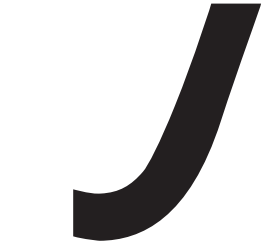

macromedia®
JRUN™4

JRun アセンブルとデプロイガイド

パーツ番号 ZJR40M500J

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、Masaki 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 13F

目 次

このマニュアルの概要

『JRun アセンブルとデプロイガイド』では、JRun で J2EE モジュールを設定、パッケージ、およびデプロイする方法について説明します。このマニュアルは、次の作業担当者を対象としています。

- **開発者**　モジュールや、どのようにモジュールをデプロイするかについて記述するデプロイメントディスクリプタの初期バージョンを作成します。
- **アセンブル担当者**　1 つ以上のモジュールを結合して、デプロイ可能なエンタープライズアプリケーションを作成します。
- **デプロイ担当者**　モジュールのデプロイメントディスクリプタリファレンスを、ターゲット JRun サーバー環境の対応するエンティティにマッピングします。
- **管理者**　モジュールが依存するシステムリソースを設定します。

目 次

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 アプリケーションの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

第 1 章
J2EE モジュールの設定

この章では、JRun でデプロイする J2EE モジュールの設定方法について説明します。

目 次

JRun デプロイメントディスクリプタについて

デプロイメントディスクリプタを使用すると、Java コードを変更せずに、多くの J2EE モジュールを設定できます。**デプロイメントディスクリプタ**とは、モジュールを JRun サーバーにデプロイする方法を記述し、モジュールが依存するリソースをリストする XML ファイルのことです。各タイプの J2EE モジュールにはそれぞれ、そのタイプのモジュールの DTD (Document Type Description) をベースにした標準デプロイメントディスクリプタがあります。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、各タイプのモジュールに固有の要素の他に、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。

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用の JRun 固有のデプロイメントディスクリプタを使用して、JRun 固有の機能を設定できます。たとえば、JRun 固有の 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 jrun-web.xml 内では、ダイナミックサーブレットコンパイルやリロード機能の有効 / 無効を設定できます。JRun 固有のデプロイメントディスクリプタの名前は、対応する標準デプロイメントディスクリプタの前に接頭辞 jrun- が付いている名前で構成されています。

メモ：JRun 3.x のエンタープライズアプリケーション、EJB、および Web アプリケーションは、デプロイメントディスクリプタを変更せずにデプロイできます。

また、RI (Sun Reference Implementation) 用にパッケージした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、およびエンタープライズリソースアダプタは、これらのモジュールが RI 固有のデプロイメントディスクリプタを含んでいてもデプロイできます。RI デプロイツールは、JRun でデプロイする予定のモジュールに使用できます。ただし、Jrun のコンテナ管理による持続性は RI より、より高度な設定をすることができます。

JRun では次のデプロイメントディスクリプタを使用しています。これらの各ファイルについては、この章の後続のセクションで説明します。詳細については、<JRun のルートディレクトリ >/docs/descriptordocs/index.html ファイルにリストされているデプロイメントディスクリプタのドキュメントを参照してください。

モジュールタイプ	デプロイメントディスクリプタ
Web アプリケーション	<Web アプリケーション >/WEB-INF/web.xml、jrun-web.xml **メモ：**JRun では、default-web.xml というデプロイメントディスクリプタも使用します。このディスクリプタは <JRun のルートディレクトリ > / server / <JRun サーバー>/SERVER-INF ディレクトリにあります。このファイルの構造は標準の web.xml ファイルの構造と同じです。このファイルを使用して、コンフィギュレーション設定を JRun サーバー内のすべての Web アプリケーションにグローバルに適用できます。default-web.xml ファイルのエントリは、Web アプリケーションの web.xml ファイルのエントリによって上書きされます。
Enterprise JavaBeans	<EJB のモジュール >/META-INF/ejb-jar.xml、jrun-ejb-jar.xml
エンタープライズリソースアダプタ	< アダプタモジュール >/META-INF/ra.xml、jrun-ra.xml
エンタープライズアプリケーション	< エンタープライズアプリケーション >/META-INF/application.xml

メモ：JRun のインストールには **XDoclet** が含まれています。XDoclet は、bean を実装したソースファイル内の特殊な Javadoc コメントをベースにして、EJB インターフェイス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。XDoclet を使用すると、サーブレットまたはタグライブラリソースファイル内のコメントをベースにして、Web アプリケーションデプロイメントディスクリプタや JSP タグライブラリディスクリプタも生成することができます。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』を参照するか、または XDoclet の Web サイト http://xdoclet.sourceforge.net をご覧ください。

モジュール設定での J2EE のロールについて

JRun サーバーで J2EE モジュールをデプロイするには、まず、ターゲット環境に合わせてモジュールを適切に設定する必要があります。設定時は、開発者、アセンブル担当者、デプロイ担当者、およびシステム管理者のロールを保持していることを前提とし、これらのロールを他のユーザーと共有することもできます。

- モジュール**開発者**は、モジュールおよびデプロイメントディスクリプタの最初のバージョンを作成します。モジュールを開発環境から運用環境のアプリケーションに移行するとき、デプロイメントディスクリプタ内の一部の情報が変更されます。

  たとえば、運用環境では EJB または Web アプリケーションデプロイメントディスクリプタ内のデータソースマッピング、セキュリティロールリファレンス、および EJB の JNDI 名に対して変更が必要になる可能性があります。これらの変更ができるように、開発者はデプロイメントディスクリプタに適切な `resource-ref`、`security-role-ref`、および `ejb-ref` 要素を含める必要があります。

- **アセンブル担当者**は、1 つ以上のモジュールを結合して、デプロイ可能なエンタープライズアプリケーションを作成します。アセンブル担当者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を作成し、アプリケーションが依存するシステムリソースおよびクラスライブラリを識別します。状況によっては、エンタープライズアプリケーションの複数のモジュールが同じクラスライブラリにアクセスする必要があります。

  アセンブル担当者はまた、個々のモジュールのデプロイメントディスクリプタを修正して、アプリケーション全体のリソースリファレンス、セキュリティロール、および EJB リファレンスを提供します。EJB の場合は、アセンブル担当者が EJB デプロイメントディスクリプタ内のメソッドへのアクセス許可やトランザクション属性を設定する場合があります。

- **デプロイ担当者**は、モジュールのデプロイメントディスクリプタを編集することによって、リソース、EJB、およびセキュリティロールリファレンスをターゲット JRun サーバー環境の対応するエントリにマッピングします。CMP bean の場合は、デプロイ担当者が CMP 設定をデータベースの実際のフィールドにマッピングします。
- **システム管理者**は、セキュリティロールおよびユーザー、JDBC データソース、JMS リソース、JavaMail セッションなど、モジュールが依存する JRun システムリソースを設定します。

Web アプリケーションの設定

このセクションでは、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である web.xml と、JRun 固有のデプロイメントディスクリプタである jrun-web.xml について説明します。詳細については、<JRun のルートディレクトリ >/docs/descriptordocs/index.html ファイルにリストされているデプロイメントディスクリプタのドキュメントを参照してください。

デプロイメントディスクリプタの設定の大部分は、Web アプリケーションのサーブレットおよび JSP に直接対応しています。サーブレットおよび JSP のプログラミング方法の詳細については、『JRun プログラマーガイド』および vi ページの「JRun ドキュメントの概要」に記載されている参考文献を参照してください。

web.xml デプロイメントディスクリプタについて

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である web.xml を使用して、次の表のような Web アプリケーションの要素を設定します。有効な web.xml ファイルに子要素は不要なので、ユーザーが設定する要素は、Web アプリケーション内の要素によって異なります。このファイルのルート要素は web-app です。

XML 要素	説明
icon、display-name、description	ツールで使用するイメージ、Web アプリケーション名、および説明
context-param (0 以上)	サーブレットコンテキスト初期化パラメータ
filter (0 以上)	サーブレットフィルタ定義
filter-mapping (0 以上)	サーブレット名または URL パターンへのサーブレットフィルタマッピング
listener (0 以上)	Web アプリケーションのリスナ bean のデプロイプロパティ
servlet (0 以上)	サーブレット定義
servlet-mapping (0 以上)	URL パターンへのサーブレットマッピング
session-config	Web アプリケーションのセッションパラメータ
mime-mapping (0 以上)	ファイル拡張子と MIME タイプとのマッピング
welcome-file-list	ウェルカムファイルの要素が順番に並んだリスト
error-page (0 以上)	エラーコードまたは例外タイプと、Web アプリケーションのリソースへのパスとのマッピング
taglib (0 以上)	JSP タグライブラリの定義
resource-env-ref (0 以上)	JRun サーバー環境で管理されているオブジェクトへの Web アプリケーションのリファレンス
resource-ref (0 以上)	外部リソースへの Web アプリケーションのリファレンス
security-constraint (0 以上)	セキュリティ制限と 1 つ以上の Web リソースコレクションとの関連付け

XML 要素	説明
login-config	フォームログインメカニズムで必要な認証方法、範囲名、および属性の宣言
security-role (0 以上)	セキュリティロールの宣言
env-entry (0 以上)	Web アプリケーションの環境エントリ
ejb-ref (0 以上)	サーブレット、JSP、またはタグライブラリコード内の、EJB のホームへのリファレンス。 アセンブル担当者のロールでは、オプションの ejb-link 要素を使用して、EJB リファレンスが Web アプリケーションと同じエンタープライズアプリケーション内の EJB にリンクされていることを指定できます。ejb-name 値の前に、EJB モジュールへの相対パスとシャープ記号 (#) を付ける必要があります。 EJB の jrun-ejb-jar.xml ファイル内で ejb-ref 要素のデフォルトの JNDI 名が変更された場合や、EJB が Web アプリケーションと同じエンタープライズアプリケーション内にない場合、jrun-web.xml ファイルにはその JNDI 名を含んでいる対応する ejb-local-ref 要素が必要です。
ejb-local-ref (0 以上)	サーブレットまたは JSP コード内の、EJB のホームへのリファレンス。 アセンブル担当者のロールでは、オプションの ejb-link 要素を使用して、EJB ローカルリファレンスが Web アプリケーションと同じエンタープライズアプリケーション内の EJB にリンクされていることを指定できます。EJB が別の EJB モジュール内にある場合は、ejb-name 値の前に、EJB モジュールへの相対パスとシャープ記号 (#) を付ける必要があります。 EJB の jrun-ejb-jar.xml ファイル内で ejb-local-ref 要素のローカルホームのデフォルトの JNDI 名が変更された場合や、EJB が Web アプリケーションと同じエンタープライズアプリケーション内にない場合、jrun-web.xml ファイルにはその JNDI 名を含む、対応している ejb-local-ref 要素が必要です。

例：web.xml デプロイメントディスクリプタ

このセクションの例では、次の web.xml ファイルを示します。

- 空の web.xml ファイル
- サーブレットおよびウェルカムファイルを設定する web.xml ファイル
- セキュリティを設定する web.xml ファイル

