macromedia®
JRUN™4

JRun 管理者ガイド

商標

Afterburner、AppletAce、Attain、Attain Enterprise Learning System、Attain Essentials、Attain Objects for Dreamweaver、Authorware、Authorware Attain、Authorware Interactive Studio、Authorware Star、Authorware Synergy、Backstage、Backstage Designer、Backstage Desktop Studio、Backstage Enterprise Studio、Backstage Internet Studio、ColdFusion、Design in Motion、Director、Director Multimedia Studio、Doc Around the Clock、Dreamweaver、Dreamweaver Attain、Drumbeat、Drumbeat 2000、Extreme 3D、Fireworks、Flash、Fontographer、FreeHand、FreeHand Graphics Studio、Generator、Generator Developer's Studio、Generator Dynamic Graphics Server、JRun、Knowledge Objects、Knowledge Stream、Knowledge Track、Lingo、Live Effects、Macromedia、Macromedia M Logo & Design、Macromedia Flash、Macromedia Xres、Macromind、Macromind Action、MAGIC、Mediamaker、Object Authoring、Power Applets、Priority Access、Roundtrip HTML、Scriptlets、SoundEdit、ShockRave、Shockmachine、Shockwave、Shockwave Remote、Shockwave Internet Studio、Showcase、Tools to Power Your Ideas、Universal Media、Virtuoso、Web Design 101、Whirlwind、および Xtra は、Macromedia, Inc. の米国およびその他の国における商標または登録商標です。このマニュアルにおける他の製品名、ロゴ、デザイン、タイトル、語句は、Macromedia, Inc. または他社の商標、サービスマーク、商号のいずれかであり、特定の法域で登録されている場合があります。

この製品には、RSA Data Security からライセンス許可されたコードが含まれています。

このマニュアルには、サードパーティの Web サイトへのリンクが含まれていますが、このリンク先の内容に関しては、当社は一切の責任を負いません。サードパーティの Web サイトには、ユーザー自身の責任においてアクセスするものとします。これらのサイトへのリンクは、参照のみを目的としてユーザーに提供されるものであり、当社がこれらのサードパーティのサイトの内容に対して責任を負うことを意味するものではありません。

保証責任の制限

Apple Computer, Inc. は、本ソフトウェアパッケージ内容、商品性、または特定用途への適合性につき、明示と黙示の如何を問わず、一切の保証を行いません。ただし、所管の行政機関によっては暗黙的な保証の制限が許可されず、前述した保証の制限が認められない場合があります。当該保証は法律上の特定の権利を付与しますが、その他の権利は所管の行政機関によって異なります。

Copyright © 2002 Macromedia, Inc. All rights reserved. このマニュアルの一部または全体を Macromedia, Inc. の書面による事前の許可なしに、複写、複製、再製造、または翻訳すること、および電子的または機械的に読み取り可能な形に変換することは禁じられています。
パーツ番号 ZJR40M400J

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 管理者ガイドは、既存の環境に Macromedia JRun® サーバーを統合される開発者を対象としています。

この章では、JRun および Macromedia に関連した、Web サイト、ドキュメント、テクニカルサポートなどのリソースを入手する方法について説明します。

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
管理者の役割

この章では、JRun 管理者の役割について説明します。また、ファイルの場所、ポート、運用デプロイなどの Macromedia JRun の重要な概念についても説明します。

目 次

管理の概要

JRun は、サーブレット、JSP (JavaServer Pages)、EJB (Enterprise JavaBeans) などの J2EE (Java 2 Platform Enterprise Edition) 仕様および J2EE テクノロジーの仕様をサポートする J2EE アプリケーションサーバーです。J2EE 仕様では、テクノロジーの定義に加え、各担当者の役割も定義します。次に示すように、各担当者は、アプリケーション開発のライフサイクルにおいて特定のタスクを実行します。

