

WebSphere® Application Server



WebSphere ビジネス・ソリューションの構築

バージョン 4.0

WebSphere® Application Server



WebSphere ビジネス・ソリューションの構築

バージョン 4.0

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、171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、SD88-7362-01 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： SC09-4432-03
WebSphere® Application Server
Building Business Solutions with WebSphere
Version 4.0

発 行： 日本アイ・ピー・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 2001.7

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

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

© Copyright International Business Machines Corporation 2000, 2001. All rights reserved.

Translation: © Copyright IBM Japan 2001

目次

図	vii	サブレットのライフ・サイクル	29
表	ix	サブレットのランタイム環境	30
本書について	xi	サブレット・コンテナ	30
本書の対象読者	xi	サブレット・キュー	30
本書の構成	xi	サブレット・グループ	31
関連情報	xii	類似性	31
本書で使用されている規則	xiii	サブレット・セッションの管理	31
		サブレットの長所	32
第1部 WebSphere Application Server の理解	1	第4章 エンタープライズ Bean の使用	35
第1章 WebSphere Application Server アドバンスド版	5	EJB 環境	35
WebSphere Application Server アドバンスド・シングル・サーバー版および WebSphere Application Server アドバンスド・デベロッパー版	5	クライアント・ビュー	35
機能	6	EJB サーバー環境	36
実行時およびシステム管理アーキテクチャー	7	コンテナ	36
アプリケーション開発環境	9	エンタープライズ Bean アプリケーションの開発	38
WebSphere Application Server アドバンスド版	11	エンタープライズ Bean のタイプ	38
アドバンスド版 Application Server の機能	11	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 Bean の使用	39
実行時およびシステム管理アーキテクチャー	11	開発チームの役割	40
オブジェクト・サービスおよびサポート・サービス	15	アプリケーション開発プロセス	41
アドバンスド版アプリケーション開発環境	16	セッション Bean	42
第2章 JavaServer Pages の使用	19	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	42
JSP ページにおける処理	20	セッション Bean コンポーネント	43
JSP ページの呼び出し	21	セッション Bean のライフ・サイクル	45
HTML 文書における Java コードの使用	22	エンティティ Bean	46
JSP タグの使用	23	Bean 管理のパーシスタンス (BMP)	46
アプリケーションにおける JSP ページの使用	23	コンテナ管理のパーシスタンス (CMP)	46
JSP ページの長所	25	エンティティ Bean のライフ・サイクル	49
第3章 サブレットの使用	27	EJB 仕様に対する機能拡張	50
サブレットのプログラミング・モデル	27	アクセス Bean	50
サブレット API	28	継承	51
		関連	55
		オブジェクト・サービス	56
		ネーミング・サービスとディレクトリー・サービス	56
		セキュリティ・サービス	59
		パーシスタンス・サービス	63
		トランザクション・サービス	68

第5章 Web アプリケーションの開発	75
Web アプリケーション・プログラミング・モデル	75
第1階層	76
第2階層	76
第3階層	77
Web アプリケーションにおける JSP ページ、サーブレット、およびエンタープライズ Bean の使用	77
Model-View-Controller アーキテクチャーのインプリメント	77
Web アプリケーションの状態情報の保持	79
Web アプリケーションでのセキュリティのインプリメント	81
第6章 WebSphere Application Server エンタープライズ版	85
エンタープライズ版 Application Server の機能	85
TXSeries	85
エンタープライズ版サービス	87
実行時およびシステム管理アーキテクチャー	87
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	89
WebSphere Studio	89
IBM VisualAge for Java エンタープライズ版	89
IBM Enterprise Access Builder (EAB)	90
IBM TeamConnection®	90
VisualAge コンポーネント開発ツールキット	90
VisualAge for C++ プロフェッショナル版	90
IBM DB2®	91
MQSeries	91
第7章 TXSeries の使用	93
TXSeries Encina	93
Encina モニター	94
リカバリー可能キューイング・サービス (RQS)	99
構造化ファイル・サーバー (SFS)	100
対等通信 (PPC) サービス	102
Encina++	104
Encina ツールキット	107
DCE-Encina Lightweight クライアント (DE-Light)	109

WebSphere アドバンスド版と Encina とのインターオペラビリティ	110
TXSeries CICS	112
CICS の基本概念	112
CICS アプリケーション・プログラミング・インターフェース	117
CICS システム間通信	121
CICS の管理	124
CICS ワークロード管理	125

第2部 WebSphere Application Server の使用 **127**

第8章 サンプル・アプリケーションの概要	129
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	129
サンプル・アプリケーションの目的	130
第9章 サンプル・アプリケーションの設計	133
アプリケーション設計	133
クライアント / サーバーの関係	134
モデル・ビュー・コントローラー・アーキテクチャー	134
オブジェクト・モデル	135
データ・モデル	137
第10章 サンプル・アプリケーションのインプリメント	139
エンタープライズ Bean を使用したアドバンスド版のインプリメント	140
エンタープライズ Bean 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	141
サンプル・アプリケーション・プラットフォーム	143
第11章 サンプル・アプリケーションの技術的な詳細	145
Web サイト	145
Web サイトの設計	145
クライアントの妥当性検査およびバックエンド処理	146
サーブレット	147
JavaServer Pages	150
WebCommand	152
WebCommand 構造	152

アクセス Bean との対話	154	Encina++ サーバー	165
エンタープライズ Bean (アドバンスド版)	155	トランザクションの管理	165
セッション Bean のインプリメント	156	配置	165
エンティティ Bean のインプリメント	156	第12章 サンプル・アプリケーションの拡張 167	
アクセス Bean のインプリメント	157	Java アプリケーションの MQSeries への接続	167
エンタープライズ Bean 間の関連	158	サンプル・アプリケーションでの WebSphere	
配置	159	Edge Server の使用	169
エンタープライズ Bean、Encina ブリッジ・		特記事項	171
サーバー、および Encina++	159	商標	173
エンタープライズ Bean	159	索引	177
wstidl の使用によるインターフェースの定			
義	161		
Encina ブリッジ・サーバー	164		



1. WebSphere Application Server アドバンスド版ランタイム・アーキテクチャー	7	23. 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	111
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・アーキテクチャー	12	24. CICS 領域	113
3. アドバンスド版 Application Server システムの一般的なノード	13	25. 複数の CICS クライアントと CICS 領域との間の通信	123
4. JSP プログラムのフロー	20	26. サンプル・アプリケーション	134
5. ブラウザーからの JSP ページの要求	21	27. オブジェクト・モデル	136
6. サブレットからの JSP ページの要求	22	28. アドバンスド版 Application Server でのエンタープライズ Bean のインプリメント	140
7. サブレットの実行モデル	28	29. Encina++ を使用したエンタープライズ Bean のインプリメント	142
8. サブレットの基本フロー	30	30. コード・サンプル: サブレット・ユーティリティ・クラス	148
9. EJB 環境のクライアント・ビュー	36	31. コード・サンプル:エンタープライズ Bean 固有のコマンド Bean のロード	149
10. クライアントとエンタープライズ Bean との対話	37	32. コード・サンプル: <BEAN> タグの使用	151
11. セッション Bean とエンティティ Bean を使用したアプリケーションの構築	39	33. WebCommand 継承構造	153
12. エンタープライズ Bean アプリケーション開発プロセス	42	34. コード・サンプル: アクセス Bean の WebCommand との使用	155
13. WebSphere Application Server のオブジェクト・サービス・システム	58	35. コード・サンプル: エンティティ Bean の CopyHelper ラッパー	157
14. EJB メソッドの許可ベースの保護	63	36. コード・サンプル: 行セット・アクセス Bean の使用	158
15. CCF アーキテクチャー	66	37. パーススタンスの管理	160
16. OTS トランザクション・モデル	72	38. コード・サンプル: TranRecord エンティティ Bean のインターフェースを含む TIDL ファイル	163
17. Web アプリケーション・コンポーネント	76	39. WebSphere Edge Server および Application Server	169
18. デジタル証明書による認証	82		
19. エンタープライズ版 Application Server のインターオペラビリティ	88		
20. モニター・セルの物理アーキテクチャー	96		
21. PPC 通信モデル	103		
22. DE-Light Java クライアント	110		

表

- | | | | |
|---------------------------|------|--|-----|
| 1. 本書で使用する規約 | xiii | 3. エンティティ Bean の wstidl ファイル | 162 |
| 2. 親と子のインターフェース | 52 | | |

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 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 のさまざまな版と、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 アドバンスド版』では、アドバンスド版 Application Server の 3 つのバージョンである、WebSphere Application Server アドバンスド・シングル・サーバー版、WebSphere Application Server アドバンスド・デベロッパー版および 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2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3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4章 エンタープライズ Bean 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5章 Web アプリケーションの開発』では、JSP ページ、エンタープライズ Bean、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6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について説明します。
- 93ページの『第7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。

- 129ページの『第8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。
- 133ページの『第9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。
- 139ページの『第10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。
- 145ページの『第11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。
- 167ページの『第12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
- *WebSphere* エンタープライズ Bean の作成
- エンタープライズ版 InfoCenter

- 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、以下の印刷上の規則およびキーの規則を使用しています。

表 1. 本書で使用する規約

規約	意味
太字 (Bold)	コマンド名を表します。グラフィカル・ユーザー・インターフェース (GUI) が対象のときには、太字は、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 (Monospace)	コマンド、関数、リソース定義属性、およびリソース定義値などの示されたとおりに使用しなければならない値を示します。モノスペースは、画面テキストおよびコード例も示します。
イタリック (<i>Italics</i>)	指定しなければならない可変値を示します (たとえば、 <i>fileName</i> にはファイル名を指定します)。イタリックは、強調するものや資料名も示します。
Ctrl- <i>x</i>	<i>x</i> はキーの名前で、制御文字シーケンスを示します。たとえば、Ctrl-c は、Ctrl キーを押したまま、c キーを押します。
リターン	Return という表示のキー、Enter という表示のキー、または左矢印のラベルが付いたキーを指します。
%	ルート特権を必要としないコマンドの UNIX コマンド・シェル・プロンプトを示します。
#	ルート特権が必要なコマンドの UNIX コマンド・シェル・プロンプトを示します。
C:¥>	Windows NT [®] のコマンド・プロンプトを表します。

表 1. 本書で使用する規約 (続き)

規約	意味
>	<p>メニュー、メニュー項目の選択を表すときに使用します。たとえば、「ファイル (File)」>「新規 (New)」の順にクリックします」は、「ファイル (File)」メニューの「新規 (New)」コマンドをクリックする」ことを意味します。</p> <hr/> <p>ツリー・ビュー、一連のフォルダーまたはオブジェクトの展開を表すときに使用します。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)」の順に展開します」は、以下のことを意味します。</p> <ol style="list-style-type: none"> 1. 「管理ゾーン (Management Zones)」フォルダーを展開する 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する 3. 「構成 (Configurations)」フォルダーを展開する <p>注: ビューの中で正符号 (+) が付いているオブジェクトは展開することができます。オブジェクトを展開すると、正符号は負符号 (-) に変わります。</p>
+	<p>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。展開する場合には、オブジェクトの左側の正符号 (+) をクリックします。</p>
-	<p>ツリー構造の分岐を縮小することを表します。分岐を縮小すると、その分岐内のオブジェクトは非表示になります。ツリー構造の分岐を縮小する場合には、分岐の先頭にあるオブジェクトの負符号 (-) をクリックします。</p>
コマンド入力	<p>コマンドを「入力」する、または「発行」すると指示された場合は、コマンドを入力してからリターンを押します。たとえば、「ls コマンドを入力してください」という指示は、コマンド・プロンプトで ls を入力し、リターンを押すという意味になります。</p>
[]	<p>構文記述内のオプション項目を囲みます。</p>
{ }	<p>構文記述内で項目を選択しなければならないリストを囲みます。</p>
	<p>構文記述内で中括弧 ({ }) で囲まれた選択項目リスト内の項目を分けます。</p>
...	<p>構文記述内の省略記号は、前の項目を 1 回または複数回繰り返すことができることを示します。例の中の省略記号は、簡潔にするために例から情報を省略したことを示します。</p>
IN	<p>関数の説明の中で、関数にデータを渡すために値が使用されるパラメーターを示します。これらのパラメーターは、変更データを呼び出しルーチンに戻すには使用しません。(ユーザーのコードに IN 宣言は組み込まないでください。)</p>

表 1. 本書で使用する規約 (続き)

