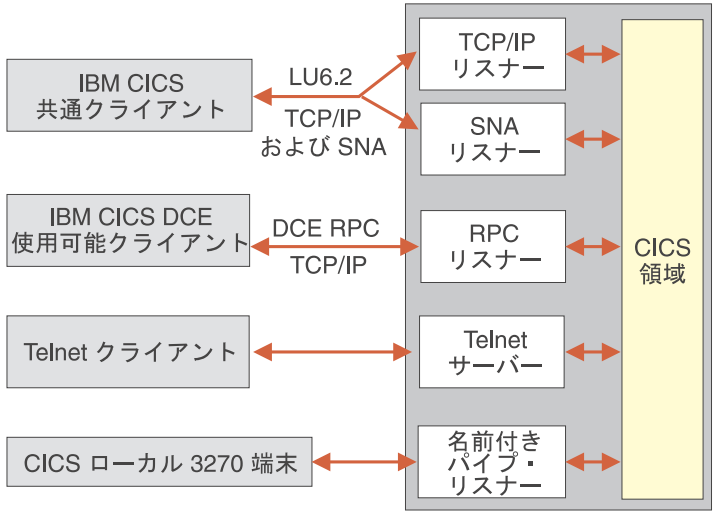

図 32. 複数の CICS クライアントと CICS 領域との間の通信

CICS クライアントは、複数の CICS 領域と通信することができます。クライアント初期設定ファイルによって、クライアント操作のためのパラメーターが判別され、関連する領域および通信に使用されるプロトコルが識別されます。

通信ゲートウェイ

CICS は、以下の通信ゲートウェイをサポートしています。

CICS トランザクション・ゲートウェイ

CICS トランザクション・ゲートウェイにより、Java アプリケーションおよび Web ブラウザーが TXSeries にアクセスすることができるようになります。このゲートウェイは Web サーバーと同じマシンに常駐し、Java 対応の Web ブラウザーまたはネットワーク・コンピューター (NC) によって使用されます。 CICS トランザクション・ゲートウェイは、Web サーバーと CICS クライアントとの間のインターフェースを提供し、クライアントを介して、その他の CICS トランザクション処理環境へのインターフェースを提供します。これはまた、Java のアプリケーションかアプレットと、CICS 領域で実行されるトランザクション・アプリケーションとの間の会話を可能にする API を提供します。CICS トランザクション・ゲートウェイは、Windows NT、AIX、Solaris、および OS/2 オペレーティング・システム上で実行されます。

CICS link for Lotus Notes

CICS link for Lotus Notes は、Lotus Notes アプリケーションが、TXSeries 環境内で管理されているデータにアクセスできるようにするゲートウェイです。このゲートウェイは Lotus Notes サーバーと同じマシンに常駐し、そのサーバーに接続可能な Lotus Notes アプリケーションによって使用されます。このゲートウェイは、Lotus Notes サーバーと CICS クライアントとの間のインターフェースを提供し、クライアントを介して残りの CICS 環境へのインターフェースを提供します。CICS link for Lotus Notes を使用するには、Lotus Notes Server およびクライアントが必要です。これらは、Lotus Domino Server を介して取得するか、(必要なバージョンの) 既存の Lotus Notes ソフトウェアを使用することができます。CICS link for Lotus Notes は、Windows NT および OS/2 オペレーティング・システムで実行されます。

CICS の管理

CICS のシステム管理は、CICS 領域を開始できるような CICS 環境の構成、実行中の領域のモニター、領域の終了、および問題の回復などから構成されています。CICS の管理には、構造化ファイル・サーバー (SFS)、DB2、DCE などのその他のコンポーネントに影響を与えるプロシージャーが必要です。

CICS の構成および管理に使用する管理ツールは、使用しているオペレーティング・システムによって異なります。たとえば、Windows NT の CICS には管理ユーティリティーを使用し、AIX の CICS にはシステム管理インターフェース・ツール (SMIT) を使用することができます。これらのツールは、管理プロ

シージャーを単純化したり自動化したりします。CICS コマンドやトランザクションなどのその他のツールも使用することができます。

CICS 管理ツールは、CICS 環境を 1 つのマシンで管理するよう設計されています。これらのツールを使用するには、システム管理者としてマシンにログインしてから、使用したいツールを起動します。CICS 環境を複数のマシンで管理するには、標準的な技法を使用して、各マシンにリモートでログインし、これらのマシンのツールを使用することができます。たとえば、あるマシンを単一の制御点として使用し、その他のマシンのツールを実行するようセッションをセットアップすることができます。このマシンへのアクセスを制御すれば、管理ツールへのアクセスを制御することができます。

CICS アプリケーションの管理についての詳細は、下記の資料を参照してください。

- *CICS 管理ガイド (オープン・システム)*
- *CICS 管理ガイド (Windows システム)*
- *CICS 問題判別ガイド*

CICS ワークロード管理

CICS ワークロード管理 (WLM) は、作業要求を処理することのできる 2 つ以上の領域を持つ CICS 環境における、タスク処理の分配を最適化する独立型ユーティリティーです。WLM は、AIX と Solaris プラットフォーム上でのみ使用することができ、それぞれ別個にインストールする必要があります。

WLM は、作業を処理することのできる複数のシステム上に、着信の作業要求を配布します。各アプリケーションでは、マシン・リソースを組み合わせて使用することができます。これには、以下のようないくつかの利点があります。

- 複数の場所からアプリケーションにアクセスすることができるため、アプリケーションの可用性が向上する。
- タスクの配布に分配アルゴリズムが使用されているため、使用可能な処理能力が最適化される。
- 追加領域の作成が可能であり、作成した領域を WLM 構成に容易に追加することができるため、スケーラビリティーが向上する。
- 正常なサーバーに要求が自動的に経路指定されるため、サーバーのシャットダウンや保守を透過的に行うことができる。
- 作業要求が WLM 構成内のすべての使用可能なサーバーに分散されるため、ハードウェアのアップグレードにかかる費用を削減することができる。

詳細については、「*CICS ワークロード管理の使用ガイド*」を参照してください。

第2部 WebSphere Application Server の使用

本節では、WebSphere Application Server のいくつかの機能を実現しているアプリケーション例について説明します。このサンプル・アプリケーションでは、WebSphere Application Server 製品ファミリーのメンバーを使用して、e-business Web サイトを構築します。

以下のトピックについて説明します。

サンプル・アプリケーションは、WebSphere Application Server のサンプル・ページ www.ibm.com/software/webservers/samples からアクセスすることができます。

第10章 サンプル・アプリケーションの概要

本節では、サンプル・アプリケーションの概要と目的について説明します。

サンプル・アプリケーションのシナリオ: オンライン・バンキング

このシナリオでは、銀行が企業の Web サイトを構築する場合のサンプル・アプリケーションを中心に説明します。このサイトの目的は、WWW を構築して顧客がオンライン・バンキングを行うことができるようにすることです。

銀行は、以下の情報を顧客に公開することを計画しています。

- 口座残高
- 口座取引の履歴
- 銀行の業務内容、支店一覧とその営業時間、および銀行の紹介など企業情報
- サイト・マップ、他のナビゲーション機能などのサイト情報

また、銀行は、Web サイトを介して顧客が以下の操作を実行できるようにすることを計画しています。

- 口座残高の表示
- 口座取引の履歴表示
- 口座間での送金
- サイト・ページのナビゲートによる他のタイプのサイト情報の表示

サイトは、さまざまなパフォーマンス要件を満たさなければなりません。たとえば、サイトに同時にログオンできるユーザー数や、許可されるユーザーの総数などの要件を満たす必要があります。また、サイトは、トランザクションの最大持続時間や、一般的なトランザクションのタイムアウト時間など、トランザクションに関するパフォーマンス要件も満たさなければなりません。

顧客がこれらのサイトの使用方法を理解できるように、銀行はいくつかの使用例を定義しています。ここでは、サイトの流れおよび例外が提示されており、これによって再設計の必要な部分を識別することができます。使用例を以下に示します。

1. 顧客は、有効なユーザー ID とパスワードを入力して、Web サイトにログオンする。
2. 顧客は、自分のユーザー ID に関連する口座リストを取得する。

3. 顧客は、いずれかの口座の取引履歴を表示する。

最終的に、このサイトは、銀行の情報システムとスムーズに対話しなければなりません。たとえば、既存の顧客データベースから口座情報にアクセスし、既存のトランザクション・モニター・システムとの通信が可能でなければなりません。この要件は、サイトのアーキテクチャーに影響します。顧客データに簡単かつ信頼性の高い状態でアクセスできる場合には (たとえば、DB2 データベースの場合)、データを直接操作するようにサイトを設計することができます。ただし、CICS トランザクションなど、既存のメカニズムを介してデータにアクセスしなければならない場合には、これらの既存のトランザクションを使用するようにサイトを設計しなければなりません。

サンプル・アプリケーションの目的

このサンプル・アプリケーションでは、WebSphere Application Server を使用して電子商取引システムをインプリメントする方法について説明します。

- ここでは、WebSphere Application Server 各部分が一緒に処理を行う方法を示します。サンプル・アプリケーションのインプリメントが異なっていても、ほとんどの WebSphere ツールおよび機能が使用されます。このアプリケーションは、WebSphere Studio、VisualAge for Java、および VisualAge Component Development Toolkit を使用して作成されています。これは、Web サーバー、Web サイト、サーブレット、JavaBeans コンポーネント、および JavaServer Pages (JSP) を使用します。ビジネス・ロジックは、Enterprise Beans および Component Broker 管理下のオブジェクトなど、分散オブジェクトでインプリメントされます。サンプル・アプリケーションのさまざまな部分が Web サーバー、WebSphere アドバンスド版 Application Server、エンタープライズ版 Application Server Component Broker、および TXSeries で実行されます。
- ここでは、アドバンスド版 Application Server およびエンタープライズ版 Application Server を使用してビジネス・システムがインプリメントされる方法を示します。サンプル・アプリケーションの別のインプリメントでは、Enterprise Beans をアドバンスド版 Application Server、TXSeries Encina++、および Component Broker の両方とともに使用する方法を示します。ここでは、また、Component Broker 管理下のオブジェクトを Enterprise Beans の代わりに使用する方法も示します。
- WebSphere Application Server を使用するアプリケーションの作成における開発および設計プロセスでのトレードオフを示します。サンプルでは、以下のソフトウェア開発の概念について説明します。

- 顧客による設計およびインプリメント。サンプル・アプリケーションのアーキテクチャーは、銀行の顧客がどのように使用するのかに直接基づいて作成されています。
- 増分アプリケーションの設計。アプリケーションは、関連するコンポーネントのセットとして作成されます。これらのコンポーネントは、個々に設計してインプリメントすることができます。アプリケーションは、コンポーネント 1 つずつを基本にして、個々の部分から作成されています。
- 再利用の可能性。アプリケーションは、そのコンポーネントが異なるインプリメント間で再利用できるように設計されています。たとえば、同じ Web サイト、サーブレット、および JSP ページが、すべてのバージョンのビジネス・ロジックに使用されます。ビジネス・ロジックの Enterprise Bean のインプリメントでは、アドバンスド版 Application Server またはエンタープライズ版 Application Server の Component Broker に配置することができます。
- モデル・ビュー・コントローラー (MVC) アーキテクチャー。これは、Web サイト・フロントエンドからビジネス処理を分離させる Web アプリケーションを設計するためのパラダイムとして機能します。

- アクセス Bean を説明します。これは、エンタープライズ JavaBeans (EJB) 仕様に対する WebSphere Application Server 拡張機能です。
- ビジネス・システムの設計に使用できるコードの例を提供します。サンプル・アプリケーションは比較的単純ですが、その基本設計およびコードは、さらに変更を行って、複雑なアプリケーションに応用することができます。

第11章 サンプル・アプリケーションの設計

本章では、サンプル・アプリケーションの高水準のアーキテクチャーについて説明します。本章では、以下のトピックについて説明します。

- 『アプリケーション設計』
- 162ページの『クライアント / サーバーの関係』
- 162ページの『モデル・ビュー・コントローラー・アーキテクチャー』
- 163ページの『オブジェクト・モデル』
- 165ページの『データ・モデル』

これらの節は、特に断りのない限り、サンプル・アプリケーションのすべてのインプリメンテーションに適用されます。サンプル・アプリケーションを別の方法でインプリメントする方法については、167ページの『第12章 サンプル・アプリケーションのインプリメント』を参照してください。

アプリケーション設計

サンプル・アプリケーションは、以下のコンポーネントから構成されます。

- お客様とのインターフェースとして機能する Web サイト
- ビジネス処理を調整するサーブレット
- WebCommand。これは、サーブレットとビジネス・ロジック間のインターフェースをカプセル化する JavaBeans コンポーネントです。
- ビジネス・ロジックをインプリメントして、実際のビジネス処理を実行するオブジェクト
- ビジネス処理の結果を表示する JavaServer Pages (JSP)
- 顧客、口座、およびトランザクション情報の永続的ストレージを処理するデータベース