また、<JRun のルートディレクトリ >/servers/samples ディレクトリ内の JRun サンプルサーバー上のアプリケーションの web.xml ファイルを表示することもできます。

web.xml：空のファイル

次の空のデプロイメントディスクリプタは、使用可能な最も基本的なものになります。開発の初期段階でこのディスクリプタを使用して、JSP を含みサーブレットを含まない Web アプリケーション (web-app) をデプロイし、アプリケーションが複雑になるにつれてディスクリプタに要素を追加できます。<JRun のルートディレクトリ >/servers/<JRun サーバー>/SERVER-INF ディレクトリの default-web.xml ファイル内で、デフォルトのウェルカムファイルは index.html および index.jsp として設定されています。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2.3.dtd">
<web-app>
</web-app>
```

メモ：デフォルトの JRun サーバー、および JMC を使用して追加された各 JRun サーバーには、単純な web.xml ファイルとともに default-war Web アプリケーションが含まれている default-ear エンタープライズアプリケーションがあります。新規 JSP およびサーブレットを短時間でデプロイしてテストするには、これらの JSP およびサーブレットを default-war Web アプリケーションに入れます。

web.xml：サーブレットおよびウェルカムファイルの設定

次のデプロイメントディスクリプタではサーブレットを設定し、サーブレットを URL パターンにマッピングして、Web アプリケーションのウェルカムファイルを指定します。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
        "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <servlet>
    <servlet-name>AdminServlet</servlet-name>
    <display-name>AdminServlet</display-name>
    <servlet-class>packagename.Administrator</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>AdminServlet</servlet-name>
    <url-pattern>/admin</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

web.xml：セキュリティ設定

次のデプロイメントディスクリプタでは、Web アプリケーションセキュリティの次の要素を設定します。

- `web-resource-collection` は、セキュリティ保護されたリソースを指定します。
- `auth-constraint` は、リソースにアクセスできる JRun セキュリティロールを指定します。
- `login-config` は認証メカニズムを指定します。
- `security-role` は JRun セキュリティロールを参照します。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
        "http://java.sun.com/j2ee/dtds/web-app_2.3.dtd">
<web-app>
<security-constraint>
   <web-resource-collection>
      <web-resource-name>Name1</web-resource-name>
      <url-pattern>/services/AdminService</url-pattern>
      <http-method>GET</http-method>
      <http-method>POST</http-method>
   </web-resource-collection>
   <auth-constraint>
      <role-name>Manager</role-name>
   </auth-constraint>
</security-constraint>
<login-config>
   <auth-method>BASIC</auth-method>
</login-config>
<security-role>
   <role-name>Manager</role-name>
</security-role>
</web-app>
```

jrun-web.xml デプロイメントディスクリプタについて

JRun 固有の 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 jrun-web.xml を使用して、次の表のような Web アプリケーションの要素を設定します。このファイルのルート要素は jrun-web-app です。Web アプリケーションにいずれの要素も必要ない場合、jrun-web.xml ファイルは不要です。

メモ：jrun-web.xml ファイルは自動的に生成できます。詳細については、第 3 章の「JRun 固有のデプロイメントディスクリプタの操作」を参照してください。

XML 要素	説明
context-root	Web アプリケーションのコンテキストルート (論理ルート) の宣言。 Web アプリケーションがエンタープライズアプリケーションの一部である場合は、jrun-web.xml ファイルではなくエンタープライズアプリケーションの application.xml ファイル内で context-root を設定する必要があります。
reload	サーブレット、サーブレットヘルパークラス、および JSP ヘルパークラスが変更されたときに、それらを自動的にリロードするかどうかを指定するブール値設定。 デフォルトは true です。
compile	ソースファイルが変更されたときに、サーブレットおよびサーブレットヘルパークラスを自動的にコンパイルするかどうかを指定します。 デフォルトは true です。

XML 要素	説明
session-config	Web アプリケーションのセッションを設定するための次のセクションが含まれています。 • persistence-config には、セッションのパーシスタンスプロパティを設定するための要素を指定します。 • replication-config には、セッションの複製を設定するための要素を指定します。 • cookie-config には、Cookie を設定する要素を指定します。
load-system-classes-first	エンタープライズアプリケーションクラスおよび Web アプリケーションクラスの前にシステムクラスをロードするかどうかを指定します。デフォルトのクラスローダー委任モデルでは、システムクラスを最初にロードすることになっていますが、サーブレット仕様書では Web アプリケーションクラスを最初にロードするように推奨しています。 デフォルトは true です。
ejb-ref	EJB 開発者が指定した ejb-ref とその JNDI 名とのマッピング。デプロイ担当者が実際の JNDI 名を指定します。 この要素は、EJB クラスを JNDI に 2 回以上バインドする場合のみ必要です。
ejb-local-ref	EJB 開発者が指定したローカルホームの ejb-ref 名とその JNDI 名とのマッピング。デプロイ担当者が実際の JNDI 名を指定します。 この要素は、EJB クラスを JNDI に 2 回以上バインドする場合のみ必要です。
resource-env-ref	Web アプリケーション開発者が指定した resource-env-ref-name とその JNDI 名とのマッピング。デプロイ担当者が JNDI 名を指定します。 この要素は、管理されたオブジェクトを JNDI に 2 回以上バインドする場合のみ必要です。

XML 要素	説明
resource-ref	Web アプリケーション開発者が指定した resource-ref-name とその JNDI 名とのマッピング。デプロイ担当者が実際の JNDI 名を指定します。 この要素は、リソースを JNDI に 2 回以上バインドする場合のみ必要です。
virtual-mapping	リソースパスと物理的な場所とのマッピング。この場所は、必ずしも Web アプリケーションのルートにあるとはかぎりません。リソースパスは常にスラッシュで始める必要があり、最後にワイルドカード (*) を付けることができます。これは、指定されたパスで始まるすべてのリソースパスがシステムパスを使用して解決されることを示します。 /WEB-INF/classes または /WEB-INF/lib を他の場所にマッピングすると、Web アプリケーションのクラスパスにその場所が含められます。 JRun 3.1 で使用した `use-web-server-root` プロパティは JRun 4 には存在しません。`virtual-mapping` プロパティに置き換えられました。`use-web-server-root` 設定を `virtual-mapping` 設定に手動で置き換える必要があります。

例：jrun-web.xml デプロイメントディスクリプタ

次のデプロイメントディスクリプタでは、コンテキストルートを設定し、サーブレットのダイナミックリロードおよびコンパイル機能を有効にし、2 つの仮想パスマッピングを定義します。

```
<?xml version ="1.0"?>

<!DOCTYPE jrun-web-app SYSTEM
      "jrun-web.dtd">

<jrun-web-app>
   <context-root>myapp</context-root>
   <reload>true</reload>
   <compile>true</compile>
   <virtual-mapping>
      <resource-path>/images/*</resource-path>
      <system-path>usr/local/images</system-path>
   </virtual-mapping>
   <virtual-mapping>
      <resource-path>/WEB-INF/classes</resource-path>
      <system-path>d:/app1/library</system-path>
   </virtual-mapping>
</jrun-web-app>
```

Enterprise JavaBeans の設定

このセクションでは、標準 EJB デプロイメントディスクリプタである ejb-jar.xml と、JRun 固有のデプロイメントディスクリプタである jrun-ejb-jar.xml について説明します。詳細については、<JRun のルートディレクトリ >/docs/descriptordocs/index.html ファイルのデプロイメントディスクリプタのドキュメントを参照してください。

EJB のプログラミング方法の詳細については、『JRun プログラマーガイド』および v ページの「このマニュアルの概要」に記載されている参考文献を参照してください。

JRun には、デプロイメントディスクリプタファイルを直接操作せずに EJB の開発、パッケージ、およびデプロイを実行できる次のツールが用意されています。

- **エンタープライズデプロイウィザード** CMP エンティティ bean のコンテナ管理パーシスタンスの設定を含む、開発、アセンブル、およびデプロイ処理全体で利用できます。既存の EJB コードのデプロイメントディスクリプタの作成にも最適です。エンタープライズデプロイウィザードは、スタンドアローンアプリケーションとして使用することも、Borland JBuilder や Forte for Java と併用することも可能です。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。
- **XDoclet** bean を実装したソースファイル内の特殊な Javadoc コメントをベースにして、EJB インターフェイス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。JRun のインストールによって XDoclet が含まれます。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』を参照するか、または XDoclet の Web サイト http://xdoclet.sourceforge.net をご覧ください。

ejb-jar.xml デプロイメントディスクリプタについて

標準 EJB デプロイメントディスクリプタである ejb-jar.xml を使用して、次の表のような EJB モジュールの要素を設定します。有効な ejb-jar.xml ファイルには、有効な enterprise-beans セクションのみが必要です。bean 開発者はこのセクションで 1 つ以上の bean を宣言します。アセンブル担当者は enterprise-beans セクションおよび assembly-descriptor セクションを処理します。このファイルのルート要素は ejb-jar です。

XML 要素	必須 / オプション	説明
description、display-name、small icon、large icon	オプション	ツールで使用する説明、名前、およびイメージ
enterprise-beans	必須	1 つ以上のセッション、エンティティ、またはメッセージによる bean の宣言。bean 開発者が初期値を設定します。 アセンブル担当者は、EJB モジュールをエンタープライズアプリケーションに追加する際に、次の情報を修正または追加できます。 • リソースリファレンス • EJB が宣言するセキュリティロールリファレンスと、アセンブル担当者が定義するセキュリティロールとの role-link を指定できるセキュリティロールリファレンス • EJB モジュール内の特定の bean への ejb-link を指定できる EJB リファレンス
relationships	オプション	コンテナ管理パーシスタンス (CMP) 2.0 を使用したエンティティ bean が関与するリレーションシップの記述
assembly-descriptor	オプション	複数の EJB をアプリケーションユニットに組み立てる方法についてのアセンブル担当者の説明。トランザクション属性、メソッドへのアクセス許可、セキュリティロール、および除外されたメソッドが含まれています。
ejb-client-jar	オプション	クライアントが EJB にアクセスするために必要なクラスが含まれている JAR ファイルの宣言。

enterprise-beans セクションについて

ejb-jar.xml ファイルの enterprise-beans セクションは唯一の必須セクションです。このセクションは、bean 開発者がアプリケーションの一部である bean を宣言する場所です。

共通の要素

次の表では、セッション、エンティティ、およびメッセージによる bean に共通する、ejb-jar/enterprise-beans 要素の下の要素のみを説明します。

XML 要素	必須 / オプション	説明
session、entity、または message-driven (親要素)	必須	EJB のタイプ
description、display-name、small icon、large icon	オプション	ツールで使用する説明、名前、およびイメージ
ejb-name	必須	bean の識別名。jrun-ejb-jar.xml ファイル内の対応する jndi-name 要素に JNDI 名が指定されていない場合、JRun は ejb-name を JNDI 名として使用します。
home および remote、local-home および local、または次の要素を 4 つとも指定する必要があります。		
home		EJB のホームインターフェイスの完全修飾名
local-home		EJB のローカルホームインターフェイスの完全修飾名
remote		EJB のリモートインターフェイスの完全修飾名
local		EJB のローカルインターフェイスの完全修飾名
ejb-class	必須	EJB の実装クラスの完全修飾名
env-entry (0 以上)	オプション	EJB の環境エントリ

