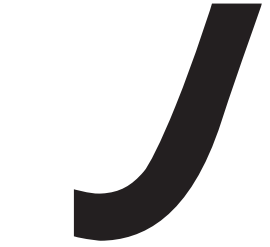


macromedia®
JRUN™4

JRun 入門

パーツ番号 ZJR40M200J

マニュアル制作

プロジェクト管理：Randy Nielsen

執筆：Michael Peterson

編集：Linda Adler、Noreen Maher

日本語版制作管理：Sawako Gensure

日本語版制作・協力：Lionbridge Technologies, Inc.、Bart Vitti、Takashi Koto、Silvio Bichisecchi、Nathalie Delarbre、Akio Tanaka、Masaaki Suga、Yoko Kurihara、Hiroshi Okugawa、IT Frontier, Inc.

初版：2002 年 5 月

Macromedia, Inc.
600 Townsend St.
San Francisco, CA 94103, USA

マクロメディア株式会社
〒 107-0052
東京都港区赤坂 2-17-22
赤坂ツインタワー本館 13 F

目 次

このマニュアルの概要 VII

JRun ドキュメントの概要 viii
印刷版ドキュメントとオンラインドキュメント viii
オンラインドキュメントへのアクセス viii

その他のリソース ix

Macromedia 社へのお問い合わせ xiii

パート I　はじめに 1

第 1 章　JRun へようこそ 3

JRun の概要 4
サーバーサイド Java の利点 4
J2EE を使用する利点 5

J2EE 規格準拠 6

JRun アーキテクチャモデル 8

JRun の機能 9

開発ツール 15
Dreamweaver MX 15
JRun と Java IDE の併用 15
Java IDE へのエンタープライズデプロイウィザードのインストール 15

次のステップ 16

第 2 章　JRun プログラミングモデル 17

エンタープライズアプリケーションのアーキテクチャ 18
エンタープライズアプリケーションの設計 18
J2EE アプリケーションアーキテクチャ 18
JRun の 3 階層モデルのサポート 20

JRun プログラミング環境 21

JRun サーバー 23
JRun サーバーの使用方法 23
JRun サーバーの起動と停止 24
インストール済みの JRun サーバー 25

Web サーバー....28
JRun Web サーバー....28
J2EE アプリケーションの JRun サポート....29
Web アプリケーションと JRun について....30
EJB と JRun について....30
エンタープライズリソースアダプタと JRun について....31
エンタープライズアプリケーションと JRun について....31
第 3 章　J2EE の概要....33
J2EE 環境....34
J2EE プラットフォームのテクノロジー....35
コンポーネント....35
コンテナ....35
通信....36
J2EE API....37
EJB 2.0....37
Java Database Connectivity 2.0....38
Java Servlet 2.3....38
JavaServer Pages 1.2....38
Java Message Service 1.0....38
Java Transaction API 1.0 と Java Transaction Service....38
JavaMail 1.2....39
JavaBeans Activation Framework 1.0....39
Java API for XML 1.1....39
J2EE Connector API 1.0....39
Java Authentication and Authorization Service 1.0....40
Java Naming and Directory Interface....40
Common Object Request Broker Architecture 対応....40
その他のリソース....41
第 4 章　サーブレットと JSP の使用....43
Java サーブレットの使用....44
サーブレットの呼び出し....44
サーブレットの利点....45
サーブレットの作成....46
JRun によるサーブレットのサポート....46
JRun による JSP のサポート....46
サーブレットと JSP....47
HTTP リクエストとレスポンス....47
クライアントへの結果の返送....48
サーブレットフィルタの使用....48
イベントリスナの使用....49
例外処理....49
ページコンテキスト情報の維持....50
セッションの処理....50
アプリケーションコンテキストの追跡....51
設定情報へのアクセス....51
Java によるサーブレットの作成....52
JSP としてのサーブレットの作成....53

第 5 章　EJB の概要 55
EJB の概要 56
EJB の構成 57
コンテナサービス 58
EJB のタイプ 59
セッション bean 59
エンティティ bean 60
メッセージ駆動型 bean 61
JRun での EJB の使用 62
スタブレスデプロイ 62
デプロイオプション 62
EJB のクラスタリング 62
XDoclet 63
エンタープライズデプロイウィザード 63
第 6 章　Web アプリケーションの開発 65
Web アプリケーションの概説 66
Web アプリケーションの利点 66
Web アプリケーションとエンタープライズアプリケーションの比較 66
Web アプリケーションの使用 67
Web アプリケーションのディレクトリ構造 68
デプロイメントディスクリプタ (web.xml) 69
Web アプリケーション、JRun サーバー、Web サーバー 71
マッピング 71
アプリケーションのマッピング 72
Web アプリケーションの開発 73
Web アプリケーションの作成 73
Web アプリケーションコンポーネントの追加 73
Web アプリケーションのデプロイ 77
クラスタ全体への Web アプリケーションのデプロイ 77
Web アプリケーションをデプロイ用にパッケージ化 78
パート II　チュートリアル 79
レッスン 1　サーブレットのチュートリアル 81
開発環境の設定 82
JRun 入門 83
JRun サーバーの追加 85
チュートリアルエンタープライズアプリケーションのデプロイ 88
JDBC データソースの追加 89
Compass Travel J2EE アプリケーション 93
ログインサーブレットのコーディング 93
要約 96
次のステップ 96

レッスン 2　JSP のチュートリアル 97
ホームページ JSP の作成 98
Compass Travel チュートリアルアプリケーション 98
JavaBean へのアクセス 101
要約 104
次のステップ 104
レッスン 3　EJB のチュートリアル 105
EJB の使用 106
JSP からの EJB の呼び出し 107
旅行の予約 109
予約用 EJB のコードの検証 109
クレジットカード用 EJB のコードの検証 110
注文用 EJB のコードの検証 110
要約 111
チュートリアルレッスンのまとめ 111
次のステップ 111
レッスン 4　Web サービスのチュートリアル 113
JRun Web サービスの使用 114
TravelNet Web サービスアプリケーション 114
開発環境の設定 115
JRun 入門 116
JRun サーバーの追加 118
TravelNet エンタープライズアプリケーションのデプロイ 121
Web サービスの作成 122
パブリッシュ済み Web サービスからの WSDL の生成 123
Web サービスプロキシクライアントの生成 126
JSP ベースのプロキシクライアントの作成および使用 128
要約 131
索引 133

このマニュアルの概要

JRun 入門 は、Java サーブレット、JavaServer ページ、Enterprise JavaBeans、Web サービスが含まれるアプリケーションを JRun を使用して開発するユーザーを対象としています。このマニュアルの第 I 部では、JRun と J2EE の一般的な概要について説明します。第 II 部には、簡単な J2EE アプリケーションを構築するチュートリアルが含まれています。チュートリアルレッスンでは、JRun サーバーを追加して、サーブレット、JSP、JavaBeans、および EJB のコードを記述します。

ここでは、Web サイト、ドキュメント、テクニカルサポートなど、JRun および Macromedia に関連したリソースの入手方法について説明します。

目 次

- JRun ドキュメントの概要 viii
- その他のリソース ix
- Macromedia 社へのお問い合わせ xiii

JRun ドキュメントの概要

JRun ドキュメントは、JSP 開発者、サーブレット開発者、EJB クライアント開発者、EJB bean 開発者、システム管理者を含むすべての JRun ユーザーにサポートを提供することを目的としています。印刷物で提供されている場合でも、オンラインの場合でも、必要な情報を速やかに探し出せるように構成されています。JRun オンラインドキュメントには、HTML 形式と Adobe Acrobat ファイル形式があります。

印刷版ドキュメントとオンラインドキュメント

JRun のドキュメントセットには、次のドキュメントが含まれます。

マニュアル	説明
JRun インストールガイド	JRun のインストールおよび設定について説明します。
JRun 入門	J2EE の概要、概念、JSP のチュートリアル、サーブレット、EJB、および Web サービスについて説明します。
JRun 管理者ガイド	JRun サーバーを既存の環境に統合する方法について説明します。
JRun プログラマーガイド	JRun を使用して JSP、サーブレット、カスタムタグ、EJB、および Web サービスを開発する方法を説明します。
JRun アセンブルとデプロイガイド	J2EE アプリケーションコンポーネントのアセンブルおよびデプロイの方法を説明します。
JRun SDK ガイド	OEM/ISV のお客様、および JRun で API の埋め込み、カスタマイズ、使用を行う上級ユーザーを対象に情報を提供します。
JSP クイックリファレンス	JSP (JavaServer Pages) のディレクティブ、アクション、およびスクリプト要素の簡単な説明とシンタックスが記載されています。
オンラインヘルプ	JMC ユーザーに、使用上の注意、方法、および概念を提供します。

オンラインドキュメントへのアクセス

すべての JRun ドキュメントは、HTML 形式と Adobe Acrobat ファイル形式でオンラインで利用できます。ドキュメントにアクセスするには、JRun を実行しているサーバー上で <JRun のルートディレクトリ>/docs/dochome.htm ファイルを開きます。<JRun のルートディレクトリ> とは、JRun がインストールされているディレクトリのことです。

Macromedia 社では、JRun の全マニュアルのオンライン版を Adobe Acrobat Portable Document Format (PDF) ファイルで提供しています。PDF ファイルは JRun CD-ROM にも含まれており、オプションで JRun /docs ディレクトリにインストールされます。JRun 管理コンソールのトップページにある製品ドキュメントへのリンクをクリックすると、これらの PDF ファイルにアクセスできます。

その他のリソース

JRun のドキュメントで説明されているトピックの詳細については、次のリソースも参照してください。

書籍

サーブレット、JavaServer Pages、タグライブラリ	
Java Server Pages Application Development	Scott M. Stirling 他著 Sams 刊、2000 年 ISBN：067231939X
< 邦訳 > JSP アプリケーション開発ガイド - 実践的アプリケーションの構築	Ben Forta 監修 ピアソン・エデュケーション刊 ISBN：489471468X
More Servlets and JavaServer Pages	Marty Hall 著 Prentice Hall PTR 刊、2001 年 ISBN：0130676144
Core Servlets and JavaServer Pages	Marty Hall 著 Prentice Hall PTR 刊、2000 年 ISBN：0130893404
< 邦訳 > コア・サーブレット & JSP	Marty Hall 著 ソフトバンクパブリッシング 刊 ISBN：4797314311
Java Servlet Programming, Second Edition	Jason Hunter、William Crawford 著 O'Reilly & Associates 刊、2001 年 ISBN：0596000405
< 邦訳 > Java サーブレットプログラミング	Jason Hunter、William Crawford 著 オライリー・ジャパン 刊 ISBN：4873110718
Java Servlets Developer's Guide	Karl Moss 著 McGraw-Hill/Osborne Media 刊、2002 年 ISBN：0-07-222262-X
Inside Servlets:Server-Side Programming for the Java Platform, Second Edition	Dustin R. Callaway 著 Addison-Wesley 刊、2001 年 ISBN：0201709066

Web Development with JavaServer Pages	Duane K. Fields、Mark A. Kolb 著 Manning Publications Company 刊、2000 年 ISBN：1884777996
<邦訳> JSP による Web 開発 サーブレットアーキテクチャを利用した新しいコンテンツ開発技法	Duane K.Fields、Mark A.Kolb 著 翔泳社 刊 ISBN：4798100048
Enterprise Java Servlets	Jeff Genender 著 Addison-Wesley 刊、2001 年 ISBN：020170921X
Advanced JavaServer Pages	David Geary 著 Prentice Hall 刊、2001 年 ISBN：0130307041
JavaServer Pages (JSP)	Hans Bergsten 著 O'Reilly & Associates 刊、2000 年 ISBN：156592746X
JSP Tag Libraries	Gal Schachor、Adam Chace、Magnus Rydin 著 Manning Publications Company 刊、2001 年 ISBN：193011009X
Core JSP	Damon Hougland、Aaron Tavistock 共著 Prentice Hall 刊、2000 年 ISBN：0130882488
<邦訳> Core JSP	Damon Hougland、Aaron Tavistock 著 ピアソン・エデュケーション 刊 ISBN：4894714574
JSP:Javaserver Pages (Developer's Guide)	Barry Burd 著 Hungry Minds Inc. 刊、2001 年 ISBN：0764535358
Enterprise JavaBeans	
Mastering Enterprise JavaBeans, Second Edition	Ed Roman 著 John Wiley & Sons 刊、2002 年 ISBN：0471417114
Enterprise JavaBeans, Third Edition	Richard Monson-Haefel 著 O'Reilly & Associates 刊、2001 年 ISBN：0596002262.
Professional EJB	Rahim Adatia 他著 Wrox Press 刊、2001 年 ISBN：1861005083

Special Edition Using Enterprise JavaBeans (EJB) 2.0	Chuck Cavaness、Brian Keeton 共著 Que 刊、2001 年 ISBN：0789725673
Applying Enterprise JavaBeans:Component-Based Development for the J2EE Platform < 邦訳 > Enterprise JavaBeans 開発ガイド	Vlada Matena、Beth Stearns 著 Addison-Wesley Pub Co 刊、2000 年 ISBN：0201702673 Vlada Matena、Beth Stearns 著 ピアソン・エデュケーション 刊 ISBN：4894714639
Enterprise Java プログラミング	
Professional Java Server Programming J2EE 1.3 Edition	Subrahmanyam Allamaraju 他著 Wrox Press 刊、2001 年 ISBN：1861005377
Server-Based Java Programming	Ted Neward 著 Manning Publications Company 刊、2000 年 ISBN：1884777716
Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition < 邦訳 > Java 2 Platform, Enterprise Edition アプリケーション設計ガイド	Nicholas Kassem 著 Addison-Wesley 刊、2000 年 ISBN：0201702770 (java.sun.com/j2ee/download.html#blueprints から無償でダウンロードできます。) ピアソン・エデュケーション 刊 ISBN：4894713233 (日本語版は、http://java.sun.com/blueprints/ja/index.html から無償でダウンロードできます。)
Building Java Enterprise Systems with J2EE	Paul Perrone、Venkata S.R. "Krishna" .R. Chaganti 共著 Sams 刊、2000 年 ISBN：0672317958
J2EE: A Bird's Eye View (e-book)	Rick Grehan 著 Fawcette Technical Publications 刊、2001 年 ISBN：B00005BAZV
Java Message Service < 邦訳 > Java メッセージサービス	Richard Monson-Haefel、David Chappell 著 O'Reilly and Associates 刊、2001 年 ISBN：0596000685 Richard Monson - Haefel、David A. Chappell 著 オライリー・ジャパン 刊 ISBN：4873110580

J2EE Connector Architecture and Enterprise Application Integration	Rahul Sharma 他著 Addison-Wesley 刊、2001 年 ISBN：0201775808
Building Web Services with Java:Making Sense of XML, SOAP, WSDL and UDDI	Sim Simeonov、Glen Daniels、他著 Prentice Hall 刊、2002 年 ISBN：0672321815
Architecting Web Services	William L. Oellermann Jr. 著 Apress 刊、2001 年 ISBN：1893115585

オンラインリソース

Java Servlet API	http://java.sun.com/products/servlet
JavaServer Pages API	http://java.sun.com/products/jsp
Enterprise JavaBeans API	http://java.sun.com/products/ejb/
Java 2 Standard Edition API	http://java.sun.com/j2se/
Servlet Source	http://www.servletsource.com
JSP Resource Index	http://www.jspin.com
Server Side	http://www.theserverside.com
Dot Com Builder	http://dcb.sun.com
Servlet Forum	http://www.servletforum.com

Macromedia 社へのお問い合わせ

開発元： Macrodedia, Inc.	600 Townsend Street San Francisco, CA 94103 U.S.A Web：http:// www.macromedia.com
販売元： マクロメディア株式会社	〒 107-0052 東京都港区赤坂 2-17-22 赤坂ツインタワー本館 13F 電話：03-5563-1980 FAX：03-5563-1990 Web：http://www.macromedia.com/jp/
テクニカルサポート	オンライン Web サポートおよび電子メールでのテクニカルサポートを提供させていただいています。ユーザー登録はがき等に記載されている方法にてお問い合わせください。 テクニカルサポートサービスの詳細は、http://www.macromedia.com/jp/support/ をご覧ください。
セールス	製品のライセンス、価格、サポート、トレーニング、コンサルティングなど、OEM/ ホスティングライセンスなどについては、次の連絡先までお問い合わせください。 電話：03-5563-1980 電子メール：service-j@macromedia.com

パート I
はじめに

ここでは、JRun と J2EE の概要を説明します。

JRun へようこそ 3
JRun プログラミングモデル 17
J2EE の概要 33
サーブレットと JSP の使用 43
EJB の概要 55
Web アプリケーションの開発 65

第 1 章
JRun へようこそ

この章では、Macromedia JRun および JRun アーキテクチャの概要について説明します。アプリケーションの開発およびデプロイで使用する、さまざまな機能およびツールについて説明します。また、この章ではさまざまなタイプの JRun ユーザーについて説明し、各タイプのユーザーが補足情報を参照できる JRun ドキュメントを記載します。

目 次

- JRun の概要 4
- J2EE 規格準拠 6
- JRun アーキテクチャモデル 8
- JRun の機能 9
- 開発ツール 15
- 次のステップ 16

JRun の概要

JRun は完全な Java アプリケーションサーバーであり、安全で信頼性のある、拡張可能なサーバーサイド J2EE (Java 2 Enterprise Edition) アプリケーションの開発とデプロイを行います。JRun では、アプリケーションの開発に使用する最新の業界標準規格をサポートしています。アプリケーションは Java サーブレット、JSP (JavaServer Pages)、EJB (Enterprise JavaBeans)、JMS (Java Message Service)、Macromedia Flash ファイル、HTML ページ、イメージ、およびその他のリソースから構成されます。

JRun はさまざまな Windows および UNIX プラットフォームをサポートします。JRun の設計はオープンなので、Apache、Microsoft IIS (Internet Information Server)、NES (Netscape Enterprise Server)、Netscape iPlanet、Zeus など、さまざまな Web サーバーとともに使用できます。JRun を使用すると、ほとんどすべてのプラットフォームに Web サイトをデプロイし、ダイナミックにコンテンツを生成できます。

JRun では、Netscape は Netscape Server API (NSAPI)、Microsoft IIS は Internet Server API (ISAPI)、Apache Web サーバーは Apache 1.3 および 2.0 Dynamic Shared Object (DSO) インターフェイスというように、各 Web サーバーに特有のプラグインメカニズムを使用して Web サーバーに接続します。しかし、JRun を使用したアプリケーションの開発およびデプロイに、別の Web サーバーを用意する必要はありません。JRun には、独自の Web サーバーが組み込まれています。

詳細については、『JRun インストールガイド』を参照してください。

サーバーサイド Java の利点

Java サーブレット、JSP、および EJB は、すべてサーバーサイド Java、つまり、Web クライアントではなくアプリケーションサーバーで実行する Java コードです。

サーバーサイド Java には、Web サーバーアプリケーションの開発に非常に役立つ、重要な機能が数多くあります。例を次に示します。

- **Java 機能の一貫性**　アプリケーションに必要な Java 機能をサーバーでサポートできます。
- **最新の Java テクノロジー**　Java 規格の進化に伴い、サーバーサイドアプリケーションはクライアントの準拠レベルにかかわらず、新しい Java 機能をすぐに利用できます。
- **JVM (Java Virtual Machine) の制御**　サーバーはアプリケーションに必要な JVM を実行します。

サーバーサイド Java は、「Write Once, Run Anywhere (一度記述すればどこででも実行可能)」という Java の目的を完全に実現します。制限や拘束を受けることなく、言語のすべての機能を使用できます。これらのアプリケーションはサーバー上で実行されるため、プラットフォーム、オペレーティングシステム、およびアプリケーションを実行する環境のあらゆる機能も制御できます。

使用する環境が完全に制御されるため、あるプラットフォームで開発したアプリケーションを別のプラットフォームにデプロイして、アプリケーションを両方のプラットフォームで正常に実行できます。

J2EE を使用する利点

J2EE プラットフォームは、多層エンタープライズアプリケーションを実装およびデプロイする単一規格です。J2EE アプリケーションは Java 固有の移植性を利用して、JRun でサポートされているすべてのプラットフォームで実行できます。

J2EE には、アプリケーションプログラマーにとって、次のような多数の利点があります。

- 豊富な標準 API およびサービス
- 高いパフォーマンス
- RDBMS (Relational Database Management Systems) や EIS (Enterprise Information Systems) などのバックエンドインフラストラクチャとの統合
- サードパーティツールや API (Application Programming Interface) を含む、業界サポート

JRun は、次のセクションで説明するように、最新の J2EE プラットフォーム仕様を基に標準化されています。

J2EE 規格準拠

Macromedia JRun は J2EE 準拠アプリケーションサーバーです。JRun は J2EE CTS (Compatibility Test Suite) に合格し、最新の Sun J2EE 1.3 プラットフォーム仕様のすべての機能を備えています。次の表に、1.3 仕様で強化された機能と新機能を示します。

機能	説明
EJB 2.0	分散型コンポーネントの開発を容易にする機能が強化されました。 • CMP (Container Managed Persistence) 2.0 を使用すると、開発者はデータベースとデータベースアクセスコードのいずれにも依存しない、移植可能なアプリケーションを構築できます。 • データベースに依存しない標準的な EJB クエリ言語 (EJBQL) が導入されました。 • 呼び出しアプリケーションと同じコンピュータにある EJB コンポーネントを効果的に起動するローカルインターフェイスが導入されました。
MDB (Message Driven Beans)	EJB 2.0 仕様では、MDB によって JMS (Java Message Service) と EJB の非同期の機能が組み合わされます。MDB は EJB コンテナで実行されるメッセージリスナであり、セキュリティやトランザクションなどのコンテナサービスを使用できます。
JCA (J2EE Connector Architecture) 1.0	ERP (Enterprise Resource Planning) システム、メインフレームトランザクション処理システム、レガシーデータベースシステムなどのエンタープライズ情報システム (EIS) と J2EE サーバーの統合をサポートする標準アーキテクチャを定義します。
JSP 1.2	JSP API に追加された次の機能をサポートします。 • `IterationTag` および `TryCatchFinallyTag` の 2 つの新しいタグタイプ • JSP ページのオーサリングの自動化を容易にする、JSP ページの XML シンタックス (JSPX) • 移植性を高めてオーサリングツールをサポートする、タグライブラリディスクリプタ (TLD) の機能強化
Servlet 2.3	次の 2 つの重要な新機能を導入します。 • **アプリケーションイベント**　アプリケーションの初期化コードで利用され、プリロードされたサーブレットを置き換えることができます。 • **フィルタ**　サーブレットのリクエストの前処理と後処理を可能にします。
JAXP (Java APIs for XML Processing) 1.1	埋め込み可能なアーキテクチャを含む、最新の XML 開発機能をサポートします。サーバーに XML パーサーや XSL プロセッサの選択肢を追加したり、JRun に付属するデフォルトのコンポーネントを書き換えたりすることができます。

機能	説明
Java Message Service (JMS) 1.0	J2EE アプリケーションコンポーネントが、メッセージの作成、送信、受信、および読み取りを行えるようにします。これによって、疎結合で信頼性の高い、非同期の分散型通信を可能にします。
JAAS (Java Authentication and Authorization Service)	J2EE アプリケーションがユーザーを認証し、アクセス制御をモジュール形式で実行するための方法を提供します。

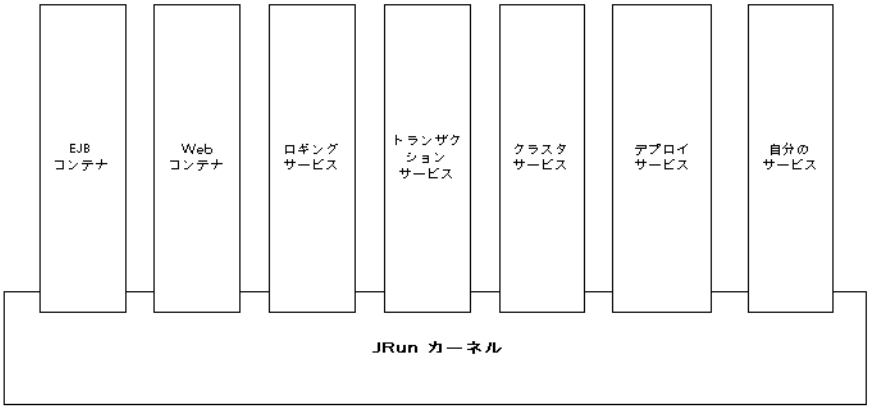
J2EE プラットフォームの詳細については、第 3 章、33 ページの「J2EE の概要」を参照してください。

JRun アーキテクチャモデル

JRun には、コアサーバーのバックボーンに埋め込み可能なサービス (コンポーネント) を含んでいるモジュール型設計である、サービスベースのアーキテクチャがあります。

サーバーのバックボーンを使用すると、埋め込まれたすべてのコンポーネントまたはサービス間の通信が容易になります。サーバーのすべてのコンポーネントと機能に共有されるサービスのコアセットを提供します。バックボーンの柔軟性によって、さまざまなサービスに対応できます。サービスは互いに独立して機能を追加し、コアサーバーを拡張します。

JRun では、コアサーバーのバックボーンはサーバーのカーネルです。次の図に示すように、各サービスが JRun に含まれます。



JRun アーキテクチャは次のような Java 規格に基づいています。

- **JMX (Java Management Extensions)** Java 環境管理の仕様。JMX は、既存の管理システムと統合できる拡張性のあるシステムのアーキテクチャおよび API を定義します。
- **JNDI (Java Naming & Directory Interface)** 分散型コンピューティング環境でアプリケーションコンポーネントとサーバーリソースにアクセスする、Java ベースアプリケーションの標準メカニズム
- **XML** JRun サービスのステートに関する情報を保管する、オープンなクロスプラットフォームの形式設定標準規格

JRun は、JMX サービスベースのアーキテクチャ上に構築されており、JMX を使用した柔軟性の高いダイナミックな管理を提供します。JMX は、管理およびカスタマイズのために登場した Java 規格です。

JRun は、その機能 (EJB コンテナ、Web コンテナ、ロギングなど) を JRun カーネルへプラグインされる JMX サービス (MBeans と呼ばれます) として実装します。これらのサービスは、JMX を使用できる JMC (JRun 管理コンソール)、またはその他の JMX を使用できる管理ツールを使用して管理できます。

サービスは相互に独立していて、別々に再起動できるため、JRun サービスベースのアーキテクチャでは高いアプリケーション利用性が保証されます。また、大幅にカスタマイズ可能で、拡張が容易なプラットフォームを提供します。不要なサービスを取り除いて、使用しない機能のオーバーヘッドを抑えることができます。管理者、上級開発者、および OEM (Original Equipment Manufacturers) は、独自のカスタムサービス (MBeans) を作成し、JRun カーネルにプラグインすることができます。すべての JRun サービスは、JRun カーネルにビルトインされているクラスタリング機能を利用できます。

JRun アーキテクチャの詳細については、『JRun 管理者ガイド』を参照してください。

JRun の機能

JRun 4 では、次のように多くの新機能および拡張機能が提供されています。

- J2EE コンポーネントのダイナミックデプロイ
- サーブレットおよびクラスのダイナミックなリロードとコンパイル
- エンティティ bean 用データベーステーブルのダイナミック作成
- JRun Server タグ (JST)
- JMC (JRun 管理コンソール)
- Web サーバー設定ツール
- クラスタリング
- セキュリティ
- エンタープライズデプロイウィザード
- XDoclet 統合
- Pointbase データベースサーバー
- JDBC ドライバ
- サンプルアプリケーション
- Web サービス

J2EE コンポーネントのダイナミックデプロイ

JRun には、オートデプロイおよびホットデプロイ機能が備わっており、Java アーカイブファイルまたは展開したディレクトリから、Web アプリケーション、EJB、エンタープライズアプリケーション、およびエンタープライズリソースアダプタをダイナミックにデプロイすることができます。J2EE モジュールアーカイブファイルまたはディレクトリをデプロイディレクトリ (デフォルトでは、<JRun のルートディレクトリ >/servers/<JRun サーバー >) にコピーすると、JRun はアプリケーションまたはモジュールを実行中のサーバーに自動的にデプロイするか、サーバーの次回起動時にデプロイします。モジュールアーカイブファイルまたはデプロイメントディスクリプタを変更すると、デプロイはダイナミックに更新されます。詳細については、『JRun アセンブルとデプロイガイド』を参照してください。

サーブレットおよびクラスのダイナミックなリロードとコンパイル

サーブレットまたは JSP が呼び出されたときに、サーブレット、JSP、サーブレットヘルパークラス、および JSP ヘルパークラスをダイナミックに再コンパイルしてリロードするように JRun を設定できます。コンパイルおよびリロード機能が有効になっている場合、JRun はサーブレットが呼び出されたときに自動的にサーブレットを再コンパイルしてリロードします。JRun は、WEB-INF/classes ディレクトリ内のクラスおよびタグライブラリクラスがサーブレットまたは JSP によって呼び出されたときに、これらのクラスをダイナミックに再コンパイルしてリロードします。この機能は、デフォルトで無効になっています。

次のテキストを含む WEB-INF/jrun-web.xml ファイルを作成して、リロードおよびコンパイル値を false または true に設定し、ダイナミックコンパイルおよびリロードを有効にします。

```
<jrun-web-app>
<reload>true</reload>
<compile>true</compile>
</jrun-web-app>
```

詳細については、『JRun アセンブルとデプロイガイド』を参照してください。

エンティティ bean 用データベーステーブルのダイナミック作成

必要とされるデータベーステーブルが存在しない状態でエンティティ bean をデプロイすると、JRun で適切な JDBC (Java Database Connectivity) データソースが設定されていれば、JRun はデータベーステーブルを作成できます。詳細については、『JRun プログラマーガイド』を参照してください。

JRun Server タグ (JST)

JST テクノロジーは JSP 1.2 および再帰をサポートします。JST により、カスタムタグハンドラ API ではなく JSP シンタックスを使用してカスタム JSP タグを記述できます。再帰を使用すると、JST は自分自身を呼び出すことができます。

JMC (JRun 管理コンソール)

再設計された、JMX を使用できる JMC は、ローカルおよびリモート JRun サーバーを管理するために、使いやすく直感的なユーザーインターフェイスを提供します。JMC を使用すると、サーバーを作成し、クラスタを定義し、アプリケーションを管理し、JAAS に基づいたセキュリティを実装できます。詳細については、JMC オンラインヘルプを参照してください。

Web サーバー設定ツール

JRun には、主要な Web サーバー用のネイティブコネクタが用意されています。Web サーバー設定ツールを使用して、各 JRun サーバーに 1 つ以上の Web サーバーを接続できます。スタンドアローンツールである Web サーバー設定ツールでは、Web サーバーホストに JRun を必要としません。外部 Web サーバーへの JRun の接続の詳細については、『JRun インストールガイド』を参照してください。Web サーバーコネクタの詳細については、『JRun 管理者ガイド』を参照してください。

クラスタリング

JRun は、信頼性と拡張性を最大にするためにエンタープライズレベルのサーバークラスタリングを提供します。クラスタリングは、Web サーバーコネクタおよび JRun カーネルにビルトインされています。

- Web サーバーコネクタレベルで、クラスタリングは、Web サーバーと Web コンテナとの間でのロードバランスおよび自動フェイルオーバーを可能にします。インメモリセッションの複製により、Web サーバーがクラスタ内の別の Web コンテナにフェールオーバーした場合にステート情報が保持されます。
- カーネルレベルで、クラスタリングは EJB、JNDI ツリー、クラスタ可能なカスタムサービスなど、クラスタ可能なサーバーオブジェクトに対するロードバランスと自動フェイルオーバーを可能にします。オブジェクトステート (ステートフル EJB のステートなど) は、最適な方法で自動的に複製され、パフォーマンスを維持しながら最高レベルの信頼性を提供します。JRun サーバーは、JINI テクノロジーを使用してピアをクラスタ内にダイナミックに配置するため、1 か所でエラーが発生するリスクを排除することができます。詳細については、『JRun 管理者ガイド』を参照してください。

セキュリティ

JRun は、次のレベルのセキュリティを含む、JAAS 1.0 仕様をサポートします。

- **JRun 管理セキュリティ**　JRun は、セキュリティフレームワークとして JAAS を使用します。JAAS は、JRun がユーザーを認証し、アクセス制御をモジュール形式で実行するためのパッケージセットです。デフォルトの JAAS セキュリティモジュールは、XML ベースのセキュリティ管理 (jrun-users.xml) を提供します。JAAS はプラグ可能なメカニズムを提供するため、LDAP やリレーショナルデータベースなどの既存の認証ユーザーストアと統合するようにシステムをカスタマイズできます。さらに JAAS 仕様をベースに、カスタムセキュリティモジュールを記述できます。