規約	意味
OUT	関数の説明の中で、呼び出しルーチンに変更データを戻すために値が使用されるパラメーターを示します。これらのパラメーターは、データを呼び出しルーチンに渡すのには使用しません。(ユーザーのコードに OUT 宣言は組み込まないでください。)
INOUT	関数の説明の中で、関数に渡され、関数より変更され、呼び出しルーチンに戻されるために値が使用されるパラメーターを示します。これらのパラメーターは、IN パラメーターおよび OUT パラメーターの両方として機能します。(ユーザーのコードに INOUT 宣言は組み込まないでください。)
\$CICS	CICS 製品がインストールされている場所の絶対パス名を表します。たとえば、Windows NT の場合には C:\opt\cics、Solaris の場合には /opt/cics になります。CICS という名前の環境変数が製品のパス名に設定されている場合、表示どおりに例を使用することができます。そうでない場合には、\$CICS のインスタンスすべてを CICS 製品のパス名に置き換えなければなりません。
オープン・システム CICS	サポートされるすべての UNIX プラットフォームで稼働する CICS 製品の総称です。
TXSeries CICS	CICS for AIX、CICS for Solaris、および CICS for Windows NT 製品の総称です。
CICS	オープン・システム CICS と CICS (for Windows NT) 製品の総称です。特定バージョンのオープン・システム CICS 製品に言及している場合は、オープン・システム CICS 間の相違点を強調しています。CICS ファミリー中の他の CICS 製品は、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 などです。

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・ストアをサポートし、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
- 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・サーバー上に配置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・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を提供し、非 WebSphere 環境とのインターオペラビリティを実現します。また、WebSphere Application Server 間のインターオペラビリティにより、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配置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)、およびエンタープライズ Bean 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

(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・アーキテクチャなどが含まれます。

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。本節で説明するトピックは以下のとおりです。

- 5ページの『第1章 WebSphere Application Server アドバンスド版』
- 19ページの『第2章 JavaServer Pages の使用』
- 27ページの『第3章 サブレットの使用』
- 35ページの『第4章 エンタープライズ Bean の使用』
- 75ページの『第5章 Web アプリケーションの開発』
- 85ページの『第6章 WebSphere Application Server エンタープライズ版』
- 93ページの『第7章 TXSeries の使用』

127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 アドバンスド版

WebSphere Application Server アドバンスド版は、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 アドバンスド版には、WebSphere Application Server アドバンスド・シングル・サーバー版、WebSphere Application Server アドバンスド・デベロッパー版および WebSphere Application Server アドバンスド版の 3 つの版があります。アドバンスド版 Application Server のそれぞれの版の詳細については、『WebSphere Application Server アドバンスド・シングル・サーバー版および WebSphere Application Server アドバンスド・デベロッパー版』と 11 ページの『WebSphere Application Server アドバンスド版』を参照してください。

WebSphere Application Server アドバンスド・シングル・サーバー版および WebSphere Application Server アドバンスド・デベロッパー版

WebSphere Application Server アドバンスド・シングル・サーバー版および WebSphere Application Server アドバンスド・デベロッパー版は、共にサブレットなどのアクティブな Web サイトや Web アプリケーションの作成に使用されます。これら 2 つの版は、サーバー・サイドのビジネス・アプリケーションの移植性と Java テクノロジーのパフォーマンスと管理の容易性を結合し、Java ベースの Web アプリケーションを設計するための包括的なプラットフォームを提供します。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、ランタイム環境、開発環境およびシステム管理環境を提供します。本節では、これら 2 バージョンのアドバンスド版 Application Server の以下の点について説明します。

- 6 ページの『機能』
- 7 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『アプリケーション開発環境』

詳細については、InfoCenter 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

機能

アドバンスド・シングル・サーバー版およびアドバンスド・デベロッパー版はともに、単一マシンの Web アプリケーション・サーバーを提供します。通常、どちらかの版をインストールすることで 1 つの Web サイトがサポートされます。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パブリッシュします。

WebSphere Application Server アドバンスド・シングル・サーバー版および WebSphere Application Server アドバンスド・デベロッパー版は、ともに HTML タグの拡張セットとアプリケーション開発ツールのセットを提供し、それらのタグを使用したページの作成 (スクリプト記述) をサポートします。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2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。サーブレット・エンジンは、アドバンスド版 Application Server に含まれています。詳細については、27ページの『第3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エンタープライズ Bean。詳細については、35ページの『第4章 エンタープライズ Bean 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5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

図1 は、WebSphere Application Server アドバンスド・シングル・サーバー版と WebSphere Application Server アドバンスド・デベロッパー版のランタイム・アーキテクチャーを示しています。

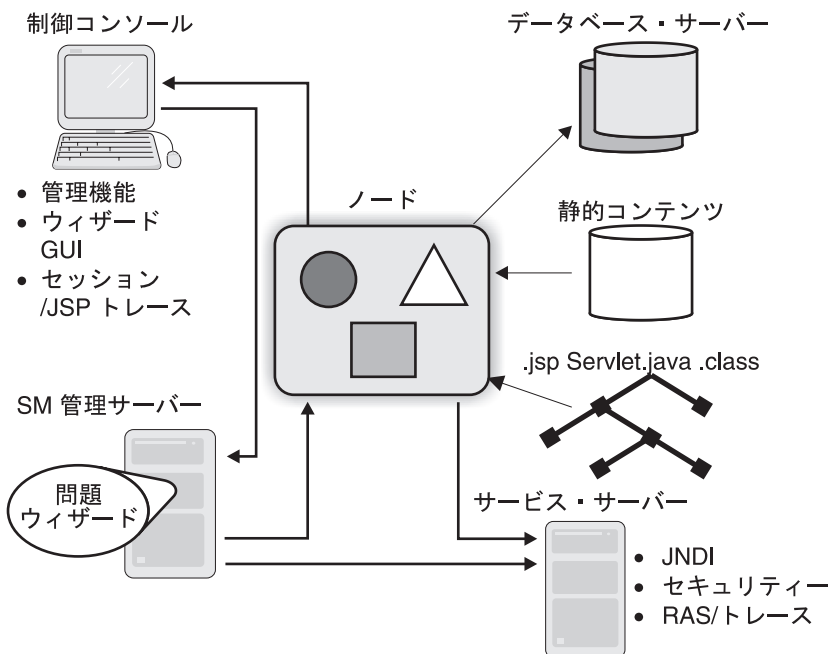


図1. WebSphere Application Server アドバンスド版ランタイム・アーキテクチャー

ランタイムおよびシステム管理のためのコンポーネントは、以下の通りです。

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

WebSphere Application Server アドバンスド・シングル・サーバー版と WebSphere Application Server アドバンスド・デベロッパー版は、IBM、Netscape および Apache の 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、アプリケーション・サーバー・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。すべてのバージョンのアドバンスド版 Application Server ランタイムおよび ORB は、これらのスレッドが Web サーバー内では、その Web サーバーにより制御されるスレッドの下で実行されるよう、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

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、およびエンタープライズ 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、エンタープライズ Bean 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。エンタープライズ 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、アプリケーション・サーバーの 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・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。

アプリケーション開発環境

WebSphere Studio は、アドバンスド版 Application Server のすべての版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。

WebSphere Application Server アドバンスド・シングル・サーバー版と WebSphere Application Server アドバンスド・デベロッパー版のアプリケーション開発環境は、以下をサポートしています。

- **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、すべてのバージョンのアドバンスド版 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3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 デベロッパーは、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2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** - すべてのバージョンのアドバンスド版 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
 - 名前付き表での作成、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

WebSphere Application Server アドバンスド版

WebSphere Application Server アドバンスド版は、WebSphere Application Server アドバンスド・シングル・サーバー版および WebSphere Application Server アドバンスド・デベロッパー版に基づいて構築されています。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「InfoCenter」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

アドバンスド版 Application Server の機能

アドバンスド版 Application Server のフル・バージョンでは、スケーラブルでトランザクション型の Web アプリケーションを開発できます。アドバンスド版 Application Server は、WebSphere Application Server アドバンスド・シングル・サーバー版や WebSphere Application Server アドバンスド・デベロッパー版によって提供される単一マシン環境とは対照的に、複数のマシン・システム上で動作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エンタープライズ Bean、分散システム管理、および共有名、セキュリティおよびトランザクション・サービスをサポートします。

WebSphere Application Server アドバンスド・シングル・サーバー版および WebSphere Application Server アドバンスド・デベロッパー版の機能の詳細については、5ページの『WebSphere Application Server アドバンスド・シングル・サーバー版および WebSphere Application Server アドバンスド・デベロッパー版』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、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・アーキテクチャーを示します。

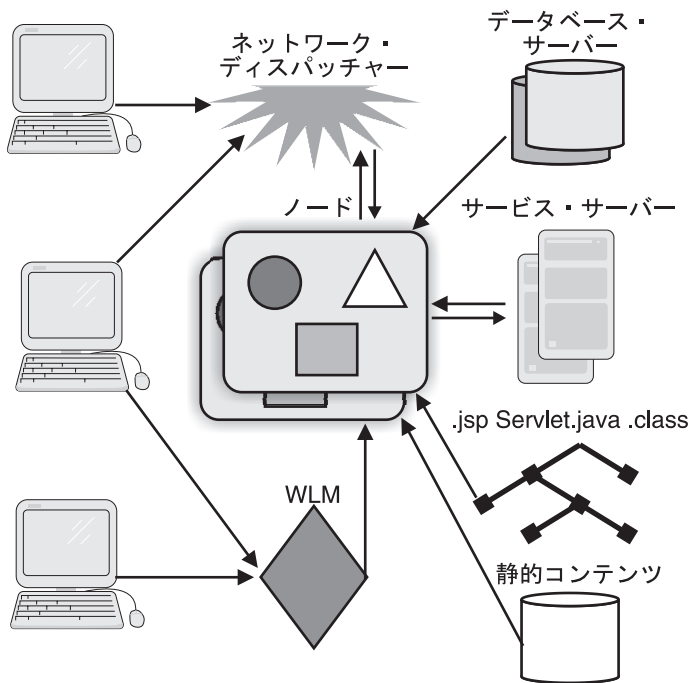


図2. ハイレベルな視点からのアドバンスド版 Application Server システム・アーキテクチャー

アドバンスド版 Application Server システムにおける要素は、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 つの要素があります。

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。

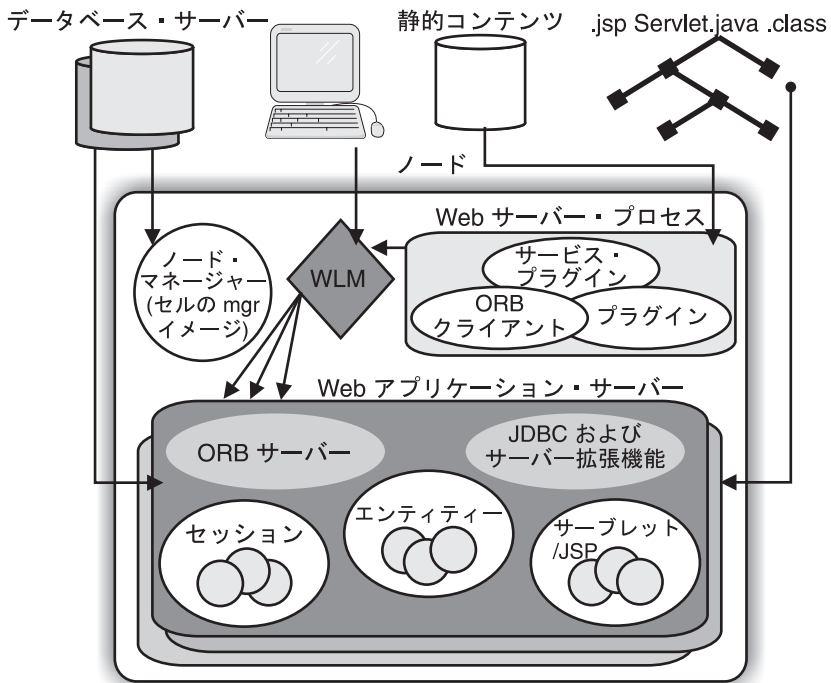


図3. アドバンスド版 Application Server システムの一般的なノード

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 ページおよびエンタープライズ Bean を実行するクローンであり、同じポリシーで構成された同じコンテナを持ちます。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 つのセル内でのみ存在できます。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エンタープライズ Bean 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、フェールオーバー (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。

クローン (アプリケーション・オブジェクトのレプリカ) とサーバー・グループは、アプリケーション・サーバー全体の複数のインスタンスを生成するのに使用することができます。アドバンスド版 Application Server は、クローンのワークロードを自動的に管理します。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 (フェールオーバー・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、障害が発生したことをセル・マネージャーに通知をします。

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管理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、エンタープライズ 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せん。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、エンタープライズ 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クローン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・グループ間で共有されます。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・スペースを共有します。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・システムに基づく、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

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、エンタープライズ Bean 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、コンポーネントはエンタープライズ Bean にアクセスします。(アプリケーション開発ツールは、JSP タグを使用してエンタープライズ Bean 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。それに代わる手段として、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提供されています。

JSP ページの詳細については、19ページの『第2章 JavaServer Pages の使用』を参照してください。

- エンタープライズ Bean 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4章 エンタープライズ Bean の使用』を参照してください。
- サブレットは、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3章 サブレットの使用』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、エンタープライズ Bean 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5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、127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『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、配置、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り、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エンタープライズ Bean 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2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、127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。サ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 製品のマニュアルおよびサンプル
- 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。