XML 要素	必須 / オプション	説明
ejb-ref (0 以上)	オプション	EJB コード内の、他の EJB のホームへのリファレンス。bean 間でのメソッド呼び出しが可能です。 アセンブル担当者のロールでは、オプションの ejb-link 要素を使用して、EJB リファレンスが同じエンタープライズアプリケーション内の EJB にリンクされていることを指定できます。EJB が別の EJB モジュール内にある場合は、ejb-name 値の前に、EJB モジュールへの相対パスとシャープ記号 (#) を付ける必要があります。 他の EJB の jrun-ejb-jar.xml ファイル内で ejb-ref 要素のデフォルトの JNDI 名が変更された場合や、EJB が 同じ EJB モジュールまたはエンタープライズアプリケーション内にない場合、この EJB の jrun-ejb-jar.xml ファイルにはその JNDI 名を含む、対応している ejb-local-ref 要素が必要です。
ejb-local-ref (0 以上)	オプション	EJB コード内の、他の EJB のローカルホームへのリファレンス。bean 間でのメソッド呼び出しが可能です。 アセンブル担当者のロールでは、オプションの ejb-link 要素を使用して、EJB ローカルリファレンスが Web アプリケーションと同じエンタープライズアプリケーション内の EJB にリンクされていることを指定できます。EJB が別の EJB モジュール内にある場合は、ejb-name 値の前に、EJB モジュールへの相対パスとシャープ記号 (#) を付ける必要があります。 他の EJB の jrun-ejb-jar.xml ファイル内で ejb-local-ref 要素のローカルホームのデフォルトの JNDI 名が変更された場合や、EJB が同じ EJB モジュールまたはエンタープライズアプリケーション内にない場合、この EJB の jrun-ejb-jar.xml ファイルにはその JNDI 名を含む、対応している ejb-local-ref 要素が必要です。
security-role-ref (0 以上) (メッセージによる bean は適用外)	オプション	EJB コード内の、セキュリティロールへのリファレンス
security-identity	オプション	EJB のメソッドの実行のために、呼び出し側に関連付けられているセキュリティロールまたは特定の run-as セキュリティロールのどちらを使用するかを指定します。空の use-caller-identity 要素、または説明 (オプション) および role-name 要素が含まれている run-as 要素のいずれかを指定します。

XML 要素	必須 / オプション	説明
resource-ref (0 以上)	オプション	EJB コード内の、外部リソースへのリファレンス
resource-env-ref (0 以上)	オプション	外部リソースに関連付けられている管理されたオブジェクトへの EJB コード内のリファレンス

必須のセッション bean 要素

あらゆるタイプの bean に必須の要素の他に、セッション bean の ejb-jar/enterprise-beans/session 要素の下には次の要素が必要です。

XML 要素	説明
session-type	セッション bean がステートフルセッション bean か、あるいはステートレスセッション bean かを指定します。値は Stateful および Stateless です。
transaction-type	EJB が bean 管理トランザクションを使用するか、あるいはコンテナ管理トランザクションを使用するかを指定します。値は Bean および Container です。

必須のエンティティ bean 要素

あらゆるタイプの bean に必須の要素の他に、エンティティ bean の ejb-jar/enterprise-beans/entity 要素の下には次の要素が必要です。

XML 要素	説明
persistence-type	パーシスタンスがコンテナ管理か、あるいは bean 管理かを指定します。値は Container および Bean です。
prim-key-class	エンティティ bean のプライマリキークラスの完全修飾名。bean 開発者がデプロイ時までプライマリキークラスの定義を保留する場合は、それを java.lang.Object に設定します。
reentrant	bean がリエントラントかどうかを指定するブール値。リエントラントとは、bean が直接メソッドを呼び出すか、あるいはその bean に対してメソッドを呼び出す他の bean を呼び出すことができるかを意味します。
abstract-schema-name	CMP バージョン 2.x および EJB Query Language (EJB QL) を使用したエンティティ bean のアブストラクトスキーマタイプの名前。

必須のメッセージによる bean 要素

あらゆるタイプの bean に必須の要素の他に、メッセージによる bean の ejb-jar/enterprise-beans/message-driven 要素の下には次の要素が必要です。

XML 要素	説明
transaction-type	EJB が bean 管理トランザクションを使用するか、あるいはコンテナ管理トランザクションを使用するかを指定します。値は Bean および Container です。

assembly-descriptor セクションについて

アセンブル担当者は ejb-jar.xml ファイルの assembly-descriptor セクションで、セキュリティロール、メソッドへのアクセス許可、コンテナ管理トランザクションを使用する EJB のトランザクション属性、およびデプロイから除外するメソッドを定義します。最上位レベルのアセンブルディスクリプタの要素はすべてオプションです。これらの要素を次の表に示します。

XML 要素	説明
security-role (0 以上)	セキュリティロールの宣言。JRun 環境で定義されたセキュリティロールと一致する必要があります。詳細については、『JRun 管理者ガイド』を参照してください。
method-permission (0 以上)	1 つ以上のセキュリティロールが 1 つ以上のエンタープライズ bean メソッドを起動できるようにする指定
container-transaction (0 以上)	EJB のメソッド呼び出しのトランザクションスコープをコンテナが管理する方法を指定します。 trans-attribute 要素に次のいずれかのトランザクション属性を指定します。 • NotSupported • Supports • Required • RequiresNew • Mandatory • Never
exclude-list	呼び出し不可として指定されたメソッドのセット。ここで指定されたメソッドを呼び出すことはできません。

例：ejb-jar.xml デプロイメントディスクリプタ

次のデプロイメントディスクリプタでは、ステートレスセッション bean、BMP エンティティ bean、CMP 1.1 エンティティ bean、および CMP 2.0 エンティティ bean の設定情報を指定します。また、<JRun のルートディレクトリ >/servers/samples ディレクトリ内の JRun サンプルサーバー上のアプリケーションの ejb-jar.xml ファイルを表示することもできます。

ejb-jar.xml：ステートレスセッション bean の宣言とアセンブルディスクリプタ

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//
         DTD EnterpriseJavaBeans 2.0//EN" "http://java.sun.com/dtd/
         ejb-jar_2_0.dtd">
<ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>CreditCard</ejb-name>
      <home>compass.CreditCardHomeRemote</home>
      <remote>compass.CreditCardRemote</remote>
      <ejb-class>compass.CreditCardBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
      <security-identity><use-caller-identity/></security-identity>
    </session>
  </enterprise-beans>
  <assembly-descriptor>
    <security-role>
      <role-name>everyone</role-name>
    </security-role>
    <method-permission>
      <role-name>everyone</role-name>
      <method>
        <ejb-name>CreditCard</ejb-name>
        <method-name>*</method-name>
      </method>
    </method-permission>
    <container-transaction>
      <method>
        <ejb-name>CreditCard</ejb-name>
        <method-name>*</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
    </container-transaction>
  </assembly-descriptor>
</ejb-jar>
```

ejb-jar.xml：BMP エンティティ bean の宣言とアセンブルディスクリプタ

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//
         DTD EnterpriseJavaBeans 2.0//EN" "http://java.sun.com/dtd/
         ejb-jar_2_0.dtd">
<ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>Order</ejb-name>
      <home>compass.OrderHomeRemote</home>
      <remote>compass.OrderRemote</remote>
      <ejb-class>compass.OrderBean</ejb-class>
      <persistence-type>Bean</persistence-type>
      <prim-key-class>java.lang.Integer</prim-key-class>
      <reentrant>False</reentrant>
      <security-identity><use-caller-identity/></security-identity>
    </entity>
  </enterprise-beans>
  <assembly-descriptor>
    <security-role>
      <role-name>everyone</role-name>
    </security-role>
    <method-permission>
      <role-name>everyone</role-name>
      <method>
        <ejb-name>Order</ejb-name>
        <method-name>*</method-name>
      </method>
    </method-permission>
    <container-transaction>
      <method>
        <ejb-name>Order</ejb-name>
        <method-name>*</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
    </container-transaction>
  </assembly-descriptor>
</ejb-jar>
```

ejb-jar.xml：CMP 1.1 エンティティ bean の宣言

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
         2.0//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_2_0.dtd">
<ejb-jar>

   <enterprise-beans>
      <entity>
         <ejb-name>Employee</ejb-name>
         <ejb-class>samples.cmp11.EmployeeBean</ejb-class>
         <home>samples.cmp11.EmployeeHome</home>
         <remote>samples.cmp11.Employee</remote>
```

```
            <persistence-type>Container</persistence-type>
            <primkey-field>employeeId</primkey-field>
            <prim-key-class>java.lang.String</prim-key-class>
            <reentrant>True</reentrant>
            <cmp-version>1.x</cmp-version>
            <cmp-field>
               <field-name>employeeId</field-name>
            </cmp-field>
            <cmp-field>
               <field-name>firstName</field-name>
            </cmp-field>
            <cmp-field>
               <field-name>lastName</field-name>
            </cmp-field>
            <cmp-field>
               <field-name>phone</field-name>
            </cmp-field>
         </entity>
      </enterprise-beans>
   </ejb-jar>
```

ejb-jar.xml：CMP 2.0 エンティティ bean の宣言

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
         2.0//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_2_0.dtd">
<ejb-jar>
   <enterprise-beans>
      <entity>
         <ejb-name>samples/cmp20/Employee</ejb-name>
         <ejb-class>samples.cmp20.EmployeeBean</ejb-class>
         <home>samples.cmp20.EmployeeHome</home>
         <remote>samples.cmp20.Employee</remote>
         <persistence-type>Container</persistence-type>
         <prim-key-class>java.lang.String</prim-key-class>
         <reentrant>True</reentrant>
         <primkey-field>employeeId</primkey-field>
         <abstract-schema-name>employeeschema</abstract-schema-name>
         <cmp-version>2.x</cmp-version>
         <cmp-field>
            <field-name>employeeId</field-name>
         </cmp-field>
         <cmp-field>
            <field-name>firstName</field-name>
         </cmp-field>
         <cmp-field>
            <field-name>lastName</field-name>
         </cmp-field>
         <cmp-field>
            <field-name>phone</field-name>
         </cmp-field>
         <query>
            <query-method>
```

```
            <method-name />
            <method-params />
         </query-method>
         <return-type-mapping>Local</return-type-mapping>
         <ejb-ql>SELECT OBJECT(o) FROM employeeschema AS o</ejb-ql>
      </query>
      <query>
         <query-method>
            <method-name />
            <method-params />
         </query-method>
         <return-type-mapping>Local</return-type-mapping>
         <ejb-ql>SELECT OBJECT(o) FROM employeeschema AS o WHERE o.lastName =
            ?1</ejb-ql>
      </query>
   </entity>
</enterprise-beans>
</ejb-jar>
```

jrun-ejb-jar.xml デプロイメントディスクリプタについて

JRun 固有の EJB デプロイメントディスクリプタである jrun-ejb-jar.xml は必ずしも必要ではありません。このディスクリプタを使用して、JNDI 名、リソースマッピング、JDBC マッピングなど、JRun 固有の要素で ejb-jar.xml ファイルに含まれていない要素を設定します。JDBC マッピングは、CMP 1.1 エンティティ bean の作成、ロード、保管、検索、および削除に使用する SQL を指定します。