- **システム管理者**　JRun のインストールと管理、JRun の起動と停止、アプリケーションの追加と削除を行います。このマニュアルでは、JRun 管理者が必要とする情報を説明します。
- **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。
- **Java サーブレットとタグライブラリの開発者**　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。
- **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。
- **アプリケーションアセンブル担当者**　J2EE アプリケーションのコンポーネントを結合し、スムーズに連動するようにします。
- **アプリケーションデプロイ担当者**　デプロイと配布のために JRun アプリケーションのパッケージ化を行います。

JRun 管理者の責務には、次のタスクが含まれます。

タスク	説明	章
Web サーバーコネクタ	JRun コネクタを設定して、ご使用の環境でサーバーを設定します。	第 2 章
クラスタリング	環境に合わせて、JRun のクラスタリングを行います。	第 3 章
ロギング	環境に合わせて、JRun ロギングの設定およびカスタマイズを行います。	第 4 章
データベースアクセス	アプリケーションが JDBC を通じてサイトのデータベースにアクセスできるようにします。	第 5 章
JMS 設定	JMS プロバイダ、ファクトリ、および送信先にアクセスできるようにします。	第 5 章
セキュリティ	JRun の認証および実装を設定して、サイトのユーザーおよびロールストアを使用します。	第 6 章
システム管理とチューニング	JRun リソースの使用率を監視し、必要に応じてチューニングを行います。	第 7 章
JNDI	JRun JNDI を設定して、サイトのネーミング / ディレクトリサービスと統合します。	第 8 章
設定	JRun 設定ファイルを管理します。	第 9 章

管理者のための JRun の概要

JRun 管理者は、次に述べる JRun および J2EE の基本を理解する必要があります。

- **ファイルとディレクトリの場所**　重要な JRun ファイルおよびディレクトリの名前、場所、および目的。詳細については、3 ページの「重要な JRun ファイルとディレクトリ」を参照してください。
- **ポート**　JRun サーバーが使用する、ポートの名前、範囲、および使用方法。詳細については、5 ページの「JRun ポート」を参照してください。
- **サーバーの起動と停止**　プラットフォーム特有の方法で JRun サーバーの起動および停止を行います。詳細については、6 ページの「JRun の起動と停止」を参照してください。
- **J2EE リソース**　JRun が JDBC、JavaMail、JMS などの J2EE リソースと対話する方法。詳細については、8 ページの「J2EE リソース」を参照してください。
- **運用環境の作成**　運用環境で使用できるように JRun を設定する方法。詳細については、8 ページの「運用環境の作成」を参照してください。

重要な JRun ファイルとディレクトリ

JRun を効果的に管理するには、重要なファイルと場所の使用方法および目的を理解する必要があります。次の表は、管理者が通常使用するファイルおよびディレクトリの名前、場所、目的を示しています。

ファイルまたはディレクトリ	説明
<JRun のルートディレクトリ>/bin/jrun.exe (Microsoft® Windows®) <JRun のルートディレクトリ>/bin/jrun (UNIX®)	JRun サーバーまたは JRun ランチャーの起動に使用する実行可能ファイルを格納します。
<JRun のルートディレクトリ>/bin/jvm.config	Java ホーム、JVM 引数、クラスパスなどの JVM 設定情報を格納します。
<JRun のルートディレクトリ>/lib/jrun.jar	JRun が使用する JRun ライブラリ、J2EE ライブラリ、および他のユーティリティライブラリを含む、JRun が必要とするすべてのライブラリを格納します。
SERVER-INF	この表にリストされている残りのファイルなど、JRun サーバーの設定ファイルを格納します。これは <JRun のルートディレクトリ>/servers/<JRun サーバー> ディレクトリの下にあります。
SERVER-INF/jrun.xml	JRun サーバーのすべてのサービス定義および設定を格納します。
SERVER-INF/jrun-resources.xml	JMS 送信先、JMS 接続ファクトリ、JDBC データソース、JavaMail セッションなどの J2EE リソースの定義を格納します。
SERVER-INF/jrun-jms.xml	JRun のビルトイン JMS プロバイダに関する設定情報を格納します。