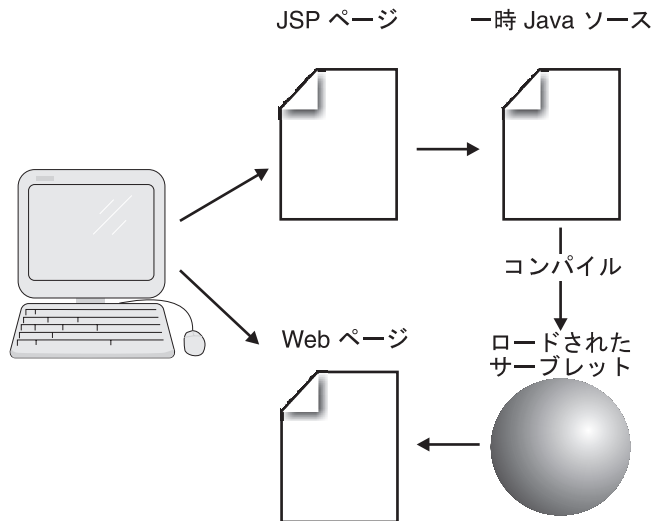


図4. JSP プログラムのフロー

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 プロセッサは、Java で記述されたスクリプト要素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 ランタイム環境が必要です。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も Java ランタイム環境が必要になります。

JSP コンパイラは 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い場合は、ロードされているサ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、サーバーのコンポーネントに情報を要求し、各コンポーネントでは必要な処理が行われます。次に、JSP ページは情報を受け取ってそれを 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。

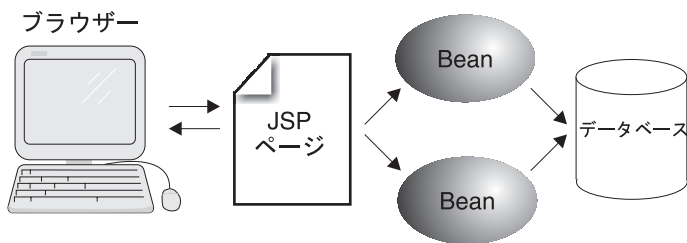


図5. ブラウザーからの JSP ページの要求

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ブレットに送信され、サブレットが JSP ページを呼び出します。サブレットは、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。

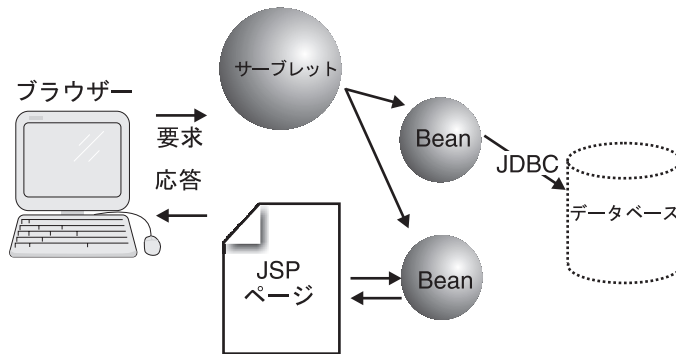


図6. サーブレットからの JSP ページの要求

JSP ページを呼び出すためのこの方式は、127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 コンポーネントのプロパティ、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、エンタープライズ Bean 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 ページはエンタープライズ Bean を使用して、バック・エンドのリソースと対話します。EJB サーバーおよびエンタープライズ 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・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、エンタープライズ Bean、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、エンタープライズ Bean、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によって大きく異なります。これらのプロパティは、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たします。

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そのまま生かされています。

従来スキルの使用

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3章 サブレットの使用

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ブレットについて説明します。サブレットは、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。サブレットは、サブレット・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、サブレットに関する以下の項目について説明します。

- 『サブレットのプログラミング・モデル』
- 28ページの『サブレット API』
- 29ページの『サブレットのライフ・サイクル』
- 30ページの『サブレットのランタイム環境』
- 31ページの『サブレット・セッションの管理』
- 32ページの『サブレットの長所』

本節では、127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の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 製品のマニュアルおよびサンプル
- WebSphere Studio の資料およびサンプル
- 「WebSphere エンタープライズ Bean の作成」 - エンタープライズ Bean を使用するサブレットの記述方法について説明しています。

サブレットのプログラミング・モデル

サブレットでは、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・サーバーからサブレットに送信されます。サ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。

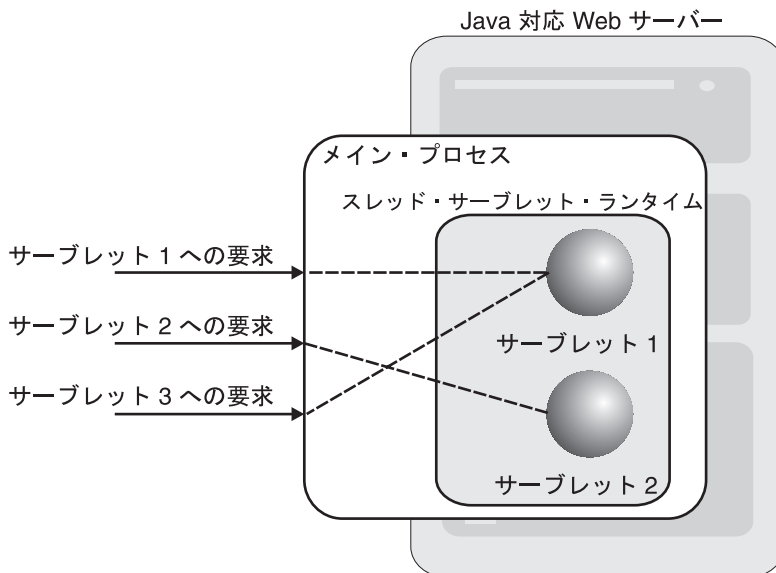


図7. サーブレットの実行モデル

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ず。
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、サーブレットのランタイム環境へ転送されて、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。

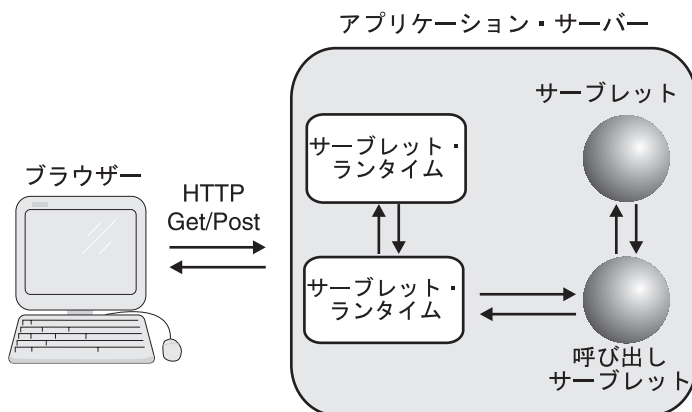


図8. サーブレットの基本フロー

サーブレットのランタイム環境

WebSphere Application Server サーブレットのランタイム環境は、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。サーブレットのランタイム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

サーブレットは、特別なサーブレット固有のコンテナ内で実行されます。このコンテナは、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・コンテナでは、トランザクション、パーシステンス、セキュリティといったサポートは、サーブレットのランタイム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・コンテナは EJB コンテナ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・コンテナをエンタープライズ Bean 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・プロセスヘディスパッチします。サーバー・グループのすべてのクローンは、同一のキューを共用し、スレッドが使用可能になるとキューからの要求を処理します。すべてのクローン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、サーブレットのクローンを実行する 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クローンに限定されます。

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。サーブレットのランタイム環境では以下の 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。クローン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。プ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除去されます。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。

サブレットの長所

サブレットは、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、サブレットと Web サーバー間の標準インターフェースがサブレットの 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、CGI 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 ページおよびエンタープライズ Bean 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、アプレットを発信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、アプレットを発信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4章 エンタープライズ Bean の使用

エンタープライズ Bean 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。エンタープライズ Bean 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』
- 38ページの『エンタープライズ 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 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、127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の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。エンタープライズ Bean 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。エンタープライズ Bean の開発の詳細については、以下を参照してください。

- *WebSphere* エンタープライズ Bean の作成
- IBM VisualAge for Java 製品のマニュアルおよびサンプル

EJB 環境

EJB 仕様には、エンタープライズ Bean 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、エンタープライズ 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、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 コンテナおよびクライアント・コンテナによって定義され

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エンタープライズ Bean と対話します。クライアントは、任意のエンタープライズ 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エンタープライズ Bean と対話します。EJB サーバー環境とコンテナについては、以下の節で説明します。

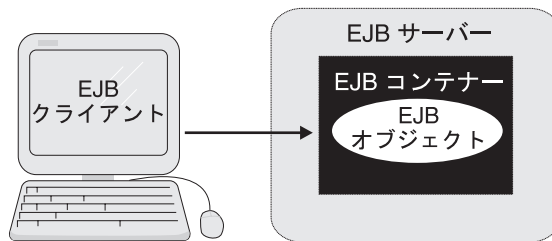


図9. EJB 環境のクライアント・ビュー

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

エンタープライズ Bean はコンテナ に配置されます。EJB 仕様で説明されているように、コンテナは 1 つまたは複数のエンタープライズ Bean 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、コンテナ内のエンタープライズ 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 と呼ばれます) は、エンタープライズ 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エンタープライズ Bean とどのように対話するのかを示したものです。

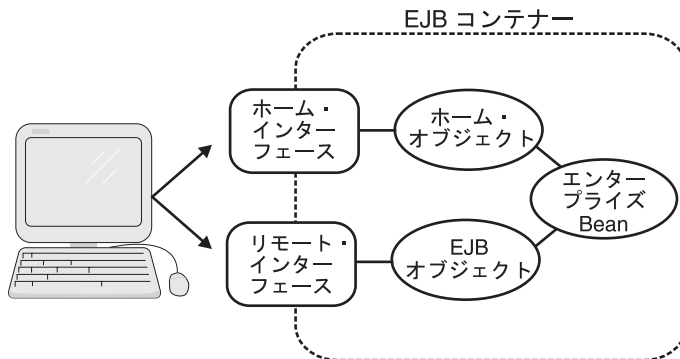


図 10. クライアントとエンタープライズ Bean との対話

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、それが使用するエンタープライズ 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、エンタープライズ 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。

エンタープライズ Bean アプリケーションの開発

本節では、エンタープライズ Bean 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。本章では、以下のトピックについて説明します。

- 『エンタープライズ Bean のタイプ』
- 39ページの『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 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』

エンタープライズ 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 Bean という 2 種類のエンタープライズ Bean 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。このタイプのエンタープライズ 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 Bean は、アプリケーションにおける永続的エンティティを表します。エンティティ 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 Bean によって表される永続データは、エンティティ Bean のコンテナまたはエンティティ 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）。このタイプのエンタープライズ Bean の詳細については、46ページの『エンティティ Bean』を参照してください。

セッション Bean とエンティティ Bean の最大の相違点は、セッション Bean にはエンタープライズ 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、1 次キーは不要です。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、除去することができます。これとは対照的に、エンティティ 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 Bean の使用

エンティティ 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 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 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。

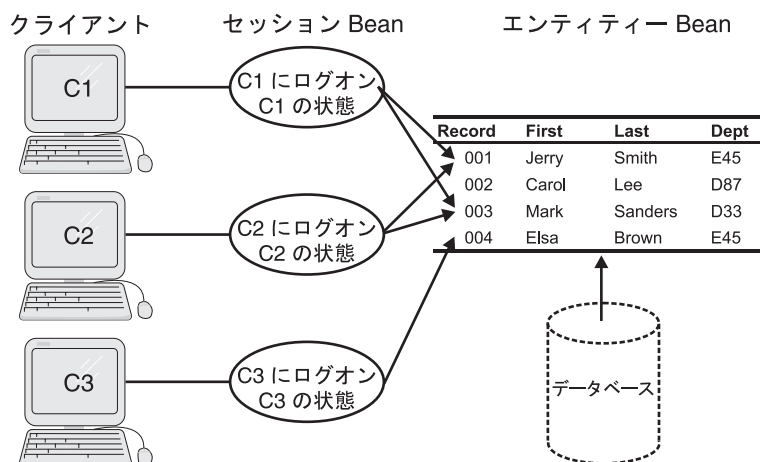


図11. セッション Bean とエンティティ Bean を使用したアプリケーションの構築

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データを取出す必要があります。エンティティ 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 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 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 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。

エンタープライズ 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 プログラミング言語でビジネス・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、エンタープライズ Bean 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、エンタープライズ Bean を永続的にするかしないかを決定します。エンタープライズ 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エンタープライズ Bean が関係する場合の、エンタープライズ Bean の振る舞いを指定します。提供者には、エンタープライズ Bean 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 コンテナ提供者

コンテナ提供者は、エンタープライズ Bean コンテナを作成します。エンタープライズ Bean コンテナは、エンタープライズ Bean に、クライアント・インターフェース、パーシスタンス、スケラビリティ、セキュリティ、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に入れます。現時点ではコンテナとサーバーとの間

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は同じベンダーが提供します。

配置担当者

配置担当者は、エンタープライズ Bean のクラスを EJB サーバーにインストールします。配置担当者はエンタープライズ Bean 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エンタープライズ 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 を介してエンタープライズ Bean 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、エンタープライズ Bean 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、既存のエンタープライズ Bean から新しいエンタープライズ Bean 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