162ページの図33 に、サンプル・アプリケーションの設計方法を示します。

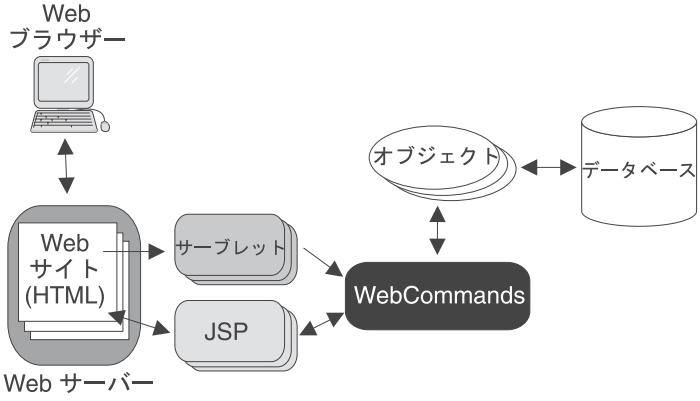

図 33. サンプル・アプリケーション

クライアント / サーバーの関係

サンプル・アプリケーションは、クライアント / サーバー設計になっています。クライアントは顧客の Web ブラウザーです。顧客は、アプリケーションの Web サイトをブラウザーにロードします。送金などの銀行取引操作の実行や口座に関する情報の表示を要求するときに、顧客はブラウザーからサンプル・アプリケーションと対話します。

サーバーは、サンプル・アプリケーションです。これは、クライアントの要求を処理して、クライアント・ブラウザーに表示される情報を戻します。Web サーバーは、クライアント・ブラウザーとビジネス・ロジックが実行されるアプリケーション・サーバーとの中間にあります。これは、Web サイトを取りまとめ、HTTP 要求などを処理します。アプリケーション・サーバーは、すべての情報処理を行い、サーブレット、JSP ページ、およびビジネス・ロジックを取りまとめます。

モデル・ビュー・コントローラー・アーキテクチャー

サンプル・アプリケーションは、モデル・ビュー・コントローラー (MVC) アーキテクチャーをインプリメントします。サンプル・アプリケーションの MVC アーキテクチャーは、さまざまな WebSphere Application Server 機能を使用することによってインプリメントされています。

モデル アプリケーションのビジネス・ロジックおよび他の対話機能を定義しま

す。モデルは、157ページの『サンプル・アプリケーションのシナリオ: オンライン・バンキング』で定義された銀行取引操作を実行します。これは、データベースにアクセスして顧客データの更新も行います。モデルは、タスク処理をカプセル化し、口座、顧客、トランザクション・レコードなどの永続的エンティティーを表すオブジェクトとしてインプリメントされます。

ビュー クライアント・ユーザー・インターフェースを定義して、操作結果をクライアントに表示します。ビューは、Web サイトおよび JSP ページによってインプリメントされます。Web サイトは、クライアントとアプリケーションが対話できるように、サンプル・アプリケーションとのユーザー・インターフェースを提供します。JSP ページは、クライアント操作 (送金、情報要求など) の結果を含む HTML を動的に生成するために使用されます。これらは、Web サイトを処理して、結果をクライアントに表示します。

コントローラー
モデルおよびビューと対話することによってクライアントがアプリケーションに入力を提供する方法を定義します。コントローラーは、サーブレットおよび WebCommand (JavaBeans コンポーネント) によってインプリメントされます。サーブレットは、クライアント要求を Web サイト (ビュー) から受信して、これらをビジネス・ロジック (モデル) に渡して処理します。また、ビジネス処理の結果を受信して、JSP ページ (ビュー) に戻し、結果を表示します。WebCommand は、サーブレットとビジネス・ロジックとの間のインターフェースとして機能し、対話を単純化して対話を独立して開発できるようにします。

オブジェクト・モデル

前述のように、ビジネス・ロジックは、オブジェクトにカプセル化され、サンプル・アプリケーションでそれぞれ異なる機能を実現します。 164ページの図 34 に、ビジネス・ロジックを構成するオブジェクトと対話方法を示します。

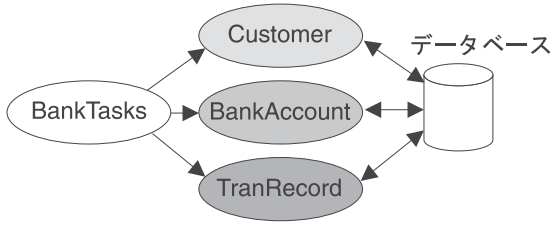

図 34. オブジェクト・モデル

オブジェクトは、サンプル・アプリケーションで以下の機能を実行します。

BankTasks

顧客、口座、およびトランザクション情報を検索して、口座間で送金を行います。BankTasks オブジェクトは、他のオブジェクトへのフロントエンドとして機能します。これは、(間接的ですが) 顧客がアクセスできるメソッドを含む唯一のオブジェクトです。このメソッドは、Customer、BankAccount、および TranRecord オブジェクトを操作して、顧客の要求したタスクを実行します。たとえば、送金を実行するために、BankTasks メソッドは、要求で指定された Account オブジェクトを検索します。次に、送金元である口座から送金金額を引き出し、受け取り口座に預金します。送金金額が 0 より少なかったり、送金元の口座の残高より多い場合は、トランザクションは行われません。

Customer

銀行の顧客を表し、データベースの顧客情報に関連しています。Customer オブジェクトには、顧客情報 (顧客 ID など) をデータベースから検索するメソッドが含まれます。顧客は複数の口座を持つことができるので、Customer オブジェクトは、複数の BankAccount オブジェクトと関連させることができます。

BankAccount

銀行口座を表します。 (サンプル・アプリケーションでは、口座のタイプを区別しません。実際のアプリケーションでは、CheckingAccount、SavingsAccount、など口座のタイプを表すオブジェクトがある場合があります。) BankAccount オブジェクトは、データベースの口座情報 (口座番号や残高など) と関連しています。これには、口座情報を検索して、口座での預金および引き出しを行うメソッドが含まれます。このメソッドは、データベースの口座残高を更新します。これらのメソッドに顧客がアクセスすることはできません。これらのメソッドは、送金時に BankTasks オブジェクトによって使用されます。口座は単一の顧客によって所有されるので、BankAccount オブジェクトは、1 つの Customer オブジェクトとしか関連させることができません。た

だし、口座には、口座に関するトランザクション・レコードが多数あるので、BankAccount オブジェクトは、複数の TranRecord オブジェクトと関連付けることができます。

TranRecord

このオブジェクトは、銀行口座のトランザクションを記録します。また、データベースのトランザクションと関連付けられています。これには、完了したトランザクションに関する情報をデータベースから検索して、新しいトランザクションに関する情報をデータベースに書き込むメソッドが含まれます。口座は多くのトランザクションを持つことができるので、複数の TranRecord オブジェクトを、Account オブジェクトに関連付けることができます。

これらのオブジェクトは、Enterprise Beans または Component Broker 管理下のオブジェクトのいずれかとしてサンプル・アプリケーションでインプリメントされます。異なるオブジェクトのインプリメントについては、167ページの『第12章 サンプル・アプリケーションのインプリメント』を参照してください。

データ・モデル

サンプル・アプリケーションは、永続的ストレージで DB2 データベースにバックアップされます。顧客、口座、およびトランザクション・レコードは、データベース内の特定の行と関連しています。ビジネス・ロジックは、照会があると、この情報をデータベースから検索します。これは、残高の送金要求に応じて、口座残高およびトランザクション・レコードの更新も行うことができます。

第12章 サンプル・アプリケーションのインプリメント

サンプル・アプリケーションは、JavaBeans コンポーネントにインプリメントされているコマンド・セットによってビジネス・ロジックがカプセル化されている Web サイトとして機能します。これらのコマンドは、ビジネス機能 (口座データへのアクセス、送金の実行など) を実際に実行するインプリメント固有のコードを呼び出します。この Web サイトは、サンプル・アプリケーションのすべてのインプリメントで、IBM HTTP Server からサービスを提供されます。アドバンスド版 Application Server は、Web サイトにサービスを提供するサーブレットおよび JavaServer Pages をサポートします。

サンプル・アプリケーションのビジネス・ロジックは、アドバンスド版 Application Server およびエンタープライズ版を使用する以下の 4 つの方法でインプリメントされます。

- アドバンスド版 Application Server に配置された Enterprise Beans によるインプリメント。エンティティー Bean をコンテナー管理のパーシスタンス (CMP) および DB2 データベースともに永続的ストレージとして使用します。このインプリメントについては、 168ページの『Enterprise Beans とアドバンスド版のインプリメント』で説明します。
- Component Broker に配置された Enterprise Beans によるインプリメント。CMP を持つエンティティー Bean とともに DB2 データベースを永続的ストレージとして使用します。このインプリメントについては、169ページの『Component Broker に配置された Enterprise Beans を使用したエンタープライズ版のインプリメント』で説明します。
- Component Broker に配置された Java ビジネス・オブジェクト (BO) によるインプリメント。DB2 へのリレーショナル・データベース・アダプターを使用します。このインプリメントについては、169ページの『Component Broker 管理下のオブジェクトを使用したエンタープライズ版のインプリメント』で説明します。
- アドバンスド版 Application Server に配置された Enterprise Beans によるインプリメント。Bean 管理のパーシスタンス (BMP) を持つエンティティー Bean と TXSeries Encina++ ブリッジ・サーバーをともに使用して、DB2 データベースを永続的ストレージとして使用する Encina++ モニター・アプリケーション・サーバーと通信します。このインプリメントについては、171ページの『Enterprise Beans と TXSeries Encina++ を使用したエンタープライズ版のインプリメント』で説明します。

173ページの『サンプル・アプリケーション・プラットフォーム』では、サンプル・アプリケーションを、さまざまなオペレーティング・システムおよびハードウェア・プラットフォーム上でインプリメントする方法について説明します。174ページの『各インプリメントでの共通点と相違点』では、サンプル・アプリケーションの 4 つのインプリメントの類似点と相違点について比較します。

各インプリメントには、それぞれ利点と欠点があります。ユーザーの組織用に選択するインプリメントは、ユーザーのビジネス要件、既存のシステム、およびどの版の WebSphere Application Server を使用するかによって異なります。

Enterprise Beans とアドバンスド版のインプリメント

図35 に、アドバンスド版 Application Server に配置された Enterprise Beans を使用することによって、サンプル・アプリケーションのビジネス・ロジックをインプリメントする方法を示します。ここでは、ビジネス・ロジックの設計のみを示します。(サンプル・アプリケーションの全体の設計については、162ページの図33 に示します。)

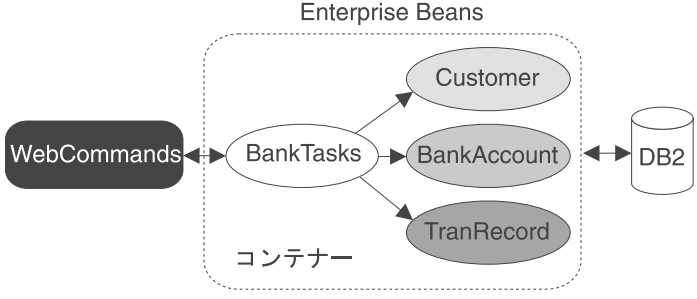

図 35. アドバンスド版 Application Server での Enterprise Bean のインプリメント

WebCommand のインプリメントは BankTasks を呼び出します。BankTasks は、エンティティー Bean Customer、BankAccount、および TranRecord の操作に必要なビジネス・ロジックを含むセッション Bean です。たとえば、BankTasks セッション Bean には、BankAccount Bean によって表される 2 つの口座間で送金を行う transfer というメソッドがあります。transfer メソッドは、BankAccount Bean の 2 つのインスタンス間での預金の引き出しと振り込みをトランザクションで処理します。また、TranRecord Bean を口座ごとにインスタンス化して、送金を文書化します。

パーシスタンスは、Enterprise Beans が配置されたコンテナーによって処理されます。エンティティー Bean による要求に応じて、コンテナーは DB2 データ

ベースから口座情報を検索し、更新された情報をデータベースに書き込みます。Enterprise Bean の開発者は、CMP を持つエンティティー Bean が配置されている場合に、使用するパーシスタンス・モデルについて考慮する必要はありません。パーシスタンス・モデルが後で変化する場合、エンティティー Bean は、アプリケーションのビジネス・ロジックを変更せずに、新しいコンテナーに配置することができます。

コンテナーは、Enterprise Beans のトランザクション管理も行います。トランザクション特性 TX_REQUIRED で配置される Enterprise Beans に関連するトランザクションは、コンテナーによって自動的に管理されます。クライアント・プログラマーおよび Enterprise Bean プログラマーは、トランザクションに関連するコードを Enterprise Beans に組み込む必要はありません。コンテナーは、トランザクションのコンテキストを自動的に管理して、適切なリソースがそれに参加できるようにします。 (この場合、リソースは DB2 データベースです。) コンテナーは、トランザクションに含まれるオブジェクトのライフ・サイクルも管理します。トランザクション特性に関する詳細については、67ページの『トランザクション・サービス』を参照してください。

Component Broker に配置された Enterprise Beans を使用したエンタープライズ版のインプリメント

Component Broker に配置された Enterprise Beans のクライアント / サーバー・プログラミング・モデルは、アドバンスド版 Application Server に配置された Enterprise Beans のものと同じです。 (詳細については、168ページの『Enterprise Beans とアドバンスド版のインプリメント』を参照してください。) パーシスタンスおよびトランザクションの管理以外に、コンテナーは Component Broker との対話をすべて処理します。

Component Broker 管理下のオブジェクトを使用したエンタープライズ版のインプリメント

170ページの図36 に、Component Broker 管理下のオブジェクトを使用して、サンプル・アプリケーションのビジネス・ロジックをインプリメントする方法を示します。ここでは、ビジネス・ロジックの設計のみを示します。(サンプル・アプリケーションの全体の設計については、 162ページの図33 に示します。)