メモ：jrun-ejb-jar.xml ファイルは自動的に生成できます。詳細については、第 3 章の「JRun 固有のデプロイメントディスクリプタの操作」を参照してください。

jrun-ejb-jar.xml ファイルには、jrun-ejb-jar の下に次の最上位レベルの要素が含まれています。

XML 要素	説明
description	EJB モジュールの説明
source	bean のデプロイや SQL ステートメントの実行に使用するデータソースの JNDI 名
enterprise-beans	ejb-jar.xml ファイルの enterprise-beans セクションに対応する要素。このセクションを使用して、JNDI マッピング、リソースマッピング、EJB クラスタリング、ステートフルセッション bean のタイムアウト値、およびステートレスセッション bean のインスタンスプールサイズを設定します。

enterprise-beans セクションについて

このセクションでは、エンタープライズ bean のサブ要素について説明します。

共通の要素

次の表では、セッション、エンティティ、およびメッセージによる bean に共通する、jrun-ejb-jar/enterprise-beans 要素の下の要素を説明します。ejb-name および jndi-name 要素のみが必須です。

XML 要素	説明
session、entity、または message-driven (親要素)	EJB のタイプ
ejb-name	EJB の識別名
jndi-name	EJB の JNDI 名。この要素に JNDI 名を指定しないと、JRun は ejb-name を JNDI 名として使用します。
tx-domain-name	EJB トランザクションが実行されるトランザクションドメインの名前
ejb-ref	bean 開発者が指定した ejb-ref-name とその JNDI 名とのマッピング。デプロイ担当者が JNDI 名を指定します。 この要素は、JNDI に EJB クラスを 2 回以上バインドする場合のみ必要です。
ejb-local-ref	EJB 開発者によって指定されたローカルホームの ejb-ref 名とその JNDI 名とのマッピング。デプロイ担当者が実際の JNDI 名を指定します。 この要素は、JNDI に EJB クラスを 2 回以上バインドする場合のみ必要です。
resource-env-ref	bean 開発者が指定した resource-env-ref-name とその JNDI 名とのマッピング。デプロイ担当者が JNDI 名を指定します。 この要素は、管理されたオブジェクトを JNDI に 2 回以上バインドする場合のみ必要です。
resource-ref	bean 開発者が指定した resource-ref-name とその JNDI 名とのマッピング。デプロイ担当者が JNDI 名を指定します。 この要素は、JNDI にリソースを 2 回以上バインドする場合のみ必要です。

XML 要素	説明
cluster-home	この bean の EJB ホームをクラスタリングするかどうかを指定します。<JRun のルートディレクトリ >/servers/<JRun サーバー >/SERVER-INF/jrun.xml ファイルでクラスタリングが有効になっている場合、この値はデフォルトで true になります。この要素を使用して bean ごとにこの動作を無効にできます。
cluster-object	この bean の EJB オブジェクトをクラスタリングするかどうかを指定します。<JRun のルートディレクトリ >/servers/<JRun サーバー >/SERVER-INF/jrun.xml ファイルでクラスタリングが有効になっている場合、この値はデフォルトで true になります。この要素を使用して bean ごとにこの動作を無効にできます。

セッション bean 要素

あらゆるタイプの bean に必須の要素の他に、セッション bean 設定の jrun-ejb-jar/enterprise-beans/session 要素の下には次のオプションの要素を指定できます。

XML 要素	説明
timeout	ステートフルセッション bean のタイムアウト値 (秒単位)。EJB インスタンスは、この時間だけアイドル状態が続くとパッシベート (メモリを節約するためにその EJB オブジェクトから分離) されます。
instance-pool	ステートレスセッション bean のインスタンスプールの最大サイズおよび最小サイズパラメータ

エンティティ bean 要素

あらゆるタイプの bean に必須の要素の他に、エンティティ bean 設定の jrun-ejb-jar/enterprise-beans/entity 要素の下には次のオプションの要素を指定できます。

XML 要素	説明
commit-option	EJB 2.0 仕様に準拠したコミットオプション。有効な値は A、B、および C です。
always-dirty	エンティティ bean のフィールドが変更されていない場合でも、トランザクションの終了時にデータソースを強制的に同期させる空の XML 要素
jdbc-mappings	CMP 1.1 エンティティ bean の CMP マッピングに関する、ejb-jar.xml ファイルで宣言されていない JRun 固有の情報。含まれている jdbc-mapping 要素は、エンティティ bean の作成、ロード、保管、検索、および削除に使用する SQL を指定します。

メッセージによる bean 要素

あらゆるタイプの bean に必須の要素の他に、メッセージによる bean 設定の jrun-ejb-jar/enterprise-beans/entity 要素の下には次のオプションの要素を指定できます。

XML 要素	説明
message-driven-subscription	永続サブスクリプションのためにメッセージによるコンテナが使用するクライアント ID
message-driven-destination	メッセージによるコンテナが使用する送信先

例：jrun-ejb-jar.xml デプロイメントディスクリプタ

次のデプロイメントディスクリプタでは、ステートフルセッション bean、ステートレスセッション bean、および CMP 1.1 エンティティ bean の設定情報を指定します。また、<JRun のルートディレクトリ >/servers/samples ディレクトリ内の JRun サンプルサーバー上のアプリケーションの jrun-ejb-jar.xml ファイルを表示することもできます。

jrun-ejb-jar.xml：ステートフルセッション bean の宣言

ステートフルセッション bean の JNDI 名を設定し、そのホームおよびオブジェクトクラスタリングを無効にして、そのタイムアウト値を設定します。

```
<?xml version="1.0"?>
<!DOCTYPE jrun-ejb-jar PUBLIC '-//Macromedia, Inc.//DTD jrun-ejb-jar 4.0//EN'
         'http://jrun.macromedia.com/dtds/jrun-ejb-jar.dtd'>
<jrun-ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>StatefulEJB</ejb-name>
      <jndi-name>StatefulEJB</jndi-name>
      <cluster-home>false</cluster-home>
      <cluster-object>false</cluster-object>
      <timeout>900</timeout>
    </session>
  </enterprise-beans>
</jrun-ejb-jar>
```

jrun-ejb-jar.xml：ステートレスセッション bean の宣言

この例では、ステートレスセッション bean の JNDI 名およびインスタンスプールサイズの最大値と最小値を設定します。16 ページの「ejb-jar.xml：ステートレスセッション bean の宣言とアセンブルディスクリプタ」も参照してください。

```
<?xml version="1.0"?>
<!DOCTYPE jrun-ejb-jar PUBLIC '-//Macromedia, Inc.//DTD jrun-ejb-jar 4.0//EN'
         'http://jrun.macromedia.com/dtds/jrun-ejb-jar.dtd'>
<jrun-ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>CreditCard</ejb-name>
      <jndi-name>ejb/CreditCard</jndi-name>
      <cluster-home>False</cluster-home>
      <cluster-object>False</cluster-object>
      <instance-pool>
```

```
        <minimum-size>1</minimum-size>
        <maximum-size>5</maximum-size>
      </instance-pool>
    </session>
  </enterprise-beans>
</jrun-ejb-jar>
```

jrun-ejb-jar.xml：CMP 1.1 エンティティ bean の宣言

CMP 1.1 エンティティ bean の JNDI 名および JDBC マッピングを設定します。17 ページの「ejb-jar.xml：CMP 1.1 エンティティ bean の宣言」も参照してください。

```
<?xml version="1.0"?>
<!DOCTYPE jrun-ejb-jar PUBLIC '-//Macromedia, Inc.//DTD jrun-ejb-jar 4.0//EN'
         'http://jrun.macromedia.com/dtds/jrun-ejb-jar.dtd'>
<jrun-ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>Employee</ejb-name>
      <jndi-name>Employee</jndi-name>
      <jdbc-mappings>
        <jdbc-mapping>
          <name>create</name>
          <statement>
            <action>INSERT INTO employees (employee_id , first_name , last_name,
               phone) VALUES ( ?, ? , ? , ?)</action>
            <source>samples</source>
            <params>
              <param>
                 <name>employeeId</name>
                 <type>VARCHAR</type>
              </param>
              <param>
                 <name>firstName</name>
                 <type>VARCHAR</type>
              </param>
              <param>
                 <name>lastName</name>
                 <type>VARCHAR</type>
              </param>
              <param>
                 <name>phone</name>
                 <type>VARCHAR</type>
              </param>
            </params>
          </statement>
        </jdbc-mapping>
        <jdbc-mapping>
          <name>load</name>
          <statement>
            <action>SELECT employee_id, first_name, last_name, phone FROM
          employees
                WHERE employee_id=?</action>
```

```
      <source>samples</source>
      <params>
        <param>
          <name>employeeId</name>
          <type>VARCHAR</type>
        </param>
      </params>
      <fields>
        <field>employeeId</field>
        <field>firstName</field>
        <field>lastName</field>
        <field>phone</field>
      </fields>
    </statement>
  </jdbc-mapping>
  <jdbc-mapping>
    <name>remove</name>
    <statement>
      <action>DELETE FROM employees WHERE employee_id=?</action>
      <source>samples</source>
      <params>
        <param>
          <name>employeeId</name>
          <type>VARCHAR</type>
        </param>
      </params>
    </statement>
  </jdbc-mapping>
  <jdbc-mapping>
    <name>store</name>
    <statement>
      <action>UPDATE employees SET first_name=?, last_name=?, phone=?WHERE
          employee_id=?</action>
      <source>samples</source>
      <params>
        <param>
          <name>firstName</name>
          <type>VARCHAR</type>
        </param>
        <param>
          <name>lastName</name>
          <type>VARCHAR</type>
        </param>
        <param>
          <name>phone</name>
          <type>VARCHAR</type>
        </param>
        <param>
          <name>employeeId</name>
          <type>VARCHAR</type>
        </param>
      </params>
```

```
      </statement>
    </jdbc-mapping>
    <jdbc-mapping>
      <name>findAll</name>
      <statement>
        <action>SELECT employee_id FROM employees</action>
        <source>samples</source>
        <fields>
          <field>employeeId</field>
        </fields>
      </statement>
    </jdbc-mapping>
    <jdbc-mapping>
      <name>findByLastName</name>
      <statement>
       <action>SELECT employee_id FROM employees WHERE last_name=?1</action>
        <source>samples</source>
        <params>
          <param>
             <name>lastName</name>
             <type>VARCHAR</type>
          </param>
        </params>
        <fields>
          <field>employeeId</field>
        </fields>
      </statement>
    </jdbc-mapping>
    <jdbc-mapping>
      <name>findByPrimaryKey</name>
      <statement>
        <action>SELECT employee_id FROM employees WHERE employee_id=?</
      action>
        <source>samples</source>
        <params>
          <param>
             <name>employeeId</name>
             <type>VARCHAR</type>
          </param>
        </params>
        <fields>
          <field>employeeId</field>
        </fields>
      </statement>
    </jdbc-mapping>
  </jdbc-mappings>
 </entity>
</enterprise-beans>
```

エンタープライズリソースアダプタの設定

このセクションでは、標準リソースアダプタデプロイメントディスクリプタである ra.xml と、JRun 固有のデプロイメントディスクリプタである jrun-ra.xml について説明します。ra.xml ファイルに display-name 要素が含まれており、管理された接続ファクトリインスタンスが 1 つだけ必要な場合、jrun-ra.xml ファイルは不要です。詳細については、<JRun のルートディレクトリ >/docs/descriptordocs/index.html ファイルにリストされているデプロイメントディスクリプタのドキュメントを参照してください。