エンタープライズ Bean 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、エンタープライズ Bean 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. コンテナ提供者は、エンタープライズ Bean が使用するコンテナを作成します。
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・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。

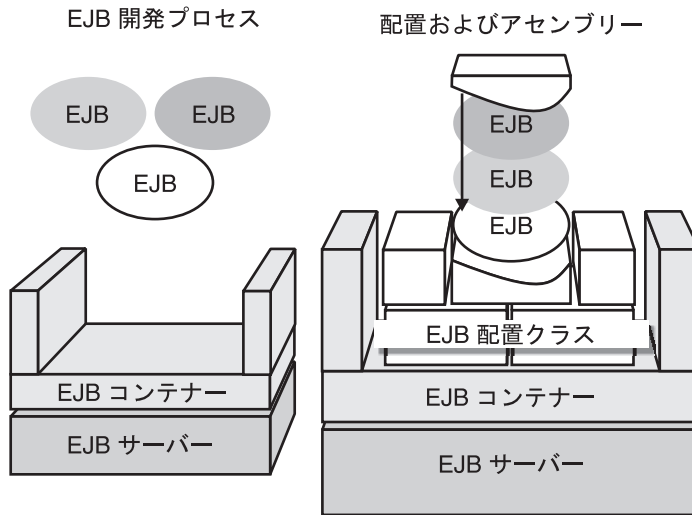


図 12. エンタープライズ Bean アプリケーション開発プロセス

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。これを活性化といいます。

コンテナはステートレス・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、エンタープライズ 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、コンテナがエンタープライズ 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。他のエンタープライズ Bean 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生成クラスを使用してエンタープライズ 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 クラスにエンタープライズ 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、エンタープライズ 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、エンタープライズ Bean の配置時にコンテナによって、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、エンタープライズ 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、エンタープライズ 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、エンタープライズ 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、エンタープライズ Bean のインスタンスの除去、およびエンタープライズ 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エンタープライズ 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、そのエンタープライズ 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、エンタープライズ Bean の配置時にコンテナによって、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、「*WebSphere* エンタープライズ Bean の作成」を参照してください。

エンティティ Bean

エンティティ 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 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 Bean の開発者は、エンティティ 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エンタープライズ 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 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 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 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 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 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 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 Bean の開発者は、CMP を持つエンティティ 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 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 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 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
- 1 次キー・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 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 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 Bean のインスタンスに通知するためにコンテナ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 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、エンタープライズ 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、エンタープライズ 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、エンタープライズ Bean に割り当てられた名前です。

コンテナは、ホーム・インターフェース、ハンドル、およびエンタープライズ 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、エンタープライズ 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。

1 次キー・クラス

すべてのエンティティ Bean は、コンテナ内に固有の 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 1 次キー の組み合わせを使用して定義されます。クライアントは 1 次キーを使用して、エンティティ Bean のインスタンスの作成または検索を行います。1 次キーは、エンティティ 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 つのエンタープライズ 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 データ型 (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 1 次キーは、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 1 次キーは、1 次キー・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。1 次キー・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、エンタープライズ Bean に割り当てられた名前です。

finder メソッド

finder メソッドは、1 次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 (アドバンスド版 Application Server) `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 Bean のライフ・サイクル

エンティティ Bean がコンテナに配置された後、クライアントはそのエンティティ 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 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 Bean のインスタンスのライフ・サイクルは、コンテナ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 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 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。その Bean クラスのプール内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 Bean のインスタンスを要求すると、コンテナ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 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 Bean クラスの新しい固有エンティティ (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 Bean クラスの既存のインスタンスを操作したとき。

エンタープライズ 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 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 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 Bean のインスタンスのコンテキストを、コンテナが除去した時点で終了します。

注: エンティティ 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 Bean のライフ・サイクルの詳細については、「*WebSphere* エンタープライズ Bean 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 つまたは複数のエンタープライズ Bean 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、エンタープライズ 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エンタープライズ Bean の使用を容易にします。

アクセス Bean を使用すると、エンタープライズ Bean 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、直接エンタープライズ Bean を使用することはありません。これにより、ネーミング・サービスとエンタープライズ 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、エンタープライズ 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

側のエンタープライズ 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 Bean 用の CopyHelper** - エンティティ 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 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 Bean 用の行セット** - 1 つ以上のエンティティ 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 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、選択されたエンタープライズ Bean 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 のエンタープライズ Bean アーキテクチャーは、ポリモアフィズムをサポートし、エンタープライズ Bean の再利用をより簡単に行うことができるエンタープライズ Bean の継承を認識します。継承を使用すると、エンタープライズ 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、エンタープライズ 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エンタープライズ 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。

継承がエンタープライズ Bean に与える影響

エンタープライズ Bean は、その親のインターフェースを直接継承しません。親のエンタープライズ Bean のインターフェースは、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、親のエンタープライズ Bean は Parent、子のエンタープライズ Bean は Child という名前が付けられています。親の 1 次キー・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	親 Bean インターフェースの名前	子 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
1 次キー・クラス	ParentKey	ParentKey

注: 継承を使用しないエンタープライズ Bean または継承階層のルートにあるエンタープライズ Bean 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、エンタープライズ 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

子のエンタープライズ 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

子のエンタープライズ 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、親のエンタープライズ 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、子のエンタープライズ 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、親のエンタープライズ 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、親と子の両方のエンタープライズ 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 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。

1 次キー・クラス

エンティティ Bean は、その 1 次キーに対して 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 Bean では、その 1 次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 Bean における継承

継承により、CMP を持つエンティティ 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 Bean 間で指定することができます。
- 1 次キー検索では、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 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` エンタープライズ Bean などの関連するエンティティ 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 Bean すべてが戻されます。

継承を使用する、CMP を持つエンティティ Bean は、1 つの例外を除き、CMP を持つエンティティ 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 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、エンタープライズ Bean 間の継承を定義することができます。これらのツールは、エンタープライズ Bean 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 `persistor` 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。)

親と子のエンタープライズ Bean クラスの間の関係は、子クラスとして識別されるエンタープライズ Bean 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エンタープライズ Bean 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 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 `persistor` 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 のエンタープライズ 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 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、関連の関係を持つエンタープライズ Bean 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 エンタープライズ 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 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 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・ブラウザを使用すると、関連を他のエンティティ 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 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 Bean のみ参照します。サブクラスは、エンティティ 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 がない場合は、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・サービス』
- 63ページの『パーシスタンス・サービス』
- 68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の問題については、127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、エンタープライズ Bean 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、エンタープライズ Bean を検索する以外にも、エンタープライズ Bean の属性がわかり、その属性でエンタープライズ 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

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。

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

58ページの図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。

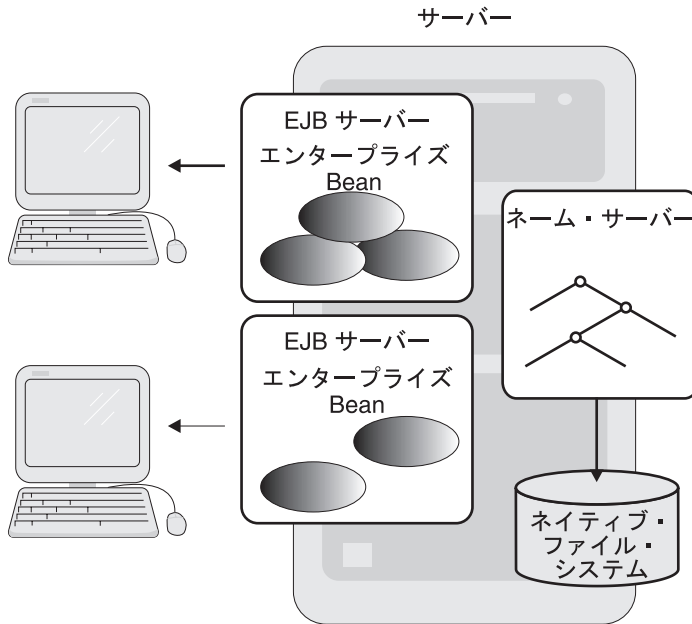


図 13. WebSphere Application Server のオブジェクト・サービス・システム

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。エンタープライズ 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・サーバーにエンタープライズ Bean 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ナルに実行されます。

EJB サーバー: エンタープライズ Bean 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・サービス

WebSphere Application Server は統一されたセキュリティ・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、およびエンタープライズ Bean のセキュリティ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・サーバー』
- 60ページの『セキュリティ・コラボレーター』
- 60ページの『セキュリティ・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』
- 62ページの『委任ポリシー』
- 62ページの『セキュリティ・サービスの使用』

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

セキュリティ・サーバー

セキュリティ・サーバーで、セキュリティ・ポリシーとセキュリティ・サービスを集中制御することができます。セキュリティ・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、エンタープライズ Bean 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 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。

エンタープライズ 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エンタープライズ Bean 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エンタ

ープライズ 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、エンタープライズ 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。

- エンタープライズ Bean 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。

エンタープライズ 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・サービスの使用

クライアントがエンタープライズ 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。

63ページの図14 は、Account エンタープライズ 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。

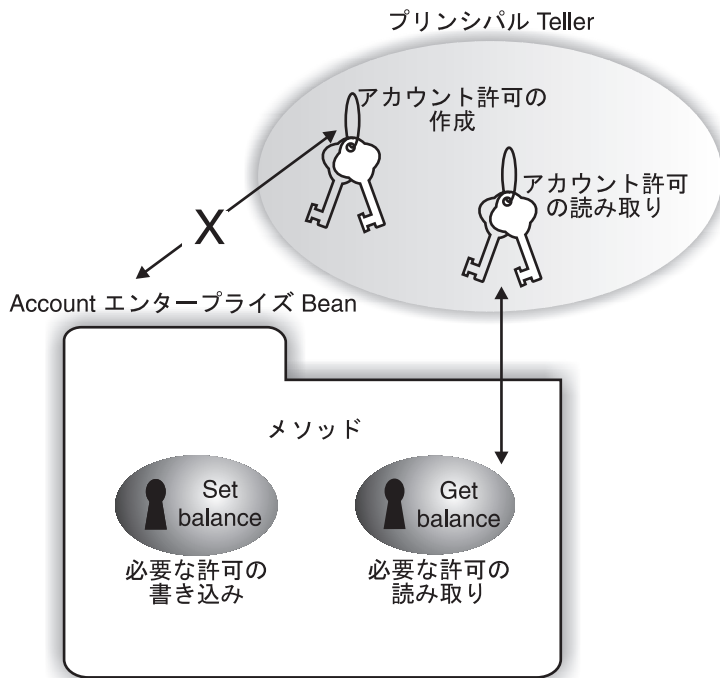


図 14. EJB メソッドの許可ベースの保護

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。

- 64ページの『コネクター Bean』
- 65ページの『アダプター Bean』

- 65ページの『Persistor』
- 65ページの『共通コネクタ・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 Bean

コネクタ *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 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 Bean には、次の 2 つのタイプがあります。

- **データ・コネクタ 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 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 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
- エンタープライズ Bean 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 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 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、エンタープライズ Bean 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3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 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2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。

Persistor

WebSphere Application Server のエンタープライズ Bean のプログラミング・モデルは、helper 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 helper クラスは、*persistor* と呼ばれます。helper class 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。

persistor は、永続データを直接保持しているエンタープライズ Bean によってインプリメントされます。CMP を持つエンティティ Bean は、*persistor* 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 Bean は、コンテナによって作成される helper 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 Bean では、*persistor* 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 Bean を使用するか、*persistor* 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。

エンタープライズ Bean 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。

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。

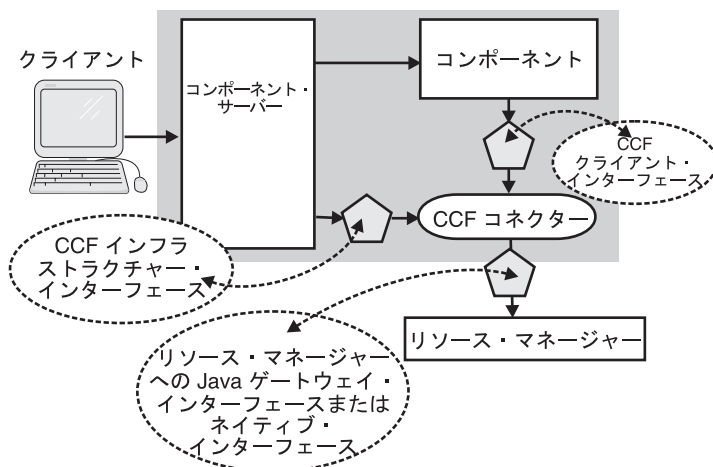


図 15. CCF アーキテクチャー

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 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、エンタープライズ Bean 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。

- 『エンタープライズ Bean 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』
- 71ページの『ロック』
- 71ページの『OTS および EJB トランザクション・モデル』
- 74ページの『セッションの同期化のインプリメント』