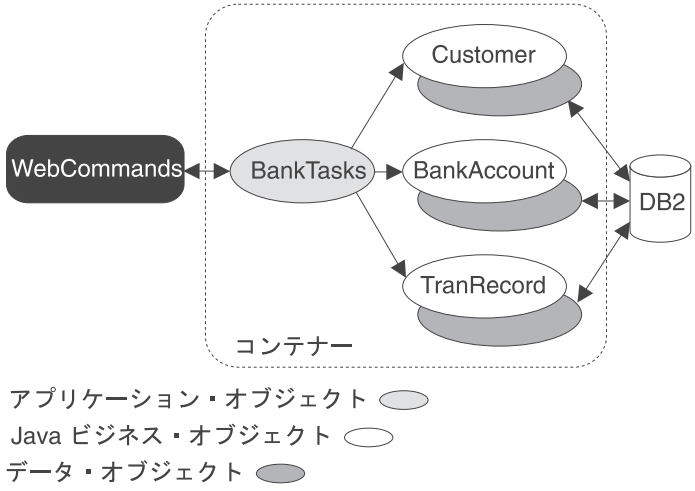

図 36. Component Broker でのビジネス・オブジェクトのインプリメント

WebCommand のインプリメントは、Java ビジネス・オブジェクト Customer、Bank Account、および TranRecord を操作するアプリケーション・オブジェクト BankTasks を呼び出します。たとえば、BankTasks には、2 つの口座間で送金を行う transfer というメソッドがあります。transfer メソッドは、BankAccount ビジネス・オブジェクトの 2 つのインスタンス間での預金の引き出しおよび振り込みをトランザクションで行います。また、TranRecords ビジネス・オブジェクトを作成して、送金を文書化します。

パーシスタンスは、データ・オブジェクトによって処理されます。このデータ・オブジェクトは、DB2 データベースから口座情報を検索して、更新された情報をデータベースに書き込みます。データ・オブジェクトはビジネス・オブジェクトに影響を与えずに変更することができるので、パーシスタンス・モデルが変更された場合に、ビジネス・ロジックを変更する必要はありません。

アプリケーション・オブジェクト、ビジネス・オブジェクト、およびデータ・オブジェクトは、コンテナーに配置されます。自動トランザクションを持つ Component Broker コンテナーは、このコンテナーに配置された管理下のオブジェクトのメソッドに関連するトランザクションをすべて管理します。オブジェクト・プログラマーは、トランザクションに関連するコードを組み込む必要はありません。コンテナーは、トランザクションのコンテキストを自動的に管理して、適切なリソースがそれに参加できるようにします。 (この場合、リソースは DB2 データベースです。) コンテナーは、トランザクションに含まれるオブジェクトのライフ・サイクルも管理します。

Enterprise Beans と TXSeries Encina++ を使用したエンタープライズ版のインプリメント

図37 に、Enterprise Beans および TXSeries Encina++ を使用して、サンプル・アプリケーションのビジネス・ロジックをインプリメントする方法を示します。ここでは、ビジネス・ロジックの設計のみを示します。(サンプル・アプリケーションの全体の設計については、 162ページの図33 に示します。)

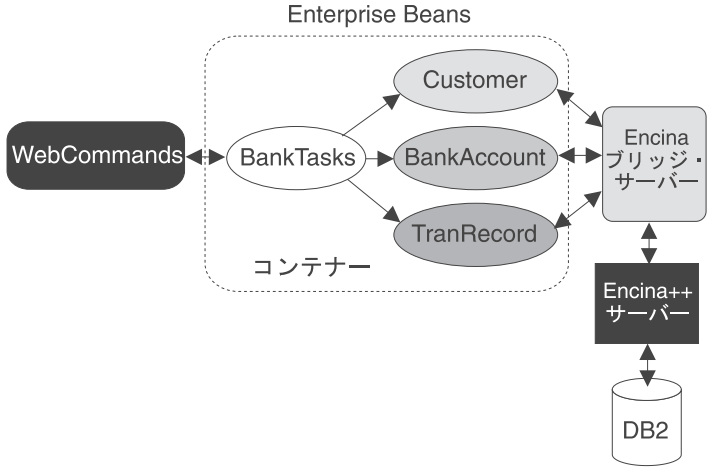

図 37. Encina++ を使用した Enterprise Bean のインプリメント

サンプル・アプリケーションは、WebSphere アドバンスド版と Encina とのインターオペラビリティー機能を使用して、パーシスタンスを管理する Encina++ サーバーと通信します。ブリッジ・サーバーによって、WebSphere Application Server に配置された Java アプリケーションは、Encina および Encina++ サーバーと通信することができます。ブリッジ・サーバーは、メソッドが Encina++ アプリケーションのトランザクション・インターフェース定義言語 (TIDL) インターフェースに対応しているインターフェースをエクスポートします。これらのインターフェースを介して、Enterprise Beans は、Encina++ アプリケーションにアクセスします。**wstidl** コマンドは、Java アプリケーションを、Encina および Encina++ アプリケーションに接続するために必要なファイルを生成します。

Enterprise Bean のインプリメントは、168ページの『Enterprise Beans とアドバンスド版のインプリメント』で説明したアドバンスド版 Application Server のインプリメントに類似しています。ただし、Customer、BankAccount、および TranRecord は、BMP を持つエンティティー Bean です (CMP を持つエンティ

ティー Bean ではありません)。エンティティー Bean は、永続データへのアクセスまたは更新のすべての要求で、**wstidl** コマンドによって生成された JavaBeans コンポーネントを参照します。

JavaBeans コンポーネントは、ファイルも **wstidl** コマンドで生成されているブリッジ・サーバーに常駐しています。JavaBeans コンポーネントは、Encina ブリッジ・サーバーを介して Encina++ アプリケーションに要求を渡します。操作に応じて、Encina++ アプリケーション・サーバーは、DB2 データベースを照会したりトランザクションで更新します。

エンティティー Bean と Encina++ サーバー間で共用されるインターフェースは、永続データにアクセスして更新するためのメソッドです。これらの共用されるインターフェースは、一連の TIDL ファイルから生成されます。さらに、**wstidl** コマンドによって、JavaBeans コンポーネントおよび Encina ブリッジ・サーバーの作成に使用されるファイル・セットが生成されます。

WebSphere アドバンスド版と Encina とのインターオペラビリティーを使用すると、いくつかの利点があります。

- 新しい Java ベースのアプリケーションは、既存の Encina および Encina++ サーバーと通信することができます。
- トランザクションは、Java アプリケーションで開始して、Encina を介してバックエンド・リソースにアクセスし続けることができます。
- 標準の IIOP および EJB インターフェースを使用します。
- クライアントおよびサーバー・ファイルの作成で手作業で行うステップは、Encina または Encina++ アプリケーションと Java アプリケーション間で共用されるインターフェースを作成する TIDL ファイルを作成することだけです。**wstidl** コマンドを実行すると、これらのインターフェースをインプリメントして、ブリッジ・サーバーを作成するのに必要なファイルが自動的に作成されます。

ただし、現段階では、Encina Java 接続機能の使用にはいくつかの欠点もあります。

- TIDL ファイルには、1 つのインターフェースしか含むことができません。(この制限は、実際は、Encina に必要な分散コンピューティング環境 (DCE) による制限です。) ただし、複数の TIDL ファイルをアプリケーションに作成することができます。
- ブリッジ・サーバーへの IIOP 接続には、ワークロード管理がありません。
- ブリッジ・サーバーと WebSphere Application Server 間にはセキュリティーがありません。

サンプル・アプリケーション・プラットフォーム

サンプル・アプリケーションは、以下のアプリケーションで使用可能です。

- AIX
- Solaris
- Windows NT
- OS/390

サンプル・アプリケーションは、WebSphere Application Server のソフトウェアがサポートする、これらのオペレーティング・システムのバージョンと同じバージョンでサポートされています。

AIX、Solaris、および Windows NT プラットフォームでは、完全なサンプル・アプリケーションが使用できます。これらの各プラットフォーム上でのアプリケーションのインプリメントは、本節で既に述べたとおりです。

OS/390 プラットフォームにおけるサンプル・アプリケーションのインプリメントは、多少異なっています。エンタープライズ版 Application Server には OS/390 をサポートしていない部分があるため、このプラットフォームにおけるビジネス・ロジックのインプリメンテーションは、下記の 2 つのバージョンのみが有効です。

- Component Broker で配置された Enterprise Beans (このインプリメンテーションの説明は、169ページの『Component Broker に配置された Enterprise Beans を使用したエンタープライズ版のインプリメント』を参照してください。)
- Component Broker ビジネス・オブジェクト (このインプリメンテーションの説明は、169ページの『Component Broker 管理下のオブジェクトを使用したエンタープライズ版のインプリメント』を参照してください。)

アドバンスド版 Application Server は、OS/390 プラットフォームをサポートしていないため、サンプル・アプリケーションの他の部分 (Web サイト、サーブレット、JSP ページ、および JavaBeans コンポーネント) は、スタンダード版 Application Server を使用して取りまとめられています。サンプル・アプリケーションのこれらのコンポーネントは、すべてのプラットフォーム上で同じように機能します。

各インプリメントでの共通点と相違点

サンプル・アプリケーションのすべてのバージョンでは、アプリケーションの基本的な設計が説明されており、多くの共通機能が含まれます。顧客の入力を収集して顧客コマンドの結果を表示するメカニズムは、4 つのすべてのバージョンで同じです。これらは、すべて同じ Web サイト、サーブレット、および JSP ページを使用します。WebCommand は、サンプルの各バージョンで同じタスクを実行しますが、インプリメントは、ビジネス・ロジックの設計方法によって異なります。

各バージョンのビジネス・ロジックは、共通の機能セットを共用します。

- サンプル・アプリケーションのすべてのバージョンが、DB2 データベースにバックアップされます。
- すべてが、同じコンポーネント・アーキテクチャーを使用します。ビジネス・ロジックの各部分が、コンポーネントにカプセル化されます。さまざまなアプリケーション・タスクを実行するコンポーネントは、永続データを変更するコンポーネントにアクセスするフロントエンドとして機能します。このアーキテクチャーのインプリメント間の類似点および相違点については、『管理下のオブジェクトおよび Enterprise Beans』で説明します。
- 可能な限り多くのデータをローカルに渡すことによって、リモート要求が削減されます。このメカニズムのインプリメント間の類似点および相違点については、 176ページの『CopyHelper オブジェクトおよびアクセス Bean』で説明します。

管理下のオブジェクトおよび Enterprise Beans

サンプル・アプリケーションのコンポーネント・アーキテクチャーは、Component Broker 管理下のオブジェクトまたは Enterprise Beans のいずれかを使用することによってインプリメントされます。管理下のオブジェクトをインプリメントすると、アプリケーション・オブジェクトが、永続データを変更するビジネス・オブジェクトおよびデータ・オブジェクトにアクセスします。Enterprise Bean をインプリメントすると、セッション Bean が、永続データを変更するエンティティー Bean にアクセスします。本節では、サンプルのアーキテクチャーの Component Broker および Enterprise Bean のインプリメントにおける類似点と相違点について説明します。

アプリケーション・オブジェクトおよびセッション Bean

Component Broker アプリケーション・オブジェクトは、さまざまなタスクを実行するように設計されています。これらは、オプションで、パーシスタンス・メカニズムによってバックアップされます。このオブジェクト・モデルは、エ

ンタープライズ JavaBeans (EJB) プログラミング・モデルのステートレスおよびステートフル・セッション Bean に類似しています。

サンプル・アプリケーションで、BankTasks コンポーネントは、Component Broker プログラミング・モデルの場合はアプリケーション・オブジェクトとして、EJB プログラミング・モデルの場合はステートレス・セッション Bean としてインプリメントされます。

ビジネス・オブジェクト、CMP のエンティティー Bean、およびデータ・オブジェクト

Component Broker ビジネス・オブジェクトには、アプリケーションのビジネス・ロジックが含まれます。これらは、パーシスタンスをインプリメントしません。CMP を持つエンティティー Bean は、EJB プログラミング・モデルと類似した役割を果たします。これらには、アプリケーションのビジネス・ロジックが含まれますが、パーシスタンスはインプリメントされません。

Component Broker では、パーシスタンスは、使用しているデータ・ストアを直接操作するデータ・オブジェクトによって処理されます。これらのデータ・オブジェクトは、VisualAge Component Development Toolkit で作成されます。EJB プログラミング・モデルでは、多くの場合、パーシスタンスはコンテナーによって処理されます。エンティティー Bean の開発者は、永続フィールドを識別します。サーバー上のコンテナーに Bean を配置するユーザーが、使用するパーシスタンス・メカニズムを決定します。

Component Broker および EJB プログラミング・モデルの両方で、パーシスタンスを個別にインプリメントすることによって、ビジネス・ロジックに影響を与えずにパーシスタンス・メカニズムを変更することができます。たとえば、データ・ストアは、データ・オブジェクトのタイプまたはエンティティー Bean が配置されるコンテナーのタイプのいずれかを変更することによって、DB2 データベースから Oracle データベースに変更することができます。ビジネス・オブジェクトまたはエンティティー Bean 自体は、変更する必要はありません。

注: BMP を持つエンティティー Bean は、パーシスタンスを直接処理するので、パーシスタンス・メカニズムが変更された場合は、変更する必要があります。171ページの『Enterprise Beans と TXSeries Encina++ を使用したエンタープライズ版のインプリメント』では、BMP を持つエンティティー Bean を使用するサンプル・アプリケーションのバージョンについて説明します。