  JRun には、JDBC ベースのログインモジュールと LDAP ベースのログインモジュールが含まれます。これらのモジュールは、リレーショナルデータベースまたは LDAP ディレクトリに格納された情報を使用して認証および承認を実行します。また、JRun は、Windows ドメインのユーザーおよびグループ用のログインモジュールを提供します。詳細については、『JRun 管理者ガイド』または JMC オンラインヘルプを参照してください。
- **Web アプリケーションセキュリティ**　Web アプリケーションに関するセキュリティ問題に対処するため、Java サーブレット API 仕様では、Web アプリケーション内のリソースへのクライアントアクセスを制御する認証メカニズムが定義されています。Web アプリケーションセキュリティでは、承認された Web クライアントのみが Web サイト上のリソースにアクセスできます。Web アプリケーションセキュリティの詳細については、『JRun 管理者ガイド』を参照してください。
- **EJB セキュリティ**　JRun には、EJB 仕様で定義されているように、EJB セキュリティの実装を設定するメカニズムがあります。デプロイメントディスクリプタでは、各 bean メソッドにアクセスできるユーザーまたはロールを指定します。確認するユーザー情報の種類と内容、ロールを構成する情報、および認証方法などを指定できます。このような柔軟性により、既に設置されているセキュリティ方式に対応できます。EJB セキュリティの詳細については、『JRun 管理者ガイド』を参照してください。

エンタープライズデプロイウィザード

JRun には、EJB の開発、パッケージ、およびデプロイを行うためのエンタープライズデプロイウィザードが用意されています。Swing ベースのグラフィカルユーザーインターフェースを使用して、あらゆるタイプの EJB を作成したり、既存の EJB のデプロイメントディスクリプタを編集して JAR ファイルにパッケージし、JRun にデプロイしたりすることができます。とくに、エンタープライズデプロイウィザードのオブジェクト関係マッピング機能を使用すると、エンティティ bean 開発処理を簡素化できます。

エンタープライズデプロイウィザードは、スタンドアローンツールとして実行したり、Java IDE (統合開発環境) の上部でプラグインとして実行することができます。

エンタープライズデプロイウィザードを起動するには、<JRun のルートディレクトリ >/bin/jrunwizard.exe (Windows) または <JRun のルートディレクトリ >/bin/jrunwizard (UNIX) を実行します。

IDE にエンタープライズデプロイウィザードをインストールするには、<JRun のルートディレクトリ >/lib ディレクトリをコンソールウィンドウに表示し、次のコマンドを実行します。

```
java -jar jrunwizard-installer.jar
```

現在サポートされている IDE のリストについては、リリースノートを参照してください。

エンタープライズデプロイウィザードの特徴は、状況に応じて表示内容が変わるオンラインヘルプシステムです。使用法については、オンラインヘルプを参照してください。

XDoclet 統合

JRun は、一般的なオープンソース Java ユーティリティである XDoclet と統合できます。XDoclet は、Javadoc スタイルのタグを使用するコンポーネントメタデータを 1 つのソースファイルで保持でき、J2EE モジュール (EJB、サーブレット、カスタムタブなど) の開発を大幅に簡略化します。

たとえば、EJB を作成する場合、これまでは、ホームインターフェイス、リモートまたはローカルインターフェイス、実装クラス、デプロイメントディスクリプタの 4 つの異なるファイルを保持する必要がありました。XDoclet を使用する場合は、1 つのソースファイル (実装クラス) のみを保持します。このソースファイルに、補助ファイルの構築方法を示す特別な Javadoc タグを使用して注釈を付けます。JRun は、ソースファイルへの変更を検出し、補助ファイル (インターフェイスやデプロイメントディスクリプタなど) を自動的に構築します。詳細については、『JRun プログラマーガイド』を参照してください。

Pointbase データベースサーバー

JRun には、Pointbase Server DBMS (Database Management Software) の制限されたバージョンが含まれています。これは、JRun のサンプルアプリケーションで使用する、すべて Java で構成されたデータベースです。このデータベースサーバーを使用して評価および開発を行うことができます。PointBase データベースの詳細については、<JRun のルートディレクトリ >/pointbase/docs にある PointBase のドキュメントを参照してください。(<JRun のルートディレクトリ > とは JRun をインストールしたディレクトリです。)

JDBC ドライバ

JRun では、主なデータベースに接続する、Type 4 JDBC データベースドライバを含んでいます。

サンプルアプリケーション

samples JRun サーバーには、多数のコーディング技法を紹介するさまざまなアプリケーションが用意されています。次の表は、JRun に付属しているサンプルアプリケーションを示しています。

アプリケーション	説明
Compass travel	オンライン旅行代理店の簡単な旅行予約システム。このアプリケーションは、JSP、サーブレット、および EJB プログラミングを具体的に紹介するもので、オープンディレクトリ構造内のエンタープライズアプリケーションとして配布されます。Compass travel は、compass データベースを使用し、JMC により表示される compass データソースを通じてこのデータベースにアクセスします。付属のチュートリアルレッスンについては、79 ページの「チュートリアル」を参照してください。
TravelNet	Compass travel の旅行を販売するオンラインの旅行代理店。このアプリケーションは、JSP および Web サービスプログラミングについて具体的に紹介するものです、オープンディレクトリ構造内のエンタープライズアプリケーションとして配布されます。付属のチュートリアルレッスンについては、113 ページのレッスン 4 の「Web サービスのチュートリアル」を参照してください。

アプリケーション	説明
World music	音楽を販売する電子商取引アプリケーション。このアプリケーションにも、admin モジュールが含まれています。World music アプリケーションでは、JSP、サーブレット、および JavaBean プログラミングを使用する一般的な設計パターンを多数紹介します。World music サンプルアプリケーションは、worldmusic データベースを使用し、JDBC を通じてこのデータベースに直接アクセスします。
Web サービス	JRun Web サービスプログラミングに使われるプログラミング技法。
Programming techniques	このマニュアルで説明されているプログラミング技法。techniques アプリケーションは、samples データベースを使用し、JMC から表示できる samples データソースを通じてこのデータベースにアクセスします。
Flash ゲートウェイ	JRun と通信する Flash アプリケーションを記述する方法を紹介する、Flash ムービーおよび Flash ゲートウェイアダプタ。
SmarTicket	Java Blueprints J2ME アプリケーション。SmarTicket アプリケーションでは、J2EE プラットフォームを J2ME (Java 2 Micro Edition) プラットフォームと同時に使用して、携帯電話、双方向ポケットベル、パームトップなどのモバイルクライアントデバイスを利用するエンタープライズアプリケーションを作成する方法を示します。 SmarTicket アプリケーションは、smarticket データベースを使用し、JMC から表示できる smarticket データソースを通じてこのデータベースにアクセスします。
Java PetStore	J2EE 1.3 プラットフォーム機能を使用して一般的な電子商取引アプリケーションを開発する方法を示す Java Blueprints J2EE アプリケーションです。これにより、発注とクレジットカード情報の取得および処理、ユーザーログイン、出荷情報、およびショッピングカートセッションの管理を行うことができます。

Web サービス

JRun を使用すると、Web サービスをパブリッシュして使用することができます。Web サービスは、HTTP などの標準インターネットプロトコルと XML を使用して、プラットフォームや場所に依存しないコンピューティングを行います。以前は互換性のなかったアプリケーションを、言語、プラットフォーム、またはオペレーティングシステムに関係なく、Web 上で相互運用できるようにすることで、Web サービスは新しいビジネスチャンスを作り出し、企業がビジネス関係の変化に適応できるようにします。たとえば、Microsoft .NET コンポーネントは、EJB (Enterprise JavaBean) などの J2EE コンポーネントと通信できます。

JRun を使用して、既存の Java コードを Web サービスとして再利用したり、Web サービスとしてパブリッシュするための新規コードを記述したりすることができます。これらのサービスが Java 以外のプラットフォームに存在する場合でも、リモートの Web サービスでメソッドを呼び出すことができるオブジェクトベースおよびタグベースのクライアントを作成することもできます。

JRun の Web サービスの実装は、Apache Software Foundation の SOAP (Simple Object Access Protocol) エンジンの第 3 世代である Apache Axis で構築されます。詳細については、『JRun プログラマーガイド』を参照してください。

JRun 4 機能の詳細については、<JRun のルートディレクトリ >/relnotes.htm にあるリリースノートを参照してください。

開発ツール

Macromedia では、JRun と連動するいくつかの開発ツール (別売) をご用意しています。また、JRun を Java IDE と併用することもできます。

Dreamweaver MX

Dreamweaver MX は、JRun で作成した Web サイトおよび JSP の視覚的な設計と管理のための HTML エディタです。Dreamweaver MX を使用すると、次のコーディングツールと機能によって JSP 開発者の生産性を向上させます。

- **ビジュアルプログラミング**　ソース編集とトランスペアレントに統合される強力な双方向ビジュアルプログラミングツールによって、開発を促進します。
- **HTML デザイン**　数々の賞を受賞した HTML デザインツールである、Macromedia HomeSite のすべての機能が含まれています。
- **JavaScript デバッガ**
- **コードエディタ**　JavaScript、XML、およびその他のテキストドキュメントを編集できるコードビューおよびコードインスペクタ。

JRun と Java IDE の併用

JRun は Java IDE と完全に統合できます。JRun と連動するように IDE を設定するには、次の指定を行います。

- JRun サーバーの `main` メソッドを含んでいるクラス：<JRun のルートディレクトリ >/lib/jrun.jar に含まれている `jrunx.kernel.JRun`

  メモ：<JRun のルートディレクトリ > とは、JRun がインストールされているディレクトリのことです。
- 作業ディレクトリ：<JRun のルートディレクトリ >/bin
- JRun 実行時クラスパスに必要な JAR ファイル：<JRun のルートディレクトリ >/lib および <JRun のルートディレクトリ >/servers/lib にあるすべての JAR ファイル

  メモ：JRun JDBC ドライバは <JRun のルートディレクトリ >/lib/jrun.jar ファイルに含まれているので、特に考慮する必要はありません。
- JRun サーバーの `main` メソッドに渡されるコマンドライン引数：

  ```
  start [<JRun サーバー >]
  ```

  メモ：コンピュータの JRun サーバーを停止してから、IDE 内にある JRun を起動してください。そうしないと、実行中のサーバーとポートが競合するため、JRun は IDE 内で起動しません。

Java IDE へのエンタープライズデプロイウィザードのインストール

IDE にエンタープライズデプロイウィザードをインストールするには、<JRun のルートディレクトリ >/lib ディレクトリをコンソールウィンドウに表示し、次のコマンドを実行します。

```
java -jar jrunwizard-installer.jar
```

現在サポートされている IDE のリストについては、リリースノートを参照してください。

エンタープライズデプロイウィザードの特徴は、状況に応じて表示内容が変わるオンラインヘルプシステムです。使用法については、オンラインヘルプを参照してください。

次のステップ

第 I 部の残りの章では、JRun と J2EE の一般的な概要について説明します。これらの章の内容は次のとおりです。

- 第 2 章、17 ページの「JRun プログラミングモデル」
- 第 3 章、33 ページの「J2EE の概要」
- 第 4 章、43 ページの「サーブレットと JSP の使用」
- 第 5 章、55 ページの「EJB の概要」
- 第 6 章、65 ページの「Web アプリケーションの開発」

JRun のユーザーは、これらの章の情報をよく理解しておく必要があります。

第 II 部には、簡単な J2EE アプリケーションを構築するチュートリアルが含まれています。チュートリアルレッスンでは、JRun サーバーを追加して、サーブレット、JSP、JavaBeans、および EJB のコードを記述します。これらを組み合わせて、柔軟性と拡張性を備えた J2EE アプリケーションを生成する方法を学習します。最後のチュートリアルレッスンでは、Web サービスを使用して JRun サンプルアプリケーションのデータにアクセスします。

次は、JRun アプリケーションの開発者やデプロイ担当者に、第 I 部の後で読むことをお勧めするマニュアルのリストです。

担当者	説明	詳細情報
すべての JRun 開発者	アプリケーションの監視、デバッグ、認証を含む一般的な開発タスクを実行します。	JRun 管理者ガイド
システム管理者	運用環境での JRun のインストールと管理、JRun の起動と停止、およびアプリケーションの追加と削除を行います。	JRun 管理者ガイド
JSP 開発者	クライアントに返されるダイナミックコンテンツを生成する JSP を作成します。これらの JSP は、Java サーブレット、カスタムタグライブラリ、および JavaBeans を参照できます。	JRun プログラマーガイド
Java サーブレットおよびタグライブラリ開発者	Java でサーブレットを開発し、JSP ページで使用するカスタムタグライブラリを開発します。	JRun プログラマーガイド
EJB 開発者	Java サーブレット開発者および JSP 開発者が使用する再利用可能なコンポーネントを作成します。	JRun プログラマーガイド
アプリケーションデプロイ担当者	デプロイと再配布用に JRun アプリケーションをパッケージします。	JRun アセンブルとデプロイガイド

第 2 章
JRun プログラミングモデル

JRun は、J2EE エンタープライズアプリケーションの開発とデプロイを行う完全な Java アプリケーションサーバーです。この章では、JRun プログラミングモデル、JRun の主要なコンポーネント、および J2EE エンタープライズアプリケーションの JRun サポートについて説明します。

目 次

- エンタープライズアプリケーションのアーキテクチャ 18
- JRun プログラミング環境 21
- JRun サーバー 23
- Web サーバー 28
- J2EE アプリケーションの JRun サポート 29

エンタープライズアプリケーションのアーキテクチャ

Web でのエンタープライズアプリケーション開発の標準は、J2EE 仕様に基づいています。JRun は、J2EE アプリケーションモデルをサポートし、エンタープライズアプリケーションを実行する実行時環境を提供します。

エンタープライズアプリケーションの設計

エンタープライズアプリケーションには、次のような共通の設計目的があります。

- **拡張性と保守性**　新規機能の追加および保守が簡単である必要があります。
- **スケーラビリティ、移植性、および可用性**　一般にエンタープライズアプリケーションには多数のユーザーをサポートするスケールが必要です。移植性の高いアプリケーションは、ベンダーの制約とシステム陳腐化のリスクを低減します。高い可用性によって、エンタープライズデータとビジネス機能へのアクセスが維持されます。
- **コードの再利用**　あるアプリケーション用に開発されたコードは、そのアプリケーション内と同様に、他のアプリケーションでも再利用できることが理想的です。
- **相互運用性**　エンタープライズアプリケーションは、そのサービスを使用し、特にレガシーシステムなどの既存の情報資産にサービスを提供することによって、他のシステムと相互運用できる必要があります。
- **ビジネスロジックを重点的に実装**　開発者は、ビジネス上の問題を解決するためのコードの記述に重点を置き、システムメカニズムの処理に費やす時間を最小にする必要があります。

アプリケーションの設計パターンの詳細については、『JRun プログラマーガイド』を参照してください。

エンタープライズアプリケーションを階層に分けることで、情報パーシスタンス、アプリケーション機能、ユーザーとの対話など、設計と実装の問題を区別するレイヤーによって、エンタープライズアプリケーションの設計目的の達成をサポートします。次のセクションでは、J2EE アプリケーションの階層のロール、および各階層で利用できるテクノロジーについて説明します。

J2EE の詳細については、第 3 章、33 ページの「J2EE の概要」を参照してください。

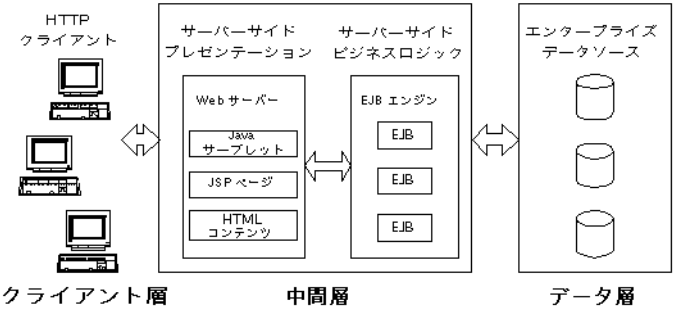
J2EE アプリケーションアーキテクチャ

J2EE アプリケーションは多層として定義されています。各階層には、アプリケーション設計に対するそれぞれの利点があります。階層アーキテクチャには、既存のシステムと今後のシステムを統合するアクセスポイントがあります。J2EE アプリケーションはその機能を複数の階層または機能レイヤーに分割します。分割された機能には、それぞれ特定の目的があります。このモデルでは、互いに独立して実装される別個のコンポーネントに、ビジネス Web サイトの機能を分割します。通常、多層アプリケーションには次のような特徴があります。

- **クライアント層**　ユーザー操作とデータ取得を行います。クライアント層のプログラムは、ユーザーアクションと入力をサーバーリクエストに変換し、サーバーレスポンスの形式を設定します。たとえば、ブラウザを使用してインターネットなどの HTTP 接続で中間層にアクセスします。この階層には、クライアントのマシン上で実行されるアプレットが含まれます。

- **中間層**　Web サイトのビジネスロジック。中間層には、プレゼンテーションロジック (Web 層) と、Web サイトを定義するビジネスルール (EJB 層) の両方が含まれます。JRun を使用して、中間層にアプリケーションを実装します。
  - **Web 層**　Web 上でアプリケーション機能を有効にします。他の階層からデータおよびビジネス機能にアクセスし、ユーザー操作の制御をカプセル化します。Web サービスのシナリオでは、他のアプリケーションがクライアントプログラムの代わりに動作し、Web 層を介して J2EE アプリケーションにアクセスします。
  - **EJB 層**　エンタープライズデータやビジネスルールへの移植性、スケーラビリティ、可用性を備えた高性能アクセスを実現します。オブジェクトパーシスタンスを提供し、EJB として実装されるビジネスロジックにアクセスします。他の階層は、Java RMI (Remote Method Invocation) API を使用して、サーバーサイドのエンタープライズ bean インスタンスにアクセスします。
- **データ層**　J2EE アプリケーションを他のエンタープライズ情報システムと統合します。データ層は、データストレージおよび J2EE アプリケーションへのその他の情報サービスを提供します。通常、データベース、統合業務パッケージシステム、メインフレームトランザクションプロセッサ、レガシーシステム、およびエンタープライズ統合テクノロジーはデータ層にあります。他の階層は、JDBC API をサポートするドライバでデータ層にアクセスし、J2EE コネクタ拡張子、CORBA、またはベンダー固有の専用プロトコルで非データベース EIS リソースにアクセスします

次の図は、3 階層アプリケーションモデルを示します。



この 3 階層アーキテクチャは、Web サイト開発者にとって次のようなメリットがあります。

- 階層または階層のコンポーネントを複数のハードウェアシステムに分散して、システムの拡張性とパフォーマンスを改善できます。
- 中間層によって、クライアントはエンタープライズデータストアにアクセスする複雑さから保護されます。
- EJB は、エンタープライズデータへのアクセスを共有するために複数のアプリケーションが再利用できる、安全なコンポーネントソリューションを提供します。Web アプリケーションは EJB にアクセスできます。また、Web クライアントはこれらに直接アクセスできます。
- コンポーネントアーキテクチャでは、アプリケーション開発を開発グループに分散できます。たとえば、JSP 開発者は通常、ビジネス ルールの実装ではなくプレゼンテーション情報にかかわります。一方、EJB 開発者はデータのアクセスと操作に関与しますが、プレゼンテーションには関与しません。

JRun の 3 階層モデルのサポート

JRun を使用して、3 階層モデルの中間層に J2EE アプリケーションを実装します。JRun では、次のコンポーネントで構成されるエンタープライズアプリケーションを開発する最新の業界標準規格を完全にサポートしています。

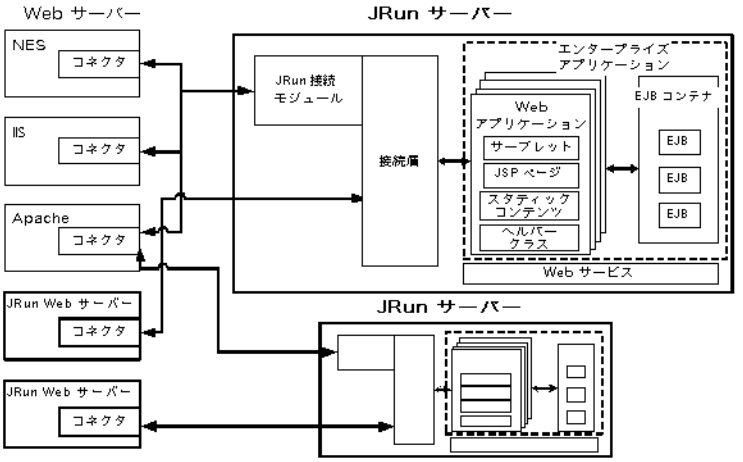
- **Java サーブレット** Java サーブレットは、Web サーバーへのカスタム機能の追加を可能にする、Java 言語で書かれたサーバーサイドコンポーネントです。サーブレットは、HTTP リクエストおよびレスポンスモデルをサポートし、データのアクセスおよび操作に使用する EJB コンポーネントにアクセスできるので、Web ベースアプリケーションに適しています。Java サーブレットの作成の詳細については、第 4 章、43 ページの「サーブレットと JSP の使用」を参照してください。
- **JavaServer Pages** JSP (JavaServer Pages) によって、HTML とスクリプトコードの組み合わせを含んでいるテキストファイルからサーブレットを作成できます。クライアントにより JSP が要求されると、そのページが Java サーブレットに変換されます。JSP のスクリプト部によって、クライアントにダイナミックコンテンツを返すことができます。また、JSP からサーブレット、カスタムタグライブラリ、および EJB にアクセスできます。JSP 作成方法の詳細については、第 4 章、43 ページの「サーブレットと JSP の使用」を参照してください。
- **Enterprise JavaBeans** EJB は、ビジネスロジックを含んでいる再利用可能な Java コンポーネントの開発およびデプロイを可能にします。EJB 仕様は、ソフトウェアコンポーネントモデルを定義します。これにより、EJB 対応アプリケーションサーバーを使用して、サーバーサイドアプリケーションロジック (エンタープライズ bean) をデプロイできます。JRun EJB サポートには、トランザクションとオブジェクトのパーシスタンス、および bean アクセスを制御するセキュリティサービスが含まれます。EJB の作成方法の詳細については、第 5 章、55 ページの「EJB の概要」を参照してください。

JRun を使用すると、EJB として実装したビジネスロジックと Web アプリケーションとして実装したプレゼンテーションロジックを組み合わせて、1 つの J2EE アプリケーションを作成できます。Web アプリケーションは、サーブレットおよび JSP を使用して EJB に含まれるビジネスロジックにアクセスし、ダイナミックコンテンツを Web クライアントに配信します。

JRun プログラミング環境

JRun を使用すると、サーブレット、JSP、および EJB を含むダイナミックな J2EE アプリケーションを開発できます。作成されたアプリケーションは、IIS や Apache などのサードパーティの Web サーバーや、JRun に統合されている JRun Web サーバー (JWS) がホストします。

次の図は、アプリケーションの開発とデプロイのために JRun を使用したシステムを示します。



この図は、JRun システムの 4 つの主要なコンポーネントを示します。

- **JRun サーバー**　サーブレット、JSP、および EJB を含んでいる、J2EE アプリケーションを処理するために必要なサービスと Web サービスを提供します。1 つのシステムで複数の JRun サーバーを作成できます。JRun をインストールして設定するとき、サーバー内の JRun 接続モジュールと通信するように Web サーバーのコネクタを設定できます。Web サーバーのコネクタの詳細については、『JRun 管理者ガイド』を参照してください。
- **Web サーバー**　クライアントのリクエストを受信して、Web コンテンツを含んでいるレスポンスを配信します。このコンテンツは、スタティック Web ページの場合と、Java サーブレットや JSP が生成して JRun が処理するダイナミックページの場合があります。1 つの JRun サーバーに 1 つ以上の Web サーバーを接続できます。逆に、1 つの Web サーバーに 1 つ以上の JRun サーバーを接続することもできます。前の図は、JRun とともに使用できる Web サーバーの例を示します。サポートされているすべての Web サーバーのリストについては、『JRun インストールガイド』を参照してください。

  JRun には、独自の Web サーバーである JWS も組み込まれています。JWS は、すべて Java で構成された、フル装備の高速で軽量な Web サーバーです。これにより、サードパーティ Web サーバーをインストールして設定しなくても、Web アプリケーションの開発、テスト、およびデプロイを行うことができます。

- **エンタープライズアプリケーション**　Web アプリケーションおよび EJB から構成されています。
  - **Web アプリケーション**　Java サーブレット仕様では Web アプリケーションを、サーブレット、JSP、HTML ページなどのスタティックコンテンツ、イメージ、および他のアプリケーションリソースから構成されるものと定義しています。JRun サーバーは、異なる URL にマッピングされる、複数の Web アプリケーションをサポートします。たとえば、JRun では、いくつかのサンプルアプリケーションをホスティングする samples JRun サーバーをインストールします。
  - **EJB**　JRun では、EJB コンポーネントの実行時環境を提供します。EJB 仕様は、ソフトウェアコンポーネントモデルを定義します。これにより、EJB 対応アプリケーションサーバーを使用して、サーバーサイドアプリケーションロジック (エンタープライズ bean) をデプロイできます。JRun EJB コンテナは、コンポーネントのライフサイクル、ネーミング、トランザクション管理、メッセージング、リソース管理、セキュリティ、分散、ステート管理、パーシスタンスなどのサービスを自動化します。
- **Web サービス**　JRun では、HTTP などの標準インターネットプロトコルと XML を使用して、Web サービスをパブリッシュして使用することができます。この Web サービスは、プラットフォームや場所に依存しないコンピューティングを行う分散型ソフトウェアコンポーネントです。JRun を使用すると、既存の Java コードを Web サービスとして再利用したり、Web サービスとしてパブリッシュするための新規コードを記述したりすることができます。これらのサービスが Microsoft .NET などの Java 以外のプラットフォームに存在する場合でも、リモートの Web サービスでメソッドを呼び出すことができるオブジェクトベースおよびタグベースのクライアントを作成することもできます。

次のセクションでは、これらのコンポーネントについて詳しく説明します。

JRun サーバー

JRun サーバーは、サーブレット、JSP、および EJB を含む J2EE アプリケーションを処理するために Web サーバーが必要とするサービスを提供します。JRun サーバーは、Web サーバー処理の外部で独自の処理を実行します。処理ごとに JRun サーバーを実行すると、次の利点があります。

- Web サーバーの安定性が向上します。
- JRun と関係なく Web サーバーを起動し、終了できます。
- Web サーバーを再起動せずにアプリケーションを変更できます。
- 1 つの JRun サーバーが複数の Web サーバーと通信できます。

JRun では 1 つのインストールによって複数の JRun サーバーをサポートできます。複数の JRun サーバーを作成する理由の 1 つは、個別のコンピュータのプロセス内でアプリケーションを分離するためです。たとえば、各 JRun サーバーは、その JRun サーバーのすべてのサーブレット、JSP、および EJB を実行する 1 つの JVM (Java Virtual Machine) と関連付けられています。JVM は JRE とも呼ばるもので、ソフトウェア的に実装された CPU です。これには、Java プラットフォーム用に作成されたプログラムを実行するのに必要なすべての機能が含まれています。アプリケーションを複数の JRun サーバー、すなわち自身の JVM に分離することで、別のアプリケーションが悪影響を与えないようにできます。さらに、各アプリケーションのクラスパス、データソース、EJB、およびその他のリソースをサーバーレベルで定義できます。

アプリケーションを異なる JRun サーバーで実行するもう 1 つの理由は、各 JRun サーバーが独自のユーザー認証メカニズムまたは一連のユーザー認証ルールを実装できるようにするためです。これにより、特定のサーバーの認証設定を利用できます。認証の詳細については、『JRun 管理者ガイド』を参照してください。

JRun サーバーの使用方法

JRun は、JRun サーバーで他の機能の起動、停止、および実行を行うユーティリティを備えています。このセクションは、さまざまな JRun プラットフォームに対応したこれらのユーティリティについて説明します。

Windows に関する検討事項

JRun サーバーは、起動しないとリクエストに応答しません。JRun を Windows NT または 2000 システムにインストールする際に、JRun を Windows NT サービスとして設定できます。サービスを選択した場合、Windows NT または 2000 システムを起動すると、JRun は 3 つのすべてのサーバーを起動します。サービスは、ユーザープロセスとしてではなく、システムプロセスとして実行されます。**[スタート] > [プログラム] > [管理ツール] > [サービス]** にあるサービスコントロールパネルを使用して、JRun の起動、停止、および再起動を行うことができます。Windows NT や 2000 以外の Windows プラットフォーム、または UNIX プラットフォームで JRun を実行する場合は、システムを起動した後で各 JRun サーバーを手動で起動する必要があります。サービスとして実行しない場合、JRun はアプリケーションとして実行されます。

JRun サーバーの起動と停止の詳細については、『JRun 管理者ガイド』または JMC オンラインヘルプを参照してください。

JRun サーバーの起動と停止

JRun サーバーは次の方法で起動および停止できます。

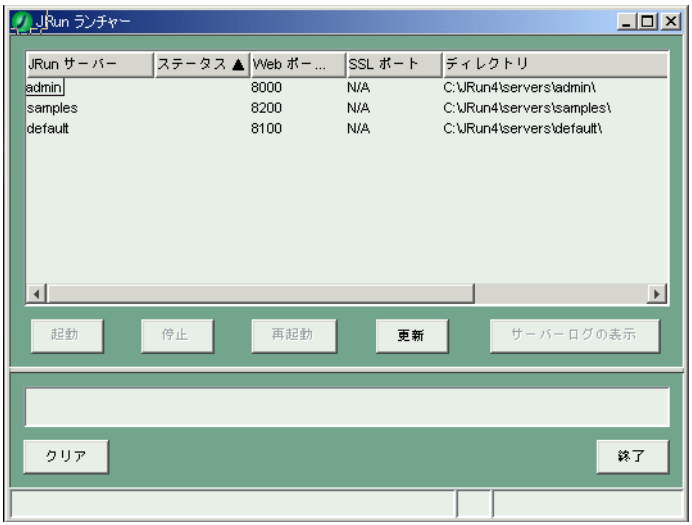
- JRun サーバーランチャー
- コマンドライン

JRun サーバーランチャーの使用

JRun サーバーランチャーは、JRun サーバーを起動、再起動、および停止するための Java Swing アプリケーションです。<JRun のルートディレクトリ >/bin ディレクトリの jrun.exe ファイル (Windows) または jrun 実行可能ファイル (UNIX) を実行して、ランチャーを実行します。ランチャーには、JRun サーバーの起動、再起動、および停止のボタンがあります。

1 [スタート] > [プログラム] > [Macromedia JRun 4] > [JRun ランチャー] を選択します。

[JRun ランチャー] ウィンドウが表示されます。



2 JRun サーバーを選択し、[起動]、[停止]、または [再起動] をクリックします。

コマンドラインの使用

jrun.exe および jrun コマンドラインユーティリティを使用して JRun を起動および停止できます。次の構文を使用します。

```
jrun {options} {server-name}
```

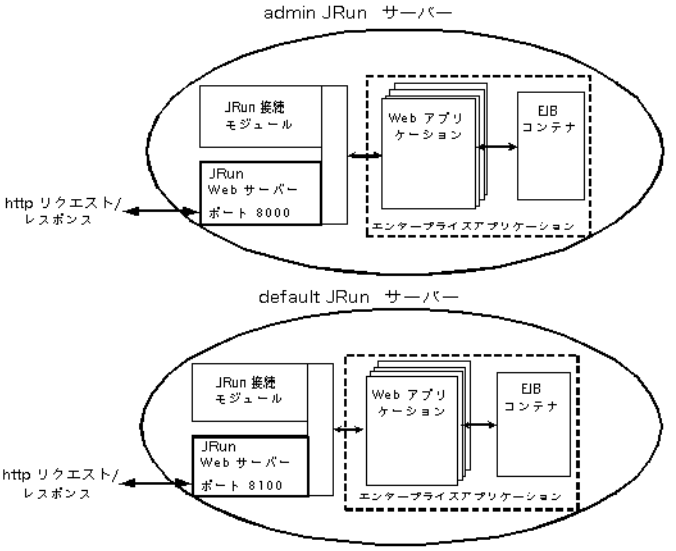
次の表では、オプションについて説明します。

オプション	説明
-start	JRun を起動します。
-stop	JRun を停止します。
-restart	JRun を再起動します。

オプション	説明
-status	すべての JRun サーバーまたは指定した JRun サーバーのステータス情報を表示します。
-nohup	別のプロセスで JRun を起動します。
-config *path-to-jvm.config*	JVM 設定ファイルのパスを指定します。デフォルトである *jrun_root*/bin/jvm.config を上書きし、JVM の設定内容を決定します。
-version	JRun のバージョン番号を表示します (主に OEM で使用)。
-info	追加情報を表示します (主に OEM で使用)。