エンタープライズ Bean のためのトランザクション管理

EJB アーキテクチャーは、エンタープライズ Bean 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、エンタープライズ Bean がトランザクションを制御します。
- コンテナ管理のトランザクション では、コンテナがそのエンタープライズ Bean 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 Bean はコンテナ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、「WebSphere エンタープライズ Bean 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、エンタープライズ 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、エンタープライズ 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 クライアントがエンタープライズ 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エンタープライズ 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、エンタープライズ 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、コンテナーがエンタープライズ 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 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 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

エンタープライズ Bean 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 の Java プログラミング言語バインディングです。JTS は、サーバー間でトランザクションを伝搬するための標準の 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 72 ページの図16 に示します。破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。

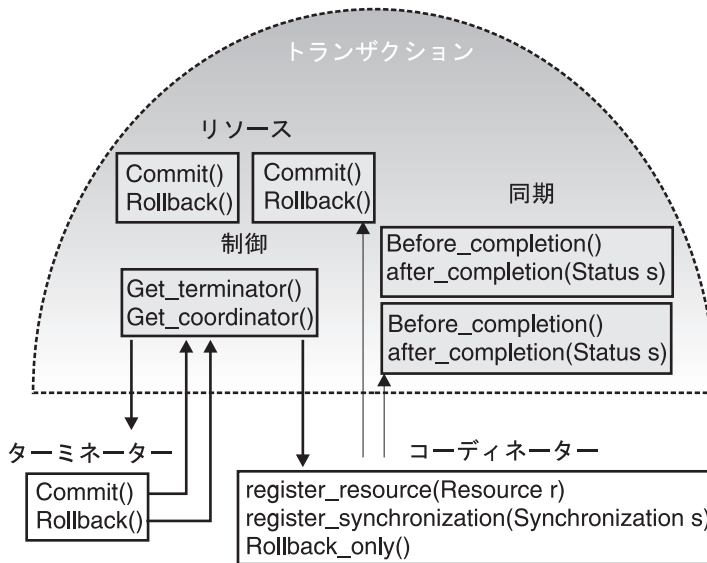


図 16. OTS トランザクション・モデル

制御オブジェクト

トランザクションを表します。エンタープライズ 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。エンタープライズ Bean はこのオブジェクトに直接アクセスしません。エンタープライズ 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。この区別はエンタープライズ Bean 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、commit メソッドおよび rollback 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 commit メソッドと rollback 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、コンテナ管理のトランザクションを使用するエンタープライズ Bean があります。コンテナは、エンタープライズ Bean に代わってトランザクションを保守します。エンタープライズ 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。エンタープライズ Bean は、commit メソッドあるいは rollback メソッドを持たないため、トランザクションはエンタープライズ 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エンタープライズ Bean を作成するには、追加作業が必要です。エンタープライズ Bean 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 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、127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 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 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・ベンダーでサポートされているわけではありません。

第5章 Web アプリケーションの開発

JSP ページ、エンタープライズ Bean、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、およびエンタープライズ Bean 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2章 JavaServer Pages の使用』
- 27ページの『第3章 サーブレットの使用』
- 35ページの『第4章 エンタープライズ Bean 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。

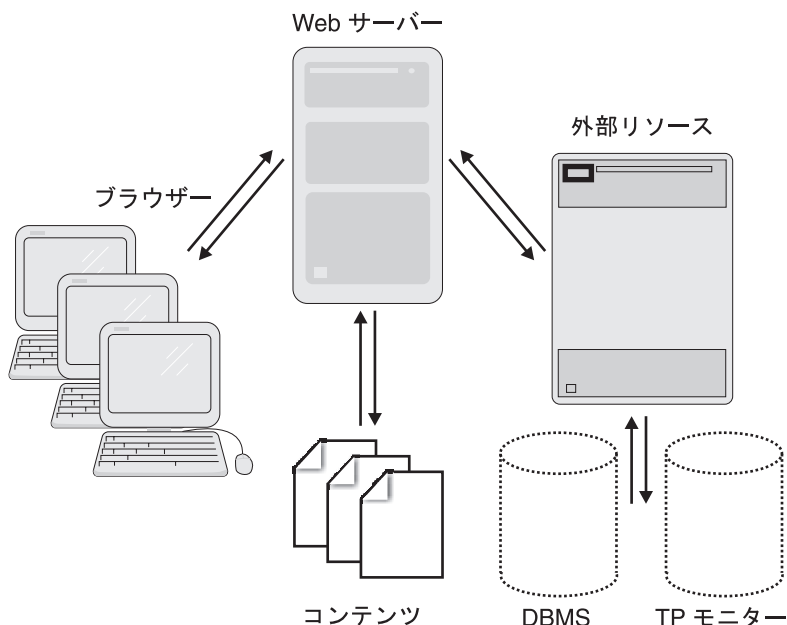


図 17. Web アプリケーション・コンポーネント

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) が含まれていません。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。エンタープライズ Bean 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、およびエンタープライズ Bean 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方法に関して説明します。第 1 部には以下の内容が記載されています。

- 『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エンタープライズ Bean 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いからです。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 のサーブレットのランタイム環境では、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。

127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、エンタープライズ Bean 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間で送受信されるデータです。ブラウザは 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、およびエンタープライズ Bean 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ンネルであり、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です。ブラウザ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、デジタル証明書およびデ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適切な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。

デジタル証明書

デジタル証明書は、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 はエンティティー B の公開鍵のコピーとメッセージを比較して、メッセージ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。

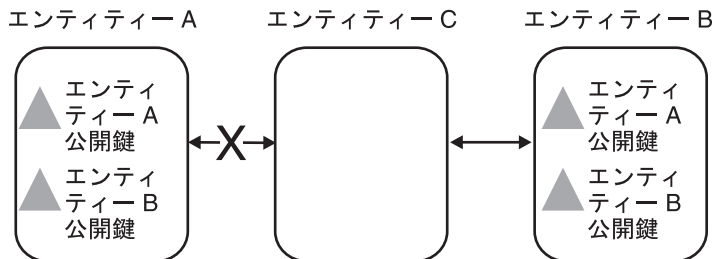


図18. デジタル証明書による認証

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6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とエンタープライズ版サービスによって提供される機能を結合します。また、エンタープライズ版は、エンタープライズ Bean 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』
- 87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。

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ラビリティ、可用性、整合性、持続性、およびセキュリティを提供します。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、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 により提供されるエンタープライズ

Bean 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様向けの 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 プログラミング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、および CICS アプリケーション上で Encina アプリケーションとエンタープライズ Bean 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。

エンタープライズ版サービス

エンタープライズ版サービスにより、アドバンスド版 Application Server は、エンタープライズ環境で機能することができます。エンタープライズ版サービスには、WorkArea サービス (ビジネス・ルール Beans の開発ツール)、国際化対応サービス、Java メッセージング・サービス (JMS) リスナー・サービス、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 WebSphere Application Server とのインターオペラビリティのサポートがあります。

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を実現しています。

88ページの図19 に、このインターオペラビリティが機能する様子を示します。

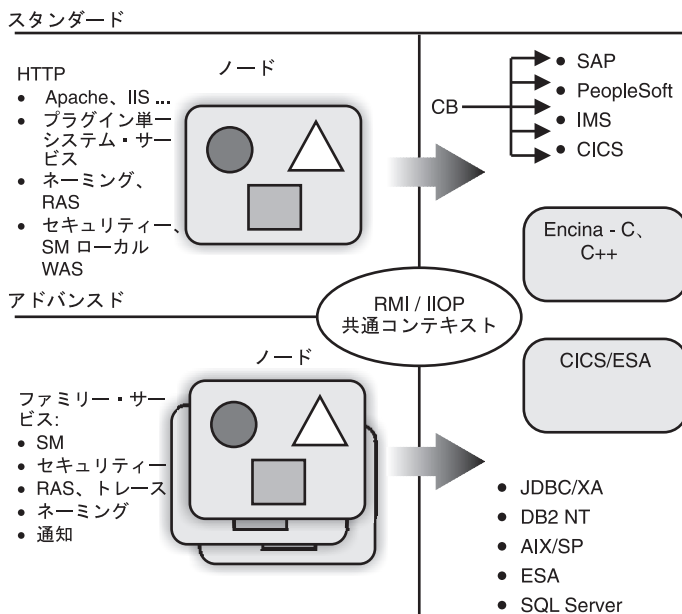


図 19. エンタープライズ版 Application Server のインターオペラビリティ

- **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のためのセキュリティ、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、共通コネクタ・フレームワーク (CCF) 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、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)、およびエンタープライズ Bean 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。
- **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 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、お客様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 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『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、エンタープライズ Bean 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・エンタープライズ Bean
- 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
- エンタープライズ 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エンタープライズ Bean 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
- エンタープライズ 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

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の処理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エンタープライズ 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、オブジェクト・ビルダー、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。

- 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
- 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

ラリは、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 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、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ラビリティ、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7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、112ページの『TXSeries CICS』を参照してください。

本節では、127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』
- 99ページの『リカバリー可能キューイング・サービス (RQS)』
- 100ページの『構造化ファイル・サーバー (SFS)』
- 102ページの『対等通信 (PPC) サービス』
- 104ページの『Encina++』
- 107ページの『Encina ツールキット』
- 109ページの『DCE-Encina Lightweight クライアント (DE-Light)』
- 110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』

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。

- **ランタイム環境。** モニターのランタイム環境は、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。

モニターのランタイム環境

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。このランタイム環境は、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があります。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。

96ページの図20 に示すモニター・セルの物理アーキテクチャーは、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。

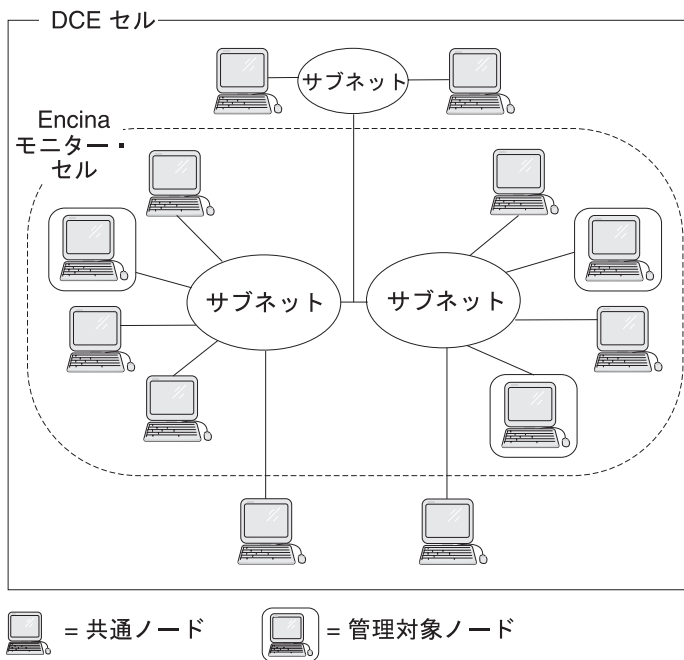


図 20. モニター・セルの物理アーキテクチャー

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。

モニターのランタイム環境および 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・モジューラの集合であり、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 ランタイム・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 ランタイム環境は、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* システム環境での *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リカバリー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の、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 サービスによって双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21 に示します。

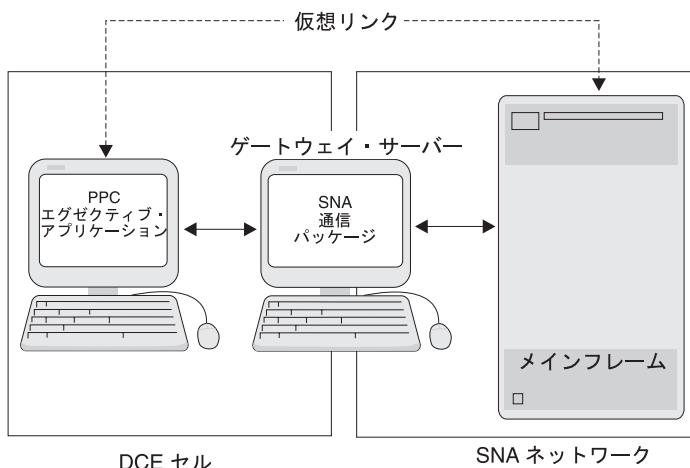


図 21. PPC 通信モデル

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

振りは双方向です。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・プロポーザル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。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汎用固有 ID (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 ではなく名前でも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安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ための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、およびサーバーのクローンなど、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 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 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 (110ページの図22 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

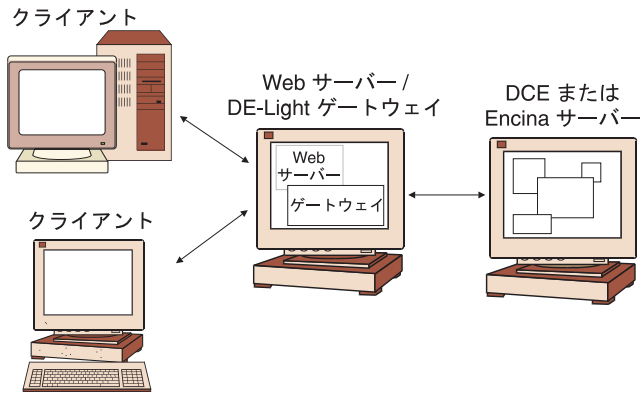


図 22. DE-Light Java クライアント

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

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23 は、ブリッジ・サーバーを使用する分散システムの場合です。