サンプル・アプリケーションでは、Customer、BankAccount、および TranRecord コンポーネントを、Component Broker プログラミング・モデルではビジネス・オブジェクトとして、EJB プログラミング・モデルではエンティティー Bean としてインプリメントします。パーシスタンスは、Component Broker プログラミング・モデルの場合はデータ・オブジェクトによって、EJB プログラミング・モデルの場合はコンテナー (CMP を持つエンティティー Bean 用) によって処理します。

CopyHelper オブジェクトおよびアクセス Bean

分散オブジェクトのプログラマーは、実際のオブジェクトが常駐し、提供されている位置を常に把握しているわけではありません。これは、同じプロセスであったり、同じマシンでの別のプロセスであったり、まったく異なるマシンでのプロセスであったりする場合があります。オブジェクトからの属性に対するクライアント要求は、情報を検索するために複数のリモート要求になる場合もあります。この情報がさらに効率的に検索できるように、現行のプロセスでオブジェクトのローカル・コピーを保管することは有効です。これは、Component Broker を使用している場合は CopyHelper オブジェクトを使用することによって、Enterprise Beans を使用している場合は アクセス Bean を使用することによって行うことができます。

CopyHelper オブジェクトおよび LocalOnly オブジェクト

Component Broker プログラミング・モデルは、CopyHelper オブジェクトを使用して、オブジェクトのローカル・コピーを作成します。CopyHelper は、create メソッドに使用されます。Component Broker 管理下のオブジェクトのインプリメントでは、CopyHelper は、ビジネス・オブジェクトおよびアプリケーション・オブジェクトのローカル・コピーを保管します。LocalOnly オブジェクトは、ビジネス・オブジェクトの状態を保持します。さらに、メソッドは、情報を LocalOnly オブジェクトからコピーするためにビジネス・オブジェクトから追加されています。

CopyHelper は、オブジェクトの作成時に、属性をクライアントからサーバーに渡すことができます。また、バイト・シーケンスで構造が保管されるので、簡単に通信できるようになります。create メソッドが複数の属性とともに呼び出されると、CopyHelper オブジェクトが作成されます。その直列化された属性は、createFromCopyhelper メソッドを呼び出すことによって渡されます。

LocalOnly オブジェクトには、CopyHelper と同じ機能があります。これらは、属性をバイト・ストリングとして保管する CopyHelper の機能を共用しています。LocalOnly オブジェクトをバイト・ストリングに変換して、リモート・ロケーションに渡し、そこにオブジェクトを戻すことができます。LocalOnly オ

ブジェクトは、属性情報の状況を更新するメソッド (refreshFromJBO メソッド) および属性をビジネス・オブジェクトに送信するメソッド (copyToJBO メソッド) も含まなければなりません。

CopyHelper および LocalOnly オブジェクトは、VisualAge Component Development Toolkit によって生成することができます。ただし、refreshFromJBO および copyToJBO メソッドは、手作業で追加しなければなりません。

アクセス Bean

アクセス Bean は、EJB プログラミング・モデルの拡張機能です。これらは、CopyHelper のアクセス Bean と類似の機能を実行します。これらは、VisualAge for Java のアクセス Bean ビルダーを使用して作成することができます。すべてのタイプの Enterprise Beans で、アクセス Bean を使用することができます。

アクセス Bean は、Enterprise Bean をローカル JavaBeans コンポーネントのように使用できるよう設計されています。これは、Enterprise Bean のホーム・インターフェースとリモート・インターフェースを隠すラッパーとして機能します。これによって、Enterprise Bean の使用が簡単になります。アクセス Bean は、Bean 属性のローカル・コピーを維持するので、リモート要求の数が削減され、パフォーマンスが向上されます。アクセス Bean には、状態情報をリモート Enterprise Bean からローカル・オブジェクトにコピーするメソッド (refreshCopyhelper メソッド) と、状態情報をローカル・オブジェクトからリモート Enterprise Bean にコピーするメソッド (commitCopyHelper メソッド) が含まれます。アプリケーション開発者は、Enterprise Bean のローカル・コピーを更新する頻度を決定しなければなりません。

エンティティー Bean Customer、TranRecord、および BankAccount は、オブジェクトからの複数の属性に対するリモート要求の数を削減するために、アクセス Bean でラップされます。

第13章 サンプル・アプリケーションの技術的な詳細

本章では、サンプル・アプリケーションの各コンポーネントに関する技術的な詳細について説明します。本章では、以下のトピックについて説明します。

- 『Web サイト』
- 181ページの『サーブレット』
- 183ページの『JavaServer Pages』
- 185ページの『WebCommand』
- 190ページの『Enterprise Beans (アドバンスド版およびエンタープライズ版/Component Broker)』
- 194ページの『Enterprise Beans、Encina ブリッジ・サーバー、および Encina++』
- 199ページの『Component Broker 管理下のオブジェクト』

Web サイト

Web サイトは、WebSphere Studio を使用して作成します。このサイトは、IBM HTTP Server 上で実行されます。これは、銀行の顧客とブラウザー・クライアントを介してと対話することを目的としています。これによって、顧客は、情報にアクセスし、銀行取引操作を行うことができます。

Web サイトは、フロントエンドとして機能し、サーブレットおよび JavaServer Pages (JSP) によって生成された情報を表示します。このサイトには、静的 HTML ファイル以外に、サーブレットと JSP ページによって動的に生成された HTML も含まれます。サーブレットは、ビジネス処理を実行したり HTML ページを動的に作成したりする Web サイト・コマンドと関連しています。サーブレットは、JavaBeans コンポーネントをコマンド・スタイル形式で呼び出します。JavaBeans コンポーネントは、フォーマットされた HTML データを生成するために、JSP ページに渡されます。

Web サイトの設計

Web サイトのすべての設計は、162ページの『モデル・ビュー・コントローラー・アーキテクチャー』で説明するモデル・ビュー・コントローラー (MVC) パラダイムに準拠しています。WebSphere Studio の「JavaBeans」ウィザードを使用すると、この設計に従ったサイトが生成されます。一般的なサイトの使用パターンを以下に示します。

1. Web サイトの訪問者が入力ページに進む (要求する)。
2. 訪問者が、データを入力してフォームを送信する。これによって、サーブレットが呼び出される。
3. サーブレットが、JSP ファイルおよび JavaBeans コンポーネントを生成して、これらのファイルおよびコンポーネントから出力ページを動的に作成する。
4. サーブレットが、出力ページをブラウザーに戻す。

WebSphere Studio の「JavaBeans」ウィザードでは、以下のファイルが生成されます。

- 入力ページ用の HTML ファイル (*.html)。このページは、ユーザーの入力を収集してサーブレットを呼び出すために使用されます。
- サーブレット (*.class および *.java ファイル)。JavaBeans コンポーネントを呼び出して、ユーザーのアクションで指定されたメソッドを実行します。
- サーブレット構成ファイル (*.servlet)。サーブレットが JavaBeans コンポーネントを呼び出した場合に処理されます。
- JSP ページ (*.jsp)。変数フィールドをフォーマットするための HTML タグおよび JSP タグが含まれます。このファイルは、要求側のブラウザーに戻す出力 HTML ファイルを動的に作成するために、サーブレットおよび JavaBeans コンポーネントによって使用されます。

注: 入力 HTML ファイル、クラス・ファイル、サーブレット・ファイル、および .jsp ファイルは、すべて公開可能です。Java ファイルは公開できません。

サンプル・アプリケーションでは、クライアントも、使用するビジネス・ロジックのインプリメントを選択します。この情報は、WebCommand の適切なインプリメントと通信するために、サーブレットによって使用されます。詳細については、181ページの『サーブレット』を参照してください。

クライアントの妥当性検査およびバックエンド処理

JavaScript 関数を使用してクライアント・サイドの妥当性検査を行うと、顧客の入力の妥当性検査を事前に行うことができます。これによって、入力エラーを識別するためのリモート要求の必要性を減らすことができます。サンプル・アプリケーションの Web サイトは、JavaScript 関数を使用して、顧客 ID および送金額用の入力フィールドに、正の値を持つ非ブランクの数値フィールドが含まれているかどうか検査します。このタイプのクライアント・サイドの妥当性検査の例として、顧客 ID を入力として取得するログイン・ページがあります。

Web サイトに対するバックエンド処理は、WebCommand という JavaBean クラス・セットにすべてカプセル化されています。これらの JavaBeans コンポーネントは、各コンポーネント (または WebCommand) がビジネス処理の操作を 1 セットずつ実行するようなコマンド・スタイル形式でセットアップされます。コマンド・オブジェクトは、コマンドのパラメーターを定義する属性グループ、コマンドを実行するメソッド、およびコマンドの結果に対して照会される属性のセットを持ちます。WebCommand のインプリメント方法に関する詳細については、 185ページの『WebCommand』を参照してください。

サーブレット

サーブレットは、サンプル・アプリケーションでいくつかの機能を実行します。サーブレットは、クライアントが入力した情報を Web サイトを介して収集し、収集した情報を WebCommand に (最終的にはビジネス・ロジックに) 送信して処理します。さらに、結果を受信して、表示する情報を JSP ページに転送します。

サンプル・アプリケーションのサーブレットは、「WebSphere Studio JavaBean」ウィザードで作成されています。また、「JavaBean」ウィザードによって、基本サーブレット・コードが生成されています。このコードは、ヘッダーやフォーマットの色などを追加するために変更されます。

同じサーブレットが、サンプル・アプリケーションのすべてのインプリメントで使用されています。これらは、以下のタスクを実行するよう設計されています。

1. 入力パラメーターを Web サイトから収集する。
2. WebCommand Bean をロードして、その入力パラメーターを設定する。
3. コマンド Bean を処理する。
4. コマンド Bean を JSP ページに渡して、結果をクライアントに表示する。別の JSP ページが、サーブレット・エラーを受信して処理する。

サーブレットによってロードされる WebCommand のバージョンは、処理に使用されるビジネス・ロジックのインプリメントによって異なります。クライアントは、Web サイトでビジネス・ロジックのインプリメントを選択します。この情報は、サーブレットに対する WebCommand の正しいバージョンを選択するために使用されます。

ユーティリティーは、クライアントの Httpsession での設定に基づいて、コマンド・クラスの特定のインプリメントをロードします。サーブレットがコマンドを呼び出す必要がある場合には、最初にユーティリティー・クラスを呼び出

し、コマンドの正しいインプリメントをロードします。次に、クライアント要求の指示に従って、コマンドを呼び出します。コマンドの特定のインプリメントに特有の例外は、すべてユーザー例外にマッピングされます。このマッピングは、コマンド自体の中にコマンドのインプリメントをカプセル化します。

サーブレットは、ユーティリティーを使用して、汎用コマンド Bean を要求します。コマンド Bean は、サブクラスが特殊化されたメソッドをインプリメントして、ビジネス・ロジックの異なるバージョンにアクセスする抽象 WebCommand です。ユーティリティーは、Httpsession の情報および汎用コマンド Bean の名前を使用して、コマンド Bean の特定のサブクラスをインスタンス化し、それをサーブレットに戻します。次に、サーブレットはメソッドを呼び出します (すべてのサブクラスのインターフェースは、汎用コマンド Bean のサブクラスと同じです)。

たとえば、図38 に、ユーティリティー・クラスのコードの一部を示します。これは、ユーティリティー・クラスが Httpsession の値を使用して、WebCommand の特定のバージョンを呼び出す方法を示しています。(WebSphere Studio では、ユーティリティー・クラスは、「Servlet (サーブレット)」ウィザードによって生成された基本サーブレットに追加されています。)

```
private static String[] subClass = {"EJB","EJB","JBO","ENC"};
...
public  Commands.CMD loadCommand(HttpSession session, String className)
            throws Commands.LoginRequired {
    int sample = 0;
    sample = Integer.valueOf((String)session.getValue("sample")).intValue();
    Commands.CMD obj =
         (Commands.CMD)java.beans.Beans.instantiate(getClass().getClassLoader(),
         "Commands." + subClass[sample] + "." + className + "_CMD");
    obj.setSession(session);
    return obj;
}
```

図 38. コード・サンプル: サーブレット・ユーティリティー・クラス

たとえば、顧客がアドバンスド版 Application Server で実行される Enterprise Bean のインプリメントをロードする場合は、183ページの図39 に示すように、そのインプリメントを処理するために、特別に設計されたコマンド・オブジェクトのバージョンがロードされます。

```
public void performTask(HttpServletRequest request, HttpServletResponse response)
{
...
HttpSession session = request.getSession(true);
// Instantiate the beans and store them so they can be accessed by the called page
    Commands.ShowAccounts_CMD showAccounts_CMD = null;
    showAccounts_CMD = (Commands.ShowAccounts_CMD)
        Util.instance().loadCommand(session, "ShowAccounts");
        "ShowAccounts");
   String customerId = (String)session.getValue("customerId");
// Setting the command bean in the HTTP request
    setRequestAttribute("showAccounts_CMD", showAccounts_CMD, request);
// Initialize the input value of the bean: the customerId property from the parameters
    showAccounts_CMD.setCustomerId(customerId);
// Call the perform method to execute the business logic.
    showAccounts_CMD.perform();

// Call the JSP output page on successful completion.
// If the output page is not passed as part of the URL, the default page is called.
    callPage(getPageNameFromRequest(request), request, response);
  catch(Throwable theException)
  {
// call the error JSP page if an error condition occurs
   handleError(request, response, theException);
   }
}
```

図 39. コード・サンプル: Enterprise Bean 固有のコマンド Bean のロード

サーブレットのこの部分を参照する JSP コードを 184ページの図40 に示します。

JavaServer Pages

JavaServer Pages (JSP) は、クライアント要求の結果を表示するために使用されます。これらは、クライアント・ブラウザーに表示される HTML を動的に生成します。