ファイルまたはディレクトリ	説明
SERVER-INF/jrun-dtd-mappings.xml	XML DOCTYPE パブリック ID と物理的 DTD ファイル間のマッピングを指定します。通常、DTD ファイルはリモートサーバーへの接続を通じて解決されます。マッピングを指定すると、DTD ファイルをローカルに読み取ることができます。DTD ファイルは、jrun.jar ファイルの META-INF ディレクトリにあります。
SERVER-INF/jrun-users.xml	デフォルトの JRun セキュリティメカニズムのユーザーおよびロールの定義を格納します。
SERVER-INF/jndi.properties	JRun サーバーの JNDI および ORB ポートを指定します。JNDI ポートは重要な値です。リモートクライアントおよびサーバーは、この値を使用して JRun サーバーとの通信を初期化します。
SERVER-INF/default-web.xml	各 Web アプリケーションの web.xml ファイルのデフォルト値を格納します。このファイルは、JRun が内部で使用しているグローバルサーブレットの定義およびマッピングを格納します。
SERVER-INF/auth.config	認証および許可に使用するログインモジュールを指定します。
<JRun のルートディレクトリ>/lib	jrun.jar など、JRun が使用する JAR ファイルを格納します。
<JRun のルートディレクトリ> /servers/lib	すべての JRun サーバーが使用する JAR ファイルを格納します。このディレクトリに、JDBC ドライバの JAR ファイルを格納することをお勧めします。
SERVER-INF/classes	JRun サーバー上のすべてのアプリケーションが使用するコンパイル済み Java クラスを格納します。
SERVER-INF/lib	JRun サーバー上のすべてのアプリケーションが使用する JAR ファイルを格納します。
WEB-INF/classes	Web アプリケーションのサーブレットクラスファイルを格納します。Web アプリケーションの jrun-web.xml ファイルによって `reload` 要素が true に設定される場合、修正したファイルを保存すると、JRun は自動的にサーブレット、サーブレットヘルパークラス、JSP、および JSP ヘルパークラスをリロードします。
WEB-INF/lib	Web アプリケーションが使用する JAR ファイルを格納します。Web アプリケーションの jrun-web.xml ファイルによって `reload` 要素が true に設定される場合、修正した JAR ファイルを保存すると、JRun は自動的にクラスをリロードします。
<JRun のルートディレクトリ> /docs/dochome.htm	オンライン JRun ドキュメントのホームページ

JRun ポート

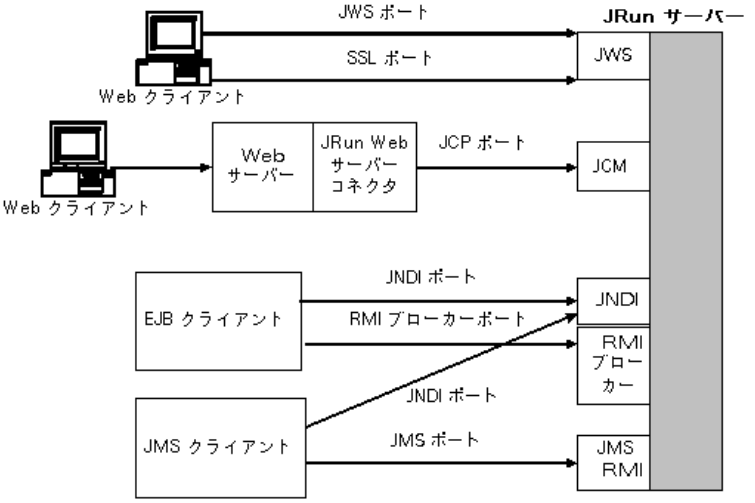
JRun サーバーが使用するポートは多数あります。JRun を管理する場合は、それらの設定および使用方法を理解しておく必要があります。新規 JRun サーバーを追加するためには、これらのポートの認識が特に重要となります。