インストール済みの JRun サーバー

インストール時に、JRun では admin、default、および samples という 3 つの JRun サーバーが作成されます。また、各 JRun サーバーごとに JWS が作成されます。次の図は、admin サーバーと default サーバーの設定を示します。



この図は、JRun の処理ビュー、すなわち各 JRun サーバー処理の内部で実行されるサービスを示しているため、前の JRun システムの図とは異なります。図に示すように、各サーバーに関連付けられた JWS はそのサーバーの処理内で実行されます。

JRun をインストールすると、未使用の Web サーバーのポート番号をデフォルトの位置からダイナミックに割り当てます。次の表は、JRun サーバーのデフォルトの JWS ポート位置をリストします。

JRun サーバー	デフォルトの JWS ポート番号
admin	8000
default	8100
samples	8200

Web サーバーは次の場所でポートを設定できます。

- JRun ランチャーの [Web ポート] の下部
- [JMC へようこそ] ページの [使用可能サーバー] テーブルの [HTTP ポート] の下部
- <JRun サーバー >/SERVER-INF/jrun.xml ファイルの次の行

```
<service class="jrun.servlet.http.WebService"
 name="WebService">
<attribute name="port">8000</attribute>
<attribute name="interface">*</attribute>
</service>
```

JRun ポートの詳細については、『JRun 管理者ガイド』を参照してください。

admin サーバー

admin JRun サーバーは、JRun 管理コンソール (JMC) など、JRun に付属するすべての管理アプリケーションを実行します。このサーバーまたはこのポートの位置で JMC を実行する場合、制限はありません。

Web アプリケーションの場合と同様に、アプリケーションリソースへの HTTP リクエストを作成して、JMC アプリケーションにアクセスします。したがって、admin サーバーは、関連付けられた JWS を実行してこれらのリクエストを処理する必要があります。

デフォルトでは、admin サーバーと関連付けられた JWS は、ポート 8000 を介して受信した HTTP リクエストに応答します。このポート番号へのリクエストは、http://localhost:8000 の後に付けられます。たとえば、JRun 管理コンソールに接続するには、http://localhost:8000 という URL を使用します。

JMC は、**[スタート] > [プログラム] > [Macromedia JRun 4] > [JRun 管理コンソール]** を選択して開くこともできます (Windows のみ)。

default サーバー

default JRun サーバーは、サーブレット、JSP、EJB、および Web アプリケーションの開発、テスト、およびデプロイに必要なサービスを提供します。default サーバーに関連付けられた JWS は、ポート 8100 を介して受信した HTTP リクエストに応答します。このポート番号へのリクエストは、http://localhost:8100 という形式を含んでいます。

デフォルトでは、JRun サーバーが起動すると、対応する JWS も起動します。サードパーティ Web サーバーをデフォルト JRun サーバーと通信するように設定する方法については、『JRun インストールガイド』を参照してください。

samples サーバー

samples サーバーは、http://localhost:8200 で次の JRun サンプルアプリケーションをホスティングします。

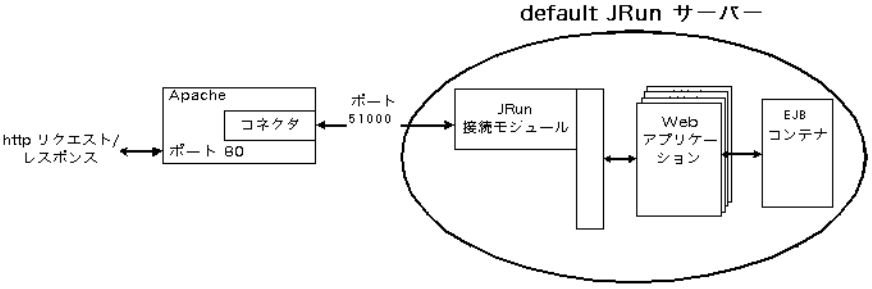
アプリケーション	説明
Compass travel	オンライン旅行代理店の簡単な旅行予約システム。このアプリケーションは、JSP、サーブレット、および EJB プログラミングを具体的に紹介するもので、オープンディレクトリ構造内のエンタープライズアプリケーションとして配布されます。Compass travel は、compass データベースを使用し、JMC により表示される compass データソースを通じてこのデータベースにアクセスします。付属のチュートリアルレッスンについては、79 ページのパート II「チュートリアル」を参照してください。
TravelNet	Compass travel の旅行を販売するオンラインの旅行代理店。このアプリケーションは、JSP および Web サービスプログラミングを具体的に紹介するもので、オープンディレクトリ構造内のエンタープライズアプリケーションとして配布されます。付属のチュートリアルレッスンについては、113 ページのレッスン 4 の「Web サービスのチュートリアル」を参照してください。
World music	音楽を販売する電子商取引アプリケーション。このアプリケーションにも、admin モジュールが含まれています。World music アプリケーションでは、JSP、サーブレット、および JavaBean プログラミングを使用する一般的な設計パターンを多数紹介します。World music サンプルアプリケーションは、worldmusic データベースを使用し、JDBC を通じてこのデータベースに直接アクセスします。
Web サービス	JRun Web サービスプログラミングに使われるプログラミング技法。
Programming techniques	このマニュアルで説明されているプログラミング技法。techniques アプリケーションは、sample データベースを使用し、JMC から表示できる sample データソースを通じてこのデータベースにアクセスします。
Flash ゲートウェイ	JRun と通信する Flash アプリケーションを記述する方法を紹介する、Flash ムービーおよび Flash ゲートウェイアダプタ。
SmarTicket	Java Blueprints J2ME アプリケーション。SmarTicket アプリケーションでは、J2EE プラットフォームを J2ME (Java 2 Micro Edition) プラットフォームと同時に使用して、携帯電話、双方向ポケットベル、パームトップなどのモバイルクライアントデバイスを利用するエンタープライズアプリケーションを作成する方法を示します。 SmarTicket アプリケーションは、smarticket データベースを使用し、JMC から表示できる smarticket データソースを通じてこのデータベースにアクセスします。
Java PetStore	J2EE 1.3 プラットフォーム機能を使用して一般的な電子商取引アプリケーションを開発する方法を示す Java Blueprints J2EE アプリケーションです。これにより、発注とクレジットカード情報の取得および処理、ユーザーログイン、出荷情報、およびショッピングカートセッションの管理を行うことができます。

Web サーバー

JRun では、Web サーバーを拡張して、JRun サーバーがホスティングするサーブレット、JSP、および EJB によって生成されるダイナミックコンテンツを配信する J2EE アプリケーションを処理できます。Web サーバーは JRun と通信するクライアントとして動作するため、Web サーバーは JRun への接続を確立する必要があります。JRun は、Web サーバーへの接続を作成するネイティブサーバー接続モジュールを提供します。

ネイティブのサーバー接続モジュール、すなわちコネクタは、特定の Web サーバー、ハードウェアアーキテクチャ、およびオペレーティングシステムに対応してコンパイルされています。JRun には、NSAPI、ISAPI、Apache 1.3 および 2.0 DSO インターフェイス用のコネクタがあり、JRun をサポートするハードウェアアーキテクチャおよびオペレーティングシステムごとに、NES、IIS、および Apache Web サーバーをサポートします。

各 JRun サーバーには複数の Web サーバーを接続できます。通常のデプロイ環境では、アプリケーションを処理する JRun サーバーに、1 つの Web サーバーを接続します。次の図は、1 つの JRun サーバーに接続した 1 つの Web サーバーを示します。



アプリケーションリソースのリクエストが作成されると、Web サーバー上のコネクタは、JRun サーバー内に常駐する JRun 接続モジュールへのネットワーク接続を開きます。接続モジュールは、トランスペアレントなコミュニケータとして動作し、コネクタから JRun サーバーにリクエストを変換します。JRun サーバーはリクエストを処理し、接続モジュールサービスにレスポンスを返します。たとえば、各 JRun サーバーは、異なるネットワークポート番号を使用して、Web サーバーからのリクエストをリスンします。前の例では、default JRun サーバーはポート 51000 をリスンします。

JRun には、Web サーバーと JRun サーバーの接続を調整するための追加パラメータがいくつかあります。接続および調整パラメータの詳細については、『JRun 管理者ガイド』を参照してください。

JRun Web サーバー

JRun には、すぐに使用できる Java Web サーバーが用意されています。このため、既存の Web サーバーがない場合でも、J2EE アプリケーションの開発を開始できます。したがって、開発チームでビルトイン JRun Web サーバー (JWS) を使用して、サーブレットの作成、テスト、およびデバッグを行い、その後、互換性のある運用サーバーにデプロイできます。

JWS は HTTP バージョン 1.0 サーバーです。ほとんどの場合、サードパーティ Web サーバーを、現在活動中の Web サイト上にある JRun と組み合わせて使用します。

J2EE アプリケーションの JRun サポート

JRun は、次の J2EE モジュールを含む J2EE アプリケーションを開発するための最新の業界標準規格を完全にサポートしています。

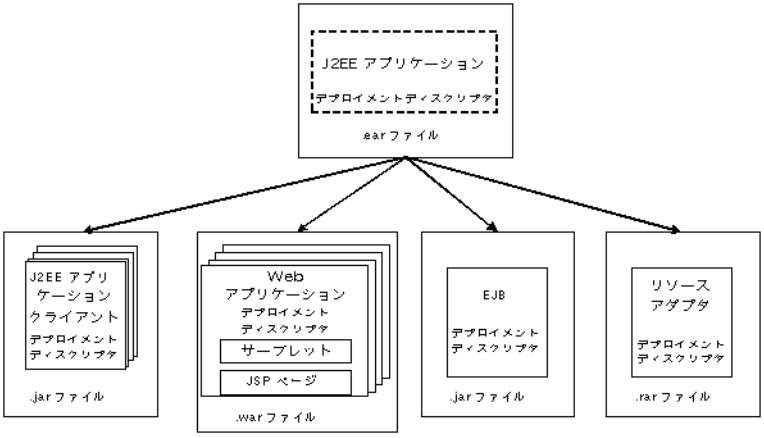
- Web アプリケーション
- EJB
- エンタープライズリソースアダプタ
- エンタープライズアプリケーション

JRun を使用すると、展開したディレクトリやアーカイブファイルから J2EE モジュールをデプロイできます。次はこれらのコンポーネントおよびファイルのリストです。

コンポーネント	アーカイブファイル
Web アプリケーション	WAR (.war ファイル)
EJB	JAR (.jar ファイル)
エンタープライズリソースアダプタ	RAR (.rar ファイル)
エンタープライズ アプリケーション	EAR (.ear ファイル)

運用環境では、移植性の高いアーカイブファイルをデプロイできます。一方、開発時には、柔軟性が高く使いやすい、展開したディレクトリをデプロイできます。

次の図は、J2EE エンタープライズアプリケーションのコンポーネントを示します。



コンポーネントをデプロイする前に、適切なディレクトリ構造に構成要素をパッケージし、コンポーネントのデプロイメントディスクリプタを作成する必要があります。各コンポーネントにはデプロイメントディスクリプタがあります。**デプロイメントディスクリプタ**とはコンポーネントを記述する XML ファイルです。たとえば、EJB デプロイメントディスクリプタは、エンタープライズ bean のトランザクション属性とセキュリティ認証を宣言します。この情報は宣言されるので、変更時に bean のソースコードを変更する必要がありません。実行時に J2EE サーバーはこの情報を読み取り、それに従って bean に作用します。

標準デプロイメントディスクリプタに加えて、JRun は Web アプリケーション、EJB、リソースアダプタ、および J2EE クライアントアプリケーションに JRun 固有のデプロイメントディスクリプタファイルを作成します。J2EE クライアントアプリケーションは、JRun 固有の機能の設定に使用します。JRun 固有のデプロイメントディスクリプタ名は、対応する標準デプロイメントディスクリプタのファイル名に接頭辞 jrun- を付けます。

詳細については、『JRun アセンブルとデプロイガイド』および『JRun 管理者ガイド』を参照してください。

Web アプリケーションと JRun について

Java サーブレット仕様で定義されているように、**Web アプリケーション**はサーブレット、JSP、HTML ページ、イメージ、および Web アプリケーションに必要なその他のリソースのコレクションです。JRun が Web アプリケーションの Java サーブレット仕様をサポートしているため、JRun を使用した Web アプリケーションの開発と、この仕様をサポートする Web アプリケーションサーバーへのアプリケーションのデプロイが可能です。

Web アプリケーションの設定は web.xml ファイルのコンテンツによって定義されます。このファイルは標準 Web アプリケーションのデプロイメントディスクリプタファイルです。このファイルには、アプリケーションサーバーがアプリケーションを実行するために必要なすべての情報が含まれています。web.xml に加えて、JRun は JRun 固有のデプロイメントディスクリプタファイルである jrun-web.xml を作成します。このファイルでは、ダイナミックなサーブレットコンパイルの有効化または無効化、リロードなどの、JRun 固有の動作を指定できます。

Web アプリケーションを配布する場合は、展開したディレクトリ構造として配布するか、または WAR (Web ARchive) ファイルと呼ばれる 1 つに圧縮したファイルとして配布することができます。WAR ファイルには、すべてのディレクトリ構造とアプリケーションを定義するすべてのファイルが含まれます。

Web アプリケーションの配布、WAR ファイルの作成、Web アプリケーションのデプロイの詳細については、第 6 章、65 ページの「Web アプリケーションの開発」を参照してください。『JRun アセンブルとデプロイガイド』および『JRun プログラマーガイド』も参照してください。

EJB と JRun について

EJB は、分散型コンポーネントベースのアプリケーションを構築するコンポーネントアーキテクチャを提供します。このアーキテクチャでは、アプリケーションサーバーがトランザクション、セキュリティ、ライフサイクル管理などのエンタープライズレベルの機能をサポートする必要があります。

JRun では、エンティティ、セッション、およびメッセージによる EJB を完全にサポートしているので、ビジネスニーズに応える強力なソリューションの開発およびデプロイを実行できます。

JRun には、EJB の開発、パッケージ、およびデプロイに役立つ 2 つのユーティリティがあります。

エンタープライズデプロイウィザード

エンタープライズデプロイウィザードを使用して、EJB テンプレートコードとデプロイメントディスクリプタを作成し、JRun サーバーに EJB をデプロイします。また、**エンタープライズデプロイウィザード**によってエンティティ bean デプロイメントディスクリプタに CMP 仕様を生成し、CMP 仕様バージョン 1.1 および 2.0 をサポートします。ウィザードはスタンドアローンツールとして実行するか、または Java IDE (Integrated Development Environment) にインストールすることができます。詳細については、『JRun アセンブルとデプロイガイド』または**エンタープライズデプロイウィザード**のオンラインヘルプを参照してください。

XDoclet (EJB および Web アプリケーション用)

JRun は、一般的なオープンソースツールである XDoclet と統合することができます。XDoclet は、エンタープライズ bean 実装ソースファイルの特定の Javadoc コメントに基づいて、EJB インターフェイスとデプロイメントディスクリプタを生成します。XDoclet を使用して、サーブレットまたはタグライブラリソースファイルのコメントに基づいて、Web アプリケーションデプロイメントディスクリプタと JSP タグライブラリディスクリプタを生成することもできます。

JRun では XDoclet を拡張して、JRun 固有のデプロイメントディスクリプタをサポートし、自動コンパイルおよびデプロイを実現します。詳細については、『JRun プログラマーガイド』を参照してください。

エンタープライズリソースアダプタと JRun について

リソースアダプタとは、特定の EIS の J2EE コネクタテクノロジーを実装する J2EE コンポーネントです。リソースアダプタには標準 API が用意されています。J2EE アプリケーションは、標準 API を介して J2EE サーバーの外にある EIS リソースと通信します。

JCA アーキテクチャは、リソースアダプタが、JRun サーバーによって管理されるトランザクション、セキュリティ、リソース管理などの J2EE 製品へのプラグインをサポートする必要があるという、一連の規約を定義します。

リソースアダプタのプロバイダは、標準デプロイメントディスクリプタファイルである ra.xml を作成します。複数の接続を管理する場合に、jrun-ra.xml を使用して、リソースアダプタの JNDI 名、設定プロパティ、セキュリティ情報、およびコネクションプール設定を指定します。

エンタープライズアプリケーションと JRun について

エンタープライズアプリケーションは、1 つ以上の J2EE モジュールとエンタープライズアプリケーションデプロイメントディスクリプタの application.xml で構成されます。これらは、EAR ファイルまたは展開したディレクトリにパッケージされています。

JRun は、J2EE アプリケーションモデルを完全にサポートし、エンタープライズアプリケーションを実行するための実行時環境を提供します。J2EE エンタープライズアプリケーションは標準化されたモジュール化コンポーネントをベースにしています。JRun はエンタープライズアプリケーションにすべてのサービスを提供し、トランザクション管理、ライフサイクル管理、リソースプールなどの詳細なアプリケーション動作を自動的に処理します。

JRun は、エンタープライズアプリケーションの開発およびデプロイを容易にするために、次の機能を提供します。

- エンタープライズアプリケーション EAR ファイルまたは展開したディレクトリをデプロイできます。

- サーバーのデプロイディレクトリにある J2EE モジュールを自動的にデプロイします。
- デプロイディレクトリにある J2EE モジュールの変更をダイナミックに検出し、リデプロイします。

エンタープライズアプリケーションの開発およびデプロイの詳細については、『JRun プログラマーガイド』および『JRun アセンブルとデプロイガイド』を参照してください。

第 3 章
J2EE の概要

この章では、J2EE (Java 2 Platform, Enterprise Edition) とその関連テクノロジーについて説明します。

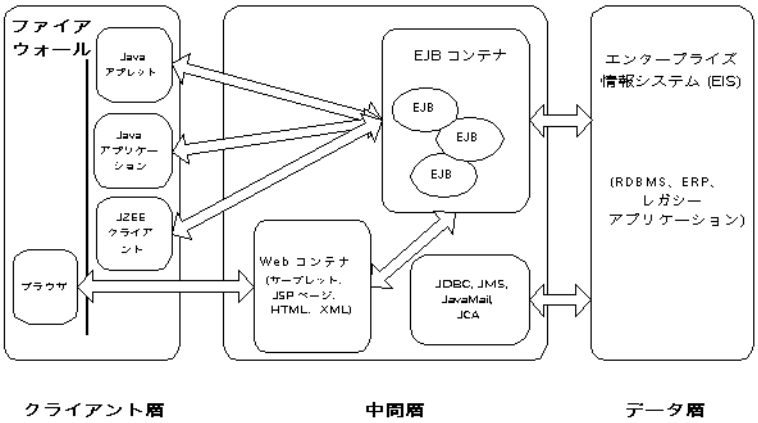
目 次

- J2EE 環境 34
- J2EE プラットフォームのテクノロジー 35
- J2EE API 37
- その他のリソース 41

J2EE 環境

J2EE プラットフォームは、多層エンタープライズアプリケーションを実装およびデプロイする単一規格です。通常、このようなアプリケーションは、ユーザーインターフェイスを提供する**クライアント層**、アプリケーションのクライアントサービスとビジネスロジックを提供する 1 つ以上の**中間層**、およびデータ管理を提供するバックエンドのエンタープライズ情報システム (EIS) から構成される**データ層**として設定されます。

次の図は、これらの階層と、一般的な J2EE 環境を構成するコンポーネントとサービスの一部を示します。



通常、分散型アプリケーションは、ネットワークを介して接続した異なるプラットフォームの、異なる実行時環境で実行される、個別のコンポーネントから構成されています。一般的な分散型アプリケーションには、2 階層 (クライアントとサーバー)、3 階層 (クライアント、ミドルウェア、およびサーバー)、および多層 (クライアント、多層ミドルウェア、および複数のサーバー) があります。

これらの階層には次のような特徴があります。

- **クライアント層**　企業のファイアウォールの内側と外側の両方で、さまざまなクライアントタイプをサポートします。クライアントは、デスクトップ、ラップトップ、PDA、携帯電話、およびその他のデバイスで実行できます。クライアントサイドのユーザーインターフェイスでは HTML および Java アプレットを使用できます。J2EE はスタンドアローンの Java アプリケーションクライアントもサポートします。
- **中間層**　Java サーブレットと JSP を含む Web コンテナとビジネスロジックコンポーネントを介してクライアントサービスをサポートし、EJB コンテナを介してサービスをサポートします。中間層は J2EE サーバーにあります。
- **データ層**　標準 API を介して既存の情報システムへのアクセスをサポートします。データ層には、ERP (Enterprise Resource Planning)、メインフレームトランザクション処理、データベースシステム、その他のレガシー情報システムなどのエンタープライズインフラストラクチャシステムがあります。

J2EE プラットフォームのテクノロジー

J2EE プラットフォームは、テクノロジーを指定して、多層エンタープライズアプリケーションをサポートします。これらのテクノロジーは次の 3 つのカテゴリに分類されます。

- **コンポーネント**　開発者はコンポーネントテクノロジーを使用して、エンタープライズアプリケーションの主な要素であるユーザーインターフェイスとビジネスロジックを作成します。複数のエンタープライズアプリケーション間で共有できる、再利用可能なモジュールとしてコンポーネントを開発します。
- **コンテナ**　コンテナは、アプリケーションコンポーネントに J2EE のシステムレベルのサービスを提供します。コンテナを使用してコンポーネントをカスタマイズし、デプロイされた環境で利用可能なリソースを使用することができます。
- **通信**　ほとんどのエンタープライズアプリケーションは既存のリソースにアクセスする必要があるので、J2EE プラットフォームはデータベース、トランザクション、ネーミングおよびディレクトリ、メッセージングサービス、およびレガシーアプリケーションにアクセスする API をサポートします。J2EE プラットフォームは、クライアントとサーバー間、および別々のサーバーがホストするコラボレーティングオブジェクト間の通信テクノロジーも提供します。

コンポーネント

J2EE コンポーネントとは、関連クラスおよびファイルで J2EE アプリケーションを構成し、他のコンポーネントと通信する自己完結型の機能を持つソフトウェアユニットです。

J2EE プラットフォームは次のタイプのコンポーネントをサポートします。

コンポーネント	内容
クライアント	Web ブラウザ、アプレット、アプリケーション。オプションで JavaBeans も含まれますが、J2EE コンポーネントとは見なされません。
Web	Java サーブレット、JSP。オプションで JavaBeans も含まれますが、J2EE コンポーネントとは見なされません。
ビジネス	EJB (セッション bean、エンティティ bean、メッセージ駆動型 bean)

コンテナ

コンポーネントを実行する前に、コンポーネントを J2EE アプリケーションに組み立て、コンテナにデプロイする必要があります。コンテナは、ライフサイクル管理、セキュリティ、デプロイ、実行時サービスなどの特定のコンポーネントサービスを提供する、標準化された実行時環境です。コンテナの各タイプ (EJB、Web、JSP、サーブレット、アプレット、およびアプリケーションクライアント) はコンポーネント特有のサービスも提供します。

アセンブル処理では、J2EE アプリケーションとアプリケーションそのものの各コンポーネントのコンテナ設定を指定します。コンテナ設定では、セキュリティチェック、トランザクション管理、JNDI (Java Naming and Directory Interface) ルックアップ、リモート接続などのサービスを含む、J2EE サーバーが提供する基本サポートをカスタマイズします。アセンブルまたはデプロイ時のアプリケーションの動作をデプロイメントディスクリプタで指定します。**デプロイメントディスクリプタ**とは、明確に定義された XML タグでコンポーネントの動作を指定するテキストファイルです。

J2EE アプリケーションのアセンブルおよびデプロイの詳細については、『JRun アセンブルとデプロイガイド』を参照してください。

コンテナはすべてのアプリケーションコンポーネントに J2EE プラットフォームの API (Application Programming Interface) を提供します。API の詳細については、37 ページの「J2EE API」を参照してください。

通信

通信テクノロジーは、クライアントとサーバー間、および別々のサーバーがホストするコラボレーティングオブジェクト間の通信メカニズムを提供します。J2EE 仕様では、次のタイプの通信テクノロジーをサポートする必要があります。

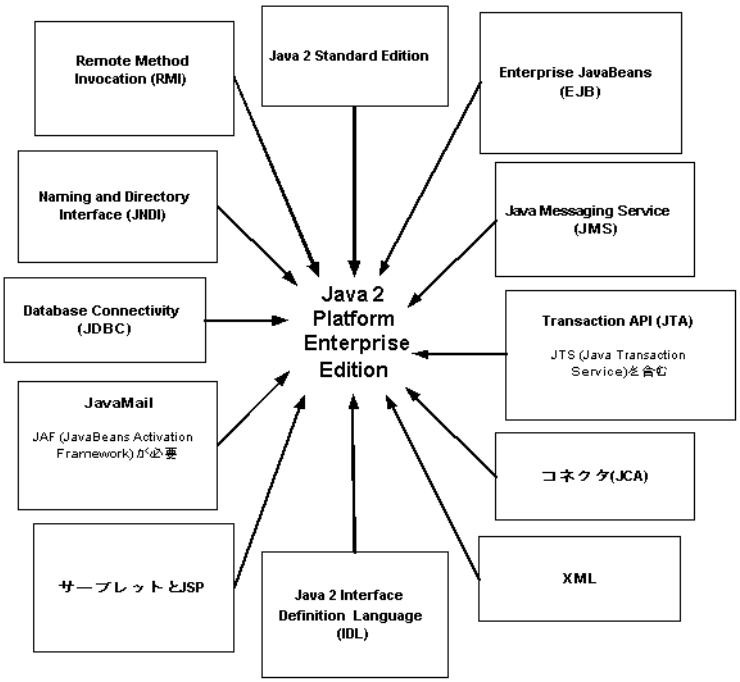
- **インターネットプロトコル** TCP/IP (Transport Control Protocol over Internet Protocol)、HTTP (Hypertext Transfer Protocol) 1.0、および SSL (Secure Socket Layer) 3.0
- **RMI (Remote Method Invocation) プロトコル**
- **OMB (Object Management Group) プロトコル** CORBA (Common Object Request Broker Architecture) テクノロジー、Java IDL (Integration Definition Language)、および IIOP (Internet Inter-ORB Protocol) を介した RMI
- **メッセージングテクノロジー** JMS (Java Message Service)、JavaMail、および JAF (JavaBeans Activation Framework)
- **データ形式** HTML、2 つの形式のイメージファイル (GIF と JPEG)、JAR ファイル、クラスファイル (コンパイル済みの Java ファイル)、および XML

次のセクションでは、J2EE API について詳しく説明します。

J2EE API

J2EE は J2SE (Java 2 Platform, Standard Edition) の既存のテクノロジーをベースにしています。J2SE には、ベース Java サポートと、アプレットおよびアプリケーションをサポートするさまざまなライブラリが含まれています。J2EE 対応アプリケーションは、J2EE および J2SE API の両方を実装します。

次の図は、J2EE のテクノロジーを示します。



次のセクションでは、J2EE プラットフォームの仕様について説明します。

EJB 2.0

EJB (Enterprise JavaBeans) アーキテクチャは、エンタープライズアプリケーションのビジネスロジックを含んでいるコンポーネントの開発とデプロイのための、サーバーサイドテクノロジーです。**エンタープライズ bean** と呼ばれる EJB コンポーネントは、拡張性があり、トランザクション処理が可能で、セキュリティ保護されています。

エンタープライズ bean は、ビジネスロジックのモジュールを実装するフィールドがあるコードの本文です。エンタープライズ bean は、J2EE サーバーでビジネスロジックを実行するときに、単独または他のエンタープライズ bean とともに使用できる構成要素です。

エンタープライズ bean は EJB コンテナがホストします。標準コンテナサービスに加えて、EJB コンテナは幅広いトランザクションおよびパーシスタンスサービスを提供し、J2EE サービスおよび通信 API にアクセスします。

EJB 2.0 仕様ではエンタープライズ bean の 3 つのタイプ、セッション bean、エンティティ bean、および メッセージ駆動型 bean (Message Driven Beans) を定義します。EJB の詳細については、55 ページの「EJB の概要」を参照してください。

Java Database Connectivity 2.0

JDBC (Java Database Connectivity) API は、データアクセスの標準 Java 拡張であり、Java プログラマーは統一されたリレーショナルデータベース API でコーディングすることができます。JDBC を使用して、SQL ステートメントを発行し、Java プログラミング言語メソッドのデータベース結果を処理できます。クライアントは、**JDBC ドライバ**によって実装される JDBC API にプログラミングします。JDBC ドライバは、特定のデータベースと独自の方法で通信するアダプタです。

JDBC API は、アプリケーションがデータベースにアクセスするために使用するアプリケーションレベルのインターフェイスと、JDBC ドライバを J2EE プラットフォームに接続するサービスプロバイダのインターフェイスから構成されます。

Java Servlet 2.3

Java Servlet テクノロジーを使用して、HTTP 固有のサーブレットクラスを定義できます。サーブレットクラスは、リクエスト / レスポンスプログラミングモデルとしてアクセスするアプリケーションをホスティングするサーバー機能を拡張します。サーブレットはすべてのタイプのリクエストに応答できますが、通常は Web サーバーがホスティングするアプリケーションを拡張するときに使用します。

サーブレットの詳細については、第 4 章、43 ページの「サーブレットと JSP の使用」を参照してください。

JavaServer Pages 1.2

JSP (JavaServer Pages) テクノロジーを使用して、サーブレットコードの抜粋をテキストベースのドキュメントに配置できます。JSP ページはテキストベースのドキュメントです。これには、HTML、WML、XML などのテキストベースの形式で表されるスタティックなテンプレートデータと、ダイナミックコンテンツのページの構築方法を指定する JSP 要素の、2 つのタイプのテキストがあります。

JSP の詳細については、第 4 章、43 ページの「サーブレットと JSP の使用」を参照してください。

Java Message Service 1.0

JMS (Java Message Service) API は、J2EE アプリケーションコンポーネントを使用してメッセージの作成、送受信、および読み取りを行うことができるメッセージング規格です。これによって、疎結合で信頼性の高い、非同期の分散型通信を可能にします。非同期にメッセージを渡すことで、最初にメッセージを送信するときにオフラインで処理したり、都合のよいときにメッセージに応答したりすることができます。

Java Transaction API 1.0 と Java Transaction Service

トランザクションとは、実行に関して一連の保証を行う作業単位です。たとえば、トランザクションのスコープ内で実行するコードをすべて実行する、またはまったく実行しないという保証があります。トランザクションでは、システムのステートを一貫して保持する必要があります。トランザクションによって、遠隔地にいる複数のユーザーが同じデータを変更できます。

J2EE アーキテクチャはデフォルトの自動コミットを提供し、トランザクションのコミットとロールバックを処理します。自動コミットとは、各データベースの読み書きオペレーション後に、データを表示する他のアプリケーションが更新済みデータを参照することです。ただし、アプリケーションが互いに依存する 2 つのデータベースアクセスオペレーションを実行する場合は、JTA (Java Transaction API) を使用して、両方のオペレーションを含むトランザクション全体をどこで開始するか、ロールバックするか、またはコミットするかを区別します。

トランザクションを容易にするために、J2EE には JTA と JTS (Java Transaction Service) の 2 つの API があります。JTA とは、アプリケーションがトランザクションを管理する高レベルのトランザクションインターフェイスです。JTS とは、EJB がバックグラウンドで使用する低レベルのトランザクションインターフェイスセットです。クライアントのコードは JTS を直接操作しません。EJB の重要な利点として、EJB コンテナはトランザクションを管理するため、コードにトランザクションロジックを必要としないことが挙げられます。

JavaMail 1.2

J2EE プラットフォームでは、アプリケーションコンポーネントがインターネットメールを送信するときに使用する JavaMail サービスプロバイダに JavaMail API を含めます。J2EE は JavaMail を含めることで、注文確認および他のユーザーのフィードバックを送信する電子商取引 Web サイトなどのアプリケーションをサポートします。JavaMail API は、アプリケーションコンポーネントがメールを送信するときに使用するアプリケーションレベルのインターフェイスと、サービスプロバイダのインターフェイスの 2 つの部分から構成されます。