サンプル・アプリケーションで使用されている JSP ページは、「WebSphere Studio JSP」ウィザードを使用して作成されています。このページは、JSP 仕様の 1.0 バージョンに準拠しています。 jsp:UseBean タグを使用することによって、情報がコマンド Bean から取得されます。このタグは、コマンド Bean への参照を定義します。Bean への参照は、要求が JSP ページに渡された場合に読み取られます。後で、コマンド Bean のプロパティーには、Bean 自体から直接アクセスすることができます。

184ページの図40 に、jsp:UseBean タグを使用してコマンド Bean のプロパティーを取得する方法を示します。コードの例で重要な部分は太字のフォントになっています。ここでは、JSP ページがコマンド Bean から情報を取得する方法を示しています。

```
<jsp:UseBean id="showAccounts_CMD" scope="request" type="Commands.ShowAccounts_CMD">
  <CENTER>
...
<!--METADATA type="DynamicData" startspan
--><%
  try {
   // throws an exception if empty
  java.lang.String _p0 = showAccounts_CMD.getAccountId(0); .
  java.lang.String _p0_0 = showAccounts_CMD.getAccountType(0);
  java.lang.String _p0_1 = showAccounts_CMD.getBalance(0); %>
  <TABLE border="1" width="600">
    <CAPTION><B><FONT size="+1">Accounts</FONT></B></CAPTION>
    <TBODY>
        <TR>
        <TH>Id</TH>
        <TH>Type</TH>
        <TH>Balance</TH>
        <TH>History</TH>
        </TR><%
      for (int _i0 = 0; ; ) { %>
        <TR>
          <TD align="center" class="TBL_ODD"><%= _p0 %></TD>
          <TD align="center" class="TBL_EVEN"><%= _p0_0 %></TD>
          <TD align="right" class="TBL_ODD">$<%= _p0_1 %></TD>
          <TD align="center" class="TBL_EVEN">
              <FORM action="/servlet/Commands.ShowHistory" method="POST"
                      target="_self" name="history<%= _i0 %>">
                <A href="javascript:invoke('history<%= _i0 %>');"
                      onmouseout="if(hover)mout('history',
                      <%= _i0 %>)" onmouseover="if(hover)mover('history',<%= _i0 %>)">
                   <IMG border="0" height="50" name="history<%= _i0 %>"
                      src="file:///D:/Studio/Projects/WSFamily/WSFamily/history_o.gif"
                      width="50">
                   <INPUT name="accountId" type="hidden" value="<%= _p0 %>">
                </A>
              </FORM>
          </TD>
        </TR><%
        _i0++;
  try {
          _p0 = showAccounts_CMD.getAccountId(_i0);
          _p0_0 = showAccounts_CMD.getAccountType(_i0);
          _p0_1 = showAccounts_CMD.getBalance(_i0);
        }
catch (java.lang.ArrayIndexOutOfBoundsException _e0) {
          break;
        }
      } %>
    </TBODY>
  </TABLE><%
}
catch (java.lang.ArrayIndexOutOfBoundsException _e0) {
} %><!--METADATA type="DynamicData" endspan-->
```

図 40. コード・サンプル: <BEAN> タグの使用

jsp:UseBean タグは、HTTP 要求に含まれる JavaBeans コンポーネント showAccounts_CMD を参照します。これは、jsp:UseBean タグ・パラメーターの値 scope="request"によって示されます。

WebCommand

WebCommand は、アプリケーションのビジネス・ロジックとの対話をカプセル化する JavaBeans コンポーネントです。これは、処理する情報を渡すための標準化された方法を提供します。サーブレットは、ビジネス・ロジック・メソッドを直接呼び出さずに、WebCommand を呼び出します。これによって、コンポーネントで適切なメソッドが実行されます。このプロセスによって、Web サイト、サーブレットおよびビジネス・ロジックの開発者はそれぞれ独立して作業することができます。WebCommand は、テキスト・エディターを使用して手作業で作成します。

WebCommand 構造

186ページの図41 に、WebCommand の構造を示します。

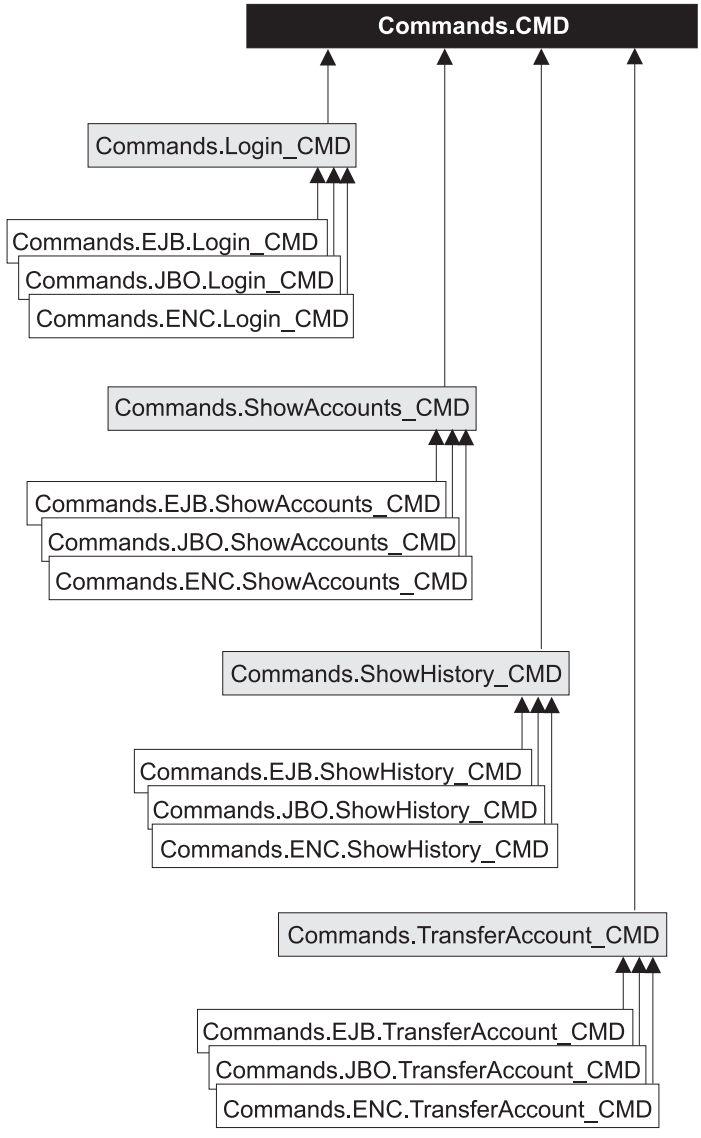

図 41. WebCommand 継承構造

すべての WebCommand は、基本 JavaBeans コンポーネント Commands.CMD から派生します。このコンポーネントには、ビジネス・ロジックと通信する、perform というメソッドが含まれます。WebCommand の各タイプは、顧客の口座の表示 (Commands.ShowAccounts_CMD) または口座間での送金 (Commands.TransferAccount_CMD) などの特定のタスクを実行します。WebCommand は、特定のタスクに対する入出力に必要な属性を追加します。各

WebCommand のサブクラスには、ビジネス・ロジックのバージョンごとに perform メソッドの異なるインプリメントがあります。

Commands.EJB.*command* _CMD
アドバンスド版 Application Server および Component Broker で、ビジネス・ロジックの Enterprise Bean のインプリメントとともに使用されます。

Commands.JBO.*command* _CMD
ビジネス・ロジックの Component Broker Java ビジネス・オブジェクトのインプリメントとともに使用されます。

Commands.ENC.*command* _CMD
ビジネス・ロジックの Enterprise Bean および Encina++ のインプリメントとともに使用されます。

たとえば、Commands.EJB.ShowHistory_CMD は、口座の取引履歴をビジネス・ロジックのアドバンスド版 Application Server または Component Broker Enterprise Bean のインプリメントから検索します。ビジネス・ロジック特有の WebCommand サブクラスは、親クラスの入力属性を使用します。これらは、親クラスの出力属性を perform メソッドのインプリメントの一部として設定します。サーブレットは、181ページの『サーブレット』で説明するように、ビジネス・ロジックのインプリメントに適切な WebCommand をロードします。

アクセス Bean および CopyHelper との対話

WebCommand の `perform` メソッドは、サーバー (EJB サーバーまたは Component Broker 管理下のオブジェクト・サーバーのいずれか) でビジネス・ロジックのインプリメントに対するクライアントとして動作します。実際のビジネス処理は、サーバーで行われます。ただし、WebCommand は、Enterprise Beans または管理下のオブジェクトに直接アクセスしません。リモート・メソッド呼び出し回数を最小にし、リモート・メソッドの検索を単純にするために、WebCommand は、アクセス Bean または CopyHelper のいずれかと対話します。

188ページの図42 に、Commands.EJB.ShowAccounts_CMD の `perform` メソッドのコードの一部を示します。この WebCommand は、アドバンスド版 Application Server および Component Broker の Enterprise Bean のインプリメントとともに使用されます。これは、顧客が所有する口座を表します。顧客 ID は入力として機能します。口座リストは出力として戻されます。

```
public void perform() throws CustNotFoundException, java.lang.Exception {
  btab = new BankTasksAccessBean();
  if (nameServiceTypeName() != null)
     btab.setInit_NameServiceTypeName(nameServiceTypeName());
  if (nameServiceURLName() != null)
     btab.setInit_NameServiceURLName(nameServiceURLName());
  try {baabt = btab.lookupAccounts(customerId);
  }
  catch (javax.ejb.FinderException fe) {
  throw new CustNotFoundException("id " + customerId
              + " does not exist in our records....");
  } finally {
    btab = null;
  }
}
```

図 42. コード・サンプル: アクセス Bean の WebCommand との使用

この例では、アクセス Bean の使用によって、Enterprise Beans のリモート・メソッドの検索が簡単になっています。 perform メソッドは、セッション Bean の BankTasks のフロントとなる新しいアクセス Bean を作成します。クライアントの URL および初期コンテキスト・ファクトリーの設定は、基本 Commands.CMD コンポーネントから読み取られます。 lookupAccounts メソッドは、サーブレットの WebCommand Bean で設定され、customerId 変数に保管されている顧客 ID を使用して呼び出されます。メソッドが正常に実行されると、戻されたアクセス Bean が WebCommand によって保管され、JSP ページに送信されて表示されます。最初のアクセス Bean の参照は削除されます。

189ページの図43 に、Commands.JBO.ShowAccounts_CMD の perform メソッドのインプリメントを示します。この WebCommand は、Component Broker 管理下のオブジェクトのインプリメントとともに使用されます。この perform メソッドは、Commands.EJB.ShowAccounts_CMD の perform メソッドと類似しており、同じ入出力パラメーターを使用します。

```
...
protected LocalCustomer lc = null;
protected LocalBankAccount lba = null;
protected byte[][] rtn = null;
...
  public void perform() throws CustNotFoundException, java.lang.Exception {
    IFamSamp.BankTasks bt = null;
    try {
         bt = SharedData.instance(this).bankTasksHome.createBankTasks();
         rtn = bt.lookupAccounts(getCustomerId());
    }
    catch (com.ibm.IManagedClient.INoObjectWKey  ino) {
          throw new CustNotFoundException();
    }
    if (bt != null) {
        bt.remove();
    }
}
```

図 43. コード・サンプル: CopyHelper の WebCommand との使用

188ページの図42 のアクセス Bean のように、CopyHelper は、Java ビジネス・オブジェクトのリモート・メソッドの検索を簡単にします。WebCommand は、Customer および BankAccount オブジェクトのローカル・コピーをインスタンス化します。Component Broker の外部化サービスは、ローカル・オブジェクトの内容を順次渡すことができるように、ローカル・オブジェクトをバイト・ストリングに変換します。

SharedData というユーティリティー・クラスは、ネーム・サービス検索の結果を検索して保管するために使用されます。これによって、WebCommand ではリモート検索が 1 つだけに制限されます。WebCommand は、リモート BankTasks オブジェクトを呼び出し、これによって、アクセス Bean テーブルと類似した一連のバイト・ストリングが戻されます。これは、Commands.ShowAccounts_CMD クラスの Commands.JBO.ShowAccounts_CMD インプリメント内のプライベート・インスタンス・データに保管されます。

バイト・ストリング出力データは、再度ローカル・オブジェクトに変換されます。情報は、JSP ページに戻されて表示されます。BankTasks オブジェクトは削除されます。

Enterprise Beans (アドバンスド版およびエンタープライズ版 /Component Broker)

Enterprise Beans は、サンプル・アプリケーションのアドバンスド版 Application Server およびエンタープライズ版 Application Server/Component Broker バージョンのバックエンド処理を行います。これらの Enterprise Bean のインプリメントは、Bean が配置されているサーバーを除いて、ほとんど同一です。

Web サイトのビジネス・ロジックは、分散オブジェクト・サーバー (アドバンスド版 Application Server またはエンタープライズ版 Application Server の Component Broker のいずれか) で実行される Enterprise Beans に常駐しています。これらの EJB サーバーは、以下を実行するように構成することができます。

- サーブレット・エンジン (アドバンスド版 Application Server) と同じプロセスで実行する。
- 同じマシン (アドバンスド版 Application Server およびエンタープライズ版 Application Server) の他のプロセスで実行する。
- 同じ環境 (アドバンスド版 Application Server およびエンタープライズ版 Application Server) で異なるマシンの他のプロセスで実行する。