リソースアダプタのプログラミング方法の詳細については、『JRun プログラマーガイド』および vi ページの「JRun ドキュメントの概要」に記載されている参考文献を参照してください。

ra.xml デプロイメントディスクリプタについて

リソースアダプタ開発者は、リソースアダプタの実装コード、設定プロパティ、およびセキュリティ情報に関する情報が含まれている ra.xml ファイルを作成します。デプロイ担当者は ra.xml ファイルの resourceadapter/config-properties セクションを修正して、リソースアダプタの設定可能なプロパティを設定します。また、デプロイ担当者は、ra.xml ファイルで定義されているトランザクション管理レベルに基づいてトランザクション管理を行うように JRun を設定し、ファイルの resourceadapter/authentication-mechanism セクションの情報に基づいてセキュリティを設定します。

ra.xml ファイルには、connector 要素の下に次の最上位レベルの要素が含まれています。

XML 要素	必須 / オプション	説明
display-name、description、icon	オプション	ツールで使用する名前、説明、およびイメージ
vendor-name	必須	名前またはリソースアダプタプロバイダ
spec-version	必須	リソースアダプタによってサポートされるコネクタのアーキテクチャ仕様のバージョン。デプロイ担当者は、仕様のデプロイおよび実行時の必要条件をサポートするようにリソースを設定できます。
eis-type	必須	アダプタを作成するエンタープライズ情報システム (EIS) のタイプ
version	必須	リソースアダプタプロバイダが指定したリソースアダプタのバージョン

XML 要素	必須 / オプション	説明
license	オプション	リソースアダプタをデプロイして使用するためにライセンスが必要かどうかを指定します。また、ライセンス条件をオプションで記述します。
resourceadapter	必須	リソースアダプタに関する次のような情報 • コネクタのアーキテクチャ規定の一部として必要なクラスおよびインターフェイスの完全修飾名 • トランザクションサポートレベル • ManagedConnectionFactory インスタンスの設定可能なレベル • 1 つ以上のサポートされている認証メカニズム • 追加の必要なセキュリティ上のアクセス許可

例：ra.xml デプロイメントディスクリプタ

J2EE Reference Implementation とともに提供されている次の ra.xml ファイルでは、JDBC データベースのリソースアダプタを設定します。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE connector PUBLIC '-//Sun Microsystems, Inc.//DTD Connector 1.0//EN'
         'http://java.sun.com/dtd/connector_1_0.dtd'>
<connector>
   <display-name>BlackBoxNoTx</display-name>
   <vendor-name>Java Software</vendor-name>
   <spec-version>1.0</spec-version>
   <eis-type>JDBC Database</eis-type>
   <version>1.0</version>
   <resourceadapter>
      <managedconnectionfactory-class>
      com.sun.connector.blackbox.NoTxManagedConnectionFactory
      </managedconnectionfactory-class>
      <connectionfactory-interface>javax.sql.DataSource
      </connectionfactory-interface>
      <connectionfactory-impl-class>
      com.sun.connector.blackbox.JdbcDataSource
      </connectionfactory-impl-class>
      <connection-interface>
      java.sql.Connection
      </connection-interface>
      <connection-impl-class>
      com.sun.connector.blackbox.JdbcConnection
      </connection-impl-class>
      <transaction-support>NoTransaction</transaction-support>
      <config-property>
      <config-property-name>ConnectionURL</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
```

```
            <config-property-value>
            jdbc:cloudscape:rmi:CloudscapeDB;create=true
            </config-property-value>
        </config-property>
        <authentication-mechanism>
            <authentication-mechanism-type>
            BasicPassword
            </authentication-mechanism-type>
            <credential-interface>
            javax.resource.security.PasswordCredential
            </credential-interface>
        </authentication-mechanism>
        <reauthentication-support>false</reauthentication-support>
    </resourceadapter>
</connector>
```

jrun-ra.xml デプロイメントディスクリプタについて

リソースアダプタの ra.xml ファイルに **display-name** 値がある場合、JRun では jrun-ra.xml ファイルを使用せずにこのリソースアダプタをデプロイできます。表示名はこのアダプタを参照する EJB などのアプリケーションコンポーネントの JNDI 名になります。jrun-ra.xml ファイルは、設定プロパティの変更、接続プール値の設定、セキュリティ情報の挿入、または複数の ManagedConnectionFactory インスタンスの使用に必要です。

メモ：jrun-ra.xml ファイルは自動的に生成できます。詳細については、第 3 章の「JRun 固有のデプロイメントディスクリプタの操作」を参照してください。

例：jrun-ra.xml デプロイメントディスクリプタ

次の jrun-ra.xml ファイルでは、認証が必要なリソースアダプタインスタンスと、認証が不要なリソースアダプタインスタンスを設定します。

```
<?xml version="1.0" encoding="UTF-8"?>
<jrun-connectionfactory>
   <managedconnectionfactory-class>
   com.sun.connector.blackbox.NoTxManagedConnectionFactory
   </managedconnectionfactory-class>
     <managedconnectionfactory-instance>
          <jndi-name>J2EEConnector/secure-blackbox-notx</jndi-name>
          <config-property>
               <config-property-name>ConnectionURL</config-property-name>
               <config-property-type>java.lang.String</config-property-type>
               <config-property-value>
               jdbc:cloudscape:CloudscapeDB;create=true
               </config-property-value>
          </config-property>
          <security-info>
               <username>victor</username>
               <password>victor</password>
          </security-info>
          <connection-pool>
              <name>mypool</name>
              <objectType>java.lang.StringBuffer</objectType>
```

```
            <minimumSize>1</minimumSize>
            <maximumSize>10</maximumSize>
            <connectionTimeout>6</connectionTimeout>
            <userTimeout>12</userTimeout>
            <skimmerFrequency>300</skimmerFrequency>
            <shrinkBy>2</shrinkBy>
            <debugging>true</debugging>
        </connection-pool>
    </managedconnectionfactory-instance>
    <managedconnectionfactory-instance>
        <jndi-name>J2EEConnector/blackbox-notx</jndi-name>
        <config-property>
            <config-property-name>ConnectionURL</config-property-name>
            <config-property-type>java.lang.String</config-property-type>
             <config-property-value>
             jdbc:cloudscape:CloudscapeDB;create=true
             </config-property-value>
        </config-property>
        <connection-pool>
            <name>mypool</name>
            <objectType>java.lang.StringBuffer</objectType>
            <minimumSize>1</minimumSize>
            <maximumSize>10</maximumSize>
            <connectionTimeout>6</connectionTimeout>
            <userTimeout>12</userTimeout>
            <skimmerFrequency>300</skimmerFrequency>
            <shrinkBy>2</shrinkBy>
            <debugging>true</debugging>
        </connection-pool>
    </managedconnectionfactory-instance>
</jrun-connectionfactory>
```

エンタープライズアプリケーションの設定

このセクションでは、標準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 と、J2EE **拡張メカニズム**について説明します。J2EE 拡張メカニズムによって、同じクラスライブラリ上にある個々のモジュールの依存性を調整することができます。

application.xml デプロイメントディスクリプタについて

アプリケーションのアセンブル担当者は application.xml ファイルを使用して、エンタープライズアプリケーションとそれに含まれている J2EE モジュールを宣言します。エンタープライズアプリケーションには、1 つ以上の Web アプリケーション、EJB、またはエンタープライズリソースアダプタが含まれています。また、アプリケーションのアセンブル担当者は application.xml ファイルを使用して、すべてのアプリケーションモジュールに適用するセキュリティロールも指定できます。

application.xml ファイルの alt-dd 要素を使用すると、アセンブル担当者はモジュールアーカイブファイル内にあるデプロイメントディスクリプタではなく、他のデプロイメントディスクリプタを使用できます。個々の WAR、JAR、および RAR ファイル内でデプロイメントディスクリプタを変更する必要はありません。

application.xml ファイルには、application 要素の下に次の最上位レベルの要素が含まれています。

XML 要素	必須 / オプション	説明
icon	オプション	ツールで使用するイメージ
display-name	必須	ツールで使用する短い名前
description	オプション	ツールで使用するエンタープライズアプリケーションの説明

XML 要素	必須 / オプション	説明
module (0 以上)	必須	エンタープライズアプリケーション内の J2EE モジュールのアーカイブファイルまたは展開したディレクトリへのパス。 次の J2EE モジュールがサポートされています。 • connector：エンタープライズリソースアダプタ • ejb：EJB • web：Web アプリケーション Web アプリケーションで EJB やリソースアダプタを使用する場合があるため、JRun は Web アプリケーションの前に EJB やリソースアダプタをデプロイします。複数の EJB モジュールは、デプロイメントディスクリプタに指定されている順にデプロイされます。 オプションの alt-dd 要素は、エンタープライズアプリケーションのルートディレクトリを基準にした別の J2EE モジュールのデプロイメントディスクリプタへのフルパスを指定します。alt-dd が設定されていない場合、JRun は最初のモジュールのデプロイメントディスクリプタを読み込みます。 オプションの web/context-root 要素は Web アプリケーションのコンテキストルート (論理ルート) を指定します。
security-role (0 以上)	オプション	アプリケーション全体のセキュリティロールの宣言

例：application.xml デプロイメントディスクリプタ

次のデプロイメントディスクリプタでは、展開したディレクトリで EJB モジュールを、JAR ファイルで EJB モジュールを、展開したディレクトリで Web アプリケーションモジュールを、WAR ファイルで Web アプリケーションを宣言し、さらにセキュリティロールも宣言します。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE application PUBLIC
   "-//Sun Microsystems, Inc.//DTD J2EE Application 1.3//EN"
   "http://java.sun.com/dtd/application_1_3.dtd">
<application>
  <display-name>samples-ear</display-name>
    <module>
      <ejb>account</ejb>
    </module>
    <module>
      <ejb>shoppingcart.jar</ejb>
    </module>
    <module>
      <web>
```

```
      <web-uri>samples-war</web-uri>
      <context-root>/samples</context-root>
    </web> <web>
      <web-uri>storefront.war</web-uri>
      <context-root>/storefront</context-root>
    </web>
  </module>
  <security-role>
    <description>Administrator role</description>
    <role-name>Admin</role-name>
  </security-role>
</application>
```

J2EE モジュールの依存性の処理

JRun には、アセンブル担当者が同じクラスライブラリ上の個々の J2EE モジュールの依存性を調整できる便利な方法があります。モジュールの META-INF/manifest.mf ファイルの Class-Path 属性に指定されているファイルが自動的にクラスパスに追加されます。Class-Path 属性には、JAR ファイル、ZIP ファイル、ディレクトリなどの標準のクラスパスで使用できる任意のファイルを指定できます。この機能は、アーカイブファイルまたは展開したディレクトリ内のエンタープライズアプリケーション、EJB、Web アプリケーション、およびリソースアダプタに使用できます。

アセンブル担当者はエンタープライズアプリケーション内にクラスライブラリの 1 つのコピーを入れ、そのクラスライブラリが必要な各モジュールの Class-Path 属性にそのクラスパスを指定できます。

manifest.mf ファイルの Class-Path 属性を使用するには

1 既存の manifest.mf ファイルを開くか、テキストエディタで新規の manifest.mf ファイルを作成します。

2 Class-Path 属性を列記した 1 行を追加します。この属性には、クラスパスに追加する各 JAR ファイルの相対 URL をスペースで区切って指定します。URL は、モジュールを含んでいるエンタープライズアプリケーションではなく、そのモジュールを基準にしている必要があります。

次に例を示します。

```
Class-Path:common/util1.jar common/log4j.jar
```

3 manifest.mf ファイルを保存します。

4 モジュールのルートディレクトリの下にある META-INF ディレクトリに manifest.mf ファイルを追加します。アーカイブしたモジュールの場合は、モジュールをアーカイブし直す必要があります。