JavaBeans Activation Framework 1.0

JAF (JavaBeans Activation Framework) は、標準サービスを提供して任意のデータタイプを判断し、そのデータへのアクセスをカプセル化し、そのデータで可能なオペレーションを見つけ、適切な JavaBeans コンポーネントを作成してこれらのオペレーションを実行します。JavaMail では JAF を使用します。

Java API for XML 1.1

XML とは、XML API を使用する任意のプログラムまたはツールでデータを読み取ったり処理したりできるように、テキストベースのデータを表示したり記述したりする言語です。たとえば、J2EE アプリケーションは JAXP を使用してレポートを作成し、そのレポートを受け取る別の企業は、ニーズに合った方法でデータを処理できます。ある企業では、XML データをプログラムで表現し、Web 上に配置するために XML を HTML に変換します。また、他の企業では、処理用の J2EE アプリケーションに XML データを読み込みます。

J2EE Connector API 1.0

J2EE は、**リソースアダプタ**と呼ばれるコネクタを介して、既存のエンタープライズ情報システム (EIS) との統合をサポートします。リソースアダプタは、ベンダー固有のブリッジで、既存のシステムと J2EE をリンクします。

J2EE ツールのベンダーとシステムインテグレータは、JCA (J2EE Connector API) を使用して、任意の J2EE 製品に埋め込まれるレガシーシステムへのアクセスをサポートするリソースアダプタを作成します。リソースアダプタはソフトウェアコンポーネントであり、これによって J2EE アプリケーションコンポーネントは基盤リソースマネージャにアクセスしたり操作したりします。リソースアダプタはリソースマネージャに固有なので、通常、データベースまたは EIS のタイプごとに個別のリソースアダプタがあります。

JCA アーキテクチャは、リソースアダプタが、トランザクション、セキュリティ、リソース管理などの J2EE 製品へのプラグインをサポートする必要があるという、一連の規約を定義しています。

Java Authentication and Authorization Service 1.0

JAAS (Java Authentication and Authorization Service) は、J2EE アプリケーションが特定のユーザーまたはユーザーグループの承認および認証を行う方法を提供します。JAAS は、標準 PAM (Pluggable Authentication Module) フレームワークの Java プログラミング言語バージョンであり、Java 2 プラットフォームのセキュリティアーキテクチャを拡張してユーザーベースの認証をサポートします。

Java Naming and Directory Interface

JNDI (Java Naming and Directory Interface) は、ネーミングサービスとディレクトリサービスの規格です。アプリケーションは JNDI に基づき、ネットワークの分散型コンポーネント (EJB) およびその他のリソース (データソース、リソースアダプタ、JavaMail、JMS など) を検索します。JNDI は JRun アーキテクチャの主要テクノロジーです。

Common Object Request Broker Architecture 対応

J2EE は、JavaIDL と RMI-IIOP という 2 つの CORBA (Common Object Request Broker Architecture) 対応テクノロジーをサポートしています。

CORBA とは、分散型オブジェクトシステムを記述する統一規格です。この規格は、プラットフォーム、言語、およびベンダーから完全に中立しており、上級ミドルウェア開発、言語間のサポート、およびレガシー統合に便利な、さまざまな重要なテクノロジーを組み込みます。

Java Integration Definition Language

Java IDL (Integration Definition Language) を使用して、リモートの CORBA オブジェクトにインターフェイスを定義します。インターフェイスはオペレーティングシステムやプログラミング言語に依存しません。Java IDL により、分散型オブジェクトは CORBA サービス全般を使用できます。J2EE は CORBA と完全に互換性があります。

Java RMI (Remote Method Invocation) と RMI-IIOP

Java RMI (Remote Method Invocation) とは、他のコンピュータからリモートでメソッドを起動するメカニズムです。これを使用すると、プロセス間の通信が可能になり、その他の通信関連サービスも提供されます。RMI-IIOP は移植可能な拡張 RMI であり、通信プロトコルとして IIOP (Internet Inter-ORB Protocol) を使用します。IIOP は J2EE アプリケーションと CORBA システムの統合にも必要です。

その他のリソース

J2EE の詳細については、ix ページの「その他のリソース」を参照してください。

第 4 章
サーブレットと JSP の使用

この章では、Java および JSP を使用したサーブレットの開発について説明します。とくに、両方の環境に共通している概念とオブジェクトについて説明します。また、Java サーブレットと JSP のサンプルコードを提示しながら、サーブレット API の違いについても説明します。

目 次

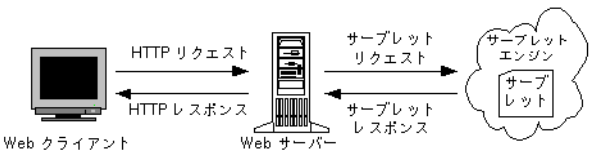
- Java サーブレットの使用 44
- サーブレットと JSP 47
- Java によるサーブレットの作成 52
- JSP としてのサーブレットの作成 53

Java サーブレットの使用

Java サーブレットは、Web サーバーへのカスタム機能の追加を可能にする、Java 言語で書かれたサーバーサイドコンポーネントです。サーブレットは Web サーバーで動作し、パフォーマンス、データベース接続性、安定性、およびセキュリティの面で高い機能を備えています。

特定の Web サーバーの実装のために C++ や Perl で書かれることの多い CGI スクリプトと違い、サーブレットは Java で書かれるため、「Write Once, Run Anywhere (一度記述すればどこででも実行可能)」という Java の利点をはじめ、Java プログラミング言語のすべての利点を活用できます。

サーブレットは HTTP のリクエスト / レスポンスプロトコルをサポートしているため、Web ベースのアプリケーションには特に適しています。次の図は、Java サーブレットの基本的な処理モデルを示しています。



この図に示すように、HTTP リクエストを受け取ると、Web サーバーはそのリクエストがサーブレットを参照しているかどうかを判別し、適切なサーブレットを起動するサーブレットエンジンに転送します。サーブレットはリクエストを処理し、Web サーバーがクライアントに転送するレスポンスを返します。

サーブレットを使用すると、特定の Web サーバーを実装するための複雑な処理や、プラットフォーム固有の動作にわずらわされることなく、最新のアプリケーションを作成できます。サーブレット API では、HTML、フォームデータ、リクエストヘッダー、Cookie などの Web アプリケーションの構成要素がサポートされています。

サーブレットの呼び出し

クライアントは URL を Web リソースとして参照し、サーブレットを呼び出します。URL がサーブレットを参照することは、Web クライアントからはわかりません。

通常、クライアントは次のいずれかの方法でサーブレットを呼び出します。

- **JSP (JavaServer Pages) のアクセス**　JSP は、HTML とスクリプトコードの組み合わせで構成されています (通常は Java または JavaScript で書かれています)。クライアントが初めて JSP ファイルにアクセスすると、このファイルは Java ソースコードに変換されます。ソースコードは Java サーブレットにコンパイルされてからロードされ、実行されます。Web サーバーからは、サーブレットのすべての HTML 出力が Web クライアントに返されます。

  サーブレットの実行可能モジュールがメモリに常駐しているため、JSP ファイルへのそれ以降のクライアントアクセスは非常に効率よく実行されます。これは、JRun がサーブレットイメージをメモリ内で参照できるため、コンパイルとロード手順が省略されることを意味します。

- **サーブレットにマッピングされた URL をダイナミックに参照**　サーブレットはメモリに常駐し、サーブレットに対する後続の呼び出しでは、そのメモリイメージが参照されます。

サーブレットの利点

サーブレットには、従来のサーバーサイドアプリケーション開発テクノロジーと比較すると、Web 開発者にとって多数の利点があります。その中には、Java プログラミング言語に関するものもあれば、サーブレットテクノロジーに関するものもあります。このセクションでは、サーブレットと Java を使用する場合のいくつかの利点について説明します。

サーブレットを使用した場合の利点

サーブレットテクノロジーを使用した場合、Web 開発者には次のような利点があります。

- **セキュリティ**　サーブレットは Web サーバーを介して起動するため、ビジネスロジックがクライアントから直接参照されることはありません。さらに、サーブレットは互いに分離されているため、1 つのサーブレットにエラーが起きても、ほかのサーブレットが破損することはありません。
- **パフォーマンス**　CGI　などの既存のサーバーサイドアプリケーションとサーブレットとの最大の違いは、パフォーマンスです。サーブレットは呼び出されたときに一度だけロードされます。サーブレットはメモリに常駐し、変更されるまでリロードされることはありません。サーブレットを修正した場合は、Web サーバーやアプリケーションを再起動せずにリロードできます。

  さらに、サーブレットはマルチスレッドであるため、サーブレット サーバーのプロセス内で実行されます。それぞれのサーブレットリクエストを処理するのに、プロセスコンテキストスイッチは必要ありません。
- **移植性**　JRun で実行されるサーブレットは、業界標準のサーブレット仕様に準拠しているため、サーブレット標準をサポートしているか、または JRun を使用している Web サーバーであれば移植可能です。移植性はサーブレットベンダにとって重要です。これは、Web サーバーやサーバープラットフォームの種類によって、異なるバージョンのサーブレットを管理する必要がないためです。
- **安定性**　サーブレットは Web サーバーのプロセスの外で実行されます。したがって、サーブレットでエラーが発生しても、影響を受けるのはサーブレットを実行しているプロセスだけです。Web サーバーのプロセスは分離されているため、影響は受けません。
- **ステートのパーシスタンス**　複数のサーブレットでスタティック情報または持続的な情報を共有できます。これにより、複数のユーザー間、またはセッション内で情報を共有できます。

Java を使用する利点

サーブレットの最も重要な利点は、サーブレットの結果が Java プログラミング言語で実装される点です。サーブレットは、Java 本来の移植性を活用するため、すべての Web サーバー、および JRun でサポートされているサーバープラットフォームで実行できます。

Java には、アプリケーションプログラマにとって、次のような多数の利点があります。

- アプリケーションの移植性
- オブジェクト指向プログラミング
- 簡易化されたプログラミングモデル
- マルチスレッドのサポート
- 自動ガーベッジコレクション

サーブレットは Java または JSP を使用して開発するため、Java プログラミング言語のその他の利点も付加されます。

サーブレットの作成

JRun には、サーブレットを作成するための 2 つの方法があります。1 つは Java プログラムを作成する方法で、もう 1 つは JSP を作成する方法です。Java プログラムを作成すると、Java のデータ処理機能と利点を十分に活用できます。通常は、Java を使用して、データベースのアクセスなどの複雑なデータ操作を実行するサーブレットを作成します。

また、HTML とスクリプトコードを組み合わせたサーバーサイドのスクリプトの JSP からもサーブレットを作成できます。JSP ページには、Java プログラミング言語の特性を十分に活用できる機能がありますが、HTML コードに Java コードを組み込むには、簡単なメカニズムを使用します。それらは、クライアントのブラウザに直接返される HTML を生成するサーブレットの実装によく使用されます。

サーブレットを作成する 2 つの方法は、いずれもこのマニュアルで説明しています。

JRun によるサーブレットのサポート

JRun を使用してサーブレットを処理する主な理由の 1 つとして、すべての Web サーバーにサーブレット機能が実装されているわけではないことが挙げられます。JRun により、Web サーバーはサーブレットを処理できるように拡張されます。

Web サーバーにサーブレットの実行機能があったとしても、実装されているサーブレット標準が、そのサーバーやサーバーのホストとなるハードウェアプラットフォームに限定されている場合があります。JRun は、完全に移植性のあるサーブレットソリューションを提供します。JRun を使用して書かれたサーブレットであれば、JRun を使用しているどの Web サーバーでも使用できます。つまり、J2EE サーブレット標準をサポートしていればどの Web サーバーでも使用できます。JRun は、Java サーブレット 2.3 の仕様をサポートしています。この仕様では、サーブレットイベントリスナおよびフィルタという 2 つの重要な機能を採用しています。

JRun には、既存の Web サーバーにアクセスできなくても、サーブレットの開発を始められるように、すぐに使用できる Java Web サーバーが用意されています。このビルトイン JRun Web サーバーを使用して、サーブレットの作成、テスト、デバッグを行ってから、互換性が保証されている実際の運用サーバーにデプロイできます。

JRun による JSP のサポート

JRun による JSP のサポートには、Java Web サーバーによって定義されたページコンパイルのすべての機能と現在の JSP 1.2 仕様が含まれています。JRun では次の機能を提供します。

- JSP 1.2 仕様との完全な互換性
- JSP の `<jsp:useBean>` タグの完全サポート
- 実際のオブジェクト指向型ページ設計用の拡張 JSP のサポート
- 従属ファイルの再帰的コンパイル機能のサポート
- プレゼンテーションテンプレートのサポート
- ほとんどの JVM (Java Virtual Machines) と Java コンパイラのサポート

JSP 1.2 の仕様書には次の機能が追加されています。

- 2 つのタグタイプ：`IterationTag` および `TryCatchFinallyTag`
- JSP ページのオーサリングの自動化を容易にする、JSP ページの XML シンタックス (JSPX)
- 移植性を高めてオーサリングツールをサポートする、TLD (タグライブラリディスクリプタ) の機能強化

詳細については、『JRun プログラマーガイド』を参照してください。

サーブレットと JSP

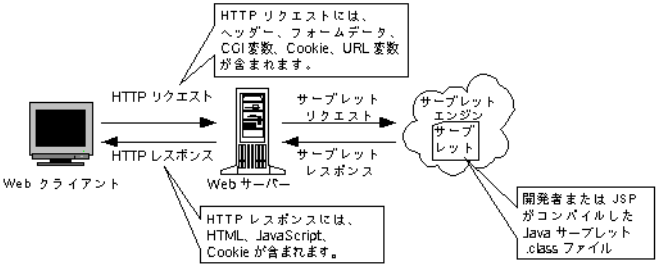
JRun では、サーブレットと JSP を使用してサーバーサイド Java アプリケーションを作成できます。JSP は JRun によってサーブレットに変換されるため、次のようなネイティブサーブレットに使用できるオブジェクトの多くは、本質的に JSP にも使用できます。

- HTTP リクエストとレスポンス
- 出力
- フィルタ
- イベントリスナ
- 例外
- ページコンテキスト
- セッション
- コンテキスト (アプリケーション)
- 設定情報
- アプリケーションのデプロイ (例：WAR ファイル、DTD)

次のページでは、これらのオブジェクトについて説明します。上記オブジェクトの使用方法の詳細については、『JRun プログラマーガイド』を参照してください。

HTTP リクエストとレスポンス

サーブレットは､HTTP リクエストがサーブレットを Java サーブレットとして直接的に参照した場合、または JSP ファイルとして間接的に参照した場合に起動されます。サーブレットは HTTP リクエストに保管された情報にアクセスし、その情報を処理してから、結果を HTTP レスポンスの一部としてクライアントに返します。次の図は、HTTP リクエストおよびレスポンスの構造を示しています。



HTTP リクエストには、クライアントからサーブレットに送信された情報が含まれています。たとえば、サーブレットがフォームによって起動されると、サーブレットは、その処理を行う前に、リクエストに保管されているフォームデータにアクセスする必要があります。フォームデータには、ユーザー、データベースに書き込まれた登録情報、またはユーザーのショッピングカートに追加された製品情報の検証に使用されるログイン情報が含まれています。

HTTP リクエスト情報には次の方法でアクセスできます。

- Java サーブレットでは、`javax.servlet.HttpServletRequest` オブジェクトを使用します。これは、リクエスト内に保管された情報のアクセスに使用できるメソッドを定義するオブジェクトです。
- JSP では、暗黙の JSP オブジェクト `request` を使用します。この request オブジェクトを使用すると、`javax.servlet.HttpServletRequest` オブジェクトの場合と同じメソッドを使用できます。

サーブレットは、HTTP レスポンスを構築し、そのレスポンスをクライアントに返すことによって、リクエストに応じます。サーブレット内で、HTTP レスポンスにアクセスし、クライアントに返されたレスポンス内の情報を書き込みます。

HTTP レスポンス情報には次の方法でアクセスできます。

- Java サーブレットでは、`javax.servlet.HttpServletResponse` オブジェクトを使用します。これは、レスポンス内に保管された情報にアクセスするメソッドを定義するオブジェクトです。
- JSP では、暗黙の JSP オブジェクト `response` を使用します。この response オブジェクトを使用すると、`javax.servlet.HttpServletResponse` オブジェクトの場合と同じメソッドを使用できます。

HTTP レスポンスには、クライアントへの結果の返送に使用する出力ストリームが含まれます。

クライアントへの結果の返送

サーブレットで、リクエストクライアントにダイナミックコンテンツを返すことができます。出力は、サーブレットが計算する情報またはサーブレットに渡された情報に基づいて生成されます。たとえば、サーブレットはフォーマットされたデータベースデータを返すために、渡されたフォーム属性を、request オブジェクトを用いてアクセスして、使用することができます。一方、アプリケーションにユーザープリファレンスを維持するメソッドがある場合は、保管されているプリファレンスに基づき、サーブレットでブラウザの表示色を設定できます。

HTTP レスポンスを使用して、次の方法で情報を返します。

- サーブレットは、`javax.servlet.HttpServletResponse` オブジェクトの `PrintWriter` または `ServletOutputStream` インターフェイスを使用します。これらのインターフェイスには、`println` および `print` メソッドが含まれています。このメソッドは、出力ストリームへの書き込みを行うためのものです。
- JSP には、暗黙の JSP オブジェクト `out` が使用されます。`out` オブジェクトにも、`println` および `print` メソッドが含まれています。

サーブレットフィルタの使用

Servlet 2.3 仕様により、前処理および後処理のための HTTP リクエストおよびレスポンスオブジェクトへのアクセスを提供するサーブレットフィルタが使用できるようになりました。**フィルタ**は、サーバーに送信される前にリクエストオブジェクトを処理したり、サーバーからクライアントに返される間にレスポンスオブジェクトを処理したりします。また、フィルタは、チェーン内で呼び出すこともできるため、チェーン内のフィルタ間でリクエストやレスポンスをやり取りできます。フィルタを使用すると、次のタスクを実行できます。

- Web アプリケーションコンポーネントからフロー制御ロジックを排除します。
- サーバーがクライアントからリクエストオブジェクトを受信する前に、リクエストオブジェクトの評価や修正を行います。

- クライアントがサーバーからレスポンスオブジェクトを受信する前に、レスポンスオブジェクトの評価や修正を行います。
- レスポンスのコンテンツを変更します。

フィルタの使用方法の詳細については、『JRun プログラマーガイド』を参照してください。

イベントリスナの使用

Java Servlet 2.3 仕様の主な新機能の 1 つに、サーブレットイベントリスナの追加があります。この仕様では、Web アプリケーション用のリスナクラスについて規定しています。**イベントリスナ**はイベントハンドラとも呼ばれ、特定のイベントの発生時に JRun が呼び出すコールバックメソッドが用意されています。イベントリスナを使用すると、Web アプリケーションのさまざまな要素を管理し、ServletContext および HttpSession オブジェクト内のステートの変化を追跡できます。

ServletContext および HttpSession オブジェクトのアクティビティを監視するサーブレットイベントリスナには 2 つの基本タイプがあります。次の表に示すように、これらのタイプにはそれぞれ、リスナインターフェイスと属性リスナインターフェイスがあります。

サーブレットオブジェクト	関連付けられているリスナインターフェイス
ServletContext	`ServletContextListener` `ServletContextAttributeListener`
HttpSession	`HttpSessionListener` `HttpSessionAttributeListener`

イベントリスナは、次のようなタスクのほか、多くの一般的なタスクに使用します。

- ロギング
- セッションの管理
- アプリケーションサーバーリソースの追跡

イベントリスナの使用方法の詳細については、『JRun プログラマーガイド』を参照してください。

例外処理

例外とは、サーブレット内で検出されるエラーのことです。例外は、JRun によって JSP が Java クラスファイルに変換されるとき、またはサーブレットが実行されるときに発生する可能性があります。

例外は次の方法で表します。

- Java サーブレットでは、`javax.servlet.ServletException` クラスのインスタンスで例外を表します。
- JSP では、`exception` オブジェクトを使用して例外を表します。

ページコンテキスト情報の維持

JSP の pageContext オブジェクトは、JSP にローカルに情報を保管するメカニズムを提供します。JRun では、ページのリクエストごとに新規 pageContext オブジェクトが作成されます。このオブジェクトは、ページが起動すると作成され、ページが終了すると廃棄されます。pageContext オブジェクトのメソッドを使用すると、JSP の情報にアクセスしたり、他のアクションを実行したりすることができます。

Java サーブレットには、これに相当するオブジェクトはありません。

セッションの処理

HTTP はステートレスプロトコルです。Web サーバーは、リクエストを受け取ってレスポンスを返すと、クライアントとの接続を終了します。Web サーバーにはクライアントの情報は維持されません。そのため、同じクライアントから別のリクエストがきても、それを判断することはできません。Web サイトからクライアントやそのナビゲーションを追跡できないことが、Web サイトにおける複雑なトランザクションの実行を困難にしています。

ただし、JRun では session オブジェクトがサポートされており、これを使用すると、Web サーバーとの対話中にユーザーを追跡できます。session オブジェクトを使用して、ショッピングするユーザーを追跡し、その登録情報や優先情報を送ることができます。ユーザーがサイトに接続するたびに、情報を再入力しなくても済むようにすることもできます。session オブジェクトは、ユーザーが Web サイトに接続している間、情報の保管および検索するための場所を 1 箇所提供します。

単純な実装ではユーザー名、セキュリティ保護されている実装ではユーザー名とパスワード、電子商取引システムではショッピングカートがアプリケーションに記憶されます。サーブレット API を使用しないアプリケーションの場合、通常はアプリケーション固有の保管メカニズムを使用してセッション情報を保管し、非表示のフォームフィールドまたは Cookie を通じてキー値をブラウザに返します。後続のリクエストでは、このキー値を使用して、以前に保存されたセッション情報を取得します。

Cookie はサーバーに保管され、クライアントとサーバー間でやり取りされるセッション ID で参照されます。Cookie は、サーバーサイドのアプリケーションが個別のブラウザに情報を保存するために使用する一般的な手段です。サーバーサイドアプリケーションでは、ブラウザに保管された Cookie を取り出すことができます。Cookie を使用すると、各ブラウザで使用する特殊変数を Web アプリケーションで作成できます。そのような変数にユーザー名または最後にアクセスした日付を含めることができます。Cookie によるセッション記録を有効にした場合、JRun は jsessionid という名前のセッショントラッキング Cookie を作成します (JMC で Cookie の名前を指定することもできます)。Cookie は、ステートレスである HTTP プロトコルを補完するパーシスタンスメカニズムを提供します。

セッション情報には次の方法でアクセスできます。

- Java サーブレットでは、`javax.servlet.http.HttpSession` オブジェクトを使用してセッション情報にアクセスします。
- JSP では、暗黙の JSP Session オブジェクトを使用します。

アプリケーションコンテキストの追跡

コンテキストオブジェクトを使用すると、アプリケーション情報を保管したり、アプリケーションのさまざまなコンポーネント間で情報を共有できます。

たとえば、アプリケーションが、複数のサーブレット (Java で書かれたものと JSP として書かれたもの)、HTML ページ、および他のサーバーサイドコンポーネントで構成されているとします。これらのアプリケーションコンポーネント間でのやり取りを可能にするために、アプリケーションコンテキストオブジェクトを使用して、その情報を保管したり、取り出したりすることができます。コンテキストオブジェクトを介して使用できる情報には、次のものがあります。

- リクエストに渡される属性
- 初期化パラメータ
- MIME タイプ
- バージョン情報
- パス情報

アプリケーション情報には次の方法でアクセスできます。

- Java サーブレットでは、`javax.servlet.ServletContext` オブジェクトを使用します。
- JSP では、暗黙の JSP Application オブジェクトを使用します。

設定情報へのアクセス

JRun は、初期化時にサーブレットに設定情報を渡します。設定情報には、初期化パラメータを示す名前 / 値のペア、およびサーブレットが実行されるコンテキストを示す servletConfig オブジェクトが含まれています。

設定情報には次の方法でアクセスできます。

- Java サーブレットでは、`javax.servlet.ServletConfig` オブジェクトを使用します。
- JSP では、暗黙の JSP Config オブジェクトを使用します。

Java によるサーブレットの作成

次の例は、正常に機能する Java サーブレットの完全なソースコードを示しています。

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SimpleServlet extends HttpServlet {

public void doGet(ServletRequest req, ServletResponse resp)
  throws IOException, ServletException {

  resp.setContentType("text/html");
  PrintWriter out = resp.getWriter();
  out.println("<html><head><title> SimpleServlet Output ");
  out.println("</title></head><body>");
  out.println("<h1> SimpleServlet Output </h1>");
  out.println("</body></html>");
  }
}
```

サーブレットのテストまたはデプロイを行うには、先にコンパイルを実行する必要があります。

リクエストに応じて、サーブレットは HTML タグを含んでいるテキストをクライアントに返します。このサーブレットは比較的シンプルですが、Java サーブレットの基本的な形態と構造を示しています。

Java サーブレットの作成方法に関するチュートリアルについては、81 ページの「サーブレットのチュートリアル」を参照してください。

JSP としてのサーブレットの作成

前のセクションでは、Java で作成したサーブレットの例を示しました。JRun には、Java のコーディングに依存しない、もう 1 つのサーブレットの開発方法、つまり JSP によって、HTML とスクリプトコードの組み合わせを含むテキストファイルからサーブレットを作成できます。

JSP 内のスクリプトコードは、JSP シンタックスと、通常は JavaScript (ECMAScript のサブセット) または Java の組み合わせになります。JSP シンタックス、およびスクリプト言語の選択方法の詳細については、『JRun プログラマーガイド』を参照してください。

JSP ファイルは、最初にリクエストされた時点で、JRun によって Java ソースファイルに変換され、その後 Java クラスファイルにコンパイルされます。したがって、Java コードを 1 行も書かずにサーブレットを作成できます。JSP ファイルの実行時イメージは Java クラスファイルになるため、Web サーバーは、Java で作成されたファイルと JSP として作成されたファイルを区別できません。

JSP ファイルからは、Java で書かれたほかのサーブレット、または JSP ファイルとして実装されたほかのサーブレットを呼び出すこともできます。

次の例は、ブラウザ画面に " ハローみなさん " と 5 回表示する簡単な JSP ページを示しています。