次のように、これらのポートは各 JRun サーバーに対して固有のものである必要があります。

- **JWS (JRun Web server) ポート**　JRun はこのポートを使用して、着信する HTTP リクエストをリスンします。このポートは、JRun サーバーの jrun.xml ファイルで指定します。
- **RMI ブローカーポート**　JRun はこのポートを使用して、リモート EJB、リモート JDBC、リモートサービス、リモートユーザートランザクションなどのリモート通信を処理します。このポートは、JRun サーバーの jrun.xml ファイルで指定します。デフォルトの JRun サーバーはポート 0 を使用します。これによって、JRun はランダムにポートを選択します。

  個々の EJB は、必要に応じて独自の RMI ブローカーを取得して、特別なソケットファクトリを使用したり (SSL を通じた EJB)、リモート呼び出し負荷を均等に分散したりすることができます。個々の RMI ブローカーは jrun-ejb-jar.xml ファイルで設定します。jrun-ejb-jar.xml ファイルに RMI ブローカーが指定されていないと、JRun はグローバル共有 RMI ブローカーを使用します。RMI ブローカーはすべてクラスタリング可能です。
- **SSL サービスポート**　JWS は SSL トラフィックにこのポートを使用します。
- **Web サーバーコネクタポート**　Web サーバーコネクタは、JCP (JRun Control Port：JRun コントロールポート) とも呼ばれるこのポートを使用して、着信した HTTP リクエストを JRun に渡します。
- **JMS ポート**　JRun は、デフォルトの JMS プロバイダの転送メカニズムとして RMI を使用する場合にこのポートを使用します。このポートは、JRun サーバーの jrun-jms.xml ファイルで指定します。デフォルトの JRun サーバーはポート 0 を使用します。これによって、JRun はランダムにポートを選択します。
- **JNDI ネーミングサーバーポート**　JRun はこのポートを使用して、クライアントに `Context` スタブを渡します。リモートクライアントは `Context.PROVIDER_URL` プロパティ内でこのポートに名前を付けます。このプロパティは、EJB、JMS、JRun サービスなどのリモート通信を起動するときに `InitialContext` コンストラクタに渡されます (ローカルクライアントは空の `InitialContext` コンストラクタを使用することができます)。このポートは、JRun サーバーの jndi.properties ファイルで定義します。
- **JNDI RMI ポート**　JNDI ネーミングサーバーポートを通じてクライアントが `Context` スタブを取得すると、JRun はすべてのリモート通信に JNDI RMI ポートを使用します。このポートは、JRun サーバーの jndi.properties ファイルで指定します。デフォルトの JRun サーバーはポート 0 を使用します。これによって、JRun はランダムポートを選択します。
- **JNDI ORB ポート**　CORBA フェデレーションが有効な場合、JRun はこのポートを使用します。これは、CORBA ネーミングサービス (COSNaming) がリスンするポートです。このポートは、JRun サーバーの jndi.properties ファイルで指定します。

次の図は、クライアントが一般的に使用する JRun と対話する方法を示しています。



JRun サーバーを追加する場合は、既存のポート設定を確認し、未使用のポートを割り当てる必要があります。

JRun の起動と停止

次の方法で JRun の起動および停止ができます。

- JRun サーバーコントロールパネル
- コマンドライン

メモ：JMC を使用しても JRun の起動および停止ができます。しかし、実際の運用環境では通常は使用しません。

次のセクションでは、これらのテクニックについて説明します。

ランチャーの使用

ランチャーとも呼ばれる JRun サーバーコントロールパネルは、JRun サーバーの起動、再起動、および停止ができる Java Swing アプリケーションです。<JRun のルートディレクトリ >/bin ディレクトリ内にある jrun.exe ファイル (Windows 用) または jrun 実行可能ファイル (UNIX 用) を実行することによってランチャーを実行します。ランチャーには、JRun の起動、再起動、および停止ボタンがあります。