5 モジュールをリデプロイします。デプロイメントディスクリプタを変更したり、モジュールアーカイブファイルを修正したりすると、ホットデプロイを実行できます。

第 2 章
J2EE モジュールのパッケージ

この章では、JRun でデプロイする J2EE モジュールのパッケージ方法について説明します。

目 次

モジュールのパッケージの概要

JRun では次のタイプの J2EE モジュールをサポートしています。

- Web アプリケーション (WAR ファイルまたはディレクトリ)
- 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 ファイルのみ)
- 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)

これらの J2EE モジュールに関する知識がない場合は、『JRun 入門』を参照してください。

モジュールをデプロイする前に、モジュールのデプロイメントディスクリプタを作成し、その構成要素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。デプロイメントディスクリプタの詳細については、第 1 章、1 ページの「J2EE モジュールの設定」を参照してください。

アーカイブファイルまたは展開したディレクトリの選択

JRun では、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 J2EE モジュールをデプロイできます。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、ディレクトリのままデプロイすると、柔軟性と使いやすさを最大に高めることができます。

J2EE モジュールは、その指定されたディレクトリ構造を維持している必要があります。モジュールのディレクトリ構造には、そのタイプのモジュールに指定されているサブディレクトリに適切な標準デプロイメントディスクリプタが含まれている必要があります。モジュールによっては、ディレクトリ構造に JRun 固有のデプロイメントディスクリプタが含まれている場合もあります。

アセンブル担当者、デプロイ担当者、および管理者のロールについて

個々の J2EE モジュールをエンタープライズアプリケーションにパッケージしたり、開発環境から運用環境に J2EE モジュールを移行したりする前に、アセンブル担当者、デプロイ担当者、および管理者としていくつかのタスクを実行しなければならない場合があります。詳細については、第 1 章の「モジュール設定での J2EE のロールについて」を参照してください。

Web アプリケーションのパッケージ

Web アプリケーションは、次の表のディレクトリ構造を維持している必要があります。単独で、またはエンタープライズアプリケーションの一部として Web アプリケーションをデプロイできます。

ディレクトリ	説明
<Web アプリケーション> (ルートディレクトリまたは WAR ルート)	WEB-INF ディレクトリおよび、ユーザーの Web ブラウザからアクセスされる JSP、HTML ページ、カスケーティングスタイルシート、画像、JavaScript ファイルなどの全てのファイルを格納します。 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 を使用しない任意のサブディレクトリに直接置くことができます。
/WEB-INF	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。デプロイメントディスクリプタの詳細については、第 1 章、1 ページの「J2EE モジュールの設定」を参照してください。
/WEB-INF/classes	サーブレット、その他の Java クラス、および JAR 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。
/WEB-INF/lib	クラスおよび関連リソースが含まれている JAR ファイルを格納します。

このセクションでは、JRun で Web アプリケーションをデプロイするための準備の概要について説明しますが、特定のツールについては説明しません。JRun には **XDoclet** というオープンソースツールが用意されています。このツールは、ファイル名が Servlet または Library で終わるサーブレットおよびタグライブラリソースファイル内に、特殊な Javadoc コメントをベースにした 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 (TLD) を生成します。JRun では、JRun 固有のデプロイメントディスクリプタをサポートするように XDoclet を拡張し、自動コンパイルおよびデプロイをサポートしています。詳細については、『JRun プログラマーガイド』を参照するか、XDoclet の Web サイト (sourceforge.xdoclet.org) をご覧ください。

Web アプリケーションをパッケージするには

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. アプリケーションの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、タグライブラリファイル (TLD) 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おく必要があります。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。JSP タグライブラリディレクティブの `uri` 属性内の TLD ファイルへのパスはスラッシュで始まり、JSP を含んでいる Web アプリケーションのルートを基準にしています。タグライブラリディレクティブは TLD ファイルを直接指すか、あるいは TLD ファイルを含んでいる JAR ファイルを指すことができます。

6 WEB-INF ディレクトリに Web アプリケーションデプロイメントディスクリプタを置きます。Web アプリケーションデプロイメントディスクリプタには次のものがあります。

- WEB-INF ディレクトリの標準 J2EE 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)
- 必要に応じて、WEB-INF ディレクトリの JRun 固有のデプロイメントディスクリプタ (jrun-web.xml)。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。

デプロイメントディスクリプタの詳細については、第 1 章、1 ページの「J2EE モジュールの設定」を参照してください。

7 パフォーマンスとセキュリティ上の理由から、Web アプリケーションを運用環境に移行する前に、JSP ページのコンパイルを無効にして、テキストベースの .jsp ファイルではなく、バイナリの .class ファイルのみを配布することもできます (オプション)。この処理は、次の 2 つの手順で構成されます。

a ダイナミック JSP コンパイルを無効にします。次の「ダイナミック JSP コンパイルの無効化」のセクションを参照してください。

b JSPC コンパイラを使用して JSP をプリコンパイルします。37 ページの「JSP のプリコンパイル」を参照してください。

メモ : JRun 3.x 用にプリコンパイルされている JSP は JRun 4 では使用できません。オリジナルのコンパイル済み JSP を削除し、JRun 4 用にコンパイルし直す必要があります。

8 開発環境で Web アプリケーションをスタンドアローンモジュールとしてデプロイするには、ディレクトリ構造を JRun デプロイディレクトリにコピーします。デフォルトのデプロイディレクトリは JRun サーバーのルートディレクトリ (<JRun のルートディレクトリ >/servers/<JRun サーバー >) です。JMC を使用すれば、デプロイディレクトリを追加または削除できます。JRun では、モジュールが自動的にデプロイディレクトリにデプロイされます。デプロイの詳細については、第 3 章、45 ページの「J2EE モジュールのデプロイ」を参照してください。

9 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 モジュール >.war
```

ここで、<Web モジュール > は Web アプリケーションの名前です。

10 Web アプリケーションがエンタープライズアプリケーションの一部である場合は、43 ページの「エンタープライズアプリケーションのパッケージ」を参照してください。

ダイナミック JSP コンパイルの無効化

パフォーマンスとセキュリティ上の理由から、ダイナミック JSP コンパイルを無効にして、テキストベースの .jsp ファイルではなく、バイナリの .class ファイルを配布できます。

ダイナミック JSP コンパイルを無効にするには

次の例に太字で示してあるように、<JRun のルートディレクトリ >/servers/<JRun サーバー >/SERVER-INF/default-web.xml ファイルで、JSPServlet の **translationDisabled** 初期化パラメータを true に設定します。

```
<servlet>
<servlet-name>JSPServlet</servlet-name>
<servlet-class>jrun.jsp.JSPServlet</servlet-class>
<init-param>
<param-name>translationDisabled</param-name>
<param-value>true</param-value>
</init-param>
</servlet>
```

JRun サーバーを再起動すると、変更内容が有効になります。

JSP のプリコンパイル

JSP をコンパイルする理由はいくつかあります。

- 開発時は、ページの Web リクエストによってコンパイルを開始するよりも、明示的にページをコンパイルしたほうがエラーを簡単に修正できる場合があります。
- セキュリティ上の理由から、JSP のコンパイルを無効にして、テキストベースの .jsp ファイルではなく、バイナリの .class ファイルのみを配布できます。

JSPC コンパイラは、Web サーバーのコンテキストの外部で JSP をコンパイルするときに使用するコマンドラインツールです。JSPC コンパイラを使用すると、JSP のリクエスト時に JRun を使用して JSP をコンパイルするのではなく、コマンドラインから明示的に JSP をコンパイルできます。JSPC が <JRun のルートディレクトリ >/bin ディレクトリに置かれている場合は IBM jikes コンパイラを使用します。そうでない場合は、Sun javac コンパイラを使用します。

メモ：JSPC コンパイラは、JRun での JSP のコンパイルに使用されるコンパイラと同じコンパイラです。唯一異なる点は、JSPC コンパイラはコマンドラインから起動することです。

JSPC の呼び出し

次のコマンドを使用して、JSPC コンパイラを呼び出します。JSPC が <JRun のルートディレクトリ>/bin ディレクトリに置かれている場合は IBM jikes コンパイラを使用します。そうでない場合は、Sun javac コンパイラを使用します。

```
jspc
[-w web_app_root]
[-d output_directory]
[-k keep processing on errors]
[JSP names]
```

JSPC コンパイラの引数

JSPC では次の引数を使用します。

引数	説明
-w	(オプション) Web アプリケーションのルートディレクトリへの絶対パス。デフォルトでは、このディレクトリは現在の作業ディレクトリです。
-d	(オプション) JSPC コンパイラが .class および .java ファイルの出力を書き込む場所を指定します。デフォルトでは、このディレクトリは Web アプリケーションのルートディレクトリを基準にした WEB-INF/jsp ディレクトリです。
-k	(オプション) エラーが発生しても処理を続行するように JSPC に指示します。
JSP names	(オプション) コンパイルする JSP の名前。JSP を指定しない場合、JSPC は Web アプリケーション内にあるすべての JSP をコンパイルします。

JSPC コンパイラを使用して、jrun-web.xml デプロイメントディスクリプタで仮想マッピングが指定されている Web アプリケーションを処理する場合は、`-w` 引数を指定する必要があります。

JSPC コンパイラの例

次に、ページ foo.jsp をコンパイルする例を示します。このページは、c:/myapps/store にドキュメントルートディレクトリを持つ Web アプリケーションの一部です。この例では、JSPC コンパイラを実行するために必要な jrun.jar は、CLASSPATH 環境変数に含まれています。

```
jspc -w c:\jrun -d c:\myapps\store\WEB-INF\classes foo.jsp
```

次の例では、複数の JSP をコンパイルします。JSP のパスにワイルドカードが使用されていることに注意してください。