```
<html>
<head>
<title> あいさつ </title>
</head>
<body>

<% for(int i=0;i<5;i++) { %>
<h1> ハローみなさん </h1>
<% } %>

</body>
</html>
```

JSP のファイル名の末尾には、拡張子 `.jsp` が付きます。JRun は JSP のリクエストを認識し、JSP を実行可能な Java サーブレットに変換します。JSP の開発方法の詳細については、『JRun プログラマーガイド』を参照してください。

第 5 章
EJB の概要

この章では、基本的な EJB (Enterprise JavaBean) の要素および概念と、JRun 特有の EJB 機能について説明します。これらの機能の詳細については、『JRun プログラマーガイド』を参照してください。

目 次

- EJB の概要 56
- コンテナサービス 58
- EJB のタイプ 59
- JRun での EJB の使用 62

EJB の概要

EJB は、分散型コンポーネントベースのエンタープライズレベル J2EE (Java 2 Platform, Enterprise Edition) アプリケーションを構築するアーキテクチャを提供します。**エンタープライズ bean** と呼ばれる EJB コンポーネントは、拡張性があり、トランザクション処理が可能で、セキュリティ保護されています。EJB 仕様に規定されている機能だけを使用して記述した EJB は、他の J2EE アプリケーションサーバーに移植できます。

エンタープライズ bean は、ビジネスロジックのモジュールを実装するフィールドがあるコードの本文です。エンタープライズ bean は、J2EE サーバーでビジネスロジックを実行するときに、単独または他のエンタープライズ bean とともに使用できる構成要素です。

エンタープライズ bean は EJB コンテナがホストします。EJB コンテナは、標準コンテナサービスだけでなく、幅広いトランザクションおよびパーシスタンスサービスを提供し、J2EE サービスおよび通信 API (Application Programming Interface) にアクセスします。EJB コンテナは、コンポーネントのライフサイクル、ステート管理、パーシスタンス、マルチスレッド処理、コネクションプール、トランザクション管理、セキュリティなどの複雑な下位レベルサービスを処理します。

詳細については、58 ページの「コンテナサービス」を参照してください。

EJB をサポートするアプリケーションサーバーを使用すると、Java 開発者は再利用可能なコンポーネントとしてエンタープライズアプリケーションのビジネスロジックを実装することに集中できます。EJB プログラミングモデルは、EJB の API を理解している熟練サーバーサイド Java 開発者に最適です。JRun では EJB 開発を容易にするために、スタブレス、オープンディレクトリデプロイ、エンタープライズデプロイウィザード、XDoclet サポートなどの機能を提供します。詳細については、62 ページの「JRun での EJB の使用」を参照してください。

EJB の構成

次の表では、EJB の**構成**について説明します。

構成	説明
ホームインターフェイス	EJB インスタンスの作成、削除、および検索を含む、EJB ライフサイクルオペレーションを制御するメソッドを提供します。ホームインターフェイスには次の 2 つのタイプがあります。 • **リモートホーム**　リモートクライアントによって使用されます。 • **ローカルホーム**　同じ JRun サーバー上で動作するクライアントによって使用されます。
コンポーネントインターフェイス	EJB クライアントに公開されるビジネスメソッドを定義します。コンポーネントインターフェイスには次の 2 つのタイプがあります。 • **リモート**　リモートクライアントによって使用されます。 • **ローカル**　同じ JRun サーバー上で動作するクライアントによって使用されます。
bean 実装	ビジネスロジックを実行するメソッドが含まれています。また、EJB 開発者が必要に応じて実装するコールバックメソッドも含まれています。
デプロイメントディスクリプタ	EJB を記述する宣言セマンティクスと、それに必要なサービスを指定します。

詳細については、『JRun プログラマーガイド』を参照してください。

コンテナサービス

EJB 仕様では、コーディングをあまり必要としない高度な機能を提供する各種サービスが、EJB コンテナによってサポートされている必要があります。次の表では、EJB コンテナサービスについて説明します。

サービス	説明
ライフサイクル管理	JRun では、現在のサーバーの負荷に応じて、使用可能な bean インスタンスの数を自動的に増減します。
セッション管理	JRun では、パッシベートおよびアクティブ化コールバックメソッドによってステートフルセッション bean のステートを維持し、エンティティ bean コールバックメソッドによってエンティティ bean のステートを維持します。
トランザクション	トランザクションとは、1 つのまとまりとして実行する必要がある作業単位です。トランザクションが成功するためには、その作業単位のすべての要素が成功する必要があります。EJB のトランザクション管理のアーキテクチャには、2 つのタイプがあります。 • **CMT (Container-managed transactions：コンテナ管理トランザクション)** CMT を使用すると、bean に代わってコンテナがトランザクションを開始してコミットまたはロールバックします。 • **BMT (Bean-managed transactions：bean 管理トランザクション)** BMT を使用すると、トランザクションを完全に制御できます。トランザクションはクライアントから管理できます。つまり、セッション bean の場合は bean 自体から管理できます。
セキュリティ	EJB では認証およびロール方式のアクセス制御をサポートしています。JRun 4 の特徴は、JAAS (Java Authentication and Authorization Service) をベースにしたセキュリティシステムです。

EJB のタイプ

EJB 仕様ではエンタープライズ bean の 3 つのタイプ、セッション bean、エンティティ bean、および メッセージ駆動型 bean (Message Driven Beans) を定義します。次のセクションでは、これらのタイプについて説明します。詳細については、『JRun プログラマーガイド』を参照してください。

セッション bean

セッション bean は、アプリケーションにおけるビジネスロジックの管理に使用します。セッション bean を使用して、EJB 間の対話の調整や、製品の価格の生成、ATM との対話、旅行の予約などのタスクを実行するために複雑な操作を行うことがよくあります。

セッション bean には、次の 2 つのタイプがあります。

- ステートレスセッション bean
- ステートフルセッション bean

ステートレスセッション bean

ステートレスセッション bean では会話型ステートを維持しません。シングルリクエストビジネスプロセスは、ステートを維持する必要がないプロセスです。ステートレスセッション bean は、これらのタイプのシングルリクエストビジネスプロセスに対応できます。

ステートレスセッション bean は、特にセッションエンティティファサードパターンで有効です。**ファサード**は、低レベルサブシステムをマスクする高レベルインターフェイスです。ステートレスセッション bean では、ビジネスプロセスに高レベルインターフェイスファサードを提供できます。ビジネスプロセスでは、セッション bean メソッドが 1 つ以上のエンティティ bean で 1 つ以上のメソッドを呼び出して作業単位を実行します。

詳細については、『JRun プログラマーガイド』を参照してください。

ステートフルセッション bean

シングルビジネスリクエストでは、商品の価格の計算やクレジットカード口座の検証といった複数のビジネスプロセスを実行できます。さらに他のビジネスプロセスも取り出され、複数のリクエストおよびトランザクションの処理が継続されます。たとえば、電子商取引の Web サイトでは、オンラインショッピングカートに商品が追加されることによって、メソッドリクエストが複数になります。ビジネスプロセスでは、リクエスト間のユーザーのステートを追跡する必要があります。ステートフルセッション bean では、クライアントに代わって会話型ステートを維持します。メソッドを呼び出すときにステートフルセッション bean が変更されると、クライアントでは次の呼び出し時にその同じステートを使用することができます。

クラスタ環境で JRun を使用すると、JRun は、クラスタ全体でステートフルセッション bean ステートを維持することによってフェイルオーバーをサポートします。

詳細については、『JRun プログラマーガイド』を参照してください。

エンティティ bean

エンティティ bean は、サーバーのシャットダウン中に存続させるオブジェクトを表します。通常、エンティティ bean のインスタンスを表現するデータはリレーショナルデータベースのテーブルの行に保管されています。このデータベースには、JDBC データストアからアクセスします。テーブルは、複数のデータベースに及ぶ場合もあります。

エンティティ bean のパーシスタンスには、次の 2 つのタイプがあります。

- **BMP (Bean-managed persistence：bean 管理パーシスタンス)** データベースアクセスおよびコールバックメソッドのステートメントの更新をコーディングすることによって、エンティティ bean の実装がパーシスタンスを管理します。
- **CMP (Container-managed persistence：コンテナ管理パーシスタンス)** デプロイメントディスクリプタに作成された仕様をコンテナが使用して、データベースアクセスを実行し、自動的にステートメントを更新します。

BMP

BMP では、コールバックメソッドの適切なデータベースの更新をコーディングすることによって、開発者がパーシスタンスを管理します。たとえば、`ejbUpdate` メソッドはデータベースを更新し、`ejbFindByPrimaryKey` メソッドはプライマリキーを使用してデータベース行を検索します。

詳細については、samples サーバーの compass-ear ディレクトリにある `CreditCard` および `Reservation` EJB を参照してください。

CMP

CMP では、ライフサイクルのある時点でデータベースと bean を自動的に同期をとることによって、コンテナがパーシスタンスを管理します。CMP を使用すると、bean 実装コーディングがより簡単になるので、ビジネスロジックに焦点を合わせることができます。

EJB 2.0 仕様では、アプリケーションサーバーで EJB 1.1 CMP および EJB 2.0 CMP がサポートされている必要があります。

EJB 2.0 CMP のサポート

EJB 2.0 仕様は CMP に対する主要な変更を特徴としており、次の機能が含まれています。

- EJB 実装クラスはアブストラクトクラスとして定義されます。
- パーシスタンスは EJBQL (EJB クエリ言語) を使用して管理されます。
- リレーションシップではエンティティ bean 間のリレーションシップを維持できます。
- CMP bean では finder メソッドだけでなく、select メソッドも使用できます。

CMP の拡張が EJB 2.0 の主な特徴になっています。このマニュアルのリソースおよび例を使用するだけでなく、EJB 2.0 に関する業界誌も参照してください。このマニュアルの序章にはこれらの本のリストが記載されています。

EJB 2.0 サポートの詳細については、『JRun プログラマーガイド』を参照してください。

EJB 1.1 CMP のサポート

JRun 4 での EJB 1.1 CMP サポートは、JRun の初期バージョンの CMP サポートとは異なります。以前は、`ejb-jar.xml` ファイルの環境エントリを使用してパーシスタンス情報を指定していました。JRun 4 では、`jrun-ejb-jar.xml` ファイルの要素を使用してパーシスタンス情報を指定します。これらの要素では、store、load、findByPrimaryKey などの適切なパーシスタンスアクションのパーシスタンス管理を実行する方法を EJB コンテナに通知します。

メモ：JRun 4 では、JRun 3.1 スタイルの CMP bean のオートデプロイもサポートしています。

EJB 1.1 サポートの詳細については、『JRun プログラマーガイド』を参照してください。

メッセージ駆動型 bean

MDB (Message Driven Beans：メッセージ駆動型 bean) は JMS メッセージリスナとしての役割を果たします。MDB にはリモート、リモートホーム、ローカル、またはローカルホームインターフェイスがない点で、セッション bean およびエンティティ bean とは異なります。MDB は、bean 実装があるという点で他の bean タイプと同じであり、ejb-jar.xml ファイルで定義されます。トランザクション、セキュリティ、ライフサイクル管理などの EJB 機能を利用できます。

MDB は、セッション bean のように、他のセッションおよびエンティティ bean に関連するタスクを調整する責任があります。メッセージ駆動型 bean とセッション bean の主な違いはそのアクセス方法です。セッション bean では、ユーザーが起動できるメソッドを定義するリモートインターフェイスが提供されますが、メッセージ駆動型 bean では提供されません。MDB は、特定の非同期メッセージを引用またはリスンします。MDB は、メッセージを処理し、それらのメッセージに応じて他の bean が行うアクションを管理することによってレスポンスします。

単純な JMS メッセージコンシューマに対する MDB の主な利点は、EJB コンテナで複数の MDB インスタンスをインスタンス化して複数のメッセージを同時に処理できることです。

MDB の詳細については、『JRun プログラマーガイド』を参照してください。

JRun での EJB の使用

このセクションでは、JRun の EJB アーキテクチャに固有の機能について説明します。

スタブレスデプロイ

JRun の主要な EJB 機能の 1 つは、**スタブレスデプロイ**です。JRun の EJB デプロイツールはありません。EJB をコンパイルし、オプションでそれらを JAR ファイルにパッケージします。コンパイルされた EJB インターフェイスをクライアントにコピーし、クライアントクラスパスでこの場所を定義します。JRun インスタンスプロキシでは、以前はスタブで管理されていた機能を処理します。

デプロイオプション

JRun には、bean の開発およびデプロイを容易にする柔軟な EJB デプロイモデルがあります。次の表では、このデプロイモデルの機能について説明します。

デプロイ機能	説明
ホットデプロイ	ホットデプロイを有効にすると、JAR ファイルまたはディレクトリ構造をデプロイディレクトリにコピーすることによって EJB をデプロイできます。デフォルトでは、JRun サーバーのルートディレクトリがホットデプロイ用に設定されます。この機能をテストするには、samples JRun サーバーを起動し、EJB を <JRun のルートディレクトリ >/servers/samples ディレクトリにコピーします。 ホットデプロイを有効にすると、JRun デプロイ担当者は EJB デプロイメントディスクリプタへの変更を検出し、bean を自動的にリデプロイします。 また、Sun リファレンス実装および JRun 3.1 を含む他のアプリケーションサーバー用に作成された EJB もデプロイされます。
JAR またはオープンディレクトリからの実行	JRun では、JAR ファイルまたはオープンディレクトリ構造から EJB を実行できます。JRun デプロイ担当者は、META-INF/ejb-jar.xml ファイルを検出すると、ファイルに指定されている EJB をデプロイします。オープンディレクトリ構造からの実行によって、従来の EJB デプロイおよびリデプロイ処理の多くの手順が不要になるため、この実行は EJB デプロイサイクル時に特に有効です。

EJB のクラスタリング

クラスタの一部である JRun サーバーで EJB を実行すると、ロードバランスおよびフェイルオーバーが有効になり、性能および信頼性が高くなります。

また、JRun ではデフォルトで EJB クラスタリングが有効になっています。EJB クラスタリングを無効にするには、jrun-ejb-jar.xml ファイルの `cluster-home` および `cluster-object` 要素を false に設定します。

メモ：JRun ではローカル bean の EJB クラスタリングを使用できません。

詳細については、『JRun 管理者ガイド』を参照してください。

XDoclet

JRun は XDoclet と統合することができます。XDoclet は、EJB、Web アプリケーション、および JSP タグライブラリのコードおよびデプロイメントディスクリプタを生成するオープンソースツールです。多くのアプリケーションサーバーでは、基本的な XDoclet タグで記述されている XDoclet および EJB のサポートを追加しています。XDoclet タグはサーバー間で移植可能です。

EJB では、XDoclet を使用して次のタスクを実行できます。

- 入力としてビジネスメソッドを含んでいるアブストラクトクラスを使用して、bean 実装に必要な EJB コールバックメソッドを生成します。
- bean　実装クラスからリモート、ローカル、ホーム、およびローカルホームインターフェイスを生成します。
- デプロイメントディスクリプタを生成します。

XDoclet が実行する処理を制御するには、bean 実装で jrun.xml 属性と JavaDoc スタイルのコメントを組み合わせて使用します。

EJB とともに XDoclet を使用する方法の詳細については、『JRun プログラマーガイド』を参照してください。

エンタープライズデプロイウィザード

JRun エンタープライズデプロイウィザードによって、EJB の開発およびデプロイ処理が簡素化されています。Swing ベースのグラフィカルユーザーインタフェースを使用すると、あらゆるタイプの EJB を作成したり、既存の EJB のデプロイメントディスクリプタを編集して JAR ファイルにパッケージし、JRun にデプロイしたりすることができます。とくに、エンタープライズデプロイウィザードのオブジェクト関係マッピング機能を使用すると、エンティティ bean 開発処理を簡素化できます。

エンタープライズデプロイウィザードは、スタンドアローンツールとして実行したり、IDE の上部でプラグインとして実行することができます。

エンタープライズデプロイウィザードを起動するには、<JRun のルートディレクトリ>/bin/jrunwizard.exe (Windows) または <JRun のルートディレクトリ>/bin/jrunwizard (UNIX) を実行します。

IDE にエンタープライズデプロイウィザードをインストールするには、<JRun のルートディレクトリ>/lib ディレクトリをコンソールウィンドウに表示し、次のコマンドを実行します。

```
java -jar jrunwizard-installer.jar
```

現在サポートされている IDE のリストについては、リリースノートを参照してください。

エンタープライズデプロイウィザードの特徴は、状況に応じて表示内容が変わるオンラインヘルプシステムです。使用法については、オンラインヘルプを参照してください。

第 6 章
Web アプリケーションの開発

この章では、最初に Web アプリケーションの構造について説明し、次に JRun で Web アプリケーションおよびそのリソースを処理する方法について説明します。Web アプリケーションの作成、パッケージ、およびデプロイの方法についても説明します。

目 次

- Web アプリケーションの概説 66
- Web アプリケーションのディレクトリ構造 68
- Web アプリケーション、JRun サーバー、Web サーバー 71
- Web アプリケーションの開発 73
- Web アプリケーションのデプロイ 77

Web アプリケーションの概説

Web アプリケーションは、サーブレット、JSP、HTML ページ、イメージ、標準デプロイメントディスクリプタで構成されます。また、オプションで JavaBeans、カスタムタグクラスなどの Web コンポーネントが含まれます。任意の Web アプリケーションサーバーへデプロイできるように、これらのリソースを標準化されたディレクトリ構造に配置します。

このセクションでは、Web アプリケーションの重要な機能と利点について説明します。

Web アプリケーションの利点

Java サーブレット 2.3 仕様では、Web アプリケーションが定義されています。この仕様は次の利点を提供します。

- アプリケーションのディレクトリ構造や、アプリケーションの定義に必要な情報など、Web アプリケーション表現の標準定義
- アプリケーションサーバーへの Web アプリケーションデプロイの標準定義。アプリケーションサーバー用に記述された Web アプリケーションは、Java サーブレットの仕様に準拠するほかのアプリケーションサーバーに移植することができます。
- アプリケーション内のアプリケーションリソースへの相対リンクが使用可能。Web アプリケーションでは絶対リファレンスを使用しないので、アプリケーションサーバー上のアプリケーションの位置は重要ではありません。そのため、開発した場所とは異なる別のディレクトリ、URL、または別のサーバーに Web アプリケーションをデプロイできます。

JRun は、Java サーブレット 2.3 の仕様に完全に従った Web アプリケーションアーキテクチャを実装します。Web アプリケーションは JRun を使用して開発でき、Java サーブレット 2.3 の仕様をサポートする Web アプリケーションサーバーにデプロイすることができます。また、JRun はエンタープライズ向けの J2EE アプリケーションの開発に必要な最新の業界標準を完全にサポートしているため、Web アプリケーションでエンタープライズ向け J2EE リソースおよびコンポーネントを利用したり相互運用することが可能です。

Web アプリケーションとエンタープライズアプリケーションの比較

エンタープライズアプリケーションは、1 つ以上の J2EE モジュール、Web アプリケーション、リソースアダプタ、および EJB から構成されます。Web でのエンタープライズアプリケーション開発の標準は、J2EE 仕様に基づいています。JRun は、J2EE アプリケーションモデルをサポートし、エンタープライズアプリケーションを実行する実行時環境を提供します。

エンタープライズアプリケーションには、エンタープライズアプリケーションのデプロイメントディスクリプタである application.xml も含まれており、通常は EAR (Enterprise ARchive) ファイルと呼ばれる 1 つのファイルに圧縮されてパッケージされます。

詳細は、18 ページの「エンタープライズアプリケーションのアーキテクチャ」を参照してください。

Web アプリケーションは、エンタープライズアプリケーションのサブセットです。ただし、スタンドアローン Web アプリケーションをデプロイできます。

Web アプリケーションの設定は web.xml ファイルのコンテンツによって定義されます。このファイルは標準 Web アプリケーションのデプロイメントディスクリプタファイルです。このファイルには、アプリケーションサーバーがアプリケーションを実行するために必要なすべての情報が含まれています。

Web アプリケーションを JRun で配布する場合は、展開したディレクトリ構造として配布するか、または WAR (Web ARchive) ファイルと呼ばれる 1 つのファイルに圧縮して配布することができます。WAR ファイルには、すべてのディレクトリ構造とアプリケーションを定義するすべてのファイルが含まれます。

Web アプリケーションとエンタープライズアプリケーションの開発およびデプロイの詳細については、73 ページの「Web アプリケーションの開発」および 77 ページの「Web アプリケーションのデプロイ」を参照してください。『JRun プログラマーガイド』および『JRun アセンブルとデプロイガイド』も参照してください。

Web アプリケーションの使用

1 つの JRun サーバーでは複数の Web アプリケーションをサポートできます。Web アプリケーションでは、同じサーバー上のほかのアプリケーション内のリソースのリファレンスや、一般的なリソースの共有が可能です。共有することによって、Web アプリケーションは EJB やデータベースドライバクラスなど、多くのアプリケーションで一般的に使用されているリソースにアクセスできます。

Web アプリケーションでは、データベースや EJB を使用してデータを共有することもできます。たとえば、電子商取引 Web サイトが、複数のアプリケーションから構成されているとします。このタイプの Web サイトの顧客は、ログイン名とパスワードによって識別できます。そのため、各 Web アプリケーションではログイン名を使用して、ショッピングカート情報、購入履歴、住所などの共有データベース内のユーザーの情報にアクセスします。

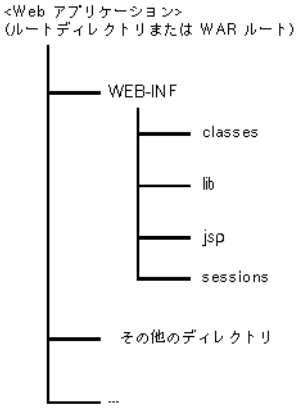
Web アプリケーションを開発するときは、アプリケーションを複数の JRun サーバーに分割するか、1 つの JRun サーバーですべてのアプリケーションをホストするか、またはサーバーをクラスタ定義してアプリケーションを処理するかを決定する必要があります。**クラスタ**はサーバーの集合で、イメージ的には 1 つのシステムと見なされ、そのクラスタに割当てられたサーバー間で、アプリケーション負荷やサーバー負荷が管理されます。クラスタの詳細については、『JRun 管理者ガイド』を参照してください。

複数の JRun サーバーを作成する理由の 1 つは、個別の Java Virtual Machine (JVM) のプロセス内でアプリケーションを分離するためです。JVM は JRE とも呼ばるもので、ソフトウェア的に実装された CPU です。これには、Java プラットフォーム用に作成されたプログラムを実行するのに必要なすべての機能が含まれています。各 JRun サーバーは、その JRun サーバーに対するすべてのサーブレット、JSP、および EJB を実行する 1 つの JVM と関連付けられています。アプリケーションを複数の JRun サーバー、すなわち独自の JVM に分離することで、別のアプリケーションが悪影響を与えないようにできます。さらに、各アプリケーションのクラスパス、データソース、EJB、およびその他のリソースをサーバーレベルで定義できます。

Web アプリケーションを異なる JRun サーバーで実行するもう 1 つの理由は、各 JRun サーバーが独自のユーザー認証メカニズムまたは一連のユーザー認証ルールを実装できるようにするためです。異なる JRun サーバーでアプリケーションを実行することによって、サーバーの認証設定を利用できます。認証の詳細については、『JRun 管理者ガイド』を参照してください。

Web アプリケーションのディレクトリ構造

次の図は、Web アプリケーションのディレクトリ構造を示します。



アプリケーションルートディレクトリ (<Web アプリケーション >) は、アプリケーションファイルを提供するためのドキュメントルートとして機能します。<Web アプリケーション > ディレクトリには、多くの場合 WAR ファイル名が使用されます。このディレクトリには、Web アプリケーションの一部として開発する JSP および HTML ページが含まれます。たとえば、Web アプリケーションが <JRun のルートディレクトリ >/servers/<JRun サーバー >/<Web アプリケーション > にある場合、デフォルトのトップページファイルは通常 <JRun のルートディレクトリ >/servers/<JRun サーバー>/<Web アプリケーション >/index.html となります。<JRun サーバー> は、JRun サーバーのディレクトリ名で、<Web アプリケーション > は、サーバーによってホストされる Web アプリケーションの名前に対応するディレクトリの名前です。

Web アプリケーションは、次のディレクトリ構造に従う必要があります。

ディレクトリ	説明
<Web アプリケーション >	JSP、HTML ページ、カスケーディングスタイルシート、イメージ、Javascript ファイルなど、クライアントのリクエストからすべてのファイルおよび WEB-INF ディレクトリを格納します。これらのファイルは、Web アプリケーションのルートディレクトリ、または予約名 WEB-INF を使用しないサブディレクトリに直接置くことができます。
WEB-INF	(必須) classes および lib ディレクトリ、標準 Web アプリケーションデプロイメントディスクリプタ (web.xml)、さらに必要に応じて JRun 固有のデプロイメントディスクリプタ (jrun-web.xml) を格納します。 このディレクトリは、アプリケーションのパブリックドキュメントツリーの一部ではありません。つまり、このディレクトリ、またはそのサブディレクトリに含まれたファイルはクライアントに直接参照させることはできません。
WEB-INF/classes	サーブレット、その他の Java クラス、および JAR (Java ARchive) ファイルにパッケージされない関連リソースを格納します。

ディレクトリ	説明
WEB-INF/lib	サーブレット、その他の Java クラス、および JAR または ZIP ファイルに含まれない関連リソースを格納します。このサブディレクトリには、タグライブラリを含む JAR ファイルが格納されます。
WEB-INF/jsp	JSP の変換時に JRun によって作成されるファイルが格納されます。
WEB-INF/sessions	デフォルトのセッションパーシスタンスメカニズムが使用するシリアル化された JRun セッションが格納されます。

エンタープライズアプリケーションは、Web アプリケーションと同様のディレクトリ構造に従います。アプリケーションのルートディレクトリである <Web アプリケーション > には、多くの場合 EAR ファイル名が使用されます。エンタープライズアプリケーションデプロイメントディスクリプタである application.xml は、<JRun のルートディレクトリ >/servers/<JRun サーバー>/<Web アプリケーション>/META-INF ディレクトリに置かれます。

Web アプリケーションは、上記のディレクトリおよびサブディレクトリに制限されません。アプリケーションには、Flash SWF ファイル、HTML ファイル、イメージ、その他のアプリケーションリソース用に、ディレクトリを追加できます。これらのディレクトリは、クライアントが直接アクセスするリソース用 Web アプリケーションのパブリックドキュメントツリーの一部になります。アプリケーションへのディレクトリ追加の詳細については、73 ページの「ディレクトリの追加」を参照してください。

デプロイメントディスクリプタ (web.xml)

Web アプリケーションは、WEB-INF/web.xml ファイルのコンテンツによって定義されます。このファイルは、デプロイメントディスクリプタとも呼ばれます。**デプロイメントディスクリプタ**は、アプリケーションサーバー上で実行するためにアプリケーションが使用する設定情報が格納された XML ファイルです。web.xml のコンテンツは Java サーブレットアプリケーション仕様で定義されているので、このファイルは JRun に固有のものではありません。J2EE 準拠の Web アプリケーションをサポートするプラットフォームはすべて、web.xml ファイルのコンテンツを認識して解釈します。

web.xml ファイルを使用して、Web アプリケーションの次のような設定情報とデプロイ情報を定義します。

- サーブレット初期化パラメータ
- セッション設定
- サーブレットと JSP の定義
- サーブレットと JSP URL のマッピング
- MIME タイプのマッピング
- welcome-file-list
- エラーページ
- フィルタ
- イベントハンドラ
- タグライブラリ
- セキュリティ情報
- データソースや JMS 接続ファクトリなどのリソース

web.xml ファイルには、Web アプリケーションについての情報だけでなく、Web アプリケーションがアクセスする EJB についてのリファレンス情報も含まれています。Web アプリケーションおよび EJB の標準デプロイメントディスクリプタには、環境エントリ、EJB リファレンス、リソースリファレンス、およびセキュリティロールの共通の要素があります。

標準デプロイメントディスクリプタに加えて、次の JRun 固有のデプロイメントディスクリプタファイルを使用できます。

- **WEB-INF/jrun-web.xml** JRun アプリケーションサーバーに固有の Web アプリケーション要素が含まれます。このデプロイメントディスクリプタを使用すると、Web アプリケーションのコンテキストルートの設定、自動サーブレットコンパイルやリロード機能の有効 / 無効の設定、仮想パスの設定、サーブレットのセッションパーシスタンスおよび Cookie の設定、および JNDI 名の設定が可能です。
- **SERVER-INF/default-web.xml** web.xml ファイルと同じ構造を持ちますが、設定内容は JRun サーバー内のすべての Web アプリケーションに適用されます。各 Web アプリケーションの web.xml ファイルのデフォルト値が含まれます。このファイルには、JRun によって内部使用される、グローバルサーブレット用の定義とマッピングが含まれます。

web.xml ファイルは標準的なテキストエディタや XML エディタを使用して編集できます。JRun では、jrun-web.xml ファイルや default-web.xml ファイルに追加要素を指定できます。標準的な web.xml ファイルのすべてのプロパティが掲載されているリストは、Java サーブレット 2.3 の仕様書を参照してください。JRun 固有のデプロイメントディスクリプタの詳細については、『JRun アセンブルとデプロイガイド』を参照してください。

Web アプリケーション、JRun サーバー、Web サーバー

Web アプリケーション開発時の最初の作業の 1 つとして、アプリケーションを JRun サーバーにデプロイします。

メモ：Web アプリケーションを admin JRun サーバーにデプロイしないでください。このサーバーは主に JRun サーバーの管理に使用します。

JRun サーバーには、複数の Web アプリケーションまたは複数のエンタープライズアプリケーションを含めることができます。両方のアプリケーションを組み合わせて含めることもできます。

JRun サーバーをホストとするさまざまな Web アプリケーションにクライアントのリクエストを割り当てるには、異なる URL パターンに対応するように各 Web アプリケーションをマッピングします。この方法で、JRun サーバーは適切な Web アプリケーションにリクエストを転送できます。

マッピング

JRun は、リクエストに応えて返すファイルを決定するときに、次のタイプのマッピングを使用します。

マッピング	目的
アプリケーションマッピング	Web アプリケーションのコンテキストルートを、そのアプリケーションが格納されている物理ディレクトリと関連付けます。
サーブレットマッピング	サーブレットを、コンテキストルート内の URL マッピング (MyServlet など)、接頭辞 (/servlet など)、または接尾辞 (.jsp など) と関連付けます。
トップページファイルマッピング	リクエストが既存のアプリケーションマッピングまたはサーブレットマッピングと一致しない場合の Web アプリケーションの動作を定義します。通常、トップページファイルマッピングは、ファイルを index.html またはその他のデフォルトページと関連付けます。たとえば、<JRun のルートディレクトリ >/servers/default/SERVER-INF/default-web.xml 内の index.html や index.jsp と関連付けます。

JRun サーバーで実行される各 Web アプリケーションには、1 つのコンテキストルートマッピングと複数のサーブレットマッピングを含めることができます。jrun-web.xml ファイルで**コンテキストルート**を定義することも、JRun にコンテキストルートを置き換えさせるようにもできます。JRun は、次の順序でコンテキストルートを決定します。

1 /WEB-INF/jrun-web.xml ファイルでコンテキストルートマッピングを調べます。

2 WAR ファイルが圧縮されていない場合、JRun は WAR ファイルのルートディレクトリ名を使用します。たとえば、WAR ファイルが <JRun のルートディレクトリ >/servers/samples/worldmusic-war/ ディレクトリにデプロイされる場合、JRun は worldmusic-war をコンテキストルートとして使用します。

3 WAR ファイルが圧縮されている場合、JRun は WAR ファイル名から ".war" を取り除いて使用します。たとえば、Web アプリケーションが techniques.war に含まれている場合、JRun は "techniques" をコンテキストルートとして使用します。

Web アプリケーションを最適な方法で使用するには、HTML ファイル、JSP、およびサーブレットに対するリクエストを処理するために JRun がどのようにアプリケーションマッピングとサーブレットマッピングを使用するかを理解する必要があります。次の表は、JRun の設定ファイルが定義するマッピングを示しています。

設定ファイル	マッピング
default-web.xml	サーブレット、フィルタ、およびトップページファイルのマッピング。 default-web.xml の設定は、この JRun サーバー上のすべての Web アプリケーションに適用されます。
web.xml	サーブレットおよびフィルタのマッピング。
jrun-web.xml	スタンドアローン Web アプリケーション (WAR ファイル) 用の仮想パスマッピングおよびオプションのアプリケーションマッピング (コンテキストルート)。
application.xml	エンタープライズアプリケーション (EAR ファイル) 内の Web アプリケーションのアプリケーションマッピング (コンテキストルート)。

アプリケーションのマッピング

アプリケーションマッピングは、コンテキストパスを Web アプリケーションの名前およびディレクトリパスに関連付けます。**コンテキストパス**は、Web アプリケーションマッピングに関連付けられたパス接頭辞を指定します。Web サーバーの URL ネーム空間のルートディレクトリにあるデフォルトのアプリケーションの場合、コンテキストパスは空の文字列になります。デフォルト以外のアプリケーションの場合、コンテキストパスは、スラッシュ (/) で始まりますが、スラッシュで終了しません。次に例を示します。/compass は、/compass を含むリクエスト URL を compass アプリケーションにマッピングします。

アプリケーションマッピングの詳細については、『JRun プログラマーガイド』を参照してください。

Web アプリケーションの開発

ここでは、Web アプリケーションの作成方法、および Web アプリケーションにリソースとコンポーネントを追加する方法について説明します。

Web アプリケーションの作成

デフォルトの JRun サーバーには、Web アプリケーション開発に使用できる空の Web アプリケーションが用意されています。Web アプリケーションルートディレクトリは、<JRun のルートディレクトリ >/servers/default/default-ear/default-war です。JRun は、アプリケーションの基本ディレクトリ構造、最小限の情報が格納された web.xml ファイル、およびアプリケーションの URL マッピングを作成します。サーブレット、JSP、および EJB の開発を開始してから、default-war ディレクトリ構造にそれらを配置できます。

さらに、JMC を使用して JRun サーバーを作成すると、そのサーバー上にデフォルトの Web アプリケーションが作成されます。Web アプリケーションに対して追加する場合と同じ規則を使用して、コンテンツをデフォルトのアプリケーションに追加します。Web アプリケーションへのリソースの追加の詳細については、73 ページの「Web アプリケーションコンポーネントの追加」を参照してください。

Web アプリケーションコンポーネントの追加

完全な Web アプリケーションは、サーブレット、JSP、HTML ページ、Flash (.swf) ファイル、イメージ、タグライブラリ、JavaBeans、Enterprise JavaBeans、その他のアプリケーションリソースから構成されます。Web アプリケーションの開発作業の一部として、アプリケーションのディレクトリ構造にこれらのコンポーネントを追加します。

次のセクションでは、Web アプリケーションに各コンポーネントを追加する方法について説明します。

- 73 ページの「ディレクトリの追加」
- 74 ページの「HTML ページの追加」
- 74 ページの「JSP の追加」
- 74 ページの「Java サーブレットの追加」
- 75 ページの「ServletInvoker の使用」
- 75 ページの「タグライブラリの追加」
- 76 ページの「EJB の追加」
- 76 ページの「リソースの追加」

ディレクトリの追加

Web アプリケーションのディレクトリ構造では、WEB-INF という名前のサブディレクトリが必ず必要です。このディレクトリについては、68 ページの「Web アプリケーションのディレクトリ構造」を参照してください。

ただし、多くの Web アプリケーションでは、アプリケーションのルートディレクトリの下に WEB-INF 以外のディレクトリが追加されています。追加のディレクトリに .class や JAR ファイルが含まれておらず、それらがアプリケーションのクラスパスに含まれている場合は、アプリケーションルートにサブディレクトリを追加する際に特別な作業を行う必要はありません。

たとえば、通常は、アプリケーションルートディレクトリの下にある images ディレクトリにイメージ ファイルを配置します。この他の一般的なディレクトリとして、複数のアプリケーションリソースで共有するファイルが格納される include ディレクトリがあります。

HTML ページの追加

アプリケーションのルートディレクトリは、アプリケーションファイルを提供するためのドキュメントルートとして機能します。アプリケーションの HTML ページをアプリケーションルートの下か、または WEB-INF ディレクトリ以外のアプリケーションルートのサブディレクトリに追加します。たとえば、Web アプリケーションが <JRun のルートディレクトリ >/servers/<JRun サーバー>/<Web アプリケーション > にある場合は、デフォルトのトップページファイルを <JRun のルートディレクトリ >/servers/<JRun サーバー >/<Web アプリケーション >/index.html に配置します。

JSP の追加

JSP は、HTML とスクリプトコードを含んでいます。JSP (.jsp ファイル) は、クライアントから最初に要求されたときに、Java ソースコードファイル (.java ファイル) に変換され、続いて Java クラスファイル (.class ファイル) にコンパイルされます。JSP 作成方法の詳細については、第 4 章、43 ページの「サーブレットと JSP の使用」を参照してください。

Web アプリケーションに JSP を追加するには、アプリケーションのルートディレクトリか、または WEB-INF にあるディレクトリ以外のアプリケーションルートの下のディレクトリに JSP をコピーします。

メモ：JRun では、JSP のリクエストに応じて作成されるファイルが、WEB-INF/jsp ディレクトリに書き込まれます。

Java サーブレットの追加

Java サーブレットは .class ファイルによって表されます。Web アプリケーションにサーブレットを追加する手順は、サーブレットの保管場所によって異なります。通常、サーブレットは次のいずれかの場所に保管されます。

- WEB-INF/classes または WEB-INF/classes のサブディレクトリ (Java パッケージ名を反映する) 内の .class ファイル
- WEB-INF/lib の JAR ファイル内の .class ファイル

ディレクトリ WEB-INF/classes と WEB-INF/lib を Web アプリケーションクラスパスに組み込みます。これらのディレクトリ内のすべての .class および JAR ファイルはリロード可能です。

重要な概念は、JRun がサーブレットを提供する方法を知ることです。定義では、アプリケーションサーバーは、WEB-INF ディレクトリまたは WEB-INF のサブディレクトリのファイルを直接提供できません。しかし、サーブレット .class ファイルは通常、JAR ファイル内の WEB-INF/classes、WEB-INF/classes のサブディレクトリ、または WEB-INF/lib に格納されます。次の手順は、サーブレットがクライアントにデータを提供する方法を示したものです。

アプリケーションにサーブレットを追加するには

1 サーブレット .class ファイル、またはサーブレット .class ファイルが含まれている JAR ファイルを、適切なディレクトリにコピーします。
   - WEB-INF/classes 内の .class ファイル
   - WEB-INF/lib 内の JAR ファイル

2 web.xml または default-web.xml ファイルでサーブレットを定義します。

a web.xml でサーブレット要素を作成します。

b web.xml でサーブレット URL マッピングを作成します。

サーブレットマッピングは、サーブレットを URL パターンと関連付けます。

指定された URL パターンに一致するサーブレットパスがリクエスト URI に含まれる場合、JRun は関連するサーブレットを呼び出します。サーブレットを明示的に定義しない場合は、暗黙のマッピング /servlet を使用して WEB-INF/classes ディレクトリに格納するサーブレットを要求できます。

ServletInvoker の使用

JRun は暗黙で /servlet を `ServletInvoker` サーブレットにマッピングします。これは、web.xml または default-web.xml ファイルで明示的に定義されていないサーブレットに対する、汎用的な呼び出しメカニズムです。定義されていないサーブレットを参照するには、http://<Web サーバー >:< ポート番号 >/<Web アプリケーション >/servlet/< サーブレットクラス名 > のように、リクエストする URL に /servlet を指定します。

メモ：サーブレットクラス名には .class を含めないようにしてください。

サーブレットを処理するためにサーブレット要素を明示的に定義する必要はありません。`ServletInvoker` サーブレットは、サーブレットクラス名を使用して仮のサーブレット登録を作成するからです。

`ServletInvoker` サーブレットは、主に開発およびテスト中に使用します。ただし、セキュリティとパフォーマンス上の理由から、すべてのサーブレットに対して明示的マッピングを定義して、運用システムのデフォルトの `ServletInvoker` マッピングを上書きする必要があります。

タグライブラリの追加

JavaServer Pages の仕様書には、タグライブラリに関するフレームワークが記述されています。開発者は、**タグライブラリ**を使用して関連する機能セットを 1 つのカスタムタグセットにカプセル化できます。JSP では、これらのタグを使用して、ライブラリに組み込まれている機能を利用できます。たとえば、タグライブラリを作成してデータベースアクセスの簡略化、電子商取引のオペレーションの実行、URL の書き換え、または XML のトランスフォームを行うことができます。

タブライブラリは、**タグ**または**カスタムタグ**と呼ばれる 1 つ以上の**アクション**から構成されます。各アプリケーションは、関連付けられた Java **タグハンドラ**クラスにコーディングされている処理を実行します。ユーザーは、カスタムタグを定義し、タグハンドラをコーディングするとともに、TLD (タグライブラリディスクリプタ) ファイルで各カスタムタグの機能 (属性を含む) を定義します。

デプロイ可能なタグライブラリでは、タグハンドラ、TLD ファイル、TEI (Tag Extra Information) クラス、およびその他のサポートクラスを JAR ファイルに組み込みます。この JAR ファイルは、通常 Web アプリケーションの WEB-INF/lib ディレクトリに配置されます。

タグライブラリの詳細については、『JRun プログラマーガイド』を参照してください。

EJB の追加

Web アプリケーションが EJB にアクセスできるようにするには、EJB を Web アプリケーションのホストである JRun サーバーか、または Web アプリケーションにアクセス可能な JRun サーバーにデプロイしなければなりません。

JRun での EJB の開発とデプロイの詳細については、55 ページの「EJB の概要」を参照してください。

リソースの追加

追加可能なリソースには、イメージディレクトリや JavaBeans の他、データベースドライバなどのリソース用のクラスファイルなどがあります。これらのリソースを Web アプリケーションに追加するときは、Web アプリケーションディレクトリ構造内の正しい場所に配置する必要があります。.class ファイルや JAR ファイルによってリソースを提供する場合は、Web アプリケーションクラスパス (通常は WEB-INF/classes または WEB-INF/lib) に含まれているディレクトリにファイルを配置する必要があります。

Web アプリケーションのデプロイ

JRun は、次のモジュールアーカイブファイルまたは展開したディレクトリから、Web アプリケーション、エンタープライズアプリケーション、および EJB やエンタープライズリソースアダプタなどの他の J2EE モジュールのダイナミックデプロイを行うためのオートデプロイおよびホットデプロイ機能を提供します。

- Web アプリケーション (WAR ファイルまたはディレクトリ)
- エンタープライズアプリケーション (EAR ファイルまたはディレクトリ)
- Enterprise JavaBeans (JAR ファイルまたはディレクトリ)
- エンタープライズリソースアダプタ (RAR ファイルまたはディレクトリ)

モジュールアーカイブファイルまたは展開したディレクトリをデプロイディレクトリ (デフォルトでは、<JRun のルートディレクトリ >/servers/<JRun サーバー >) にコピーすると、JRun はアプリケーションまたはモジュールを自動的にデプロイする (サーバーが実行中の場合) か、サーバーの次回起動時にデプロイします。モジュールアーカイブファイルまたはデプロイメントディスクリプタを変更すると、デプロイはダイナミックに更新されます。

運用環境では、移植性を高めるためにアーカイブファイルをデプロイするのが最適です。開発時は、展開したディレクトリをデプロイすると、柔軟性と使いやすさを最大に高めることができます。詳細については、『JRun アセンブルとデプロイガイド』を参照してください。

クラスタ全体への Web アプリケーションのデプロイ

また、クラスタデプロイディレクトリでアーカイブファイルが検出されると、JRun はサーバークラスタ全体に Web アプリケーションをデプロイします。この機能は、展開したディレクトリをサポートしません。デフォルトのクラスタデプロイディレクトリは、<JRun のルートディレクトリ >/servers/<JRun サーバー >/SERVER-INF/cluster ですが、JMC または jrun.xml ファイルを使用してディレクトリを変更できます。JMC を使用してクラスタにアプリケーションをデプロイすると、JRun はクラスタされたいずれかのサーバーのクラスタデプロイディレクトリに、アーカイブファイルをコピーします。クラスタ全体で Web アプリケーションのデプロイと同期を行うには、サーバーを実行している必要があります。

JMS デスティネーション、JMS 接続ファクトリ、JDBC データソース、JavaMail セッションなど、<JRun サーバー>/SERVER-INF/jrun-resources.xml に定義されている Web アプリケーションリソースが、クラスタ内の各サーバー上に存在するか、アクセス可能であることを確認する必要があります。

JRun クラスタの詳細については、『JRun 管理者ガイド』を参照してください。

Web アプリケーションをデプロイ用にパッケージ化

Web アプリケーションをデプロイする前に、アプリケーションのデプロイメントディスクリプタを作成し、Web アプリケーションの構成を適切なディレクトリ構造にパッケージする必要があります。

デプロイ用に Web アプリケーションを準備する方法は多数ありますが、基本的な Web アプリケーションのデプロイでは、次の手順を実行します。

1. エンドユーザーの Web ブラウザからアクセスする JSP、HTML ファイル、イメージ、その他のリファレンスファイルを含むステージングディレクトリツリーを作成します。

   このディレクトリを Web アプリケーションディレクトリとして簡単に識別できるように、-war で終わるディレクトリ名を使用します。また、リファレンスファイルの元のディレクトリ構造を維持するようにしてください。

2. ステージングディレクトリで WEB-INF というディレクトリを作成します。

3. WEB-INF ディレクトリで classes および lib というディレクトリを作成します。

4. アプリケーションサーブレットおよびその他のアーカイブされていないクラスファイルを WEB-INF/classes ディレクトリにコピーします。

5. Web アプリケーションで JSP タグライブラリを使用する場合は、JAR ファイルにパッケージしたタグライブラリを WEB-INF/lib ディレクトリにコピーします。アーカイブされていないタグライブラリクラスを WEB-INF/classes ディレクトリにコピーします。

   JAR ファイル内にデプロイする場合は、TLD ファイルを META-INF ディレクトリまたは META-INF ディレクトリのサブディレクトリに置く必要があります。Web アプリケーションに直接デプロイする場合は、TLD ファイルを WEB-INF ディレクトリまたはそのサブディレクトリに置く必要があります。

6. Web アプリケーションデプロイメントディスクリプタである web.xml、および必要に応じて jrun-web.xml を WEB-INF ディレクトリに置きます。

7. 運用環境向けに Web アプリケーションを準備するには、ステージングディレクトリの最上位レベルから次の jar ユーティリティコマンドを使用して WAR ファイルにパッケージします。

   ```
   jar cvf <Web アプリケーション>.war
   ```

   ここで、<Web アプリケーション> は、Web アプリケーションの名前です。

詳細については、『JRun アセンブルとデプロイガイド』を参照してください。

パート II
チュートリアル

第 II 部には、簡単な J2EE アプリケーションを構築するチュートリアルが含まれています。チュートリアルレッスンでは、JRun サーバーを追加して、Java サーブレット、JSP、JavaBeans、および EJB のコードを記述します。これらを組み合わせて、柔軟性と拡張性を備えた J2EE アプリケーションを生成する方法を学習します。最後のチュートリアルレッスンでは、Web サービスを使用して JRun サンプルアプリケーションのデータにアクセスします。

サーブレットのチュートリアル 81
JSP のチュートリアル 97
EJB のチュートリアル 105
Web サービスのチュートリアル 113

レッスン 1
サーブレットのチュートリアル

このチュートリアルレッスンでは、サーブレット、JavaServer Page (JSP)、JavaBeans、および Enterprise JavaBeans (EJB) から構成されている基本的な J2EE アプリケーションの開発、デプロイ、および実行の方法について説明します。

このレッスンでは、開発環境の設定、チュートリアル用の JRun サーバーの追加、およびチュートリアルテンプレートアプリケーションのデプロイを行います。最初の手順では、ビジネスロジックを提供し、ユーザー ID およびパスワードを確認するログオンサーブレットをコーディングします。

目 次

- 開発環境の設定 82
- JRun 入門 83
- チュートリアルエンタープライズアプリケーションのデプロイ 88
- Compass Travel J2EE アプリケーション 93
- 要約 96

開発環境の設定

チュートリアルレッスンを開始する前に、開発環境を正しく設定してください。

開発環境を設定するには

1 次のソフトウェアをインストールします。

ソフトウェア	必要条件
Web サーバー	• JRun 提供の Web サーバー (JWS) • サポートされている外部 Web サーバー (オプション)
テキストエディタ	• 任意のテキストエディタ • Java 統合開発環境 (IDE) • Macromedia Dreamweaver などの HTML エディタ
JRun 4	システム必要条件については、『JRun インストールガイド』を参照してください。

2 チュートリアルファイルは、<JRun のルートディレクトリ >/docs/tutorialfiles ディレクトリにあります。このディレクトリは、< チュートリアルルートディレクトリ > と呼ばれます。

3 (オプション) JRun と連動する Java IDE を設定します。

詳細については、15 ページの「JRun と Java IDE の併用」を参照してください。

JRun 入門

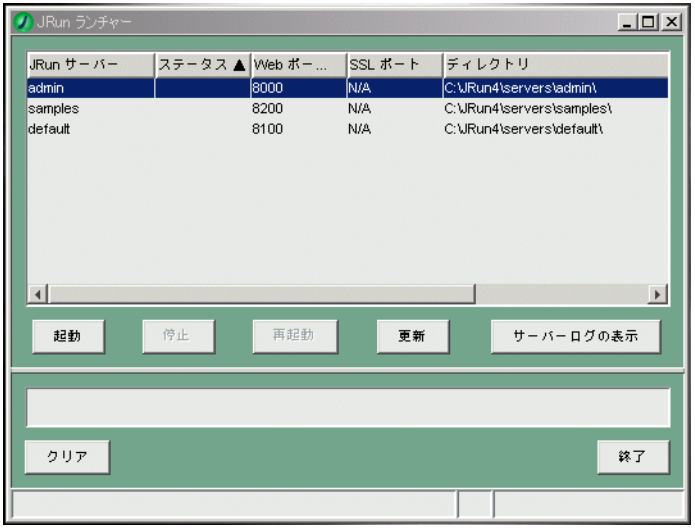
admin JRun サーバーを起動して、JRun の操作を開始します。

admin JRun サーバーを起動するには

- <JRun のルートディレクトリ >/bin ディレクトリで、JRun コマンドラインユーティリティを使用します (<JRun のルートディレクトリ> は JRun がインストールされているディレクトリです)。