ビジネス・ロジックには、ステートレス・セッション Bean を呼び出してアクセスします。これらのステートレス・セッション Bean は、作業単位を監視し、それに応じてエンティティー Bean を操作します。エンティティー Bean は、DB2 データベース内のデータを更新して、ビジネス処理を行います。

顧客は、Web サイトからビジネス・ロジックに直接アクセスしません。顧客の要求は、サーブレットに渡され、これによって、WebCommand の適切なセットが使用され、ビジネス・ロジックにアクセスされます。詳細については、185ページの『WebCommand』を参照してください。

セッション Bean のインプリメント

ステートレス・セッション Bean、BankTasks には、顧客が要求するほとんどのビジネス・ロジックが含まれています。これには、以下のメソッドが含まれます。

- lookupCustomer － 顧客に関する情報を戻します。
- lookupAccounts － 顧客の口座のリストを戻します。
- lookupHistory － 指定された口座の取引履歴を戻します。
- deposit － 口座に振り込みを行い、その取引履歴を更新します。このメソッドは、Web サイトでは使用できません。

- withdraw － 口座から引き出しを行い、その取引履歴を更新します。このメソッドは、Web サイトでは使用できません。
- transfer － 口座間で送金して、その取引履歴を更新します。

セッション Bean は、TX_REQUIRED 属性 (必須トランザクション) とともにコンテナー内に配置されます。セッション Bean が呼び出されると、コンテナーは、新しいトランザクションを開始し、顧客の要求を処理します。処理が終了したらトランザクションをコミットします。

このタイプの Enterprise Bean に関する詳細については、 42ページの『セッション Bean』を参照してください。

エンティティー Bean のインプリメント

3 つのエンティティー Bean (Customer、BankAccount、および TranRecord) は、銀行の異なるエンティティーを表します。Customer および TranRecord エンティティーは、使用しているデータベース内の行を更新します。これらには、追加のビジネス・ロジックは含まれません。BankAccount エンティティー Bean も、使用しているデータベース内の行を更新します。ただし、これには、預金を行ったり、口座から引き出すためのメソッドが含まれます。これは、0 またはマイナスの金額である場合、あるいは残高不足で引き出しができない場合にはエラーを送出します。

注: セッション Bean の BankTasks における withdraw メソッドとエンティティー Bean の BankAccount における withdraw メソッド間の違いは、セッション Bean は、口座に対する変更のトランザクション・レコードも、TranRecords Bean を介して生成する点です。

このタイプの Enterprise Bean に関する詳細については、 46ページの『エンティティー Bean』を参照してください。

アクセス Bean のインプリメント

すべての Enterprise Beans は、アクセス Bean でラップされます。アクセス Bean は、Enterprise Bean のホーム・インターフェースとリモート・インターフェースを隠すことによってユーザーのプログラミング・モデルを単純化します。これによって、Bean をローカル JavaBeans コンポーネントとして扱うことができます。ただし、サンプル・アプリケーションの WebCommand は、Enterprise Beans に直接アクセスしません。アクセス Bean を使用して、Enterprise Beans との対話を単純化します。3 つのタイプのアクセス Bean (ラップ Bean、CopyHelper、および行セット) は、サンプル・アプリケーションの Enterprise Bean インプリメントで使用されます。

セッション Bean、BankTasks はラップ Bean を使用します。このタイプのアクセス Bean は、セッション Bean インスタンスとともに使用されます。以下のコードの例に、ラップ Bean アクセス Bean を使用するセッション Bean のメソッドにクライアントがアクセスするための方法を示します。

```
BankTasksAccessBean bt = new BankTasksAccessBean();
CustomerAccessBean cab = bt.lookupCustomer("1234");
System.out.println("Customer title is " + cab.getTitle());
System.out.println("Customer first name is " + cab.getFirstName());
System.out.println("Customer last name is " + cab.getLastName());
```

エンティティー Bean、Customer、TranRecord、および BankAccount は、すべて CopyHelper アクセス Bean を使用します。このタイプのアクセス Bean は、Enterprise Bean のローカル・コピーを維持し、オブジェクトから属性を取得するときのリモート要求の回数を削減します。 図44 に、CopyHelper アクセス Bean でラップされるエンティティー Bean からクライアントが情報を取得する方法を示します。

```
public BankAccountAccessBean lookupAccount(String accountId)
throws javax.naming.NamingException, java.rmi.RemoteException, FinderException {
  try {
       BankAccountAccessBean baab = new BankAccountAccessBean();
       baab.setInitKey_accountId(accountId);
       baab.refreshCopyHelper();
       return baab;
      } catch (CreateException ce) {
        throw new java.rmi.RemoteException(ce.getMessage());
      }
}
```

図 44. コード・サンプル: エンティティー Bean の CopyHelper ラッパー

BankAccount および TranRecord オブジェクトも、行セット・アクセス Bean を使用します。このタイプのアクセス Bean によって、エンティティー Bean の複数のインスタンスを個々にインスタンス化せずに処理することができます。これは、Enterprise Beans のローカル・コピーの維持も行います。アクセス Bean は、索引付けされた結果を戻し、戻されたデータが JSP ページに簡単に表示できるようにします。 193ページの図45 に、JSP クライアントが行セット・アクセス Bean を介してループする方法を示します。

```
BankAccountAccessBeanTable baabt = bt.lookupAccounts("1234");
   for (int i=0; i < baabt.numberOfRows(); i++) {
       BankAccountAccessBean baab = (BankAccountAccessBean)
              baabt.getBankAccountAccessBean(i);
       System.out.println("Balance is " + baab.getBalance());
       System.out.println("Account Id is " +
              ((BankAccountKey)baab.__getKey()).accountId);
       System.out.println("Account Type is " + baab.getAccountType());
}
```

図 45. コード・サンプル: 行セット・アクセス Bean の使用

アクセス Bean に関する詳細については、 50ページの『アクセス Bean』を参照してください。VisualAge for Java の資料では、アクセス Bean の作成方法および使用方法について説明しています。

Enterprise Beans 間の関連

関連は、Enterprise Beans 間の関係を定義します。サンプル・アプリケーションのエンティティー Bean 間には、1 対多の関連がいくつかあります。顧客は、複数の銀行口座を持つことができます。したがって、Customer エンティティー Bean は、多くの BankAccount Bean と関連付けることができます。同様に、銀行口座に対して行われた取引ごとに 1 つのレコードが存在するので、各銀行口座には、複数のトランザクション・レコードを持つことができます。したがって、BankAccount Bean は、関連する複数の TranRecord Bean を持つことができます。

サンプル・アプリケーションでは、移植性を高めるために、これらの関連は手作業で定義されています。 (VisualAge for Java の EJB ツールでも、CMP エンティティー Bean 間の関連を定義できます。) Java Network Directory Interface (JNDI) を使用すると、メソッドは、関連するエンティティー Bean のホーム・インターフェースが一度検索されます。後続の検索要求で使用するために、この情報が保存されます。

この例では、Customer Bean が多くの BankAccounts Bean と関連しています。Customer エンティティー Bean には、さらに getBankAccounts および getBankAccountHome の 2 つのメソッドがあります。これらのメソッドは、以下のように使用されます。

- getBankAccountHome メソッドは、JNDI 検索を行い、BankAccount Bean のホームへの参照を戻します。
- getBankAccounts メソッドは、BankAccount オブジェクトの個々のインスタンスの列挙を戻します。BankAccount ホーム・インターフェースの custom

finder メソッドを呼び出し、Customer Bean の基本キー情報を渡すことによって、この操作を行います。以下に例を示します。

```
findBankAccountsByCustomerId(((CustomerKey)
     entityContext.getPrimaryKey()).customerId)
```

エンティティー Bean 間の関連に関する詳細については、 54ページの『関連』を参照してください。

配置

Enterprise Beans は、EJB サーバーのコンテナーに配置されます。コンテナーは、トランザクションを管理し、サーバーとのすべての対話およびパーシスタンス・メカニズムを処理します。

Enterprise Beans、Encina ブリッジ・サーバー、および Encina++

サンプル・アプリケーションの Encina++ のインプリメント (171ページの『Enterprise Beans と TXSeries Encina++ を使用したエンタープライズ版のインプリメント』で説明) は、Enterprise Beans、Encina ブリッジ・サーバー、および Encina++ サーバーから構成されます。本節では、これらのコンポーネントおよびインターフェースについて説明します。

Enterprise Beans

サンプル・アプリケーションのエンタープライズ版 Application Server/Encina++ インプリメントの Enterprise Beans は、サンプル・アプリケーションのアドバンスド版 Application Server および Component Broker インプリメントの Enterprise Beans と類似しています。この両方のインプリメントは、以下を使用します。

- セッション Bean BankTasks。ビジネス・ロジックを含みます。
- 3 つのエンティティー Bean Customer、BankAccount、および TranRecord のセット。銀行取引操作で呼び出されるエンティティーを表します。
- アクセス Bean。各 Enterprise Bean をラップします。

Enterprise Beans およびアクセス Bean のインプリメント方法に関する詳細については、 190ページの『Enterprise Beans (アドバンスド版およびエンタープライズ版 /Component Broker)』を参照してください。

サンプル・アプリケーションのエンティティー Bean インプリメント間の主な違いは、パーシスタンスの管理方法にあります。Encina++ Enterprise Bean のインプリメントでは、BMP を持つエンティティー Bean を使用します。アドバ

ンスド版および Component Broker Enterprise Bean のインプリメントでは、CMP を持つエンティティー Bean を使用します。

図46 に、パーシスタンスの Encina++ インプリメントを示します。

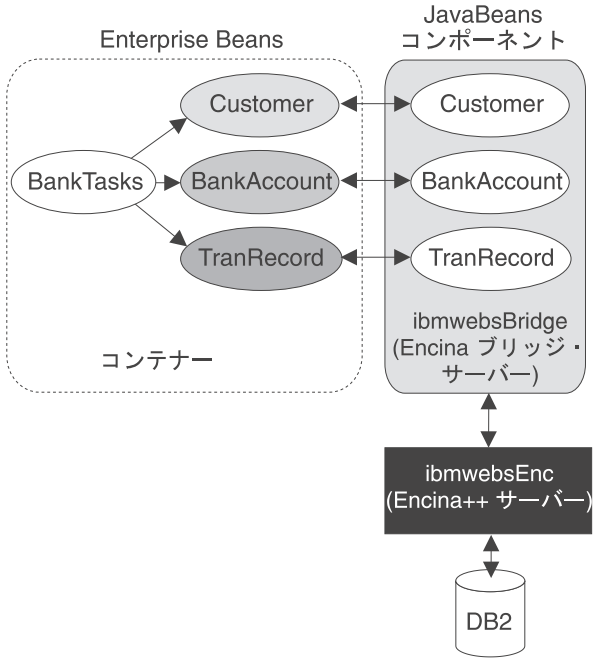

図 46. パーシスタンスの管理

エンティティー Bean は、永続データへのアクセス要求に対して、Encina ブリッジ・サーバー上に常駐している JavaBeans コンポーネントを参照します。JavaBeans コンポーネントには、Encina++ サーバーで実行可能な操作に対するメソッドが含まれます。ブリッジ・サーバーは、トランザクションで DB2 データベースを更新する Encina++ サーバーにパーシスタンス要求を渡します。

エンティティー Bean メソッドは、Encina++ アプリケーションのメソッドに対応しています。これらのメソッドは、トランザクション・インターフェース定義言語 (TIDL) ファイルで定義されています。Encina ブリッジ・サーバーを構成する JavaBeans コンポーネントおよびファイルは、**wstidl** コマンドによって生成されます。

wstidl の使用によるインターフェースの定義

Enterprise Beans にカプセル化されたビジネス・ロジック・メソッドは、最初は TIDL で作成されます。(一般的に、Encina または Encina++ アプリケーション用にもともと作成された TIDL ファイルは、Java アプリケーションで対応するインターフェースを定義するために使用されます。)**wstidl** コマンドは、TIDL ファイルで定義されたインターフェースから以下を生成します。

- ホーム・インターフェースおよびブリッジ・オブジェクト・インターフェースを含む CORBA IDL ファイル。ホーム・インターフェースは、Encina++ バインディング・タイプに対応する特定の create メソッドを使用して、オブジェクトを作成し、戻します。ブリッジ・オブジェクト・インターフェースおよびメソッドは、TIDL インターフェースおよび操作に対応します。
- C++ ホームおよびブリッジ・オブジェクトのインプリメント (C++ ブリッジ層)。この層は、Encina++ アプリケーションと通信します。
- JavaBeans コンポーネント (Java ブリッジ層)。これは、CORBA ホーム・インターフェースおよびブリッジ・オブジェクト・インターフェースのクライアント・サイドの使用をカプセル化します。エンティティー Bean は、パーシスタンス要求に対して、これらのコンポーネントを参照します。 (**wstidl** コマンドは、オプションで、Enterprise Bean インプリメントを生成して、クライアント・サイドの Java ブリッジ層として機能します。)
- 追加の Java クラス。操作の出力パラメーターと戻り値およびアプリケーション例外を処理します。
- ブリッジ・サーバーのメイン関数。

エンティティー Bean インターフェースは、対応する TIDL ファイルに手作業でマッピングされています。TIDL ファイルはインターフェースのセットを 1 つしか含むことができないので、TIDL ファイルは、各エンティティー Bean と関連付けられます。表3 に、サンプル・アプリケーションで使用される TIDL 入力ファイルのリストを示します。

表 3. エンティティー Bean の wstidl ファイル

エンティティー Bean	TIDL ファイル
BankAccount	ibmwebsAcctIf.tidl
Customer	ibmwebsCustIf.tidl
TranRecord	ibmwebsTranIf.tidl