ランチャーに表示されているサーバーリストは、<JRun のルートディレクトリ >/lib/servers.xml ファイルが制御します。JMC 内にサーバーを作成すると、JRun は自動的にサーバーを servers.xml ファイルに追加します。また、必要に応じてテキストエディタでこのファイルを編集して次の属性を指定します。

- server　名前およびディレクトリ要素をラップします。
- name　サーバー名を指定します。
- directory　JRun サーバーのルートディレクトリを指定します。

次のコードはデフォルトの servers.xml ファイルを示しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE servers PUBLIC "-//Macromedia, Inc.//DTD servers 4.0//EN"
        "http://jrun.macromedia.com/dtds/servers.dtd">
<servers>
  <server>
    <name>default</name>
    <directory>{jrun.home}/servers/default</directory>
  </server>
  <server>
    <name>samples</name>
    <directory>{jrun.home}/servers/samples</directory>
  </server>
  <server>
    <name>admin</name>
    <directory>{jrun.home}/servers/admin</directory>
  </server>
</servers>
```

JMC の [JVM 設定] パネルを使用して、クラスパス、JNI 設定、VM サイズなどの JRun サーバーの起動指定を作成します。

コマンドラインの使用

jrun.exe (Windows 用) および jrun 実行可能ファイル (UNIX 用) のシェルスクリプトコマンドラインユーティリティを使用して、JRun を起動できます。次のシンタックスを使用します。

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

次の表に、各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。
-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 (UNIX のみ)	別のプロセスで JRun を起動します。
-version	JRun のバージョン番号 (主に OEM 用) を表示します。
-info	補足情報 (主に OEM 用) を表示します。

J2EE リソース

J2EE アプリケーションは、リソースと呼ばれるさまざなま外部システムと対話します。代表的な外部リソースは、次のものがあります。

- **データベース**　JDBC を使用してアクセス
- **メール**　JavaMail を使用してアクセス
- **レガシーシステム**　JCA (J2EE Connector Architecture) を使用してアクセス
- **メッセージシステム**　JMS を使用してアクセス

リソース管理の詳細については、第 5 章、53 ページの「リソース」を参照してください。

運用環境の作成

アプリケーションの開発およびテスト段階では、セキュリティ、パフォーマンス、および他の運用に関する検討事項は重要ではありません。しかし、運用環境でサーバーを最適かつ安全に稼働させるのは JRun 管理者の役目です。

次の表では、JRun サーバーを運用環境に移行する手順を説明します。

アクション	説明
JWS を無効にする	アプリケーションが Web サーバーコネクタを使用している場合は、JWS (JRun Web サーバー) を無効にします。詳細については、116 ページの「WebService」を参照してください。
サーブレットマッピングを作成する	SERVER-INF/default-web.xml ファイルの /servlet マッピングを無効にします。web.xml ファイル内にサーブレットおよびサーブレットマッピングを作成します。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。
ホットデプロイを無効にする	Macromedia では、運用環境ではホットデプロイ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、113 ページの「DeployerService」を参照してください。
FileServlet マッピングを削除する	アプリケーションがサーブレットおよび JSP のみで構成されている場合は、SERVER-INF/default-web.xml ファイル内の / の FileServlet サーブレットマッピングを無効にします。
Web サービスを無効にする	アプリケーションが Web サービスを使用しない場合は、SERVER-INF/default-web.xml ファイル内の AxisServlet マッピングを無効にします。
JSP 変換を無効にする	既にすべての JSP をアプリケーションにコンパイルしている場合は、SERVER-INF/default-web.xml ファイル内の translationDisabled 属性を true に設定します。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。
自動コンパイルを無効にする	jrun-web.xml ファイル内の reload および compile 要素を false に設定します。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。