  ```
  jrun -start admin
  ```

- (Windows のみ) **[スタート] > [プログラム] > [Macromedia JRun 4] > [JRun ランチャー]** を選択します。admin サーバーを選択し、[起動] ボタンをクリックします。



JMC (JRun 管理コンソール) を使用するには、admin JRun サーバーを実行しておく必要があります。最初のチュートリアル手順では、JMC を使用して、JRun アプリケーションサーバーを追加し、JDBC データソースを定義します。

JMC を開始するには

1 次のいずれかの方法で JMC を起動します。

- Web ブラウザを開き、次の URL を入力します。
  http://localhost:8000

メモ：この手順は、デフォルトの JWS ポート 8000 で JMC に接続している場合を想定しています。デフォルトのポートを使用しない場合は、8000 を admin Web サーバーのポート番号に変更してください。admin Web サーバーのポート番号は、<JRun のルートディレクトリ>/servers/admin/SERVER-INF/jrun.xml ファイル内の次の行にあります。

```
<service class="jrun.servlet.http.WebService"
 name="WebService">
<attribute name="port">8000</attribute>
<attribute name="interface">*</attribute>
</service>
```

- (Windows のみ) **[スタート] > [プログラム] > [Macromedia JRun 4] > [JRun 管理コンソール]** を選択します。

  JMC のログインウィンドウが表示されます。

2 ユーザー名とパスワードを入力し、[ログイン] をクリックします。

メモ : ユーザー名とパスワードは JRun のインストール時に設定しています。詳細については、『JRun インストールガイド』を参照してください。

JRun のホームページが表示されます。

次のセクションでは、JMC を使用してチュートリアルアプリケーションのために JRun サーバーを追加します。

JRun サーバーの追加

インストール時に、JRun では admin、default、および samples という 3 つの JRun サーバーが作成されます。admin サーバーはデフォルトで JMC をホスティングします。次のセクションでは、JMC を使用して JRun サーバーを追加する方法について説明します。JRun サーバーの追加や削除を行う場合は、それぞれのサーバーに固有のポートが必要であることに注意してください。JRun ポートの詳細については、『JRun 管理者ガイド』を参照してください。

JRun サーバーは、次のような理由により追加します。

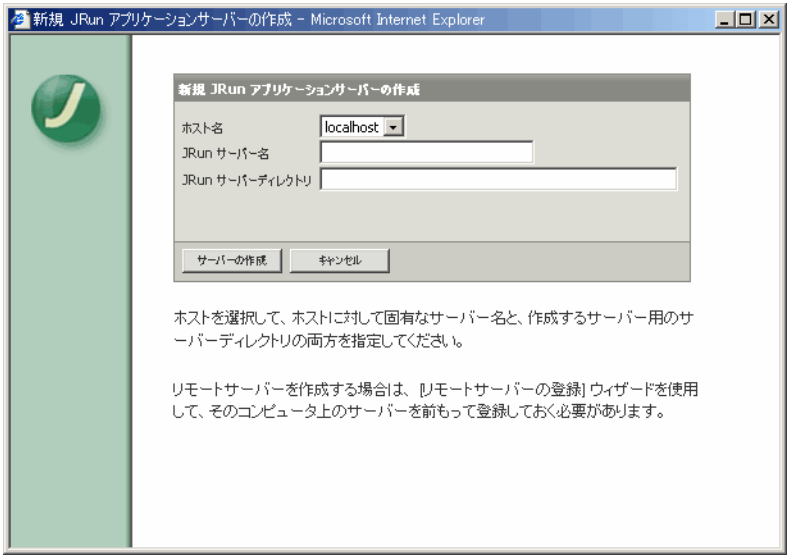
- 多くの企業では、複数の Web アプリケーションから構成されるサイトを構築していますが、複数の JRun サーバーを使用すると、これらの Web アプリケーションを分離できます。いずれかの JRun サーバーが応答しなくなったり再起動が必要になっても、他のサーバーやそれらのアプリケーションは実行を継続できます。
- ユーザーのタイプによって JRun サーバーを論理的に分けることができます。たとえば、生産、品質管理、および管理用に複数の JRun サーバーを使用すると、開発のワークフローを構築するうえで役立ちます。
- JMC を使用すると、サーバークラスタを簡単に設定および管理できます。JRun はクラスタのすべてのメンバーにアプリケーションを自動的にデプロイし、コネクタベースのロードバランスおよびフェイルオーバーのサービスを提供して、Web サーバーおよび JRun サーバーの高い可用性を実現します。

次の手順では、チュートリアルアプリケーション用 JRun サーバーを追加します。

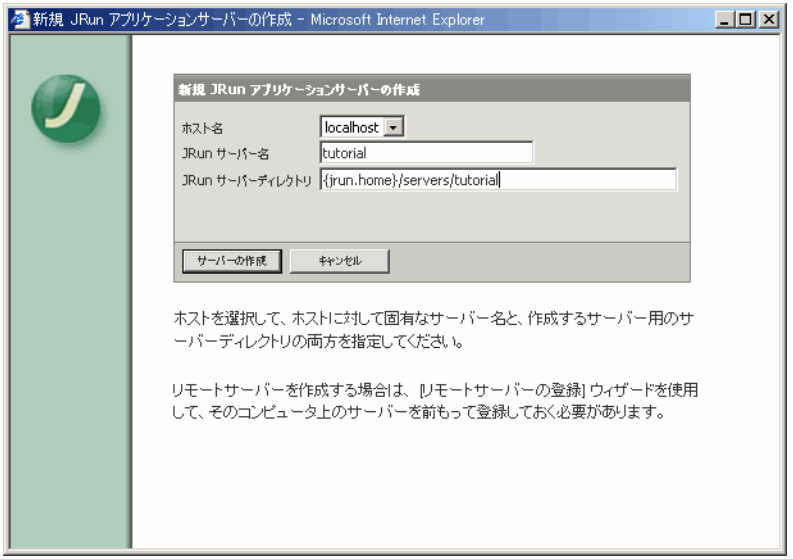
JRun サーバーを追加するには。

1 JMC の上部メニューバーにある [新規サーバーの作成] をクリックします。

[新規 JRun アプリケーションサーバーの作成] ウィンドウが表示されます。



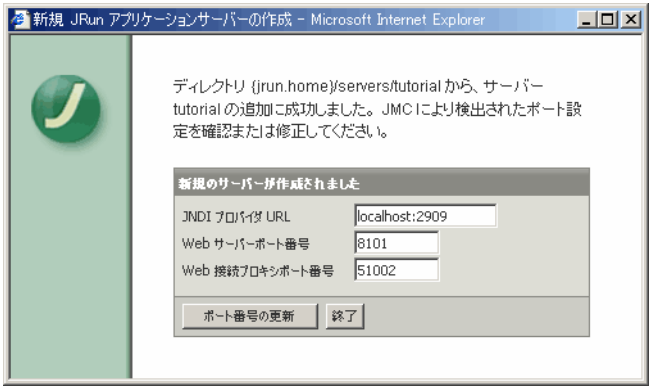
2 デフォルトの [ホスト名] (localhost) はそのままにします。チュートリアルアプリケーションのサーバー名には tutorial と入力し、サーバーディレクトリにはディレクトリ (デフォルトでは {jrun.home}/servers/Tutorial が追加されます) を入力します。



サーバーの作成処理によって、シンボリック変数 {jrun.home} は JRun がインストールされているディレクトリに置き換えられます。

3 [サーバーの作成] をクリックします。

ポート設定のウィンドウが表示されます。



メモ：Web サーバーのポート番号を書き留めておいてください。このポート番号は、tutorial サーバーの compass チュートリアルアプリケーションにアクセスするときに使用します。

4 デフォルトのポート設定をそのままにして、[終了] をクリックします。

5 JMC を更新すると、JMC ホームページの「使用可能サーバー」テーブルおよび左側のメニューに tutorial サーバーが表示されます。

6 [起動] アイコンをクリックして、tutorial サーバーを起動します。

サーバーのステータスは [停止済み] から [実行中] に変わります。

チュートリアルエンタープライズアプリケーションのデプロイ

エンタープライズアプリケーションには、Web アプリケーションおよび EJB が含まれています。通常、エンタープライズアプリケーションは 1 つの圧縮されたエンタープライズアーカイブ (EAR) ファイルとしてデプロイします。EAR ファイルには、すべてのディレクトリ構造と、アプリケーションを定義するすべてのファイルが含まれています。

JRun では、次のいずれかの方法で EAR ファイルをデプロイします。

- **オートデプロイ**　オートデプロイ EAR ファイルをターゲットサーバーのオートデプロイ先ディレクトリにデプロイします。デフォルトのオートデプロイ先は、<JRun のルートディレクトリ >/servers/<JRun サーバー > です。アプリケーションをデプロイする JRun サーバー名は tutorial です。
- **JMC デプロイ**　JMC で、左側のペインにある JRun サーバー名をクリックしてサーバーデプロイウィンドウにアクセスします。「エンタープライズアプリケーション」テーブルで [追加] ボタンをクリックし、EAR ファイルへのパスを参照するかまたは直接入力して、[デプロイ] をクリックします。

また、JRun を使用すると、オープンディレクトリ構造からエンタープライズアプリケーションをデプロイできます。通常、アプリケーションの開発時にはこの方法を使用します。

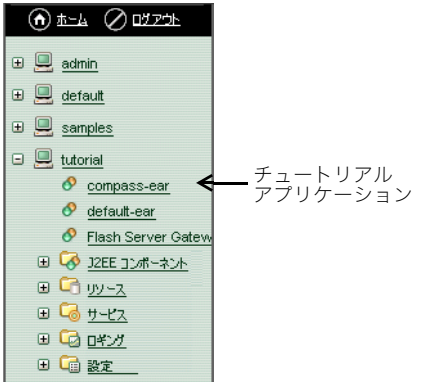
次の手順では、tutorial サーバーに compass チュートリアルアプリケーションのオープンディレクトリをオートデプロイします。次に、JMC を使用して JDBC データソースを追加し、データベースとの接続を確認します。

チュートリアルアプリケーションファイルをデプロイするには

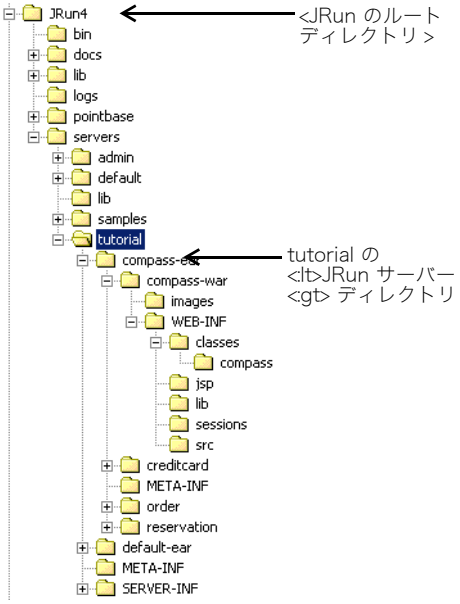
1　< チュートリアルのルートディレクトリ > から compass-ear ディレクトリをコピーします。

2　<JRun のルートディレクトリ >/servers/Tutorial に compass-ear ディレクトリをペーストします。

JMC で、compass チュートリアルアプリケーションが tutorial サーバーにデプロイされたのがわかります。



次の図は JRun のディレクトリ構造を示しています。tutorial サーバーの下を展開するとデプロイ済みチュートリアル (compass) エンタープライズアプリケーションがあります。



JDBC データソースの追加

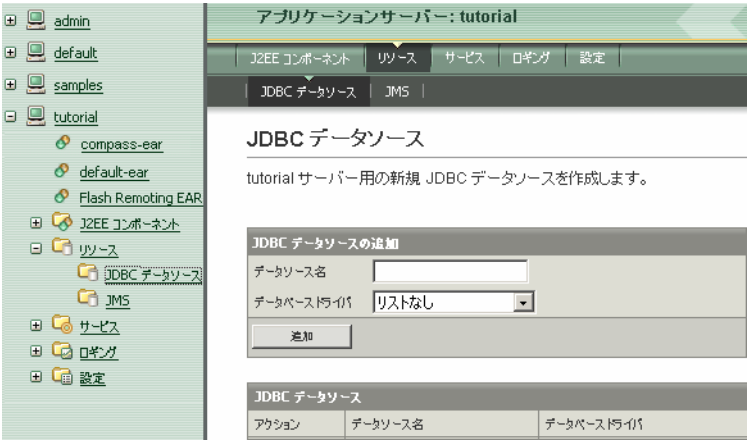
JMC を使用すると、JDBC データソースの追加、削除、編集、およびテストを行うことができます。JMC はデータソース設定値とのインターフェイスを提供し、ほとんどのドライバおよび URL を設定します。

次のセクションでは、JMC を使用してチュートリアルアプリケーションのデータソースを作成します。

新しい JDBC データソースを追加するには

1 JMC の左側のペインで [tutorial] サーバーのメニューを展開し、[リソース] を展開し、[JDBC データソース] をクリックします。

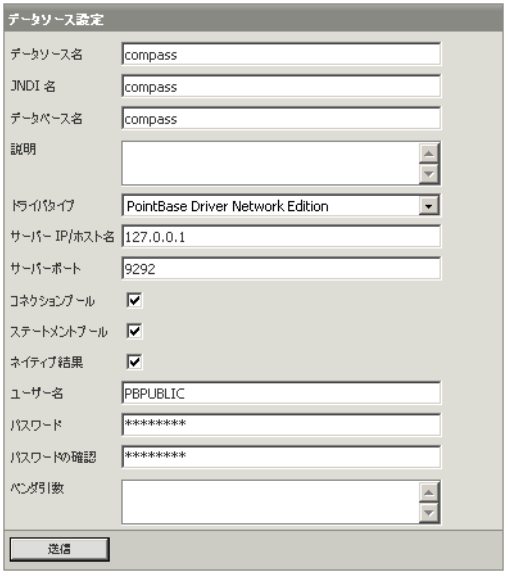
[JDBC データソース] ウィンドウが表示されます。



2 [データソース名] に、「compass」と入力します。[データベースドライバ] ドロップダウンリストボックスで、「 Pointbase Server 」を選択します。[追加] をクリックします。

[データソース設定] ウィンドウが表示されます。

3 [データベース名] に「compass」と入力し、[サーバーポート] を「 9292 」に変更し、[ユーザー名]、[パスワード]、および [パスワードの確認] フィールドに「PBPUBLIC」と入力します。



4 [送信] をクリックします。

compass データソースが「 JDBC データソース」テーブルに追加されます。

5 compass チュートリアルデータベースを起動します。

- (Windows) <JRun のルートディレクトリ >/pointbase ディレクトリに移動し、compassdb.bat をダブルクリックします。
- (Linux/UNIX) compassdb シェルスクリプトを実行します。

6 次のいずれかの方法でサーバーを再起動します。

- JMC で、tutorial サーバーの隣にある [再起動] アイコンをクリックします。
- JRun ランチャーで、tutorial サーバーを選択し、[再起動] ボタンをクリックします。

7 JDBC データソース compass の隣にある [確認] アイコンをクリックして、compass データベースへの接続を確認します。

確認メッセージが表示されます。

8 Web ブラウザを開いて次の URL を入力し、アプリケーションのデプロイを確認します。http://localhost:8101/compass/logon.jsp

メモ：8101 または tutorial サーバーを作成したときに JRun によって割り当てられた Web サーバーのポート番号を使用します (87 ページ を参照)。

チュートリアルアプリケーションのログインページを表示します。

次のチュートリアルレッスンでは、アプリケーションページおよびビジネスロジックをコーディングします。ビジネスロジックには、ログイン、旅行データへのアクセス、旅行の予約が含まれています。

Compass Travel J2EE アプリケーション

Compass Travel チュートリアルアプリケーションでは、アドベンチャー旅コースの説明、予約情報、旅行予約を参照できます。アプリケーションのスタティックおよびダイナミックな部分を作成するには、HTML、サーブレット、JSP、Java Beans、および EJB を使用します。

ログインサーブレットのコーディング

サーブレットとは、サーバーサイド Java プログラムで、Web サーバーの機能を拡張します。サーブレットを使用してダイナミックなコンテンツやロジックを HTML ページに適用します。通常は、Java を使用して、データベースへのアクセスなどの複雑なデータ操作を行うサーブレットを作成します。

サーブレットをテストまたはデプロイするにはサーブレットをコンパイルする必要があります。JRun で WEB-INF/classes ディレクトリのサーブレットのソースコードを変更すると、JRun が呼び出されたときに自動的にサーブレットがリコンパイルおよびリロードされます (この動作はデフォルトで有効になっています)。

次の手順では、ログオンサーブレットをコーディングします。logon.jsp 内のユーザーが入力したユーザー ID およびパスワードを確認します。ログインが有効であれば、home.jsp が表示されます。それ以外の場合は、LogonError.htm が表示されます。この手順では、データベースにアクセスするサーブレットを記述する方法について説明します。

ログオンサーブレットをコーディングするには

1 テキストエディタまたは Java IDE を開き、次のコードをコピーしてペーストします。

```
package compass;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.sql.*;
import javax.sql.DataSource;
import javax.naming.InitialContext;

public class Logon extends HttpServlet {
  private String validLogonPage;
  private String invalidLogonPage;
public void init() throws ServletException {

/* web.xml の初期化パラメータを読み取ります。
  - validLogonPage は、ログオンが有効な場合の移動先ページを指定します。
  - invalidLogonPage は、ログオンが無効な場合の移動先ページを指定します。
  この技法を使用すると、サーブレットにページ名をハードコーディングする必要がなく、
  サーブレットの再利用度が高くなります。 */

validLogonPage = getInitParameter("validLogonPage");
if (validLogonPage==null) throw new ServletException
     ("web.xml エラー：初期化パラメータ 'validLogonPage' が見つかりません。
     ");
invalidLogonPage = getInitParameter("invalidLogonPage");
```

```
if (validLogonPage==null) throw new ServletException
     ("web.xml エラー：初期化パラメータ 'invalidLogonPage' が見つかりません。
     ");
}
protected void doPost(HttpServletRequest request,
     HttpServletResponse response)
    throws ServletException, IOException {
  String userId = request.getParameter("userId");
  String password = request.getParameter("password");

       String firstName="";
       String lastName="";

  String sql = "SELECT password, first_name, last_name
     FROM user WHERE user_id=?";
  Connection connection=null;
  PreparedStatement ps=null;
  ResultSet rs=null;
  boolean validLogon=false;
  try {
    InitialContext ctx=new InitialContext();
    DataSource ds=(DataSource) ctx.lookup("compass");
    connection=ds.getConnection();
    ps=connection.prepareStatement(sql);
    ps.setString(1, userId);
    rs = ps.executeQuery();
    if (rs.next()) {
      if (password.equals(rs.getString("password"))) {
                  validLogon=true;
                  firstName=rs.getString("first_name");
                  lastName=rs.getString("last_name");
            }
    }
  } catch (Exception e) {
    System.out.println(e);
    throw new ServletException(e);
  } finally {
    try {
      rs.close();
      ps.close();
      connection.close();
  } catch (Exception e) {
      System.out.println(e);
    }
  }
```

```
  if (validLogon) {
    HttpSession session = request.getSession();
    session.setAttribute("userId", userId);
    session.setAttribute("firstName", firstName);
    session.setAttribute("lastName", lastName);
    response.sendRedirect(validLogonPage);
  } else {
    response.sendRedirect(invalidLogonPage);
  }
  }
}
```

2 <JRun のルートディレクトリ >/servers/Tutorial/compass-ear/compass-war/WEB-INF/classes/compass に Logon.java としてファイルを保存します。
次の手順で、ダイナミックサーブレットおよびクラスのコンパイルと再ロードの作業を有効にします。

3 次のコードをコピーして *jrun_root*/servers/tutorial/compass-ear/compass-war/WEB-INF/**jrun-web.xml** として保存します。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<jrun-web-app>
<reload>true</reload>
<compile>true</compile>
</jrun-web-app>
```

4 チュートリアルサーバーを次の方法で再起動します。

- JMC で、tutorial サーバーの隣にある [再起動] のアイコンをクリックします。

5 Web ブラウザで次の URL を開きます。

http://localhost:8101/compass/logon.jsp

6 登録されているユーザー ID およびパスワードを入力します。

メモ：compass データベースには、john (パスワード：john) および mary (パスワード：mary) の 2 人が定義されています。

7 [ログオン] ボタンをクリックします。

ログインが有効であれば、空白の「アドベンチャー旅コース」ページが表示されます。

要約

このレッスンでは、JRun を使用して J2EE アプリケーションを記述する開発環境を設定しました。JMC を使用してチュートリアル用 JRun サーバーを追加し、チュートリアルテンプレートアプリケーションをデプロイし、JDBC データソースを定義しました。