TIDL ファイル以外にも、インターフェース定義言語 (IDL) ファイル common.idl に、インターフェースのデータ構造を定義します。これは、 表3 にリストされた TIDL ファイルに含まれます。

図47 に、TranRecord エンティティー Bean のインターフェース定義を含む TIDL ファイル ibmwebsTranIf.tidl を示します。

```
/* WebSphere Family Sample TIDL file
ibmwebsTranInterface */
[
  uuid(c7cc49d0-572f-11d3-9da4-0008c7b2e356),
  version(1.0)
]
interface ibmwebsTranIf
{
  import "common.idl";
 /* get TransactionRecord given transaction id */
 /* this should cover all get TransactionRecord EJB get methods */
 [nontransactional] void getTranRec
 (
   [in] estring_t tranId,
   [out] transactionRecord_t *txRecord
 );
 /* get all transactions of all banking accounts of the
  * specified customer using customer id */
 [nontransactional] void getAllTR
 (
   [in] estring_t customerId,
   [out] transactionRecordArray_t *arrayP
 );
 /* create new transaction record if txRecord.id == null
  * otherwise update other fields */
 /* this should cover all TransactionRecord EJB set and add methods */
 [transactional] void updateTR
 (
   [in, out] transactionRecord_t *txRecord
 );
 /* remove transaction given tranId */
 [transactional] void removeTR
 (
   [in] estring_t tranId
 );
}
```

図47. コード・サンプル: TranRecord エンティティー Bean のインターフェースを含む TIDL ファイル

この TIDL ファイルは、Encina++ サーバーとの通信に使用される 4 つのインターフェースを定義します。

- getTranRec － トランザクション・レコードをデータベースから取得します。
- getAllTR － 顧客の口座に関連するすべてのトランザクション・レコードを取得します。

- updateTR － 新しいトランザクション・レコードを作成したり、既存のレコードを更新したりします。このメソッドは (TIDL ファイル内の [transactional] 指定によって示される) トランザクションのコンテキスト内で実行されることに注意してください。
- removeTR － トランザクションを削除します。このメソッドも、トランザクションのコンテキスト内で実行されます。

これらの各インターフェースは、TranRecord エンティティー Bean および Encina++ サーバーで定義されたメソッドに対応します。TranRecord エンティティー Bean は、これらのインターフェースをインプリメントします。これは、永続データの取得または更新の要求に対して、TIDL ファイルから生成された JavaBeans コンポーネントを参照します。Encina ブリッジ・サーバーは、TIDL ファイルで定義されたメソッドを実行する Encina++ アプリケーション・サーバーにこの要求を渡します。Java 順次化可能ローカル・クラスは、TIDL 呼び出しが 1 回だけ必要になるリモート呼び出しを避けるために使用されます。

以下の **wstidl** コマンドは、ブリッジ・サーバー用のファイルを生成するために使用されます。

```
wstidl -Ic:¥opt¥encina¥include -main ¥
  -module CustomerIf -javabean  ¥
  -pkgPrefix com.ibm.ibmwebs.ibmwebsCustIf.tidl
```

各 TIDL ファイルは、このコマンドでコンパイルされます。適切なファイル名およびモジュールで置換してください。-javabean オプションを使用すると、**wstidl** は JavaBeans コンポーネントをクライアント層として生成します。ブリッジ・サーバーにはメイン関数を 1 つだけ生成する必要があるので、最初の TIDL ファイルにのみ -main オプションを使用します。

common.idl ファイルは、対応するヘッダー・ファイルを生成するために、IDL コンパイラーで処理されます。生成されたヘッダー・ファイルは、ブリッジ・サーバーを生成するために、他の生成されたファイルとリンクされます。

Encina ブリッジ・サーバー

Encina ブリッジ・サーバー ibmwebsBridge は、実際には、Java ベースのサンプル・アプリケーションと Encina++/DCE アプリケーション間のブリッジとして機能する特化したモニター・アプリケーション・サーバーです。これは、**wstidl** コマンドによって生成されたファイルから構築されます。生成されたメイン関数は、TIDL インターフェースのヘッダー・ファイルを含み、インターフェースを初期化します。ブリッジ・サーバーは複数のインターフェースをエクスポートするので、3 つの TIDL インターフェースおよび common.idl イン

ターフェースすべてのヘッダー・ファイルを含むように、ファイルを編集しなければなりません。各インターフェースの初期化も行う必要があります。

Encina++ サーバー

Encina++ サーバー ibmwebsEnc は、「Encina Server (Encina サーバー)」ウィザードによって作成されます。このウィザードは、TXSeries アプリケーション開発キットの一部です。これには、サンプル・アプリケーション用に定義された 3 つの TIDL ファイルすべてが組み込まれます (196ページの表3 を参照してください)。

ibmwebsEnc サーバーは、DB2 データベースにトランザクション・アクセスを行います。エンティティー Bean からのデータ要求は、JavaBeans コンポーネントおよびブリッジ・サーバーを介して Encina++ サーバーにパススルーされます。Encina++ サーバーは、組み込み構造化照会言語 (SQL) ステートメントを使用して、データベースを照会して必要に応じて更新します。各 SQL ステートメント (複数の結合を含む場合もあります) は、エンティティー Bean メソッドにほぼ対応しています。

トランザクションの管理

トランザクションは、エンティティー Bean によって開始され、コンテナーによって管理されます。コンテナーは、Encina ブリッジ・サーバーを処理して、Encina++ アプリケーション・サーバーを呼び出すトランザクションを調整します。トランザクションは、ブリッジ・サーバーをパススルーして、Encina++ アプリケーション・サーバーに制御されているバックエンド・リソースにアクセスします。Encina++ サーバーのトランザクション・メソッドは、TIDL ファイルで識別されます。

配置

Enterprise Beans は、デプロイメント・ディスクリプター TX_MANDATORY とともに配置されます。 (アプリケーションがトランザクションを管理する方法については、 『トランザクションの管理』を参照してください。) インプリメントでは BMP を持つエンティティー Bean を使用するので、コンテナーはパーシスタンスを処理しません。

Component Broker 管理下のオブジェクト

Web サイトのビジネス・ロジックは、分散オブジェクト・サーバーで実行されている Component Broker 管理下のオブジェクトに常駐しています。
Component Broker 分散オブジェクト・サーバーは、サーブレットが実行されるアドバンスド版 Application Server プロセスと同じマシンで稼働させること

も、異なるマシンで稼働させることもできます。サーブレットは、ビジネス・ロジックに対する顧客として機能し、Component Broker のアプリケーション・オブジェクト (AO) でメソッドを呼び出します。アプリケーション・オブジェクトは作業単位を監視し、銀行取引エンティティーを表す Component Broker ビジネス・オブジェクトを操作します。

Java ビジネス・オブジェクトは、永続的ストレージ、つまり DB2 データベースにバックアップされます。アプリケーション・オブジェクトなどのクライアントが、これらのビジネス・オブジェクトを操作すると、オブジェクトが活動化されます。これによって、データが永続的ストレージからオブジェクトにロードされます。作業単位がコミットされると、データへの変更が永続的ストレージに書き込まれます。

アプリケーション・オブジェクトのインプリメント

アプリケーション・オブジェクト BankTasks には、顧客が要求するほとんどのビジネス・ロジックが含まれます。これには、いくつかのメソッドが含まれます。

- lookupCustomer － 顧客に関する情報を戻します。
- lookupAccounts － 顧客の口座のリストを戻します。
- lookupHistory － 指定された口座の取引履歴を戻します。
- deposit － 口座に振り込みを行い、その取引履歴を更新します。このメソッドは、Web サイトでは使用できません。
- withdraw － 口座から引き出しを行い、その取引履歴を更新します。このメソッドは、Web サイトでは使用できません。
- transfer － 口座間で送金して、その取引履歴を更新します。

アプリケーション・オブジェクトは、トランザクションをサポートするコンテナーに配置されます。メソッドが呼び出されると、コンテナーは新しいトランザクションを開始し、ビジネス・ロジックを実行して、終了したらトランザクションをコミットします。

ビジネス・オブジェクトのインプリメント

3 つのビジネス・オブジェクト (Customer、BankAccount、および TranRecord) は、銀行の異なるエンティティーを表します。Customer および TranRecord エンティティーは、使用しているデータベース内の行を直接更新します。これらには、追加のビジネス・ロジックは含まれません。BankAccount エンティティー Bean も、使用しているデータベース内の行を更新します。ただし、これに

は、預金を行ったり、口座から引き出すためのメソッドが含まれます。これは、0 またはマイナスの金額である場合、あるいは残高不足で引き出しができない場合にはエラーを送出します。

注: アプリケーション・オブジェクト BankTasks の withdraw メソッドとビジネス・オブジェクト BankAccount の withdraw メソッド間の違いは、アプリケーション・オブジェクトは、口座に対する変更のトランザクション・レコードも、TranRecords Bean を介して生成する点です。

データ・オブジェクトのインプリメント

データ・オブジェクトは、ビジネス・オブジェクト Customer、BankAccount、および TranRecord のパーシスタンスを処理します。データ・オブジェクトは、Component Broker のキャッシュ・サポートを使用して、基礎となるデータ・ストアと通信する永続オブジェクトです。キャッシュ・サポートは、最適なロックを使用します。このサポートによって、データ・オブジェクトを使用するビジネス・オブジェクトが活動化または非活性化されたときに、既存のデータを再利用することでデータ・オブジェクトにアクセスする速度が速くなります。

管理下のオブジェクト間の関連

さまざまなオブジェクト間には、1 対多の関連がいくつかあります。顧客は、銀行口座を多数持つことができます。したがって、Customer ビジネス・オブジェクトは、複数の BankAccount ビジネス・オブジェクトと関連付けることができます。これらの関連は、Object Builder を使用することで定義されます。関連するオブジェクトを検索するのに必要なコードは、すべて Object Builder によって自動的に生成されます。

CopyHelper オブジェクトおよび管理下のオブジェクト

Component Broker は、LocalOnly オブジェクトを作成することができます。これらのオブジェクトをバイト・ストリームに直列化し、同じバイト・ストリームからオブジェクトを再作成することができます。Component Broker は、LocalOnly オブジェクトである CopyHelper オブジェクトのフォームを使用して、オブジェクトの作成時に、複数の属性をクライアントからサーバーに渡します。

CopyHelper オブジェクトの構造がバイト・シーケンスに直列化されると、他のオブジェクトおよびアプリケーションの他の部分と簡単に通信することができます。顧客が `create` メソッドを複数の属性とともに呼び出すと、CopyHelper オブジェクトが作成されます。この直列化されたバイト・ストリングは、createFromCopyHelper メソッド呼び出しで渡されます。

LocalOnly オブジェクトは、汎用化された CopyHelper オブジェクトです。アクセス Bean に類似した機能を行うために、LocalOnly オブジェクトがビジネス・オブジェクトごとに作成され、これらの状態が保持されます。さらに、リモート Java ビジネス・オブジェクトに情報をコピーしたり、このオブジェクトから情報をコピーするために、2 つのメソッド (copyFromJBO および copyToJBO) がローカル・ビジネス・オブジェクトに追加されています。

LocalOnly オブジェクトに関する詳細については、 176ページの『CopyHelper オブジェクトおよびアクセス Bean』を参照してください。

配置

Component Broker は、コマンド行メカニズムを使用することによって、オブジェクトをサーバーに配置することができます。コマンド行メカニズムを使用するスクリプトは、サンプル・アプリケーションのビジネス・オブジェクトを Component Broker 内でロードして構成することができます。このスクリプトは、ビジネス・オブジェクトを含むアプリケーションを定義およびロードして、サーバーを定義し、ロードされたアプリケーションに合わせてビジネス・オブジェクトを構成します。

第14章 サンプル・アプリケーションの拡張

本節では、サンプル・アプリケーションの拡張方法について説明します。本節は、他の IBM 製品および WebSphere プロダクトを、WebSphere Application Server を用いて使用する方法の手引きとなるものであって、拡張機能をすべてリストしたり、それらの拡張機能のインプリメント方法を詳細に解説するものではありません。

本節では、以下のトピックについて説明します。

- 『Component Broker の管理下のオブジェクトの他のシステムへの接続』
- 205ページの『Java アプリケーションの MQSeries への接続』
- 207ページの『サンプル・アプリケーションでの WebSphere Edge Server の使用』

Component Broker の管理下のオブジェクトの他のシステムへの接続

サンプル・アプリケーションにおける Component Broker の管理下のオブジェクトのインプリメンテーションを拡張して、Component Broker アプリケーション・アダプターを介して、リソース・マネージャーなどの他のシステムを使用することができます。アプリケーション・アダプターは、それを使用するコンポーネントから独立しています。アプリケーション・アダプターは、識別、キャッシング、およびパーシスタンスなどのサービスを提供し、特定のリソース・マネージャーへのアクセスをサポートします。Component Broker のビジネス・オブジェクトと Component Broker で配置された Enterprise Beans は、どちらもアプリケーション・アダプターを使用して、他のシステムに接続することができます。

MQSeries への接続

サンプル・アプリケーションと統合可能なリソース・マネージャーの 1 つに MQSeries があります。MQSeries は、エンタープライズ版 Application Server に組み込まれています。MQSeries は、メッセージ形式でのデータ交換のメカニズムを提供します。MQSeries を使用することにより、タイプの異なる多数のシステム間で作業を分散することができ、また、2 フェーズ・コミットの伝達が現実的ではない環境において、トランザクションの保全性を提供することができます。