Web サーバーアクセスを管理する	ProxyService の interface 属性が運用 Web サーバーの IP アドレスに設定されていることを確認します。詳細については、17 ページの「単純な分散型インストール」を参照してください。

アクション	説明
パスワードを暗号化する	アプリケーションが JRun のデフォルトのセキュリティ実装を使用している場合は、SERVER-INF/jrun-users.xml ファイル内のパスワードが暗号化されていることを確認します。詳細については、73 ページの「暗号化」を参照してください。
ディレクトリリファレンスを無効にする	SERVER-INF/default-web.xml ファイル内の FileServlet の browseDirs パラメータを false に設定して、Web ブラウザのディレクトリリファレンスを無効にします。
サンプルサーバーを削除する	サンプル JRun サーバーがデプロイされていないことを確認します。これによる安全上の危険性はありませんが、外部ユーザーによるサンプルアプリケーションへのアクセスにともなうリソースの浪費を防ぐことができます。
ドキュメントを削除する	<JRun のルートディレクトリ >/docs ディレクトリを削除します。サンプルサーバーによる安全上の危険性はありませんが、外部ユーザーによるドキュメントへのアクセスにともなうリソースの浪費を防ぐことができます。

Macromedia では、運用デプロイの最適な方法および推奨事項を頻繁に更新しています。JRun 運用デプロイの最新情報については、JRun サポートセンター (http://www.macromedia.com/support/jrun/) をご覧ください。

第 2 章
Web サーバーコネクタ

この章では、外部 Web サーバーに接続する場合の JRun について説明します。また、いくつかの一般的な事例と、分散環境で JRun を使用する場合に役立つセキュリティと要求のチェーン化などの問題も取り上げます。

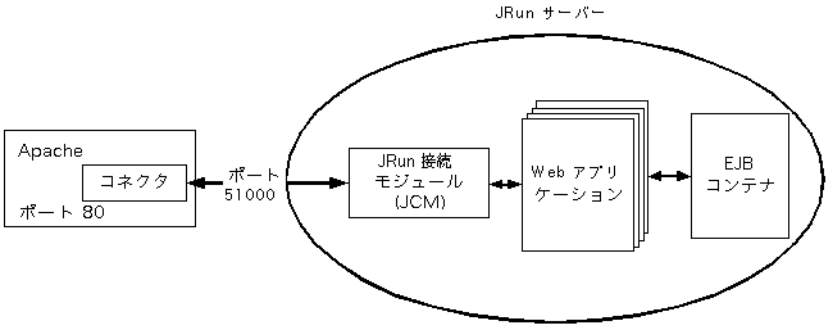
目 次

Web サーバーコネクタについて

ネイティブの Web 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。たとえば、JRun は Apache モジュールを使用して、各ハードウェアアーキテクチャおよび JRun でサポートされているオペレーティングシステムに対して、Apache web Server 用のコネクタを作成します。JRun Web サーバーコネクタを、バックエンドシステムと統合できる Sun J2EE JCA (Java Connector Architecture) コネクタと混同しないでください。

JRun 接続モジュール (JCM)

各 JRun サーバーに 1 つ以上の Web サーバーを接続できます。基本的なデプロイ環境では、アプリケーションを処理する 1 つの 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 サーバーに接続された 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に対してリクエストがあると、Web サーバー上のコネクタにより、JRun サーバー内に常駐する JCM (JRun 接続モジュール) へのネットワーク接続が確立されます。JCM によって透過的に通信ができるようになり、コネクタから JRun サーバーにリクエストが転送されます。JRun サーバーはリクエストを処理して、JCM に返信します。

JRun プロキシポート

ポート番号およびオプションのバインドアドレスを使用して、Web サーバーと JRun サーバー間の接続を定義します。マシン上の各 JRun サーバーは、異なるネットワークポートを使用して、Web サーバーからのリクエストをリスンします。このポートを、JRun プロキシポートまたは JCP (JRun コントロールポート) と呼びます。前の例では、デフォルト JRun サーバーはポート 51000 をリスンします。コンピュータの各 JRun サーバーが、このポート番号に合わせて一意に設定されていることを確認してください。このポート番号は、JMC の [外部 Web サーバー接続設定] パネルを使用して指定します。

また、2 番目のパラメータである Interface を使用して、Web サーバーと JRun サーバー間の接続を定義することもできます。インターフェイスアドレスには、JRun サーバーがリクエストを受信できるネイティブ Web サーバーの IP アドレスを指定します。デフォルトのインターフェイス属性は 127.0.0.1 に設定されます。これは、JRun サーバーがローカルホストからのリクエストをリスンすることを意味します。JMC を使用して、これらの IP アドレスを指定できます。

Web サーバーコネクタの確立

Web サーバー設定ツールを実行することによって Web サーバーコネクタを作成します。このツール (JRun 3.x で使用されたコネクタウィザードの代わりとなる) は、Swing アプリケーションです。Web サーバー上にコネクタをインストールし、Web サーバーと JRun サーバー間の接続を定義します。

Web サーバー設定ツールを実行するには

1 接続先の JRun サーバーを起動します。JRun サーバーがクラスタの一部となっている場合は、クラスタ内ですべての JRun サーバーを起動します。

2 Web サーバーがあるマシンで、Web サーバー設定ツールを実行します。次のいずれかの方法で、Web サーバー設定ツールを実行します。

- (Windows のみ) [スタート] > [プログラム] > [Macromedia JRun 4] > [Web サーバー設定ツール] を選択します。
- コンソールウインドウを開き、<JRun のルートディレクトリ >/lib ディレクトリに移動し、次のコマンドを入力します。