```
jspc -d c:\myapps\store\WEB-INF\classes *.jsp store\*.jsp
```

Enterprise JavaBeans のパッケージ

EJB モジュールには 1 つ以上の EJB を含めることができ、次の表のディレクトリ構造を維持している必要があります。単独で、またはエンタープライズアプリケーションの一部として EJB モジュールをデプロイできます。

ディレクトリ	説明
<EJB モジュール> (ルートディレクトリまたは WAR ルート)	Java パッケージ構造に合ったサブディレクトリに、コンパイル済み EJB のインターフェイスおよび実装クラスを格納します。
/META-INF	標準の EJB デプロイメントディスクリプタ (ejb-jar.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-ejb-jar.xml) を格納します。デプロイメントディスクリプタの詳細については、第 1 章、1 ページの「J2EE モジュールの設定」を参照してください。

このセクションでは、JRun で EJB をデプロイするための準備の概要について説明しますが、特定のツールについては説明しません。JRun には、EJB の開発、パッケージ、およびデプロイに使用できるツールが 2 つあります。

- **エンタープライズデプロイウィザード**は、EJB の開発、アセンブル、およびデプロイ処理全体で役立ちます。
- **XDoclet** は、bean を実装したソースファイル内の特殊な Javadoc コメントをベースにして、EJB インターフェイス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。JRun では、JRun 固有のデプロイメントディスクリプタをサポートするように XDoclet を拡張し、自動コンパイルおよびデプロイをサポートしています。

これらのツールの詳細については、『JRun プログラマーガイド』を参照するか、XDoclet の Web サイト (sourceforge.xdoclet.org) をご覧ください。

アプリケーションの bean はまとめてパッケージすることも、個別にパッケージすることもできます。

- bean を個別にパッケージすると、再利用可能性が高くなります。
- すべての bean をまとめてパッケージするほうが簡単です。この方法は、すべての bean を使用する予定があることがわかっている場合のみ有効です。
- 特定のカテゴリに分類される多数の bean が含まれているアプリケーションの場合は、関連 bean をまとめてパッケージすることをお勧めします。

EJB モジュールをパッケージするには

1. Java パッケージ構造に合ったサブディレクトリに、コンパイル済み EJB のインターフェイスおよび実装クラスが含まれているステージングディレクトリを作成します。このディレクトリを EJB ディレクトリとして簡単に識別できるように、-ejb で終わるディレクトリ名を使用します。
2. ステージングディレクトリで META-INF というディレクトリを作成します。

3 META-INF ディレクトリにデプロイメントディスクリプタを置きます。デプロイメントディスクリプタには次のものがあります。

- 標準 EJB デプロイメントディスクリプタ (ejb-jar.xml)
- 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-ejb-jar.xml)。このデプロイメントディスクリプタを使用すると、JNDI 名、コンテナ管理パーシスタンス (CMP)、メッセージによる bean の設定など、EJB モジュールに JRun 固有の機能を定義できます。

メモ: デフォルトの jrun-ejb-jar.xml ファイルを自動的に生成して、ニーズに合わせて編集できます。詳細については、第 3 章の「JRun 固有のデプロイメントディスクリプタの操作」を参照してください。

JRun エンタープライズデプロイウィザードまたは XDoclet を使用してデプロイメントディスクリプタを作成したり、手動でコーディングしたりできます。EJB デプロイメントディスクリプタの詳細については、第 1 章、1 ページの「J2EE モジュールの設定」を参照してください。

4 開発環境で EJB モジュールをスタンドアローンモジュールとしてデプロイするには、ディレクトリ構造を JRun デプロイディレクトリにコピーします。デフォルトのデプロイディレクトリは JRun サーバーのルートディレクトリ (<JRun のルートディレクトリ >/servers/<JRun サーバー >) です。JMC を使用すれば、デプロイディレクトリを追加または削除できます。JRun では、モジュールが自動的にデプロイディレクトリにデプロイされます。デプロイの詳細については、第 3 章、45 ページの「J2EE モジュールのデプロイ」を参照してください。

5 運用環境向けに EJB モジュールを準備するには、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 JAR ファイルにディレクトリ構造をパッケージします。

```
jar cvf <EJB モジュール>.jar
```

ここで、<EJB モジュール> は EJB モジュールの名前です。

6 EJB がエンタープライズアプリケーションの一部である場合は、43 ページの「エンタープライズアプリケーションのパッケージ」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のパッケージ

エンタープライズリソースアダプタでは、次の表のディレクトリ構造を維持している RAR アーカイブファイルに 1 つ以上のリソースアダプタを含めることができます。単独で、またはエンタープライズアプリケーションの一部としてエンタープライズリソースアダプタをデプロイできます。

ディレクトリ	説明
< リソースアダプタモジュール > (RAR ルート)	Java パッケージ構造に合ったサブディレクトリに、META-INF ディレクトリおよびリソースアダプタクラスを格納します。
/META-INF	META-INF 標準 リソースアダプタデプロイメントディスクリプタ (ra.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-ra.xml) を格納します。デプロイメントディスクリプタの詳細については、第 1 章、1 ページの「J2EE モジュールの設定」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モジュールをパッケージするには

1 Java パッケージ構造に合ったサブディレクトリにリソースアダプタクラスファイルが含まれているステージングディレクトリを作成します。オプションで、ステージングディレクトリの JAR ファイル内にリソースアダプタクラスファイルをパッケージできます。このディレクトリをリソースアダプタディレクトリとして簡単に識別できるように、-rar で終わるステージングディレクトリ名を使用します。

2 ステージングディレクトリで META-INF というサブディレクトリを作成します。

3 META-INF ディレクトリにデプロイメントディスクリプタを置きます。デプロイメントディスクリプタには次のものがあります。

- 標準リソースアダプタデプロイメントディスクリプタ (ra.xml)。このデプロイメントディスクリプタではリソースアダプタ実装コード、設定プロパティ、およびセキュリティ情報についての情報を提供します。
- 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-ra.xml)。ra.xml ファイルに `display-name` が含まれており、管理された接続ファクトリインスタンスが 1 つだけ必要な場合、jrun-ra.xml ファイルは不要です。このデプロイメントディスクリプタを使用すると、リソースアダプタインスタンスの JNDI 名、設定プロパティ、セキュリティ設定、接続プール設定などの JRun 固有の機能を定義できます。

リソースアダプタデプロイメントディスクリプタの詳細については、第 1 章、1 ページの「J2EE モジュールの設定」を参照してください。

4 デプロイまたは配布するためにリソースアダプタを準備するには、ステージングディレクトリの最上位レベルから次の Java jar ユーティリティコマンドを使用して RAR ファイルにディレクトリ構造をパッケージします。

```
jar cvf <リソースアダプタモジュール>.rar
```

ここで、< リソースアダプタモジュール > はリソースアダプタモジュールの名前です。

5 リソースアダプタモジュールをスタンドアローンモジュールとしてデプロイするには、RAR ファイルを JRun デプロイディレクトリにコピーします。デフォルトのデプロイディレクトリは JRun サーバーのルートディレクトリ (<JRun のルートディレクトリ>/servers/<JRun サーバー>) です。JMC を使用すれば、デプロイディレクトリを追加または削除できます。JRun では、モジュールが自動的にデプロイディレクトリにデプロイされます。デプロイの詳細については、第 3 章、45 ページの「J2EE モジュールのデプロイ」を参照してください。

6 リソースアダプタがエンタープライズアプリケーションの一部である場合は、43 ページの「エンタープライズアプリケーションのパッケージ」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のパッケージ

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、および 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た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 (application.xml) から構成されています。エンタープライズアプリケーションには、そのモジュールが依存するすべてのクラスも含まれています。

エンタープライズアプリケーションは、次の表のディレクトリ構造を維持している必要があります。

ディレクトリ	説明
<エンタープライズアプリケーション>(ルートディレクトリまたは EAR ルート)	META-INF ディレクトリと次の要素が含まれています。 • Web アプリケーションの WAR ファイルまたはディレクトリ • EJB の 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の RAR ファイルまたはディレクトリ • パッケージした依存クラスおよびユーティリティクラス ルートレベルまたはサブディレクトリに J2EE モジュールを置くことができます。
/META-INF	適切に設定された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 (application.xml) が含まれています。デプロイメントディスクリプタの詳細については、第 1 章、1 ページの「J2EE モジュールの設定」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パッケージするには

1 アーカイブファイルまたは展開したディレクトリにエンタープライズアプリケーションの J2EE モジュールが含まれているステージングディレクトリを作成します。このディレクトリをエンタープライズ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、-ear で終わるディレクトリ名を使用します。

2 エンタープライズアプリケーションを移植可能にするには、JDBC、JMS、JavaMail リソースなどの外部リソースへの依存性を除き、その展開したディレクトリまたは EAR ファイルの外部に依存しないようにします。たとえば、Web アプリケーションモジュールが仮想マッピングに依存した状態で Web アプリケーション内にないライブラリにアクセスする場合は、必要なファイルを Web アプリケーションの WEB-INF/lib ディレクトリにコピーしてください。

3 すべてのモジュールの依存性を調整します。エンタープライズアプリケーションの複数のモジュールが、同じユーティリティクラスへのアクセスを必要とする場合があります。

このような状況の対処方法の詳細については、第 1 章、1 ページの「J2EE モジュールの設定」を参照してください。

4 エンタープライズアプリケーションディレクトリの META-INF ディレクトリに application.xml ファイルを置きます。

5 開発環境でエンタープライズアプリケーションをデプロイするには、そのディレクトリ構造を JRun サーバーのルートディレクトリ (<JRun のルートディレクトリ>/servers/<JRun サーバー>) にコピーします。JRun では、モジュールが自動的に JRun サーバーのルートディレクトリにデプロイされます。

デプロイの詳細については、第 3 章、45 ページの「J2EE モジュールのデプロイ」を参照してください。

6 運用環境向けにエンタープライズ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 EAR ファイルにディレクトリ構造をパッケージします。

```
jar cvf <エンタープライズモジュール>.ear
```

ここで、<エンタープライズモジュール> はエンタープライズアプリケーションの名前です。

展開したディレクトリにエンタープライズアプリケーションのモジュールが置かれている場合は、オプションで適切なアーカイブファイルにエンタープライズアプリケーションをパッケージし、さらに EAR ファイルにパッケージできます。

第 3 章
J2EE モジュールのデプロイ

この章では、JRun での J2EE モジュールのデプロイ方法について説明します。

目 次

モジュールのデプロイの概要

JRun では、次のタイプの J2EE モジュールを 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簡単にデプロイしたりアンデプロイしたりできます。

- Web アプリケーション (WAR ファイルまたはディレクトリ)
- 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 ファイルのみ)
- 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)

J2EE モジュールに関する知識がない場合は、『JRun 入門』を参照してください。

モジュールを**オートデプロイディレクトリ**に置いて自動的にデプロイしたり、JMC を使用してモジュールをデプロイ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 (EJB のみ) 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たり、JRun サーバーに EJB をデプロイしたりすることができます。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。

JMC を使用してモジュールをアンデプロイおよびリデプロイできますが、デプロイディレクトリにあるモジュールはアンデプロイできません。デプロイディレクトリのモジュールをアンデプロイするには、モジュールをディレクトリから手動で削除します。

この章では、モジュールのデプロイおよびアンデプロイについてのみ説明します。モジュールのパッケージ方法の詳細については、第 2 章、33 ページの「J2EE モジュールのパッケージ」を参照してください。

ダイナミックモジュールデプロイ

概要

JRun では、新規および修正済みの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、EJB、およびエンタープライズリソースアダプタのダイナミックデプロイを実行できます。新規モジュールを追加したり、デプロイしたモジュールを変更したりした後に JRun サーバーを再起動する必要はありません。ダイナミックデプロイは、次の機能によって実行できます。

- **オートデプロイディレクトリ** JRun サーバーの起動時に、デプロイディレクトリでモジュールアーカイブファイルまたは展開したモジュールディレクトリが検出されると、モジュールが自動的にデプロイされます。デフォルトでは、この機能はホットデプロイ機能と結合されています。ホットデプロイ機能では、実行中の JRun サーバーのデプロイディレクトリにコピーしたモジュールが自動的にデプロイされます。デフォルトの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> ですが、追加のデプロイディレクトリを作成して、既存のデプロイディレクトリを修正または削除できます。その方法としては、JMC を使用するか、あるいは `deployDirectory` 属性を JRun サーバーの <JRun のルートディレクトリ >/servers/< サーバー名 >/SERVER-INF/jrun.xml ファイルの `DeployerService` セクションに追加または削除する方法があります。たとえば、ディレクトリ c:/my_deployments をデプロイディレクトリとして設定するには、次のエントリを jrun.xml ファイルに追加します。