MQSeries アプリケーション・アダプターにより、Component Broker アプリケーションは、MQSeries アプリケーションとの間で、メッセージを送受信することができます。MQSeries アプリケーション・アダプターは、Windows NT および Solaris プラットフォーム上の Component Broker と、それと同じホスト上の MQSeries キュー・マネージャーとの間の通信をサポートします。アプリケーション・アダプターを使用することにより、Component Broker アプリケーションは、MQSeries キューにメッセージを入れたり、そこからメッセージを取り出したりすることができます。

MQSeries アプリケーション・アダプターのフレームワークは、Component Broker アプリケーションの開発者にインターフェースの詳細を隠す、MQSeries インターフェースのラッパーを提供します。アプリケーション・アダプター・インターフェースは、データ・オブジェクトを介して操作され、ビジネス・ロジックをインプリメントするオブジェクトからの MQSeries へのアクセスをカプセル化します。

MQSeries を、サンプル・アプリケーションの Component Broker インプリメンテーション用の第 3 層リソース・マネージャーとして使用する場合は、データ・オブジェクトのインプリメンテーションを以下のように変更します。

- アプリケーションは、データベースからキャッシュ情報にアクセスする代わりに、MQSeries キューからデータを取り出す。データは、データ・オブジェクトによって処理される InboundMessage でカプセル化されます。
- 更新データは、ビジネス・オブジェクトによって処理された後、OutboundMessage オブジェクトでカプセル化され、MQSeries キューに戻される必要がある。

MQSeries アプリケーション・アダプターの詳細については、「*MQSeries アプリケーション・アダプター開発ガイド*」を参照してください。

CICS、SAP、または IMS への接続

Component Broker アプリケーションは、SAP、情報管理システム (IMS)、および CICS に接続することもできます。手続き型アプリケーション・アダプター (PAA) は、これらのシステムを、Component Broker システム用の第 3 層リソース・マネージャーとして使用するためのフレームワークを提供します。

PAA ランタイム・ライブラリーは、PAA サポートを Component Broker に統合します。PAA ランタイム・ライブラリーはまた、ホスト・オンデマンド、CICS トランザクション・ゲートウェイなどの第 3 層システムによって使用される通信テクノロジーをサポートします。

CICS、SAP、または IMS システムを、サンプル・アプリケーションの Component Broker インプリメンテーション のための第 3 層リソース・マネージャーとして使用するためには、それぞれのデータ・オブジェクトを手続きアダプター・オブジェクト (PAO) に関連付ける必要があります。PAO は、第 3 層システム内で保持されているデータをカプセル化し、作成、検索、更新、および削除 (CRUD) などのデータ・アクセス操作を使用して、データを変更することができます。

たとえば、Component Broker アプリケーションが第 3 層システムのデータを必要とする場合、PAO は、データ・オブジェクトをインスタンス化します。次に、データ・オブジェクト上で検索メソッドを呼び出します。検索メソッドにより、PAO は、トランザクションを呼び出して、データを取得します。

- CICS および IMS アプリケーションへのアクセスには、3270 ターミナルと互換性のあるデータ・ストリームを使用します。データ・フォーマットは、CICS 基本マッピング・サービス (BMS) および IMS メッセージ形式サービス (MFS) によって定義されています。これらのサービスは、3270 データ・ストリームを取り扱うためのインターフェースを提供します。PAA は、ホスト・オンデマンド、CICS 外部呼び出しインターフェース (ECI)、拡張プログラム間通信機能 (APPC) を使用して、CICS および IMS アプリケーション内のルーチンを実行することができます。
- SAP R/3 アプリケーションへのアクセスには、Component Broker ビジネス・オブジェクトへの Java ベースの SAP コネクターを介した、リモート・ファンクション・コール (RFC) を使用します。これにより、Component Broker は、SAP R/3 サーバーにリクエストを送出し、SAP RFC モジュールを実行して、データにアクセスすることができます。

VisualAge for Java、CICS/IMS コネクター (CICON) ツール、および Enterprise Access Builder を使用して、上記の第 3 層システム用の PAA コネクターを作成することができます。

CICS/IMS アプリケーション・アダプターの詳細については、「*手続き型アプリケーション・アダプター開発ガイド*」を参照してください。

Java アプリケーションの MQSeries への接続

MQSeries を使用することにより、アドバンスド版 Application Server の機能を拡張することができます。そのためには、サーブレット、アプレット、JavaBeans および EJB コンポーネント・アーキテクチャーに基づくアプリケーションなどの Java アプリケーションが、メッセージの形式でデータを交換で

きるようにする必要があります。これにより、これらの Java アプリケーションが、広範なプラットフォーム上で、MQSeries アプリケーションと通信できるようになります。

サンプル・アプリケーションが MQSeries と通信できるようにするには、サンプルのサーブレットあるいは Enterprise Beans を変更して、MQSeries キュー・マネージャーに接続し、メッセージの形式でデータ交換ができるようにします。MQSeries は、メッセージングのインプリメントに使用することができる、以下の 3 つの Java API をサポートしています。

- *Java 用 MQSeries クラス* (MQ ベース Java) *Java* バインディング・パッケージ (com.ibm.mqbind) は、MQSeries アプリケーションとの通信のための完全な Java API を提供します。Java アプリケーションは、この API を使用して、MQSeries のキュー・マネージャーに直接接続することができます。この場合、キュー・マネージャーは、アプリケーションと同じホスト上になくてはなりません。この API は、ローカル・ホスト上のキュー・マネージャーに高速で接続できますが、異なるホスト上のキュー・マネージャーへの接続には使用することができません。
- MQ ベース Java *Java* クライアント・パッケージ (com.ibm.mq) は、クライアント接続を介して、MQSeries サーバー・チャネルと通信するための完全な Java API を提供します。アプリケーションはこの API を使用して、ネットワーク接続されているあらゆるホスト上の MQSeries キュー・マネージャーと通信することができます。ただし、Java クライアント API は、Java バインディング API に比べ、ローカル・キュー・マネージャーへの接続が遅くなります。これは、Java アプリケーションの接続が直接接続ではなく、チャネルを介した接続となるためです。
- *Java* メッセージング・サービス (JMS) API は、Java プログラム言語における、メッセージング・アプリケーションをインプリメントするための標準です。この API は、移植可能で、ベンダーに依存しない API を提供します。JMS インターフェースの MQSeries のインプリメンテーションは、JMS API の基礎を成すものです。MQSeries のキューおよびキュー・マネージャーを表すオブジェクトは、ディレクトリー・ネーミング・サービス (MQSeries メッセージ・サービス) を使用することにより、JMS に定義されます。アプリケーションは JMS API を使用して、ネットワーク上にある、あらゆるホスト上の MQSeries キュー・マネージャーと接続することができます。JMS はまた、MQ ベース Java API では利用不能な、以下のような機能を提供することができます。
  - 非同期メッセージング
  - メッセージ・セレクター

– MQSeries パブリッシュ / サブスクライブ・メッセージ
– 構造化されたメッセージ・クラス

これらの 3 つの API により、Java アプリケーションは、MQSeries をクライアント・マシンにインストールすることなく、インターネットを利用するなどの方法で、MQSeries に呼び出しや照会を送信することができます。これらの API は、企業アプリケーションへのアクセス、および Web ベースのアプリケーションの開発のためのインフラストラクチャーを提供します。MQSeries のキューを、ネットワーク的に、よりユーザーに近い位置に設置することにより、システムのロードをバイパスすることができるとともに、より高速に応答を送信することができます。さらに、MQSeries メッセージングでは、バックエンド・リソースへのトランザクション・アクセスも可能になります。

WebSphere Application Server 環境では、MQSeries を使用して、統合サーバー、つまり、複数のプレゼンテーション層の統合ポイントとして機能するサーバーをインプリメントすることができます。統合サーバーにより、第 1 層のクライアントが、第 2 層および第 3 層上のアプリケーションおよびリソースを共用することができます。統合サーバーは、データベース、トランザクション処理モニター、アプリケーション、およびバックエンド・データ・ストアと通信し、企業情報が他のアプリケーションによって処理され、エンド・ユーザーに表示される前に、それらの情報を検索および操作します。柔軟性に富む MQSeries のメッセージング・フォーマットにより、MQSeries ベースの統合サーバーは、多くの異なるタイプのシステム上のさまざまなリソースと通信することができます。

MQ ベース Java および JMS API は、MQSeries SupportPac MA88 で使用することができます。これは、MQSeries のサポート Web サイトからダウンロード可能です。これらの API の使用についての詳細は、下記の資料を参照してください。

- *MQSeries Using Java*。これは、MQSeries のサポート Web サイトから参照できます。
- JMS 仕様。これは、Sun Microsystems の Java Web サイト www.java.sun.com.から参照できます。

サンプル・アプリケーションでの WebSphere Edge Server の使用

WebSphere Edge Server は、WebSphere Application Server とともに使用することができます。Edge Server は、208ページの図48 に示すように、WebSphere Application Server システムへのフロントエンドとして機能します。

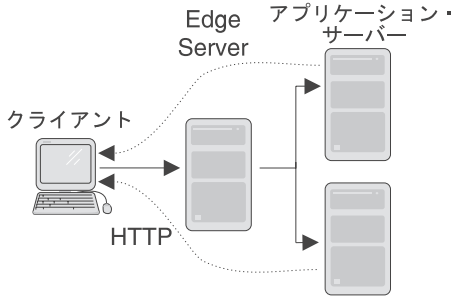

図 48. WebSphere Edge Server および Application Server

Edge Server は、サーバーによるロードのモニター、アプリケーションおよびデータベースの保全性の検証、サーバー・クラスターの作成、フェイルオーバー・サポートの提供、Web サーバーからの静的コンテンツのオフロードを行うことができます。

以下に述べる Edge Server のコンポーネントは、両方とも、サンプル・アプリケーションなどの WebSphere アプリケーションで使用することができます。

- キャッシング・プロキシー・コンポーネントは、エンド・ユーザーからのデータの要求を代行受信し、要求された情報を検索して、検索した情報をエンド・ユーザーに戻します。ほとんどの場合、要求は、Web サーバー上に保管された文書に対するものであり、HTTP 経由で送達されます。キャッシング・プロキシーは、サンプルの静的な Web ページのキャッシュに使用され、それらの Web ページのより高速なロードを可能にします。
- ネットワーク・ディスパッチャーは、データの要求を代行受信し、その時点において、要求の処理に最適なサーバー・マシンに、それらのデータを転送します。ネットワーク・ディスパッチャーは、同じタイプの要求を処理するマシン群に着信要求を配布します。ネットワーク・ディスパッチャーは、Web サーバーのロード・バランサーとして使用することができます。たとえば、サンプル・アプリケーションを複数のサーバーに複製する場合などに有効です。ネットワーク・ディスパッチャーは、異なるマシン上に均等に要求を分散し、クライアントとアプリケーション・サーバー間の信頼性のある接続を提供します。

Edge Server の使用に関する詳細は、プロダクト資料を参照してください。

特記事項

本書はアメリカ合衆国で提供されている製品およびサービス用に作成されたものであり、本書に記載の製品、サービス、またはフィーチャーが日本においては提供されていない場合があります。日本で利用可能な製品、サービス、およびフィーチャーについては、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の 知的所有権を侵害することのない、機能的に同等な製品、プログラム、またはサービスを使用することができます。ただし、IBM 製以外の製品と組み合わせた場合、その操作の評価と検証については、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む。) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権の許諾については、下記の宛先に、書面にてご照会ください。

〒106-0032 東京都港区六本木 3 丁目 2-31
AP 事業所
IBM World Trade Asia Corporation
Intellectual Property Law & Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。

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

本書は定期的に見直され、必要な変更 (たとえば、技術的に不適確な表現や誤植など) は、本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するもので

はありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

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

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

Component Broker については、
IBM Corporation
Department LZKS
11400 Burnet Road
Austin, TX 78758
U.S.A.

TXSeries については、
IBM Corporation
ATTN: Software Licensing
11 Stanwix Street
Pittsburgh, PA 15222
U.S.A.

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

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

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

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。また、IBM 以外の製品に関するパフォーマンスの正確性、互換性、またはその他の要求は確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

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

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

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は現れない場合があります。

商標

以下は、IBM Corporation の米国およびその他の国における商標です。

Advanced Peer-to-Peer Networking
AFS
AIX
APPN
AS/400
CICS
CICS OS/2
CICS/400
CICS/6000
CICS/ESA
CICS/MVS
CICS/VSE
CICSPlex
DB2
DCE Encina Lightweight Client
DFS
Encina
IBM
IBM System Application Architecture
IMS
IMS/ESA
Language Environment
MQSeries
MVS/ESA
NetView
Open Class
OS/2
OS/390
OS/400
Parallel Sysplex
PowerPC
RACF
RAMAO
RMF
RISC System/6000
RS/6000
S/390
SecureWay
TeamConnection
Transarc
TXSeries
VSE/ESA
VTAM
VisualAge
WebSphere

Domino、Lotus、および LotusScript は、Lotus Development Corporation の米国およびその他の国における商標または登録商標です。

Tivoli は、Tivoli Systems, Inc. の米国およびその他の国における登録商標です。

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

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

UNIX は、The Open Group がライセンスしている米国およびその他の国における登録商標です。

本書の一部は、以下の著作権表示を持つ Object Management Group からの資料を基にして作成されています。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

[ワ行]

[数字]

A

B

C

D

E

I

J

L

M

N

O

P

R

S

T

V

W

X

Printed in Japan

SD88-7362-01

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