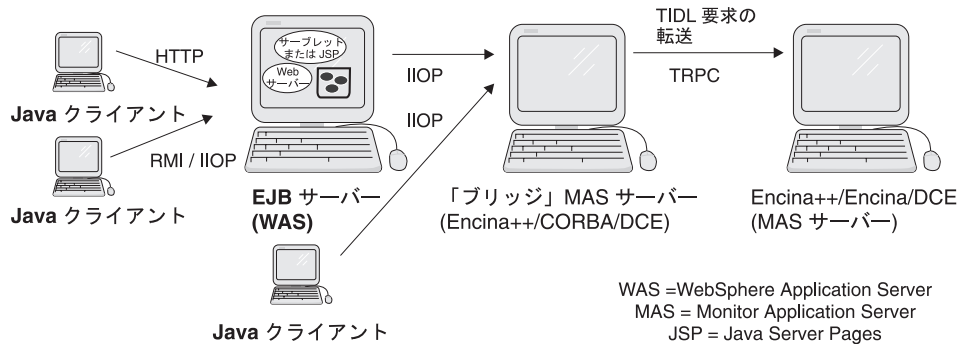


図 23. Java アプリケーションと Encina/Encina++ サーバーとのインターオペラビリティ

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 インターフェイス、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 またはエンタープライズ 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 インターフェイス、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 の類似の要素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、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)」および「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・サイクルとプロセッサ・メモリーを使用し、タスクとユーザー・データには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、データ通信チャンネル、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、およびリカバリー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が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24 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。

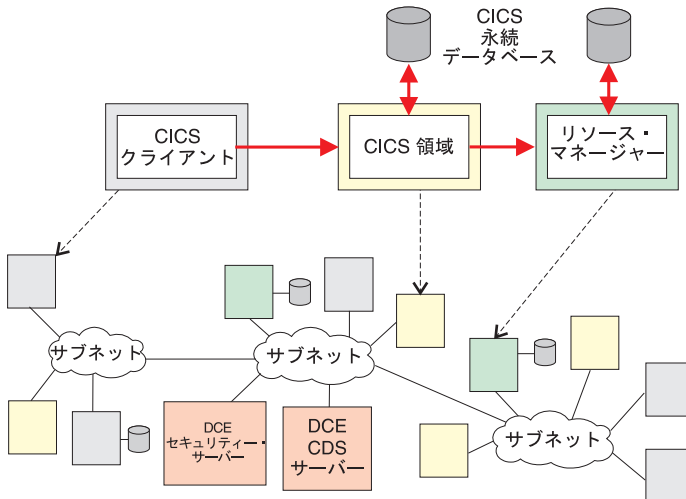


図 24. CICS 領域

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、領域間 (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、121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。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・システムなどがすべて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論理作業単位 (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いければ、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、データ安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リカバリーします。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 -- 論理作業単位の処理。

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。

- *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、ユーザー入力に回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きます。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せん。

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) に制御を渡す位置です。ユーザーの出口プログラムが処理を終了すると、CICS に制御が戻されます。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、102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、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25 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。

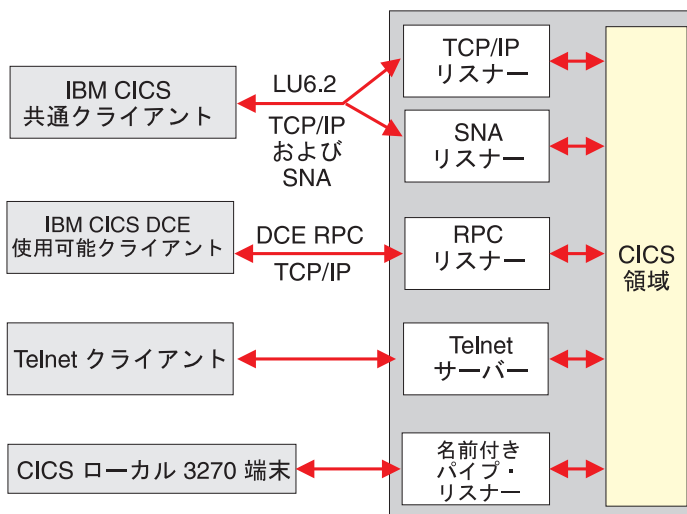


図25. 複数の CICS クライアントと CICS 領域との間の通信

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、および問題のリカバリー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が関係します。

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ます。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で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、「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。

- 129ページの『第8章 サンプル・アプリケーションの概要』
- 133ページの『第9章 サンプル・アプリケーションの設計』
- 139ページの『第10章 サンプル・アプリケーションのインプリメント』
- 145ページの『第11章 サンプル・アプリケーションの技術的な詳細』
- 167ページの『第12章 サンプル・アプリケーションの拡張』

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8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利用可能に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 各部分が共同で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も、ほとんどの WebSphere ツールおよび機能が使用されます。サンプル・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、エンタープライズ Bean のような分散オブジェクトでインプリメントされています。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、およびエンタープライズ版 Application Server 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テーションでは、エンタープライズ Bean をアドバンスド版 Application Server および TXSeries Encina++ とともに使用する方法を示します。
- 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、コンポーネントごとに、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。
- 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9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』
- 134ページの『クライアント / サーバーの関係』
- 134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』
- 135ページの『オブジェクト・モデル』
- 137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、139ページの『第10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

134ページの図26 に、サンプル・アプリケーションの設計方法を示します。

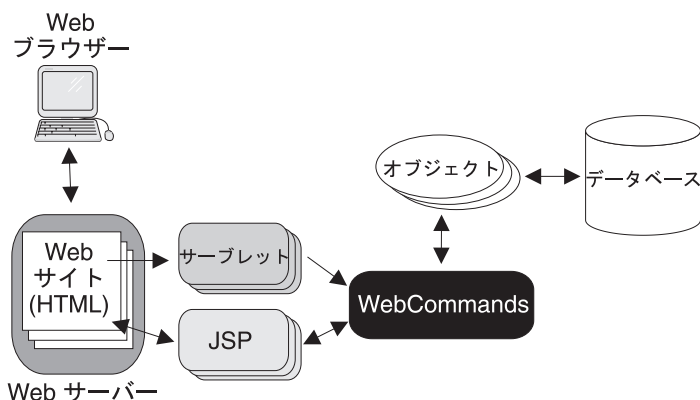


図 26. サンプル・アプリケーション

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、129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。136ページの図 27 に、ビジネス・ロジックを構成するオブジェクトとオブジェクトの対話方法を示します。

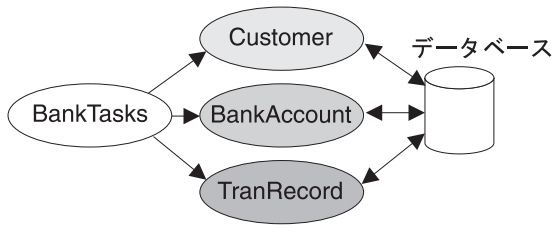


図 27. オブジェクト・モデル

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ますので、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、エンタープライズ **Bean** 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ています。詳細については 139ページの『第10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の各レコードは、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10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せん。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 2 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 に配置されたエンタープライズ Bean によるインプリメント。コンテナ管理のパーシスタンス (CMP) を持つエンティティ Bean、および永続的ストレージとして DB2 データベースを使用します。このインプリメントについては、140ページの『エンタープライズ Bean を使用したアドバンスド版のインプリメント』で説明します。
- アドバンスド版 Application Server に配置されたエンタープライズ Bean 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 Bean、および 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・サーバーと通信する TXSeries Encina++ ブリッジ・サーバーを使用します。このインプリメントについては、141ページの『エンタープライズ Bean 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。

143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。

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。

エンタープライズ Bean を使用したアドバンスド版のインプリメント

図28 に、アドバンスド版 Application Server に配置されたエンタープライズ Bean 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、134ページの図26 に示します。)

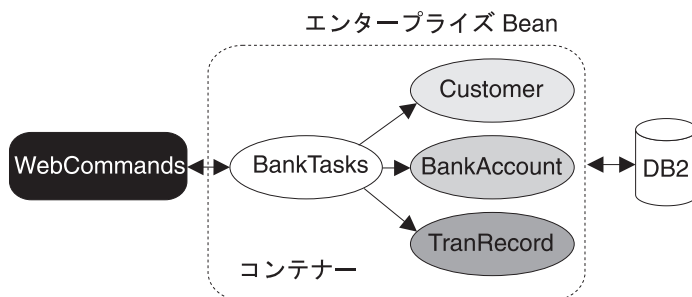


図28. アドバンスド版 Application Server でのエンタープライズ Bean のインプリメント

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 Bean の 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、エンタープライズ Bean が配置されたコンテナによって処理されます。エンティティ Bean による要求に応じて、コンテナは 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。エンタープライズ Bean の開発者は、CMP を持つエンティティ 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 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、新しいコンテナに配置することができます。

コンテナは、エンタープライズ Bean 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 で配置されるエンタープライズ Bean に関連するトランザクションは、コンテナ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エンタープライズ 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エンタープライズ Bean に組み込む必要はありません。コンテナは、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、68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。

エンティティ Bean の 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。

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、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。すべてのタイプのエンタープライズ Bean で、アクセス Bean を使用することができます。

アクセス Bean は、エンタープライズ Bean をローカルの JavaBeans コンポーネントのように使用できるように設計されています。これは、エンタープライズ 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、エンタープライズ 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。

エンタープライズ Bean 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

142ページの図29 に、エンタープライズ Bean 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、134ページの図26 に示します。)

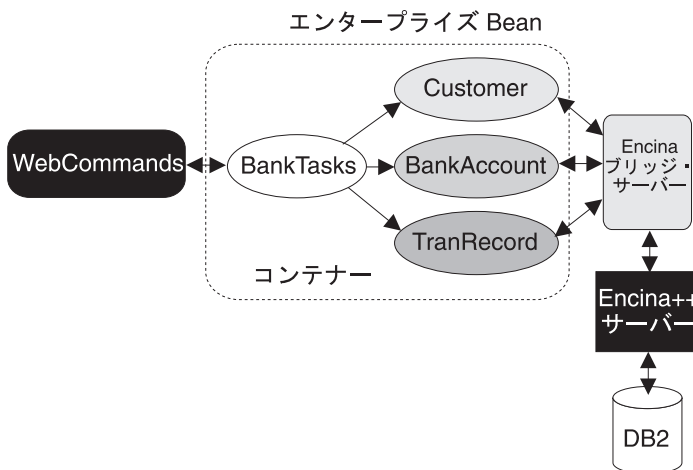


図 29. Encina++ を使用したエンタープライズ Bean のインプリメント

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、エンタープライズ Bean 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。

エンタープライズ Bean のインプリメントは、140ページの『エンタープライズ Bean を使用した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 Bean です (CMP を持つエンティティ Bean ではありません)。エンティティ 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、以下のプラットフォームで使用可能です。

- AIX
- Solaris
- Windows NT

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。

第11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。本節で説明するトピックは以下のとおりです。

- 『Web サイト』
- 147ページの『サーブレット』
- 150ページの『JavaServer Pages』
- 152ページの『WebCommand』
- 155ページの『エンタープライズ Bean (アドバンスド版)』
- 159ページの『エンタープライズ Bean、Encina ブリッジ・サーバー、および Encina++』

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・クライアントを介して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、134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、147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、152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・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30 に、ユーティリティ・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・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・クラスは、「サーブレット (Servlet)」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30. コード・サンプル: サーブレット・ユーティリティ・クラス

たとえば、顧客がアドバンスド版 Application Server で実行されるエンタープライズ Bean のインプリメントをロードする場合は、149ページの図31 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31. コード・サンプル:エンタープライズ 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 151ページの図32 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 仕様のバージョン 1.0 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 のプロパティには、Bean 自体から直接アクセスすることができます。

151ページの図32 に、jsp:UseBean タグを使用してコマンド Bean のプロパティ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 32. 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