チュートリアルアプリケーションを起動するために、compass データベースにアクセスするログオンサーブレットをコーディングし、ユーザー ID およびパスワードを確認しました。ログインに成功すると、Compass の「アドベンチャー旅コース」ホームページが表示されます。

次のステップ

次のレッスンでは、JSP コードを Compass の「アドベンチャー旅コース」ホームページに追加します。詳細については、97 ページのレッスン 2 の「JSP のチュートリアル」を参照してください。

レッスン 2
JSP のチュートリアル

このレッスンでは、Compass Travel チュートリアルアプリケーションの本体を構成する JavaServer Page (JSP) をコーディングします。データベース情報にアクセスする Java コードを埋め込んだ JSP を構築し、JavaBeans を使用する JSP を開発します。

目 次

- ホームページ JSP の作成 98
- JavaBean へのアクセス 101
- 要約 104

ホームページ JSP の作成

JSP を使用すると、HTML、Java、およびスクリプトコードを含むダイナミックページを作成できます。初めてクライアントが JSP を要求するときにページは Java サーブレットに変換およびコンパイルされるので、JSP は高いパフォーマンスを実現します。実行時には、コンパイル済みサーブレットを実行します。

Compass Travel チュートリアルアプリケーション

Compass Travel チュートリアルアプリケーションでは、アドベンチャー旅コースの説明、予約情報、旅行予約を参照できます。アプリケーションのスタティックおよびダイナミックな部分を作成するには、HTML、サーブレット、JSP、Java Beans、および EJB を使用します。

チュートリアルレッスンを開始するには、81 ページのレッスン 1 の「サーブレットのチュートリアル」を参照してください。この章では、チュートリアルのホームページに JSP コードを追加し、JSP を使用してデータベースから情報を検索する JavaBeans にアクセスします。

チュートリアルアプリケーションのホームページである home.jsp では、次の JSP 要素をコーディングします。

- ページ全体の属性を定義し、JSP ページにテキストを挿入する JSP ディレクティブ
- ページ出力をクライアントに送信する前に、データベースにクエリを実行し、サーバーで評価される文を定義する JSP スクリプト要素

home.jsp では、顧客が予約できる旅行のリストを表示します。このレッスンでは、データベース情報にアクセスする、Java コードが埋め込まれた JSP の構築方法について説明します。

JSP コードを Compass Adventures ホームページに追加するには

1 テキストエディタまたは Java IDE で、<JRun のルートディレクトリ >/servers/Tutorial/compass-ear/compass-war にあるファイル **home.jsp** を開きます。

2 ファイルの所定の場所に、次のコードを追加します。

- JSP ページの `import` ディレクティブ

```
<%@ page import="java.sql.*,javax.naming.*"%>
```

- ヘッダーおよび Welcome ページを呼び出す JSP `include` アクション

```
<jsp:include page="header.jsp" flush="true"/>
<jsp:include page="welcome.jsp" flush="true"/>
```

- フッターファイルテキストを挿入する JSP `include` ディレクティブ

```
<%@ include file="footer.htm"%>
```

- アドベンチャー旅コースに関する情報のデータベースに対してクエリを実行する JSP スクリプトレットコード

```
<%
java.text.NumberFormat nf =
   java.text.NumberFormat.getCurrencyInstance();
String sql="SELECT trip_id, name, teaser, price FROM trip";
Connection connection=null;
Statement stmt=null;
ResultSet trips=null;
```

```
    try {
      InitialContext ctx=new InitialContext();
      DataSource ds=(DataSource) ctx.lookup("compass");
      connection=ds.getConnection();
      stmt=connection.createStatement();
      trips=stmt.executeQuery(sql);
      while (trips.next()){
    %>

  <!-- 旅行情報の出力 (tripID はリンクに使用されていますが、名前は表示されているもの
       です。)-->
    <tr>
      <td class="contentBG">
       <a href="tripdetail.jsp?tripId=<%=trips.getInt("trip_id")
       %>">
       <%=trips.getString("name") %></a></td>
      <td class="contentBG"><%=trips.getString("teaser") %></td>
      <td class="contentBG"><%=nf.format(trips.getDouble("price"))
       %></td>
    </tr>
    <%
      }
    } catch (Exception e) {
      out.println(e);
    } finally {
      connection.close();
    }
  %>
```

3 home.jsp を保存し、Web ブラウザで次の URL を開きます。
http://localhost:8101/compass/home.jsp

メモ：8101 または tutorial サーバーを作成したときに JRun によって割り当てられた Web サーバーのポート番号を使用します (87 ページ を参照)。

アドベンチャー旅コース情報のテーブルが表示されます。

次のセッションでは、JavaBeans を使用する JSP の開発方法について説明します。

JavaBean へのアクセス

JavaBeans は JavaBeans API で指定された一連の設計基準に従う特定のタイプの Java クラスです。JavaBeans は特定のベースクラスまたはインターフェイスを継承しませんが、プロパティにアクセスするメソッドを提供する必要があります。JavaBean は EJB とは異なります。

Java 技術には、次に示す方法など、JavaBeans を使用できる方法にはいくつかあります。

- JSP またはサーブレットは bean のインスタンスを作成し、そのメソッドを直接呼び出します。
- JSP は **useBean** アクションを使用して、JavaBeans プロパティを取得および設定し、オプションでそのメソッドを直接呼び出します。

次のチュートリアル手順では、JSP はビジネスロジックをカプセル化した JavaBeans にアクセスして、データベースからデータを検索します。JSP アクションをコーディングし、詳細な旅行情報を検索する JavaBeans のインスタンスの作成および参照を行います。

この手順では、次の JSP アクションを使用します。

要素名	目的
`jsp:useBean`	JavaBeans のインスタンスを定義します。
`jsp:setProperty`	1 つ以上の JavaBeans のプロパティ値を設定します。
`jsp:getProperty`	bean プロパティ値を文字列としてページ出力に書き込みます。

tripdetail.jsp では、**TripBean** JavaBeans のインスタンスを作成し、それを使用して選択した旅行の詳細情報にアクセスします。

JavaBeans のインスタンスを作成するには

1 <JRun のルートディレクトリ >/servers/Tutorial/compass-ear/compass-war にある **tripdetail.jsp** ファイルを開きます。

2 ファイルの所定の場所に、次の JSP コードを追加します。

```
<jsp:useBean id="trip" scope="session" class="compass.TripBean"/>
<jsp:setProperty name="trip" property="tripId"/>
```

3 `jsp:getProperty` コードを確認します。このコードは、旅行名、説明、旅行代金、出発日、お帰り日を出力します。

TripBean JavaBean はデータアクセスロジックを提供し、データベースから旅行に関する情報を検索します。

このチュートリアルレッスンでは、J2EE アプリケーションの開発におけるアプリケーション分割およびロール分離の例を示しています。JSP のユーザーインターフェイスは Web 開発者が構築します。Java についての知識は不要です。ビジネスロジックは Java 開発者が記述します。ページがどのように表示されるかを考える必要はありません。このレッスンでは、**useBean**、**setProperty**、および **getProperty** タグの使用方法についても説明します。

次の手順では、JSP で使用されている JavaBeans の記述方法について説明します。

JavaBeans コードを検証するには

- <JRun ルートディレクトリ >/servers/Tutorial/compass-ear/compass-war/WEB-INF/classes/compass にあるファイル **TripBean.java** を開き、次の Java コードを確認します。
  - 宣言された複数のプライベートフィールドを持つパブリッククラス `TripBean`
  - 宣言されたすべてのプライベートフィールドのパブリック accessor (get) メソッド
  - 次のような ID フィールド用のパブリック mutator (set) メソッド

```
public void setTripId(int tripId) {
  this.tripId=tripId;
  retrieve(tripId);
  }
```

  - データベースにアクセスし、渡された tripID 用の旅行の詳細を検索するパブリック `retrieve` メソッド

```
public void retrieve (int id)
```

次の手順では、`TripBean` Java コードを再使用し、検索したデータを別の形式で表示します。

同じ JavaBeans を別の方法で使用するには

1 Compass Adventures ホームページで、旅行の名前をクリックします。

「旅行コースの詳細」ページが表示されます。

2 「旅行コースの詳細」ページで、[旅の予約] ボタンをクリックします。

予約ページが表示されるので、URL で選択した tripID を参照します。

3 <JRun ルートディレクトリ >/servers/Tutorial/compass-ear/compass-war にある **reservation.jsp** のコードを確認します。

JSP の **include** アクションがファイル tripsummary.jsp を挿入します。

```
<jsp:include page="tripsummary.jsp" flush="true"/>
```

4 <JRun ルートディレクトリ L>/servers/Tutorial/compass-ear/compass-war にある **tripsummary.jsp** のコードも確認します。

このコードは、JSP アクションを使用して **TripBean** JavaBeans のインスタンスを作成および参照して、旅行の要約を表示します。

```
<jsp:useBean id="trip" scope="session" class="compass.TripBean"/>
```

予約ページでは、**TripBean** JavaBeans を再利用して旅行名、出発日、お帰り日、および旅行代金を取得し、別の形式でデータを表示します。

旅行コースの予約

コース	出発日	お帰り日	旅行代金
数学激流コース	2001-08-01	2001-08-05	¥139,599

要約

このレッスンでは、Compass Travel ホームページに JSP コードを追加しました。これには、JSP ディレクティブおよびスクリプトコードが含まれています。JavaBeans を使用してデータベースから詳細な旅行情報を検索する JSP を開発しました。さらに、JavaBeans を再使用して旅行情報を検索し、それを概要テーブル形式で表示しました。

次のステップ

次のレッスンでは、EJB を使用してビジネスロジックを提供し、クレジットカードの支払い確認および旅行予約を行います。

レッスン 3
EJB のチュートリアル

このレッスンでは、EJB を使用して、クレジットカードの支払いの確認および注文の作成を含む、旅行の予約手続きを処理します。

JSP から EJB を呼び出す方法および他の EJB を呼び出す EJB を記述する方法について説明します。

目 次

- EJB の使用 106
- 旅行の予約 109
- 要約 111
- チュートリアルレッスンのまとめ 111

EJB の使用

EJB は Java を使用した分散型のオブジェクト指向ビジネスアプリケーションの構築にコンポーネントアーキテクチャを提供します。EJB をサポートするアプリケーションサーバーを使用すると、Java 開発者は再利用可能なコンポーネントとしてエンタープライズアプリケーションのビジネスロジックを実装することに集中できます。EJB サーバーは、コンポーネントのライフサイクル、状態の管理、パーシスタンス、マルチスレッド処理、コネクションプール、トランザクション管理、セキュリティなどの複雑な下位レベルサービスを処理します。

EJB 仕様では次のタイプの Enterprise JavaBeans を定義しています。

- **セッション bean**　ビジネスプロセスを表します。
- **エンティティ bean**　ソースからのデータのオブジェクト表示を表します。通常はリレーショナルデータベースの行です。
- **メッセージ駆動型 bean** (Message Driven Beans)　EJB および Java Message Service (JMS) のリスナの機能を結合します。

次のコンストラクトは EJB にアクセスするときに使用できます。

コンストラクト	拡張クラス	説明
リモートホームインターフェイス	`EJBHome`	EJB コンテナ外のアプリケーションからアクセスできる EJB のライフサイクルメソッドを定義します。これは EJB インスタンスを作成 (取得)、検索、または削除するメソッドです。
ローカルホームインターフェイス (EJB 2.0 の新機能)	`EJBLocalHome`	クライアントの代わりに、EJB インスタンスを作成 (取得)、検索、または削除するメソッドを定義します。このメソッドは同じ EJB コンテナに配置されています。
リモートインターフェイス	`EJBObject`	EJB コンテナ外のアプリケーションからアクセスできる EJB のビジネスメソッドを定義します。EJB が外部に向けてその動作を示すビジネスメソッドです。
ローカルインターフェイス (EJB 2.0 の新機能)	`EJBLocalObject`	同じ EJB コンテナにある他の EJB が使用できる EJB ビジネスメソッドを定義します。ローカルインターフェイスを使用すると、分散型のオブジェクトプロトコルのオーバーヘッドがなくても、EJB は相互作用できます。
EJB 実装クラス		ビジネスおよびライフサイクルメソッドの実装を提供します。リモートインターフェイスおよびローカルインターフェイスでの署名が一致するメソッド、およびリモートおよびローカルホームインターフェイスの両方のメソッドの一部に対応するメソッドがある必要があります。

JSP からの EJB の呼び出し

次の手順では、JSP スクリプトレットをコーディングして、セッション bean である予約用 EJB にアクセスします。次に、リモートインターフェイス、リモートホームインターフェイス、および bean の実装クラスのコードを検証します。

JSP から EJB にアクセスするには

1 <JRun のルートディレクトリ>/servers/tutorial/compass-ear/compass-war にあるファイル **reservationaction.jsp** を開きます。

2 ファイル内の所定の場所に、次の JSP コードを入力します。

```
<%
// ログオンサーブレットで設定します。
String type=request.getParameter("type");
String number=request.getParameter("number");
String expiration=request.getParameter("expiration");
String userId=(String) session.getAttribute("userId");
int tripId=trip.getTripId();
double price=trip.getPrice();

try {
  // ネームサーバーと通信する初期コンテキストオブジェクトが必要です。
  javax.naming.InitialContext ctx=new
     javax.naming.InitialContext();
  //lookup メソッドを使用すると、ネームサーバーを照会して EJB を配置し、
     // ホームインターフェイスへのリファレンスを返します。
  Object obj=ctx.lookup("Reservation");
  // リモートホームインターフェイスのデータタイプのリファレンスを配列します。
  ReservationHomeRemote home=(ReservationHomeRemote)
     javax.rmi.PortableRemoteObject.narrow
     (obj, ReservationHomeRemote.class);
  // リモートインターフェイスへのリファレンスを取得します。
  ReservationRemote reservation=home.create();
  // 旅行を予約する reserve メソッドを呼び出します。
  int id=reservation.reserve(userId, tripId, price, type, number,
     expiration);
  %>
  ありがとうございます。予約内容は確認されました。<br><br>
  予約番号は次のとおりです：<%= id %><br>
  <%
  } catch (Exception e) {
  %>
  申しわけありません。この旅行を予約できませんでした。
    <br><%= e.getMessage() %>
  <%
    }
  %>
```

3 ファイルを保存し Web ブラウザで Compass Adventures ホームページを開きます。
http://localhost:8101/compass/home.jsp

メモ：8101 または tutorial サーバーを作成したときに JRun が割り当てた Web サーバーのポート番号を使用します (87 ページ の 手順 3 を参照)。

4 旅行を選んで予約します。

メモ：旅行を予約するには、定義済みユーザー (名前 john と パスワード john、または名前 mary とパスワード mary) としてログインする必要があります。

- Visa カードは使用できますが、他のクレジットカードは使用できません。
- [番号] フィールドに番号を入力します。
- [有効期限] フィールドに MM/YY (月 / 年) を入力します。

予約ができると Reservation.jsp によって確認 ID が表示され、問題が発生した場合はエラーメッセージが表示されます。

セッション bean を使用して、EJB 間の相互作用の調整や、タスクを実行するために複雑な操作を行うことはよくあります。旅行リストの検索、ATM との相互作用、予約の作成など、アプリケーションでのタスクとロジックを記述するためにセッション bean を使用します。次の手順では、予約用セッション bean を使用して、アドベンチャー旅コースを予約します。

旅行の予約

旅行を予約するには、クレジットカードを確認し、旅行の注文を作成する必要があります。予約用 EJB は旅行の予約手続きを処理するワークフローセッション bean です。予約用 EJB は、クレジットカード用 EJB および注文用 EJB を呼び出します。

- クレジットカード用 EJB は、`validate` メソッドを持つセッション bean です。
- 注文用 EJB はエンティティ bean です。エンティティ bean はデータベースの行を追加 (`ejbCreate`)、削除 (`ejbRemove`)、更新 (`ejbStore`)、および検索 (`ejbLoad`) するメソッドを提供します。

予約用 EJB は、旅行の予約手続きを処理するステートレスセッション bean です。まず、クレジットカード用 EJB を呼び出し、ユーザーのクレジットカードを確認します。次に予約用 EJB が注文用 EJB を呼び出して新規注文を作成します。

事前にデプロイされた次の EJB の Java ソースコードを検証します。

予約用 EJB のコードの検証

予約用 EJB の Java ソースコードは、<チュートリアルのルートディレクトリ>/compass-ear/reservation/compass ディレクトリにあります。次のファイルを開いて、コードを確認します。

- **ReservationRemote.java**　リモートインターフェイスには、EJB　で使用できるビジネスメソッドがリストされます。この EJB には、旅行を予約できる `reserve` と呼ばれるビジネスメソッドが 1 つだけあります。
- **ReservationHomeRemote.java**　この EJB では、引数を取らない `create` メソッドを使用してインスタンスが生成されます。
- **ReservationBean.java**　EJB 実装クラスは次のタスクを実行します。
  - `ejbRemove`、`ejbPassivate`、`ejbActivate` のような `SessionBean` インターフェイスで定義されたメソッドの実装を行います。
  - ホームインターフェイスで定義された各 `create` メソッドに対応する `ejbCreate` メソッドを提供します。
  - リモートインターフェイスに定義された各ビジネスメソッドの実装を行います。
- **ejb-jar.xml**　<JRun のルートディレクトリ>/servers/tutorial/compass-ear/reservation/META-INF にある予約用 EJB の実行時の属性を含む XML ファイルである、デプロイメントディスクリプタを確認します。予約用 EJB をステートレスセッション bean として定義します。

`reserve` メソッドは旅行の予約手続きを処理します。まずクレジットカード用 EJB のインスタンスを取得し、次にクレジットカード用 EJB `validate` メソッドを呼び出します。その後、注文用 EJB のインスタンスを取得し、注文用 EJB の `create` メソッドを呼び出して、データベースに新規注文を挿入します。

この手順では、他の EJB を呼び出す EJB を記述する方法について説明し、セッション bean を使用して手続きを処理する方法も示しています。

クレジットカード用 EJB のコードの検証

クレジットカード用 EJB の Java ソースコードは、< チュートリアルのルートディレクトリ >/compass-ear/creditcard/compass ディレクトリにあります。次のファイルを開いて、コードを確認します。

- **CreditCardRemote.java**　この EJB には、ユーザーのクレジットカードを確認する `validate` と呼ばれるビジネスメソッドが 1 つだけあります。
- **CreditCardHomeRemote.java**　この EJB では、引数を取らない `create` メソッドを使用してインスタンスが生成されます。
- **CreditCardBean.java**　この EJB はビジネスロジックを提供してクレジットカードを確認するセッション bean です。この例では、確認のロジックは簡素化されています。Visa カードは使用できますが、他のクレジットカードは使用できません。
- **ejb-jar.xml**　<JRun のルート ディレクトリ >/servers/tutorial/compass-ear/creditcard/META-INF にあるクレジットカード用 EJB デプロイメントディスクリプタを確認します。クレジットカード用 EJB をステートレスセッション bean として定義します。

この手順では、基本的なセッション bean を記述する方法について説明します。

注文用 EJB のコードの検証

注文用 EJB の Java ソースコードは、< チュートリアルのルートディレクトリ >/compass-ear/order/compass ディレクトリにあります。次のファイルを開いて、コードを確認します。

- **OrderRemote.java**
- **OrderHomeRemote.java**　この EJB には、新規注文の作成方法を定義する `create` メソッドがあります。新規注文を作成するには、customerId、tripId、ccType、ccNumber、ccExpiration パラメータを提供する必要があります。
- **OrderBean.java**　この EJB は注文を追加 (`ejbCreate`)、削除 (`ejbRemove`)、更新 (`ejbStore`)、および検索 (`ejbLoad`) するメソッドを提供します。
- **ejb-jar.xml**　<JRun のルートディレクトリ >¥servers¥tutorial¥compass-ear¥order¥META-INF にある注文用 EJB デプロイメントディスクリプタを確認します。注文用 EJB を、bean 管理パーシスタンスを持つエンティティ bean として定義します。

この手順では、bean 管理パーシスタンスを使用してエンティティ bean を記述する方法について説明します。

要約

このチュートリアルレッスンでは、EJB を使用して、クレジットカードの支払いの確認および注文の作成を含む、旅行の予約手続きを処理しました。

JSP から EJB を呼び出す方法および他の EJB を呼び出す EJB を記述する方法について学習しました。また、基本的なセッション bean およびエンティティ bean のコードを検証しました。セッション bean が手続きを処理する方法についても学習しました。

チュートリアルレッスンのまとめ

チュートリアルレッスンでは、旅行を選択し、予約する基本的な J2EE アプリケーションを作成しました。JRun を使用して、チュートリアル用のアプリケーションサーバーを追加し、Compass チュートリアルアプリケーション EAR ファイルをデプロイしました。その後のレッスンでは、Java サーブレットをコーディングし、Java コードを JSP (JavaServer Page) に追加しました。JSP から JavaBeans および EJB にアクセスしました。他の EJB を呼び出して、EJB が手続きを処理する方法について学習しました。基本的で機能的な J2EE Web アプリケーションは、これらすべてのコンポーネントによって構成されています。

次のステップ

J2EE アプリケーションのコード作成の詳細については、JRun samples サーバーを起動し、http://localhost:8200 で Compass Travel のアプリケーションを確認して、jrun_root/servers/samples/compass-ear にあるアプリケーションファイルを参照してください。

次のレッスン 113 ページのレッスン 4 の「Web サービスのチュートリアル」に進み、Web サービスを使用して Compass Travel サンプルアプリケーションからアドベンチャー旅コースにアクセスして予約する Web アプリケーションをデプロイしましょう。

レッスン 4
Web サービスのチュートリアル

このチュートリアルレッスンでは、TravelNet という Web アプリケーションをデプロイします。TravelNet では、Web サービスを使用して Compass Travel サンプルアプリケーションからアドベンチャー旅コースにアクセスして予約します。

Web サービスチュートリアルレッスンは、前の章のサーブレット、JSP、または EJB のチュートリアルレッスンを終了していなくても行うことができます。

目 次

- JRun Web サービスの使用 114
- JRun 入門 116
- TravelNet エンタープライズアプリケーションのデプロイ 121
- Web サービスの作成 122
- パブリッシュ済み Web サービスからの WSDL の生成 123
- Web サービスプロキシクライアントの生成 126
- 要約 131

JRun Web サービスの使用

JRun を使用すると、**Web サービス**をパブリッシュして使用することができます。Web サービスは、HTTP などの標準インターネットプロトコルと XML を使用して、プラットフォームや場所に依存しないコンピューティングを行います。以前は互換性のなかったアプリケーションを、言語、プラットフォーム、またはオペレーティングシステムに関係なく、Web 上で相互運用できるようにすることで、Web サービスは新しいビジネスチャンスを作り出し、企業がビジネス関係の変化に適応できるようにします。たとえば、Microsoft .NET コンポーネントは、EJB (Enterprise JavaBean) などの J2EE コンポーネントと通信できます。

JRun を使用すると、既存の Java コードを Web サービスとして再利用したり、Web サービスとしてパブリッシュするための新規コードを記述したりすることができます。これらのサービスが Java 以外のプラットフォームに存在する場合でも、リモートの Web サービスでメソッドを呼び出すことができるオブジェクトベースおよびタグベースのクライアントを作成することもできます。

Web サービスを開発、デプロイ、および使用するユーザーを次に示します。

- **作成者**　Web サービスを作成して定義します。Java を使用して Web サービス機能を作成し、XML を使用して WSDD (Web Services Deployment Descriptor) ファイルに Web サービスを定義します。Java コードを記述する人は、必ずしもそのコードを Web サービスとしてパブリッシュする人ではありません。
- **デプロイ担当者**　Web サービスデプロイ担当者は、Web サービスパブリッシャでもある場合があります。デプロイ担当者は WSDL (Web Services Description Language) ファイルを生成します。このファイルは、Web サービスの目的、場所、およびアクセス情報を記述する XML ドキュメントです。Web サービスのエンドポイント URL にアクセスすると、WSDL ファイルをクライアントアプリケーションからダイナミックに生成することもできます。

  さらに、Web サービスプロキシを生成し、クライアントコードを記述する人もいます。
- **コンシューマ**　コンシューマはアプリケーションで Web サービス機能を使用します。

JRun Web サービスの詳細については、『JRun プログラマーガイド』を参照してください。

TravelNet Web サービスアプリケーション

このチュートリアルレッスンでは、旅行代理店がアドベンチャー旅コースの販売および予約に使用できる Web サービスクライアントアプリケーション TravelNet をデプロイします。TravelNet は、独自の旅行データベースを持っていませんが、Compass Travel という旅行業者の旅行商品を転売します。TravelNet では、Compass Travel アプリケーションの Web サービスを使用します。

Web サービスチュートリアルレッスンでは、次の作業を行います。

1. Compass Travel アプリケーションの WSDD ファイルで Web サービスとして公開される既存の Java クラスを検証します。
2. WSDL ファイルを作成して、Web サービスの呼び出しに必要な情報を提供します。
3. 特定の Web サービスの WSDL からプロキシクライアントを生成します。

開発環境の設定

Web サービスチュートリアルレッスンは、前の章のサーブレット、JSP、または EJB のチュートリアルレッスンを終了していなくても行うことができます。レッスン 1、81 ページの「サーブレットのチュートリアル」を終了している場合は、121 ページの「TravelNet エンタープライズアプリケーションのデプロイ」に直接進むことができます。

チュートリアルレッスンを開始する前に、開発環境を正しく設定してください。

開発環境を設定するには

1 次のソフトウェアをインストールします。

ソフトウェア	必要条件
Web サーバー	• JRun 提供の Web サーバー (JWS) • サポートされている外部 Web サーバー (オプション)
テキストエディタ	• 任意のテキストエディタ • Java 統合開発環境 (IDE) • Macromedia Dreamweaver などの HTML エディタ
JRun 4	

JRun 4 をインストールするためのシステム必要条件については、『JRun インストールガイド』を参照してください。

2 チュートリアルファイルは、<JRun のルートディレクトリ >/docs/tutorialfiles ディレクトリにあります。このディレクトリは、< チュートリアルのルートディレクトリ > と呼ばれます。

3 (オプション) JRun と連動する Java IDE を設定します。

詳細については、第 1 章の「JRun と Java IDE の併用」を参照してください。

JRun 入門

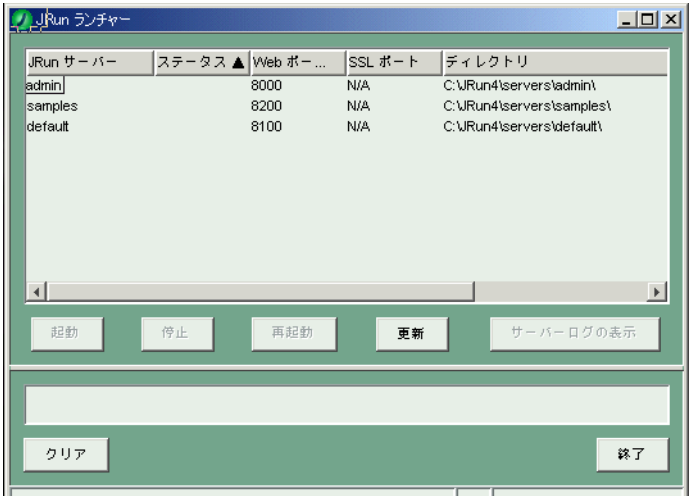
admin JRun サーバーを起動して、JRun の操作を開始します。

admin JRun サーバーを起動するには

- <JRun のルートディレクトリ >/bin ディレクトリで、JRun コマンドラインユーティリティを使用します (<JRun のルートディレクトリ> は JRun がインストールされているディレクトリです)。

  ```
  jrun -start admin
  ```

- (Windows のみ) **[スタート] > [プログラム] > [Macromedia JRun 4] > [JRun ランチャー]** を選択します。admin サーバーを選択し、[起動] ボタンをクリックします。



JMC (JRun 管理コンソール) を使用するには、admin JRun サーバーを実行しておく必要があります。最初のチュートリアル手順では、JMC を使用して JRun アプリケーションサーバーを追加します。

JMC を開始するには

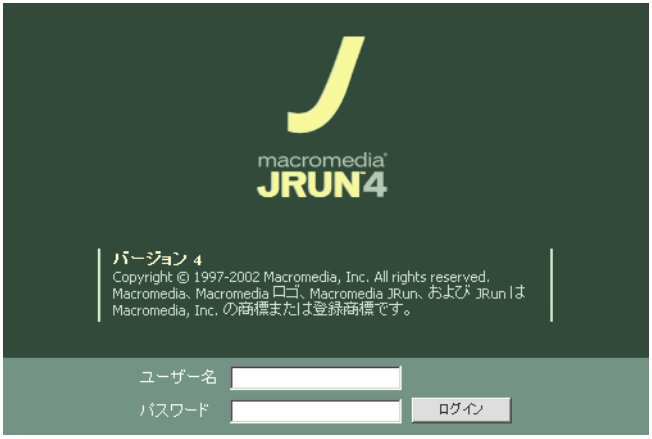
1 次のいずれかの方法で JMC を起動します。

- Web ブラウザを開き、次の URL を入力します。
  http://localhost:8000
- この手順は、デフォルトの JWS ポート 8000 で JMC に接続している場合を想定しています。デフォルトのポートを使用しない場合は、8000 を admin Web サーバーのポート番号に変更してください。admin Web サーバーのポート番号は、<JRun のルートディレクトリ >¥servers¥admin¥SERVER-INF¥jrun.xml ファイル内の次の行にあります。

  ```
  <service class="jrun.servlet.http.WebService"
   name="WebService">
  <attribute name="port">8000</attribute>
  <attribute name="interface">*</attribute>
  </service>
  ```

- (Windows のみ) **[スタート] > [プログラム] > [Macromedia JRun 4] > [JRun 管理コンソール]** を選択します。

JMC のログインウィンドウが表示されます。



2 ユーザー名とパスワードを入力し、[ログイン] をクリックします。

メモ : ユーザー名とパスワードは JRun のインストール時に設定しています。『JRun インストールガイド』を参照してください。

JRun のホームページが表示されます。

次のセクションでは、JMC を使用してチュートリアルアプリケーションのために JRun サーバーを追加します。

JRun サーバーの追加

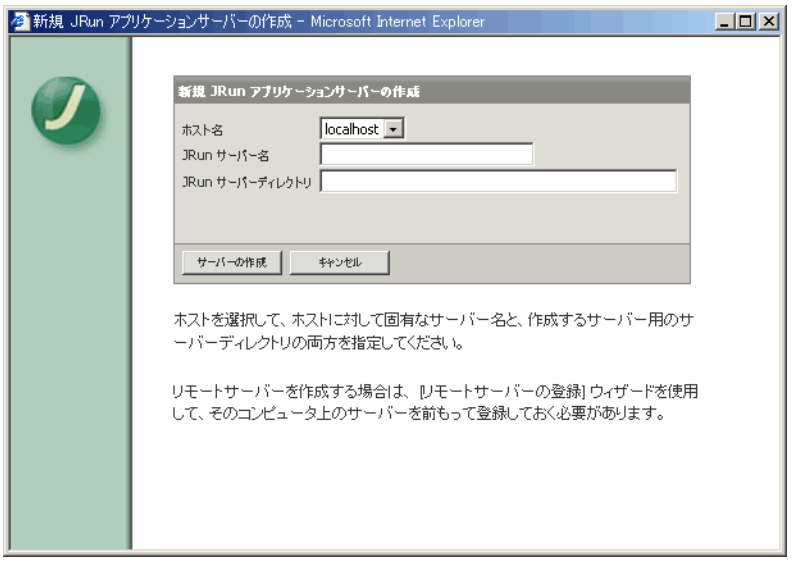
インストール時に、JRun では admin、default、および samples という 3 つの JRun サーバーが作成されます。admin サーバーはデフォルトで JMC をホスティングします。次のセクションでは、JMC を使用して JRun サーバーを追加する方法について説明します。JRun サーバーの追加や削除を行う場合は、それぞれのサーバーに固有のポートが必要であることに注意してください。JRun ポートの詳細については、『JRun 管理者ガイド』を参照してください。