  ```
  <attribute name="deployDirectory">c:/my_deployments
      </attribute>
  ```

  また、クラスタデプロイディレクトリでモジュール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モジュールを自動的にデプロイします。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、追加のデプロイディレクトリを作成して、既存のデプロイディレクトリを修正または削除できます。その方法としては、JMC を使用するか、あるいは `deployDirectory` 属性を JRun サーバーの <JRun のルートディレクトリ >/servers/< サーバー名 >/SERVER-INF/jrun.xml ファイルの `ClusterDeployerService` セクションに追加または削除する方法があります。JMC を使用してクラスタにモジュールをデプロイすると、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、モジュールアーカイブファイルをコピーします。

- **オートデプロイファイル (JMC)** JRun では、モジュールの WAR、JAR、RAR、または EAR ファイルや、JMC を使用して追加する展開したモジュールディレクトリを自動的にデプロイします。

  また、JRun サーバーの `jrun.xml` ファイル内にある `DeployerService` の `url` または `file` 属性にオートデプロイファイルを指定すると、そのオートデプロイファイルを手動で設定できます。たとえば、c:/my_files/mybean.jar にある EJB モジュールを自動的にデプロイするには、<JRun のルートディレクトリ >/servers/<JRun サーバー >/SERVER-INF/jrun.xml ファイルの `DeployerService` セクションに次のいずれかのエントリを追加します。

  ```
  <attribute name="file">c:/my_files/mybean.jar</attribute>
  <attribute name="url">file:c:/my_files/mybean.jar</attribute>
  ```

- **ホットデプロイ**　ホットデプロイ機能を使用した場合、モジュールのアーカイブファイルまたは展開したディレクトリのデプロイメントディスクリプタに行われた変更が検出されると、モジュールは自動的にリデプロイ（再起動）されます。たとえば、展開したディレクトリ内の jrun-web.xml ファイルが修正されると、WAR ファイルがホットデプロイされます。また、実行中の JRun サーバーのデプロイディレクトリにコピーされたモジュールも、ホットデプロイ機能によってデプロイされます。

メモ：Macromedia では、運用環境ではホットデプロイ機能を無効にすることを強くお勧めしています。詳細については、次のセクション「ダイナミックデプロイの制御」をリファレンスしてください。開発環境でホットデプロイを使用する場合は、展開したディレクトリの使用によって、ディスクスペース、処理動作などのシステムリソースを節約できます。アーカイブされたモジュールをホットデプロイすると、JRun によってそのアーカイブされたモジュールがテンポラリディレクトリに拡張されます。

モジュールのオートデプロイでは、モジュールのディレクトリ名または拡張子なしのアーカイブファイル名をベースにして、JRun によってモジュール名が作成されます。Web アプリケーションの場合は、モジュールディレクトリ名または拡張子なしのアーカイブファイル名をベースにしてコンテキストルート URL マッピングも作成されます。

たとえば、Web アプリケーション newapp.war をデフォルトのサーバーのルートディレクトリに追加すると、次のように作成されます。

- Web アプリケーション名：newapp
- コンテキストルート：/newapp

ダイナミックデプロイの制御

オートデプロイディレクトリ機能を制御するには、JMC を使用するか、あるいは <JRun のルートディレクトリ >/servers/<JRun サーバー >/SERVER-INF/jrun.xml ファイルの `DeployerService` セクション（クラスタの場合は `ClusterDeployerService` セクション）にある `deployDirectory` 属性を作成または編集します。デフォルトではデプロイディレクトリは 1 つですが、追加のディレクトリを作成して、既存のディレクトリを修正または削除できます。

JRun では、`deployDirectory` 属性で指定されたディレクトリで検出された新規モジュールアーカイブファイルまたはディレクトリが自動的にデプロイされます。サーバーの起動時にこの場所が確認され、サーバーの実行中は新しいファイルの監視も行われます。すべての `deployDirectory` 属性を削除すると、オートデプロイおよびホットデプロイは無効になります。デフォルトの `deployDirectory` 属性は次のとおりです。

```
<attribute name="deployDirectory">{jrun.server.rootdir}</attribute>
```

オートデプロイを有効にしたままでホットデプロイを無効にするには、JMC を使用するか、または `hotDeploy` 属性を <JRun のルートディレクトリ >/servers/<JRun サーバー >/SERVER-INF/jrun.xml ファイルに追加します。デフォルトでは、ホットデプロイは有効になっており、`jrun.xml` ファイルには `hotDeploy` 属性が設定されていません。jrun.xml ファイルの `DeployerService` または `ClusterDeployerService`（クラスタ用）セクションに次の行を追加すると、ホットデプロイが無効になります。

```
<attribute name="hotDeploy">false</attribute>
```

開発目的でのモジュールのデプロイ

展開したモジュールディレクトリを JRun サーバーのデプロイディレクトリにホットデプロイすると、開発中にモジュールをパッケージし直してリデプロイしたり、JRun を再起動したりせずに変更内容をテストできるので、非常に便利です。デフォルトの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 > です。

メモ : デプロイディレクトリのファイルは直接操作できますが、JRun によって他のディレクトリにコピーされたファイルは操作できません。この機能は JRun 3.1 の機能とは異なります。

デフォルトでは、JRun のホットデプロイ機能により、展開したディレクトリのデプロイメントディスクリプタファイルに行われた変更が監視され、そのファイルが変更されるとモジュールがリデプロイされます。サーバーの <JRun のルートディレクトリ >/servers/<サーバー名 >/SERVER-INF/jrun.xml ファイル内のプロパティを編集することによって、指定されているデプロイディレクトリを変更したり、ダイナミックデプロイを無効にしたりできます。詳細については、47 ページの「ダイナミックモジュールデプロイ」を参照してください。

最初に JRun 3.x にデプロイされていた Web アプリケーションを同じ名前の JRun サーバーにデプロイするには、まず <Web アプリケーション >/WEB-INF/jsp ディレクトリの .java ファイルを削除する必要があります。また、JRun 3.x ではデフォルトで生成された JSP が保持されましたが、JRun 4 では保持されなくなりました。生成された JSP を保持するには、次の例に太字で示すように、<JRun のルートディレクトリ >/servers/<JRun サーバー >/SERVER-INF/default-web.xml ファイルで、JSPServlet の **keepGenerated** 初期化パラメータを **true** に設定します。JRun サーバーを再起動すると、変更内容が有効になります。

```
<servlet>
<servlet-name>JSPServlet</servlet-name>
<servlet-class>jrun.jsp.JSPServlet</servlet-class>
<init-param>
<param-name>keepGenerated</param-name>
<param-value>true</param-value>
</init-param>
</servlet>
```

運用目的でのモジュールのデプロイ

運用環境では、モジュールを適切なアーカイブファイルにパッケージしてデプロイすることをお勧めします。そのほうが移植性が高まり、モジュールの変更を簡単に制御できるようになります。また、運用環境ではホットデプロイ機能を無効にしてください。JMC を使用すると、ホットデプロイ機能を簡単に無効にできます。ホットデプロイ機能を無効にする方法の詳細については、47 ページの「ダイナミックモジュールデプロイ」を参照してください。

モジュールを運用環境にデプロイする前に、パッケージ作業が必要になる場合もあります。詳細については、第 2 章の「アセンブル担当者、デプロイ担当者、および管理者のロールについて」を参照してください。

モジュールのアンデプロイとリデプロイ

JMC を使用すると、サーバーが実行中かどうかにかかわらず、JRun サーバーのモジュールをアンデプロイできます。デプロイディレクトリを使用してデプロイされたモジュールを JMC を使用してアンデプロイした場合、そのモジュールは、デプロイディレクトリから削除しないかぎり、次回の JRun 起動時にリデプロイされます。JMC を使用しない場合は、ディレクトリから削除することによって、デプロイディレクトリにあるモジュールをアンデプロイできます。

モジュールをリデプロイするには次のオプションがあります。

- ホットデプロイ機能を使用すると、JRun サーバーの実行中に、モジュールのアーカイブファイルおよび展開したディレクトリのデプロイメントディスクリプタに行われた変更が監視され、それらのファイルが変更されるとモジュールはダイナミックにリデプロイされます。また、モジュールをデプロイディレクトリにコピーすると、そのモジュールはホットデプロイ機能によってリデプロイされます。
- JMC を使用すると、モジュールを個別にリデプロイできます。
- JRun サーバーを再起動すると、デプロイディレクトリ内のモジュールまたは JMC を使用してデプロイしたモジュールがリデプロイされます。

認証のためのユーザー、グループ、およびロールの定義

認証を使用する J2EE モジュールでは、デプロイ担当者は管理者と協力して、ターゲット JRun サーバーのユーザー、グループ、およびロールを定義する場合があります。これらのユーザー、グループ、およびロールは、J2EE モジュールで定義されているセキュリティロールに対応している必要があります。詳細については、『JRun 管理者ガイド』を参照してください。

JRun 固有のデプロイメントディスクリプタの操作

JRun 固有のデプロイメントディスクリプタの生成

モジュールをデプロイすると、EJB、Web アプリケーション、およびリソースアダプタの JRun 固有のデプロイメントディスクリプタを自動的に生成できます。この機能には次のような利点があります。

- 生成されたディスクリプタにはデフォルト値があらかじめ入力されているので、ここからカスタム設定を開始できます。デプロイメントディスクリプタを手動で作成する場合は、最初から jrun-ejb-jar.xml ファイルを作成するよりもこの方法のほうが簡単です。
- 以前にデプロイしたモジュールに対して JMC を使用して設定を修正した場合、生成されたディスクリプタには修正した設定に対応する要素が含まれています。たとえば、JMC を使用して自動サーブレットコンパイルを無効にした場合、対応する要素は jrun-web.xml ファイルに追加されます。
- JRun 3.x の EJB の場合は、JRun 3.x の ejb-jar.xml ファイルの Ejipt 固有の環境エントリに対応する要素を使用して jrun-ejb-jar.xml が生成されます。EJB 2.0 の ejb-jar.xml ファイルも生成されます。
- J2EE Reference Implementation (RI) の EJB の場合は、RI 固有のデプロイメントディスクリプタ内の要素に対応する要素を使用して jrun-ejb-jar.xml ファイルが生成されます。

JRun 固有のデプロイメントディスクリプタを生成するには

1 <JRun のルートディレクトリ>/servers/<JRun サーバー>/SERVER-INF/jrun.xml ファイルで、`DeployerService` の `persistXML` 属性を `true` に設定します。

2 JRun サーバーを再起動します。

3 標準デプロイメントディスクリプタが含まれている J2EE モジュールをデプロイします。

展開したディレクトリにモジュールがある場合、デプロイメントディスクリプタは標準デプロイメントディスクリプタと同じディレクトリに生成されます。アーカイブファイル内にモジュールがある場合、デプロイメントディスクリプタはそのアーカイブファイルと同じディレクトリに生成されます。

アーカイブしたモジュールの外部での JRun 固有のデプロイメントディスクリプタの使用

アーカイブファイル内のモジュールの場合は、JRun 固有のデプロイメントディスクリプタをアーカイブファイルに追加する必要はありません。代わりに、デプロイメントディスクリプタをアーカイブファイルと同じディレクトリに入れ、名前を < モジュール名 >.< モジュールタイプ >.jrun.xml に変更します。ここで、モジュールタイプは ejb、war、または rar のいずれかです。

たとえば、mywebapp.war というアーカイブファイル内にある Web アプリケーションの JRun 固有のデプロイメントディスクリプタには、mywebapp.war.jrun.xml という名前を使用します。

索引