153ページの図33 に、WebCommand の構造を示します。

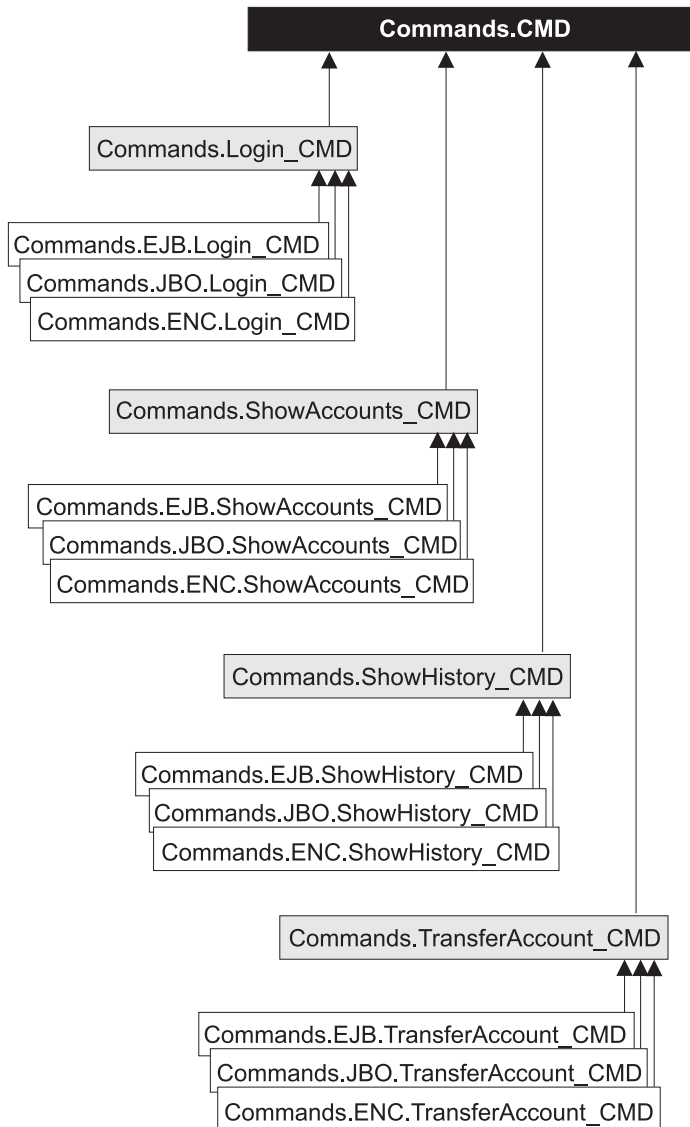


図 33. WebCommand 継承構造

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 で、ビジネス・ロジックのエンタープライズ Bean 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・ロジックのエンタープライズ 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 またはエンタープライズ 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。サブレットは、147ページの『サブレット』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 との対話

WebCommand の perform メソッドは、EJB サーバーで、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、エンタープライズ Bean 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。

155ページの図34 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 のエンタープライズ 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 が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 34. コード・サンプル: アクセス Bean の WebCommand との使用

エンタープライズ Bean (アドバンスド版)

エンタープライズ Bean は、サンプル・アプリケーションのアドバンスド版 Application Server バージョンのバックエンド処理を行います。

Web サイトのビジネス・ロジックは、アプリケーション・サーバーで実行されるエンタープライズ Bean に常駐しています。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。

- サブレット・エンジンと同じプロセスで実行する。
- 同じマシンの他のプロセスで実行する。
- 同じ環境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 Bean を操作します。エンティティ 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、152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 の 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。

このタイプのエンタープライズ 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 Bean のインプリメント

3 つのエンティティ Bean (Customer、BankAccount、および TranRecord) は、銀行の異なるエンティティを表します。Customer および TranRecord エンティティは、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 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 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。

このタイプのエンタープライズ Bean に関する詳細については、46ページの『エンティティ Bean』を参照してください。

アクセス Bean のインプリメント

すべてのエンタープライズ Bean は、アクセス Bean でラップされます。アクセス Bean は、エンタープライズ 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、エンタープライズ Bean に直接アクセスしません。アクセス Bean を使用して、エンタープライズ Bean 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) が、サンプル・アプリケーションのエンタープライズ Bean インプリメントで使用されます。

セッション Bean の 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 Bean の 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、エンタープライズ 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35 に、CopyHelper アクセス Bean でラップされるエンティティ 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 35. コード・サンプル: エンティティ 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 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、エンタープライズ Bean 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。図36 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36. 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。

エンタープライズ Bean 間の関連

関連は、エンタープライズ Bean 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 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 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、各銀行口座は、複数のトランザクション・レコードを持っています。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 Bean 間の関連を定義できます。) Java Network Directory Interface (JNDI) を使用して、メソッドは、関連するエンティティ Bean のホーム・インターフェースを一度検索します。後続の検索要求で使用するために、この情報がそのメソッドにより保管されます。

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 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 1 次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 Bean 間の関連に関する詳細については、55ページの『関連』を参照してください。

配置

エンタープライズ Bean は、EJB サーバーのコンテナに配置されます。コンテナは、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。

エンタープライズ Bean、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 (141ページの『エンタープライズ Bean 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、エンタープライズ Bean、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。

エンタープライズ Bean

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++ インプリメントのエンタープライズ Bean は、サンプル・アプリケーションのアドバンスド版 Application Server のエンタープライズ Bean 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 の BankTasks。ビジネス・ロジックを含みます。
- 3 つのエンティティ Bean の 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を表します。
- アクセス Bean。各エンタープライズ Bean をラップします。

エンタープライズ Bean 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、155ページの『エンタープライズ Bean (アドバンスド版)』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 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++ エンタープライズ Bean のインプリメントでは、BMP を持つエンティティ Bean を使用します。アドバンスド版エンタープライズ Bean のインプリメントでは、CMP を持つエンティティ Bean を使用します。

図37 に、パーシスタンスの Encina++ インプリメントを示します。

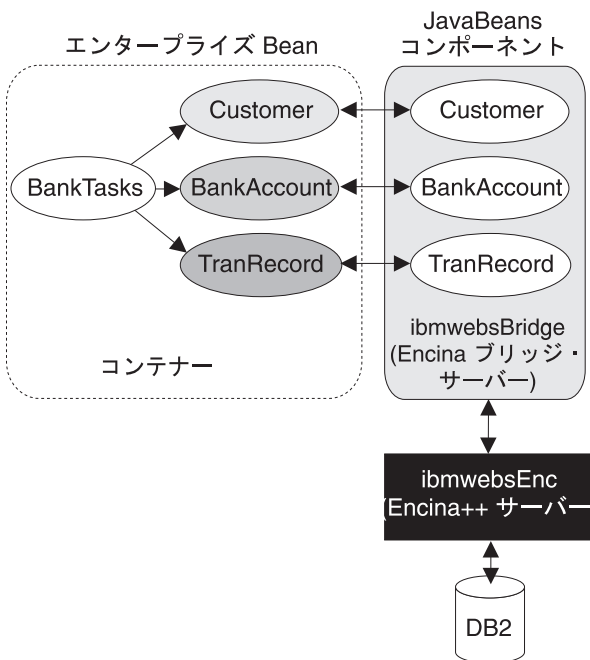


図 37. パーシスタンスの管理

エンティティ 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 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

エンタープライズ Bean 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 Bean は、パー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、エンタープライズ 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 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 Bean と関連付けられます。162ページの表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 Bean の wstidl ファイル

エンティティ 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。

163ページの図38 に、TranRecord エンティティ 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 38. コード・サンプル: *TranRecord* エンティティ・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 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 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 直列化可能ローカル・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 -javabeans \  
-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。-javabeans 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ため、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 サーバー (Encina Server)」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 (162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 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 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 Bean によって開始され、コンテナによって管理されます。コンテナは、Encina ブリッジ・サーバーと協働して、Encina++ アプリケーション・サーバーが関与する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

エンタープライズ Bean 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 Bean を使用するので、コンテナはパーシスタンスを処理しません。

第12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。

- 『Java アプリケーションの MQSeries への接続』
- 169ページの『サンプル・アプリケーションでの WebSphere Edge Server の使用』

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エンタープライズ Bean 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もなりません。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・チャン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、チャン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のと同時に、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、図39 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。

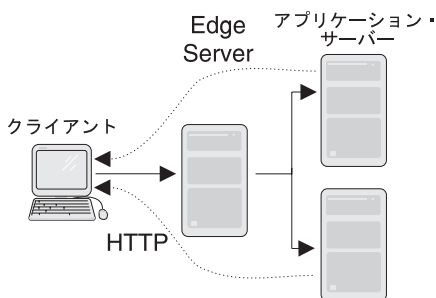


図39. WebSphere Edge Server および Application Server

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、フェールオーバー・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。

次の 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。

- **Caching Proxy** 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。**Caching Proxy** 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。
- **Network Dispatcher** 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。**Network Dispatcher** 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。**Network Dispatcher** 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クローンする場合などに有効です。**Network Dispatcher** 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、プロダクトの資料を参照してください。

特記事項

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

IBM は、本書に記載されている内容に関して特許権（特許出願中のものを含む。）を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権の許諾については、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を提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

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 の場合:

IBM Corporation
Department LZKS
11400 Burnet Road
Austin, TX 78758
U.S.A.

TXSeries の場合:

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。

本書で記述されたライセンス・プログラムおよびそれに関して使用可能なすべてのライセンス資料は、IBM プログラムのご使用条件またはそれと同等な、お客様と IBM の間の契約の条件に基づいて、IBM によって提供されるものです。

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。その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、また

はその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願います。

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ます、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表示されない場合があります。

商標

以下は、IBM Corporation の商標です。

Advanced Peer-to-Peer Networking	MVS/ESA
AFS	NetView
AIX	Open Class
APPN	OS/2
AS/400	OS/390
CICS	OS/400
CICS OS/2	Parallel Sysplex
CICS/400	PowerPC
CICS/6000	RACF
CICS/ESA	RAMAO
CICS/MVS	RMF
CICS/VSE	RISC System/6000
CICSplex	RS/6000
DB2	S/390
DCE Encina Lightweight Client	SecureWay
DFS	TeamConnection
Encina	Transarc
IBM	TXSeries
IBM System Application Architecture	VSE/ESA
IMS	VTAM
IMS/ESA	VisualAge
Language Environment	WebSphere
MQSeries	

Domino、Lotus、および LotusScript は、Lotus Development Corporation の商標です。

Tivoli は、Tivoli Systems, Inc. の商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、の 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. 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。

Copyright 1995, 1996 AT&T/NCR

Copyright 1995, 1996 BNR Europe Ltd.

Copyright 1991, 1992, 1995, 1996 by Digital Equipment Corporation

Copyright 1996 Gradient Technologies, Inc.

Copyright 1995, 1996 Groupe Bull

Copyright 1995, 1996 Expersoft Corporation

Copyright 1996 FUJITSU LIMITED

Copyright 1996 Genesis Development Corporation

Copyright 1989, 1990, 1991, 1992, 1995, 1996 by Hewlett-Packard Company

Copyright 1991, 1992, 1995, 1996 by HyperDesk Corporation

Copyright 1995, 1996 IBM Corporation

Copyright 1995, 1996 ICL, plc

Copyright 1995, 1996 Ing. C. Olivetti &C.Sp

Copyright 1997 International Computers Limited

Copyright 1995, 1996 IONA Technologies, Ltd.

Copyright 1995, 1996 Itasca Systems, Inc.

Copyright 1991, 1992, 1995, 1996 by NCR Corporation

Copyright 1997 Netscape Communications Corporation

Copyright 1997 Northern Telecom Limited

Copyright 1995, 1996 Novell USG

Copyright 1995, 1996 02 Technolgies

Copyright 1991, 1992, 1995, 1996 by Object Design, Inc.

Copyright 1991, 1992, 1995, 1996 Object Management Group, Inc.

Copyright 1995, 1996 Objectivity, Inc.

Copyright 1995, 1996 Oracle Corporation
Copyright 1995, 1996 Persistence Software
Copyright 1995, 1996 Servio, Corp.
Copyright 1996 Siemens Nixdorf Informationssysteme AG
Copyright 1991, 1992, 1995, 1996 by Sun Microsystems, Inc.
Copyright 1995, 1996 SunSoft, Inc.
Copyright 1996 Sybase, Inc.
Copyright 1996 Taligent, Inc.
Copyright 1995, 1996 Tandem Computers, Inc.
Copyright 1995, 1996 Teknekron Software Systems, Inc.
Copyright 1995, 1996 Tivoli Systems, Inc.
Copyright 1995, 1996 Transarc Corporation
Copyright 1995, 1996 Versant Object Technology Corporation
Copyright 1997 Visigenic Software, Inc.
Copyright 1996 Visual Edge Software, Ltd.