次の手順では、チュートリアルアプリケーション用の JRun サーバーを追加します。

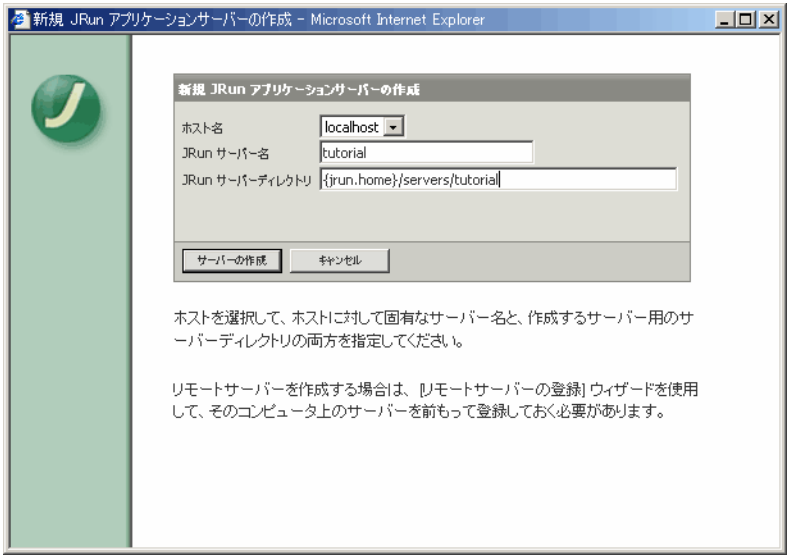
JRun サーバーを追加するには

1 JMC の上部メニューバーにある [新規サーバーの作成] をクリックします。

[新規 JRun アプリケーションサーバーの作成] ウィンドウが表示されます。



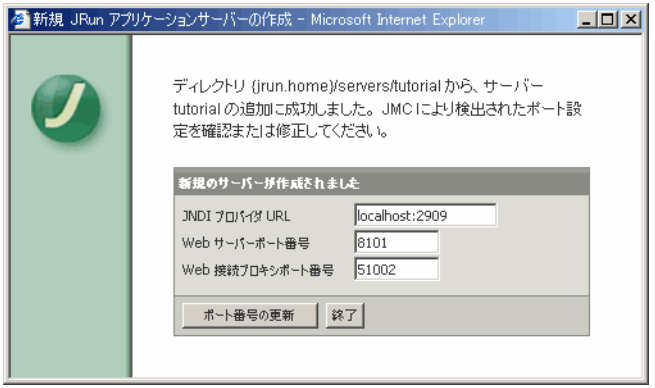
2 デフォルトの [ホスト名] (localhost) はそのままにします。チュートリアルアプリケーションのサーバー名には tutorial と入力し、サーバーディレクトリにはディレクトリ (デフォルトでは {jrun.home}/servers/Tutorial が追加されます) を入力します。



サーバーの作成処理によって、シンボリック変数 {jrun.home} は JRun がインストールされているディレクトリに置き換えられます。

3 [サーバーの作成] をクリックします。

ポート設定のウィンドウが表示されます。



メモ：Web サーバーのポート番号を書き留めておいてください。このポート番号は、tutorial サーバーの TravelNet アプリケーションにアクセスするときに使用します。

4 デフォルトのポート設定をそのままにして、[終了] をクリックします。

5 JMC を更新すると、JRun ホームページの「使用可能サーバー」テーブルおよび左側のペインに tutorial サーバーが表示されます。

6 [起動] アイコンをクリックして、tutorial サーバーを起動します。

サーバーのステータスは [停止済み] から [実行中] に変わります。

TravelNet エンタープライズアプリケーションのデプロイ

通常、エンタープライズアプリケーションは 1 つの圧縮されたエンタープライズアーカイブ (EAR) ファイルとしてデプロイします。EAR ファイルには、すべてのディレクトリ構造と、アプリケーションを定義するすべてのファイルが含まれています。

JRun では、次のいずれかの方法で EAR ファイルをデプロイします。

- **オートデプロイ** EAR ファイルをターゲットサーバーのオートデプロイ先ディレクトリにデプロイします。デフォルトのオートデプロイ先は、<JRun のルートディレクトリ >/servers/<JRun サーバー> です。アプリケーションをデプロイする JRun サーバー名は tutorial です。
- **JMC デプロイ** JMC で、左側のペインの JRun サーバー名をクリックしてサーバーデプロイページにアクセスします。「エンタープライズアプリケーション」テーブルで [追加] ボタンをクリックし、EAR ファイルへのパスを参照するかまたは入力し、[デプロイ] をクリックします。

また、JRun を使用すると、オープンディレクトリ構造からエンタープライズアプリケーションをデプロイできます。通常、アプリケーションの開発時にはこの方法を使用します。

次の手順では、tutorial サーバーに TravelNet アプリケーションのオープンディレクトリをオートデプロイします。

チュートリアルアプリケーションファイルをデプロイするには

1 < チュートリアルのルートディレクトリ > から travelnet-ear ディレクトリをコピーします。

2 <JRun のルートディレクトリ >/servers/tutorial に travelnet-ear ディレクトリをペーストします。

TravelNet チュートリアルアプリケーションは tutorial サーバーにオートデプロイされました。

3 Web ブラウザを開いて次の URL を入力し、アプリケーションのデプロイを確認します。http://localhost:8101/travelnet/home.jsp

メモ : 8101 または tutorial サーバーを作成したときに JRun によって割り当てられた Web サーバーのポート番号を使用します (119 ページ を参照)。

Web サービスの作成

Compass Travel サンプルアプリケーションでは、アドベンチャー旅コースの説明や予約情報を参照したり、旅行を予約したりできる旅行予約システムをシミュレーションします。Compass Travel では、WSDD ファイルで Trip および Reservation という 2 つの Web サービスを公開します。

JRun での Web サービスのオーサリングは、Web アプリケーションの WEB-INF ディレクトリで正しく設定された WSDD ファイルと Java クラスの挿入、およびアプリケーションのデプロイから構成されます。次の手順では、Compass Travel アプリケーションの WSDD ファイルで Trip および Reservation Web サービスコードを検証します。

Compass Travel アプリケーションの WSDD ファイルには、一般的な設定情報の他に、Web サービスとして公開する Trip および Reservation Java クラスの **service** 要素があります。**className** 要素は Java 実装クラスの名前であり、**methodName** は許可されたメソッドの名前です。tutorial サーバーで実行する TravelNet では、samples サーバーで実行する Compass Travel アプリケーションからこれらの Web サービスを使用します。

WSDD コードを検証するには

- <JRun のルートディレクトリ >/servers/samples/compass-ear/compass-war/WEB-INF/**server-config.wsdd** ファイルを開き、次のコードを確認します。

```
<service name="Trip" provider="java:RPC">
  <parameter name="methodName" value="getInfo getList"/>
  <parameter name="className" value="compass.Trip"/>
  <beanMapping languageSpecificType="java:compass.TripInfo"
     qname="ns1:TripInfo" xmlns:ns1="http://compass"/>
 </service>

 <service name="Reservation" provider="java:RPC">
  <parameter name="methodName" value="reserve"/>
  <parameter name="className" value="compass.Reservation"/>
 </service>
```

このコードでは、次の情報に注意します。

- Trip Web サービスでは、Java クラス **compass.Trip** で **getInfo** および **getList** という 2 つのメソッドを公開します。
- Reservation Web サービスでは、Java クラス **compass.Reservation** で **reserve** というメソッドを公開します。

次の手順では、Web サービスごとに WSDL ファイルを生成します。

パブリッシュ済み Web サービスからの WSDL の生成

WSDL ファイルを作成することによって、Web サービスの呼び出しに必要な情報を提供します。この WSDL ファイルとは、目的、場所、および Web サービスへのアクセス方法に関する情報を記述する XML ドキュメントです。また、WSDL ファイルには、ユーザーが呼び出すメソッドおよびそれに関連するデータタイプも記述されています。

JRun を使用すると、URL の最後に **?WSDL** を追加することによって、パブリッシュ済み Web サービスの WSDL ドキュメントを生成できます。次のチュートリアル手順では、samples サーバー上の Compass Travel アプリケーションから WSDL ファイルを生成し、そのファイルを tutorial サーバー上の TravelNet アプリケーションに保存します。

WSDL ファイルを生成するには

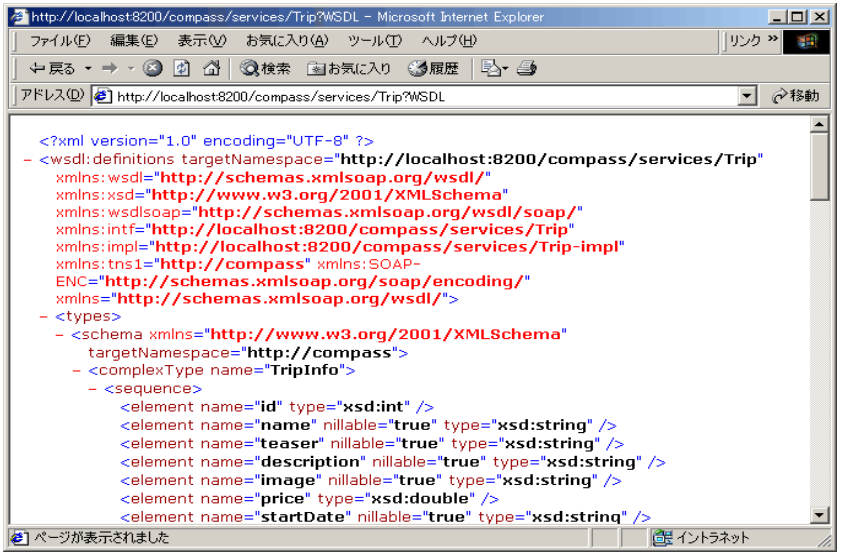
1 次のいずれかの方法で samples サーバーを起動します。

- JMC で、samples サーバーの隣にある [起動] アイコンをクリックします。
- JRun ランチャーで、samples サーバーを選択し、[起動] ボタンをクリックします。

2 Web ブラウザに次の URL を入力します。

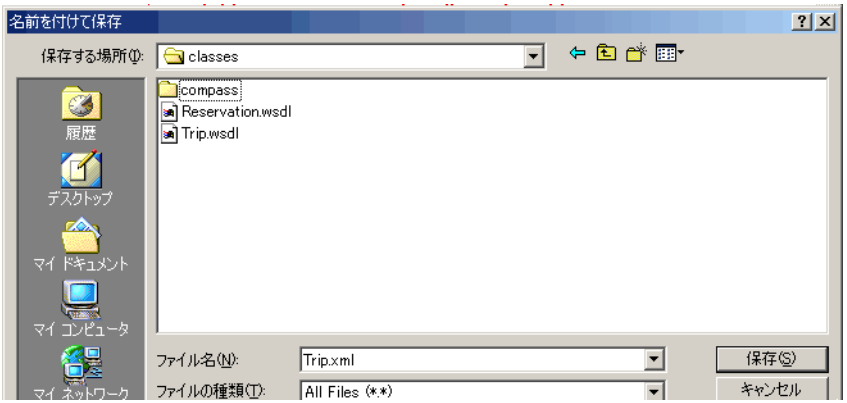
http://localhost:8200/compass/services/Trip?WSDL

Trip.wsdl ドキュメントが表示されます。



3 Web ブラウザのメニューバーで **[ファイル] > [名前を付けて保存]** を選択します。

[名前を付けて保存] ウィンドウが表示されます。



4 ファイル名に Trip.wsdl と入力します。

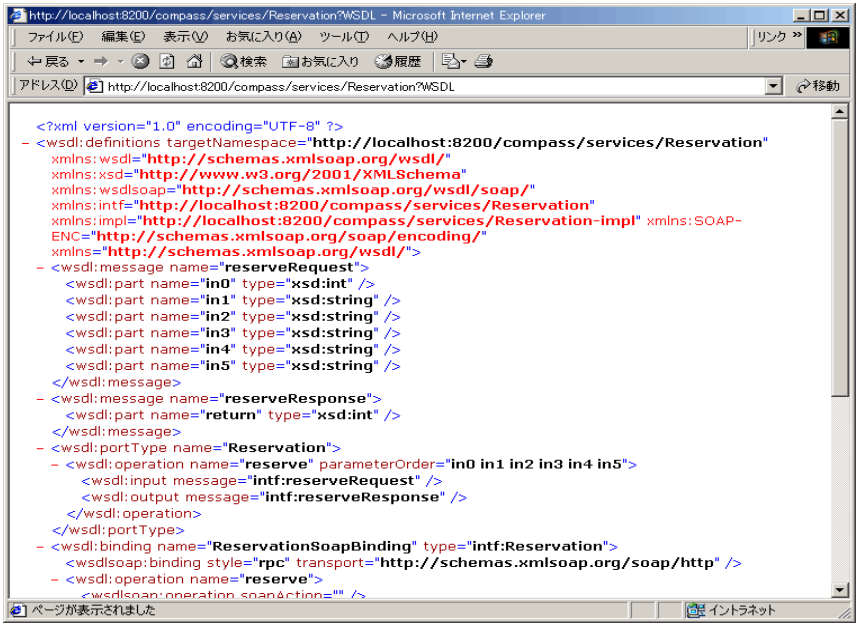
メモ：[ファイルの種類] ドロップダウンリストボックスで [All Files (*.*)] を選択し、[保存] をクリックします。ファイル名に .xml が追加されます。

5 このファイルを <JRun のルートディレクトリ >¥servers¥tutorial¥travelnet-ear¥travelnet-war¥WEB-INF¥classes ディレクトリに保存します。

6 Web ブラウザに次の URL を入力します。

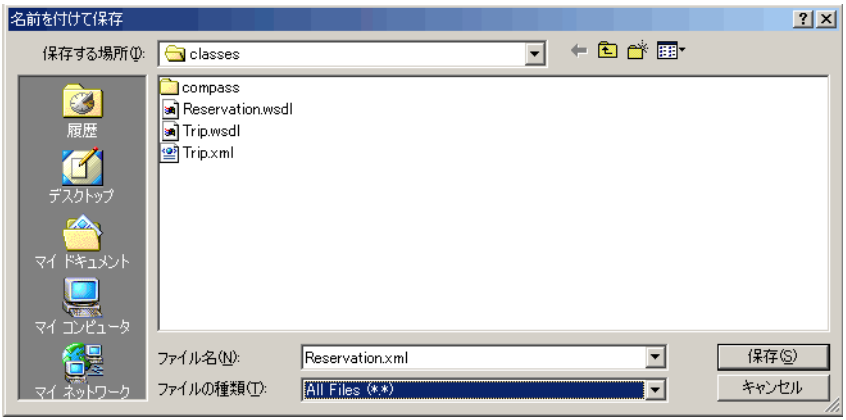
http://localhost:8200/compass/services/Reservation?WSDL

Reservation.wsdl ドキュメントが表示されます。



7 Web ブラウザのメニューバーで **[ファイル] > [名前を付けて保存]** を選択します。

[名前を付けて保存] ウィンドウが表示されます。



8 ファイル名 Reservation.wsdl を入力します。

メモ : [ファイルの種類] ドロップダウンリストボックスで [All Files (*.*)] を選択し、[保存] をクリックします。ファイル名に .xml が追加されます。

9 このファイルを <JRun のルートディレクトリ >¥servers¥tutorial¥travelnet-ear¥travelnet-war¥WEB-INF¥classes ディレクトリに保存します。

WSDL ファイルを使用して Trip および Reservation Web サービスのプロキシクライアントを生成します。

Web サービスプロキシクライアントの生成

プロキシクライアントによって Web サービスオペレーションが呼び出されます。**プロキシクライアント**は、特定の Web サービスのために、WSDL ファイルから生成されたローカルプロキシ上にメソッドを呼び出します。プロキシオブジェクトは、リモート Web サービスとの相互作用を処理します。プロキシの生成には WSDL2Java ツールを使用します。

WSDL2Java ツールは、Web サービスの WSDL ドキュメントの情報に基づいて Web サービスプロキシを生成します。**プロキシ**とは、実際の Web サービスと同じインターフェイスを持つローカルオブジェクトのことです。これを使用すると、Web サービスをローカルオブジェクトであるかのように呼び出すことができます。

次の手順では、WSDL2Java ツールを使用して Trip および Reservation Web サービスのプロキシコードを生成します。その後、プロキシコードを使用して、ターゲット Web サービスを呼び出すプロキシクライアントを作成します。

プロキシコードを生成するには

- コマンドウィンドウを開き、次のコマンドを入力します。

```
>cd <JRun のルートディレクトリ>¥servers¥tutorial¥travelnet-ear¥
     travelnet-war¥WEB-INF¥classes
>setenv
```

メモ：setenv.bat ファイルでは、JRUN_HOME がデフォルトの JRun インストールディレクトリ (<JRun のルートディレクトリ >) に設定されています。デフォルトのディレクトリ以外の場所に JRun をインストールした場合は、setenv.bat ファイルの JRUN_HOME を編集してください。

UNIX ユーザー　login、profile ファイル、またはシェル rc/profile ファイルに次のように追加してください。

```
JRUN_HOME=/path/to/JRun4
PATH=$PATH:$JRUN_HOME/bin
CLASSPATH=$CLASSPATH:$JRUN_HOME/lib/jrun.jar:$JRUN_HOME/lib/
     webservices.jar
>wsdl2java -p compass Trip.wsdl
>wsdl2java -p compass Reservation.wsdl
>cd compass
>javac *.java
```

WSDL2Java のコマンドラインスイッチの詳細については、『JRun プログラマーガイド』を参照してください。

<JRun のルートディレクトリ >¥servers¥tutorial¥travelnet-ear¥travelnet-war¥WEB-INF¥classes¥compass ディレクトリには、次の .java および .class ファイルがあります。

ファイル	説明
Reservation (エンコードされた WSDL 用の ServicePortName.java)	このサービスが提供する Java インターフェイスです。このインターフェイスには、Web サービスオペレーションのメソッド署名が含まれています。
ReservationService (ServiceName.java)	ServiceNameLocator.java クラスが実装しているファクトリインターフェイスです。 インターフェイス名には、WSDL ドキュメントのサービス名が付けられます。

ファイル	説明
ReservationServiceLocator (ServiceNameLocator.java)	プロキシのインスタンスを取得するファクトリ。ServiceNameService.java インターフェイスを実装します。WSDL ファイルのサービス名から生成されます。 クラス名は、WSDL ドキュメントのサービス名に接尾辞 Locator を付けて作成されます。WSDL ファイルに複数のサービスが記述されている場合は、サービスごとに 1 つのクラスが生成されます。このクラス内の `get` メソッドを使用して、Web サービスインターフェイスを実装するスタブオブジェクトを取得します。
ReservationSoapBindingStub (ServiceBindingNameStub.java)	Web サービスプロキシクラスです。Axis のクライアント API を使用して Web サービスを呼び出す ServicePortName.java インターフェイスを実装します。 クラス名は、サービスバインディング名に接尾辞 Stub を付けて作成されます。
Trip (エンコードされた WSDL 用の ServicePortName.java)	このサービスが提供する Java インターフェイスです。このインターフェイスには、Web サービスオペレーションのメソッド署名が含まれています。
TripInfo	返された Trip サービスの `getList` および `getInfo` メソッドを処理するために生成される JavaBean です。
TripService (ServiceName.java)	ServiceNameLocator.java クラスが実装しているファクトリインターフェイスです。 インターフェイス名には、WSDL ドキュメントのサービス名が付けられます。
TripServiceLocator (ServiceNameLocator.java)	プロキシのインスタンスを取得するファクトリ。ServiceNameService.java インターフェイスを実装します。WSDL ファイルのサービス名から生成されます。 クラス名は、WSDL ドキュメントのサービス名に接尾辞 Locator を付けて作成されます。WSDL ファイルに複数のサービスが記述されている場合は、サービスごとに 1 つのクラスが生成されます。このクラス内の `get` メソッドを使用して、Web サービスインターフェイスを実装するスタブオブジェクトを取得します。
TripSoapBindingStub (ServiceBindingNameStub.java)	Web サービスプロキシクラスです。Axis のクライアント API を使用して Web サービスを呼び出す ServicePortName.java インターフェイスを実装します。 クラス名は、サービスバインディング名に接尾辞 Stub を付けて作成されます。

JSP ベースのプロキシクライアントの作成および使用

プロキシコードを作成したら、Web サービスプロキシオブジェクトをインスタンス化してそのメソッドを呼び出すことによって、ターゲット Web サービスオペレーションを呼び出すプロキシクライアントを作成する必要があります。

JSP ベースのプロキシクライアントを作成するには次の手順を実行します。

1 プロキシファクトリをインスタンス化します。

2 プロキシをインスタンス化します。

3 Web サービスメソッドを呼び出します。

JSP ベースのプロキシクライアントを作成するには

1 ファイル <JRun のルートディレクトリ >¥servers¥tutorial¥travelnet-ear¥travelnet-war¥**home.jsp** を開きます。ファイルの所定の場所に、次のコードを追加します。

```
// 1. プロキシファクトリをインスタンス化します。
TripService factory = new TripServiceLocator();
// 2. プロキシをインスタンス化します。
Trip trip = factory.getTrip();
// 3. Web サービスの getList() メソッドを呼び出します。
TripInfo[] trips = trip.getList();

int length = trips.length;
for (int i=0; i<length ; i++) {
%>
<tr>
  <td class="contentBG"><a href="tripdetail.jsp?tripId=
   <%= trips[i].getId() %>"><%= trips[i].getName() %></a></td>
  <td class="contentBG"><%= trips[i].getTeaser() %></td>
  <td class="contentBG"><%= trips[i].getPrice() %></td>
</tr>
<%
   }
```

home.jsp には、Compass で提供される Trip Web サービスの `getList` メソッドを呼び出すことによって、旅行のリストが表示されます。

2 ファイル <JRun のルートディレクトリ >¥servers¥tutorial¥travelnet-ear¥travelnet-war¥**tripdetail.jsp** を開きます。ファイルの所定の場所に、次のコードを追加します。

```
int tripId=Integer.parseInt(request.getParameter("tripId"));
// 1. プロキシファクトリをインスタンス化します。
TripService factory = new TripServiceLocator();
// 2. プロキシをインスタンス化します。
Trip trip = factory.getTrip();
// 3. Web サービスの getInfo() メソッドを呼び出します。
 TripInfo info = trip.getInfo(tripId);
```

Tripdetail.jsp には、Compass で提供される Trip Web サービスの `getInfo` メソッドを呼び出すことによって、home.jsp で選択された旅行の詳細な情報が表示されます。

3 ファイル <JRun のルートディレクトリ >¥servers¥tutorial¥travelnet-ear¥travelnet-war¥**reservationaction.jsp** を開きます。ファイルの所定の場所に、次のコードを追加します。

```
// 1. プロキシファクトリをインスタンス化します。
ReservationService factory = new ReservationServiceLocator();
// 2. プロキシをインスタンス化します。
Reservation stub = factory.getReservation();
// 3. Web サービスの reserve() メソッドを呼び出します。
 int id = stub.reserve(tripId, firstName, lastName, ccType,
    ccNumber, ccExpiration);
```

Reservationaction.jsp では、Compass で提供される Reservation Web サービスの `reserve` メソッドを呼び出すことによって、旅行が予約されたことを Compass に通知します。

最後の手順では、Web サービスを使用して Compass Travel と通信する TravelNet で旅行を予約します。

旅行を予約するには

1 Web ブラウザで次の URL を開きます。

http://localhost:8101/travelnet/home.jsp

メモ：samples JRun サーバーを実行しておく必要があります。

旅行情報が含まれている TravelNet ホームページが表示されます。

2 旅行を選んで予約します。

- 名前を入力します。
- 姓を入力します。
- クレジットカードを選択します。
- [番号] フィールドに番号を入力します。
- [有効期限] フィールドに MM/YY (月 / 年) を入力します。

予約ができると Reservation.jsp によって確認 ID が表示され、問題が発生した場合はエラーメッセージが表示されます。

要約

このチュートリアルレッスンでは、tutorial JRun サーバーの TravelNet という Web サービスクライアントアプリケーションを使用して、samples サーバー上の Compass Travel アプリケーションの Web サービスを使用しました。

このチュートリアルでは、次の実行方法について学習しました。

- WSDD ファイルでの Web サービス作成
- Web サービスの WSDL ファイル生成
- WSDL ファイルからの Java プロキシコード生成
- JSP ベースのプロキシクライアント作成による Web サービスメソッドの呼び出し

Web サービスを使用した演習については、http://localhost:8200 の samples サーバー上にある JRun Web サービスのサンプルをご覧ください。

Web サービスの詳細については、『JRun プログラマーガイド』を参照してください。

索引

A
admin JRun サーバー　26

B
bean 管理パーシスタンス (BMP)、「BMP」を参照
BMP
　定義　60

C
CMP
　定義　60
config オブジェクト　51
　「ServletConfig オブジェクト」も参照

D
default JRun サーバー　26

E
EAR ファイル　66
EJB
　JRun アーキテクチャ　62
　XDoclet、併用　31
　インターフェイス　57, 106
　エンタープライズデプロイウィザード、併用　31
　エンティティ bean　60
　概要　56
　構成　57
　仕様　37
　セッション bean　59
　デプロイモデル　62
　呼び出し　107
EJB コンテナ　56
EJB のチュートリアル　105
Enterprise ARchive (EAR) ファイル、「EAR ファイル」を参照
Enterprise JavaBeans、「EJB」を参照

H
HttpServletRequest オブジェクト　48
HttpServletResponse オブジェクト　48
HttpSession オブジェクト、「セッションオブジェクト」を参照
HTTP リクエスト / レスポンス
　アクセス　47
　サポート　44

I
IDE　15
invoker サーブレット
　Web アプリケーション　75

J
J2EE
　API　37
　コンテナ　35
　コンポーネント　35
　準拠　6
　新機能　6
　通信テクノロジー　36
　プラットフォーム　34
　利点　5
J2EE Connector API (JCA)、「JCA」を参照
J2EE アプリケーション
　3 階層モデル　18
　JRun によるサポート　29
J2EE プラットフォーム
　階層　34
　テクノロジー　35
JAAS　40
Java
　サーバーサイド　4
　利点　45
Java 2 Platform Enterprise Edition (J2EE)、「J2EE」を参照
Java Authentication and Authorization Service 、「JAAS」を参照
Java Database Connectivity、「JDBC」を参照
Java Management Extensions (JMX)、「JMX」を参照
Java Message Service、「JMS」を参照
Java Naming and Directory Interface、「JNDI」を参照
JavaBeans
　アクセス　101
　インスタンス作成　101
JavaServer Pages、「JSP」を参照
Java サーブレット、「サーブレット」を参照
Java 統合開発環境、「IDE」を参照
JDBC
　JMC、データソース、追加　89
　仕様　38
JMC
　JDBC データソース、追加　89
　JRun サーバー、追加　85
　起動　83
JMS　38
JMX
　JRun アーキテクチャ　8
JNDI　40
JRun
　Web サーバー　28
　Web サーバーのサポート　4
　アーキテクチャ　8
　カーネル　8
　概要　4
　起動と停止　24
　新機能　9
　プログラミングモデル　21
　ユーザータイプ　16
　ランチャー　24
JRun Web サーバー (JWS)
　使用　28
　定義　21
jrun.exe (Windows)　24

JRun 管理コンソール (JMC)、「JMC」を参照
jrun コマンド　24
JRun サーバー
　admin　26
　default　26
　samples　27
　Web アプリケーション　71
　Web サーバー　21
　概要　25
　起動と停止　24
　クラスタリング　67
　使用　23
　追加　85
　複数の Web アプリケーション　67
　ランチャーでの制御　24
jrun 実行可能ファイル (UNIX)　24
JRun の起動　24
JRun ランチャー　24
JSP
　JSP アクション　101
　Web アプリケーション、追加　74
　スクリプト要素　98
　ディレクティブ　98
JSP のチュートリアル　97

M
Macromedia
　日本オフィス　xiii
　販売 (米国)　xiii
MDB
　概要　61

N
NT サービス
　アプリケーション、相違　23

P
PrintWriter インターフェイス　48

S
samples JRun サーバー　27
ServletConfig オブジェクト　51
　「アプリケーションオブジェクト」も参照
ServletContext オブジェクト　51
　「アプリケーションオブジェクト」も参照
ServletOutputStream インターフェイス　48
session オブジェクト　50
　「HttpSession オブジェクト」も参照

T
tutorial サーバー　87

U
URL パターン　75

W
WAR ファイル　67
　Web アプリケーション　30, 67
Web ARchive ファイル、「WAR ファイル」を参照
Web Services Deployment Descriptor、「WSDD」を参照
Web Services Description Language ファイル、「WSDL」を参照
web.xml ファイル　69
　Web アプリケーション　30
Web アプリケーション
　EJB、追加　76
　HTML ファイル、追加　74
　invoker サーブレット　75
　JRun サーバー　67
　JSP、追加　74
　WAR ファイル　30
　web.xml ファイル　30, 69
　WEB-INF ディレクトリ　68
　アプリケーションマッピング　71
　概要　30
　クラスタ、全体へデプロイ　77
　コンポーネント、追加　73
　サーブレット　74
　作成　73
　使用　67
　タグライブラリ、追加　75
　定義　22
　ディレクトリ構造　68
　ディレクトリ、追加　73
　デプロイメントディスクリプタ　30, 69
　パッケージ化　78
　利点　66
　ルートディレクトリ　68
Web サーバー
　JRun サーバー　21
　JRun への接続　28
　概要　28
　定義　21
Web サーバー設定ツール
　概要　10
Web サービス
　WSDD　122
　WSDL2Java　126
　WSDL ファイル、生成　123
　概要　14
　クライアントアプリケーション　114
　作成　122
　プロキシクライアント、生成　126
　ユーザー　114
Web サービスのチュートリアル　113
WSDD　122
WSDL ファイル　123

X
XDoclet　12
　EJB、併用　31, 63

あ
アプリケーションオブジェクト　51
　「ServletContext オブジェクト」も参照
アプリケーションマッピング　71

い
イベントハンドラ、「イベントリスナ」を参照
イベントリスナ
　使用　49

え
エンタープライズ bean　37, 56
エンタープライズアプリケーション
　JRun によるサポート　20, 31
　設計　18
　定義　22
エンタープライズデプロイウィザード
　EJB、併用　31, 63
　起動　11
エンティティ bean　60
　パーシスタンス　60

お
オートデプロイ
　概要　9
　チュートリアルアプリケーション　88

か
開発ツール
　Dreamweaver MX　15
　IDE　15

く
クライアントクラスパス　62
クラスタリング
　JRun サーバー　67
　サーバーオブジェクト　10

こ
コネクタ　28
コマンドライン
　JRun の起動と停止　24
コンテキストオブジェクト　51
コンテキストパス　72

コンテキストルート　71
コンテナ管理パーシスタンス (CMP)、「CMP」を参照

さ
サーブレット
　ServletInvoker　75
　Web アプリケーション、定義　75
　Web アプリケーション、への追加　74
　概要　44
　作成　46, 93
　呼び出し　44
　リクエスト / レスポンスプロセス　44
　利点　45
サーブレットのチュートリアル　81
サーブレットフィルタ、使用　48
サーブレットマッピング　71
サンプルアプリケーション　12

す
スタブレスデプロイ　62

せ
セキュリティ
　明示的サーブレットマッピング　75
セッション bean　59
　使用　108
　ステートフル　59
　ステートレス　59

ち
チュートリアル　79
　Compass Travel アプリケーション　93
　EJB のチュートリアル　105
　EJB、ソースコード　109
　J2EE アプリケーション、開発　81
　JavaBeans　101
　JRun サーバー、追加　85
　JSP、コーディング　98
　JSP のチュートリアル　97
　TravelNet アプリケーション　114
　Web サービスのチュートリアル　113
　WSDL2Java　126
　アプリケーション、デプロイ　88
　サーブレット、コーディング　93
　サーブレットのチュートリアル　81
　ファイル　82
　プロキシクライアント、生成　126

て
ディレクトリ構造
　Web アプリケーション　68
　エンタープライズアプリケーション　69
データベースアクセス、JDBC　89
デフォルトの Web アプリケーション
　使用　73
デプロイ
　EJB クライアントクラスパス　62
　アーカイブファイル　77
　オープンディレクトリ　77
　クラスタ全体　77
　スタブレス　62
デプロイメントディスクリプタ
　application.xml ファイル　69
　JRun 固有　70
　web.xml ファイル　69
　定義　69

ふ
プロキシクライアント　126
　JSP ベース　128
プログラミングモデル　21

へ
ページコンテキスト情報　50

ほ
ホットデプロイ　9, 62

ま
マッピング
　アプリケーション　71
　servlet　71
　設定ファイル　72

め
メッセージ駆動型 bean、「MDB」を参照

ゆ
ユーザータイプ　16

ら
ランチャー　24

り
リクエスト / レスポンスプロセス　47
リソース
　オンライン　xii
　書籍　ix
リソースアダプタ
　概要　31
リファレンス実装
　オートデプロイ　62

れ
例外、処理　49