上記の各著作権者は、本書に示す仕様を使用したり、あるいはコンピューター・ソフトウェアをこの仕様に適合させたりしても、これらの著作権者の資料の著作権が侵害されたと見なさないことに同意しています。

本書に記載する情報は正確を期していますが、Object Management Group および上記の各社は、法律上の瑕疵担保責任、商品性の保証および特定目的適合性の保証を含む本書に関するいかなる保証もいたしません。Object Management Group および上記の各社は、本書に含まれる誤り、または本書の提供、適用、もしくは使用に関連して発生した付随的損害その他の拡大損害について、何ら責任を負うものではありません。



このソフトウェアには RSA 暗号コードが組み込まれています。



他の会社名、製品名およびサービス名等はそれぞれ各社の商標または登録商標です。

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]

アクセス Bean 50
 サンプル・アプリケーションでの使用 141
 WebCommand との対話 154
アダプター Bean 65
アドバンスド版 Application Server
 サンプル・アプリケーションでの使用 155
アプリケーション類似性 31
アプリケーション例
 アクセス Bean 157
 アクセス Bean の使用 141
 アドバンスド版 Application Server のインプリメント 140, 155
 インプリメント 139
 エンタープライズ Bean のインプリメント (アドバンスド版 Application Server) 140, 155
 エンタープライズ Bean のインプリメント (Encina++) 141, 159
 オブジェクト・モデル 135
 関連の使用 (エンタープライズ Bean) 158
 銀行口座 136
 銀行取引操作の実行 136
 顧客 136
 サーブレット 147
 シナリオ 129
 設計 133
 データベース 137
 トランザクションの管理 (アドバンスド版 Application Server) 140
アプリケーション例 (続き)
 トランザクションの管理 (Encina++) 142, 165
 トランザクション・レコード 137
 パーシスタンス (アドバンスド版 Application Server) 140
 パーシスタンス (Encina++) 142, 165
 目的 130
 Encina++ のインプリメント (エンタープライズ Bean) 141, 159
 JavaBeans コンポーネント (WebCommand) の使用 152
 JSP ページ 150
 アプレット 34
 MQSeries への接続 167
 委任ポリシー 62
 永続的ネーム・サーバー 58
 エレメント 99
エンタープライズ Bean
 アドバンスド版 Application Server 6, 17
 アプリケーション開発プロセス 41
 および OTS トランザクション・モデル 71
 関連 55
 継承 51
 コンテナーへの配置 36
 サンプル・アプリケーションでの使用 140, 141, 155
 タイプ 38
 トランザクション 68
 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の比較 73
 MQSeries への接続 167
エンタープライズ版 Application Server 85
 アプリケーション開発ツール 89

エンタープライズ版 Application Server 85 (続き)
 サンプル・アプリケーションでの使用 141, 159
 システム管理ツール 88
 ランタイム・アーキテクチャー 87
エンタープライズ版サービス 87
エンティティ Bean 38, 46
 アプリケーションでの使用 39
 関連の定義 55
 継承 53
 サンプル・アプリケーションでの使用 156
 ライフ・サイクル 49
 BMP を持つ 46
 CMP を持つ 46
オブジェクト
 ビジネス・ロジックのインプリメントに使用 135
 MVC アーキテクチャーでの役割 135
オブジェクト・サービス 15
 エンタープライズ Bean の 56
オンライン・バンキングの例 129

[カ行]

開発チームの役割
 エンタープライズ Bean 40
活動化 43
可用性管理 14
関連 55
 エンタープライズ Bean 間 158
 サンプル・アプリケーションでの使用 158
キュー 99
行セット・アクセス Bean 51
 サンプル・アプリケーションでの使用 158
許可サービス 61

クライアント / サーバー
 サンプル・アプリケーションでの
 インプリメント 134
クライアント証明書 82
クライアントの妥当性検査 146
クローン 14
継承
 エンタープライズ Bean コンポー
 ネントへの影響 52
 エンタープライズ Bean での 51
 親と子のインターフェース 52
コネクター Bean 64
コマンド Bean 67
コンテナ 36
 サブレット 30
コンテナ管理の永続性 (CMP) 46,
 159
コンテナによるトランザクション
 の管理 69

【サ行】

サーバー証明書 83
サーバー・グループ 13
サービス品質 66
サブレット 27, 139
 からの JSP ページの呼び出し
 21
キュー 30
グループ 31
サンプル・アプリケーションでの
 使用 147
セキュリティ 81
セッション管理 31, 80
長所 32
プログラミング・モデル 27
ライフ・サイクル 29
ランタイム環境 30
類似性 31
MQSeries への接続 167
MVC アーキテクチャーでの役割
 135
Web アプリケーションでの使用
 77
サブレットのリダイレクト 14
状態管理 66
シン・クライアント 33

スクリプト 22
スクリプトレット 22
ステートフル・セッション Bean 42
ステートレス・セッション Bean 43
セキュリティ
 Web アプリケーションでの 81
セキュリティ・アプリケーション
 59
セキュリティ・コラボレーター
 60
セキュリティ・サーバー 59
セキュリティ・サービス 59
セキュリティ・プラグイン 60
セッション Bean 38, 42
 アプリケーションでの使用 39
 継承 54
 コンポーネント 43
 サンプル・アプリケーションでの
 使用 156
 ステートレスとステートフルの比
 較 42
 ライフ・サイクル 45
 Java クラス 45
セッションの同期化 74
セッション類似性 31
セル 14
セル・マネージャー 15
疎結合アプリケーション 24

【夕行】

第 1 階層 76
第 2 階層 76
第 3 階層 77
多階層アーキテクチャー 24
タスク管理 115
デジタル証明書 82
ディレクトリー・サービス 56
デプロイメント・ディスクリプター
 36
統合サーバー 168
トランザクション
 エンタープライズ Bean での 68
 サンプル・アプリケーションでの
 記録 137
CICS 114

トランザクション (続き)
 EJB トランザクション・モデル
 71
トランザクション処理モニター 94
トランザクション属性 69
トランザクション分離レベル属性
 70
トランザクション・オブジェクト
 73
トランザクション・サービス 68

【ナ行】

ナビゲーター Bean 67
認証サービス 60
ネーミング・サービス 56
ノード 12
ノード・マネージャー 15

【ハ行】

パシスタンス 63
 インプリメント 67
 コンテナ管理の 46, 140
 Bean 管理の 46
非活性化 43
ビジネス・ロジック
 エンタープライズ Bean でのイン
 プリメント 140, 141
 オブジェクト・モデル 135
非表示フィールド 80
ブラウザー
 からの JSP ページの呼び出し
 21
ブリッジ・サーバー 110, 142
 作成 164
 サンプル・アプリケーションでの
 使用 160
プログラム管理 115
ホーム・インターフェース 36
 エンティティ Bean の 47
 継承の影響 52
 セッション Bean の 44

[マ行]

- モデル 14
- モデル・ビュー・コントローラー・アーキテクチャー 77
 - サンプル・アプリケーションでのインプリメント 134
 - サンプル・アプリケーションでの使用 145
- JSP ページでサポートされている 25

[ヤ行]

- 要求のリダイレクト 25

[ラ行]

- ライフ・サイクル
 - エンティティ Bean の 49
 - セッション Bean の 45
- ラップ Bean アクセス Bean 51
 - サンプル・アプリケーションでの使用 157
- リカバリー可能オブジェクト 73
- リモート・インターフェース 37
 - エンティティ Bean の 48
 - 継承の影響 52
 - セッション Bean の 44
- 領域 112
 - アプリケーション・プログラミング 117
- リレーショナル・データベース 17, 91
 - エンタープライズ Bean におけるサポート 63
 - CICS でのサポート 119
- ロケーション・サービス・デーモン 58
- 論理作業単位 114

[ワ行]

- ワークロード管理 14
 - CICS 125

[数字]

- 1 次キー 48
- 1 次キー・クラス 48
 - 継承の影響 53
- 2 階層アーキテクチャー 23
- 3 階層アーキテクチャー
 - Web アプリケーションでの 76

A

- Apache Web サーバー 8

B

- BankAccount オブジェクト 136
 - エンタープライズ Bean のインプリメント (アドバンスド版 Application Server) 140, 156
 - エンタープライズ Bean のインプリメント (Encina++) 142, 159
- BankTasks オブジェクト 136
 - エンタープライズ Bean のインプリメント (アドバンスド版 Application Server) 140, 156
 - エンタープライズ Bean のインプリメント (Encina++) 142, 159
- Bean 管理の永続性 (BMP) 46, 160
- Bean クラス
 - エンティティ Bean 47
 - 継承の影響 53
 - セッション Bean 43
- Bean によるトランザクションの管理 68

C

- CCF 65
 - VisualAge for Java ツール 67
- CDS 97
- CGI 23, 32, 76
- CICS 85, 112
 - アプリケーションの開発 117
 - インターネット・アプリケーション 119
 - 管理 124
 - クライアント 122

- CICS 85, 112 (続き)
 - クライアント・インターフェース 118
 - ゲートウェイ 124
 - システム間通信 121
 - 領域 112
 - ワークロード管理 125
- CICS link for Lotus Notes 124
- CICS トランザクション・ゲートウェイ 124
- Cookies 80
- CopyHelper
 - WebCommand との対話 154
- CopyHelper アクセス Bean 51
 - サンプル・アプリケーションでの使用 157
- CORBA
 - CosNaming ネーミング・サービス 57
- Customer オブジェクト 136
 - エンタープライズ Bean のインプリメント (アドバンスド版 Application Server) 140, 156
 - エンタープライズ Bean のインプリメント (Encina++) 142, 159
- D
- DB2 91, 137, 140
- DCE 93, 97
- DE-Light 109
- E
- EAB 67
- EJB 環境 35
 - オブジェクト・サービス 56
 - クライアント・ビュー 35
- EJB クライアント 35
- EJB サーバー 59
- EJB サーバー環境 36
- EJB 仕様
 - 機能拡張 50
- EJB トランザクション 71
- Encina 86, 93
 - アプリケーション開発 98
 - 管理 94

Encina 86, 93 (続き)
サーバー・ウィザード 98
ツールキット 107
トレース・ツール 98
COM ウィザード 98
DE-Light 109
Encina++ 104
PPC 102
RQS 99
SFS 100
WebSphere アドバンスド版と
Encina とのインターオペラビリティ 110
「Encina サーバー (Encina Server)」
ウィザード 165
Encina ツールキット 107
Encina モニター 94
ランタイム環境 95
Encina++ 104
サンプル・アプリケーションでの
使用 141, 165

I

IBM Enterprise Access Builder 90
IBM HTTP Server 139
IBM TeamConnection 90
IBM VisualAge for Java エンタープライズ版 89
IDL
サンプル・アプリケーションでの
使用 162

J

J2EE 2
Java メッセージング・サービス
(JMS) 168
JavaBeans コンポーネント
アドバンスド版 Application
Server 6, 17
サンプル・アプリケーションでの
使用 152
JSP ページと共に使用する 26
MVC アーキテクチャーでの役割
135

JavaBeans コンポーネント (続き)
WebCommand 147
wstidl コマンドでの生成 142
JDBC 36
JNDI 36, 56
JSDK 28
JSP タグ 23
JSP ページ 19, 139
アドバンスド版 Application
Server 6, 17
開発における役割 26
サンプル・アプリケーションでの
使用 150
スクリプト 22
セキュリティ 81
長所 25
プログラムのフロー 20
呼び出し 21
MVC アーキテクチャーでの役割
135
Web アプリケーションでの使用
77
jsp:UseBean タグ 150

L

LU 6.2 122

M

MQ ベース Java 167
MQSeries
Java API 167

N

Netscape Web サーバー 8

O

OTS 36
OTS トランザクション 71

P

persistor 65
PPC 102

R

RQS 99
エレメント 99
キュー 99

S

SFS 100
SNA 122
SSL 81

T

TIDL 98, 142
サンプル・アプリケーションでの
使用 161
ブリッジ・サーバーでの使用
164
TranRecord オブジェクト 137
エンタープライズ Bean のインプ
リメント (アドバンスド版
Application Server) 140, 156
エンタープライズ Bean のインプ
リメント (Encina++) 142, 159
Transactional-C 98
TRPC 98
TXSeries 85, 93
TX_BEAN_MANAGED 69
TX_MANDATORY 69
TX_NOT_SUPPORTED 69
TX_REQUIRED 70
TX_REQUIRES_NEW 70
TX_SUPPORTS 70

V

VisualAge for C++ プロフェッショナル
版 90
VisualAge for Java 18, 130
継承のサポート 54
VisualAge コンポーネント開発ツール
キット 90, 130

W

WAS アドバンスド版 11, 139
アプリケーション設計 17

- WAS アドバンスド版 11, 139 (続き)
 - サンプル・アプリケーションでの使用 140
 - システム管理ツール 15
 - ランタイム・アーキテクチャー 11
 - Encina との通信 110
 - HTML タグ 6
 - Web サーバー 8
- Web アプリケーション 75
 - 状態の保持 79
- Web サーバーによる認証 79
- Web サイト 139, 145
 - MVC アーキテクチャーでの役割 135
- WebCommand 152
 - 継承階層 152
- WebSphere Application Server 1
 - アドバンスド版 11
 - アドバンスド・シングル・サーバー版 5
 - アドバンスド・デベロッパー版 5
 - アプリケーション例 127
 - エンタープライズ版 85
 - および J2EE 2
 - 使用 130
 - WebSphere Edge Server での使用 169
- WebSphere Application Server アドバンスド・シングル・サーバー版
 - ランタイム・アーキテクチャー 9
- WebSphere Application Server アドバンスド・シングル・サーバー版
 - システム管理 7
- WebSphere Application Server アドバンスド・デベロッパー版
 - システム管理 7
 - ランタイム・アーキテクチャー 9
- WebSphere Edge Server 169
- WebSphere Studio 9, 18, 89, 130
 - サンプル・アプリケーションでの使用 145
- WebSphere アドバンスド版と Encina とのインターオペラビリティ
 - 110
 - サンプル・アプリケーションでの使用 142
- wstidl コマンド 111, 142
 - インターフェースの定義 160
 - 生成されたファイル 161
 - 呼び出し 164

X

- XML 19
 - JSP ページにおける使用 24



Printed in Japan

SD88-7362-02



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