```
java -jar wsconfig.jar
```

通常は Web サーバー設定ツールを使用して、同じコンピュータ上で動作している Web サーバーと JRun サーバー間の接続を設定します。ただし、Web サーバーと JRun サーバーを同じコンピュータ上にインストールする必要はありません。Web サーバーコネクタの設定オプションについては、後の章で詳しく説明します。Web サーバー設定ツールの実行の詳細については、『JRun インストールガイド』を参照してください。

メモ : クラスタ内に JRun サーバーがあると、Web サーバーコネクタは自動的にロードバランスおよびフェイルオーバーを有効にします。詳細については、35 ページの「コネクタベースのクラスタリング」を参照してください。

Web サーバー設定ツールは次のように、Web サーバーのコンピュータ上にある設定ファイルにプロパティを保管します。

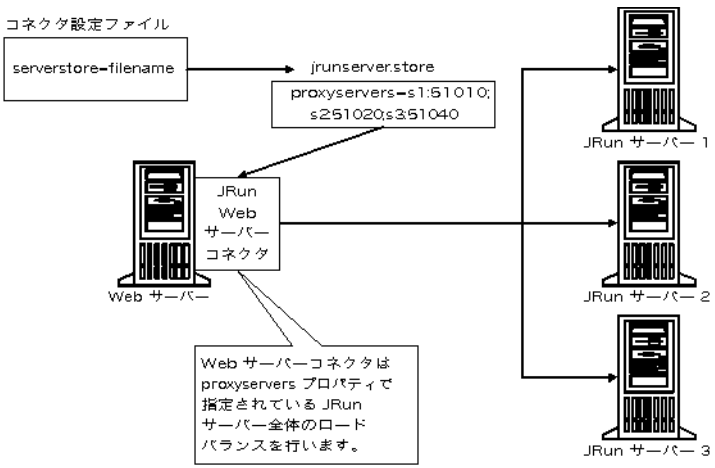
- **IIS** <JRun のルートディレクトリ >/lib/wsconfig ディレクトリにある jrun.ini ファイル内です。
- **Apache** <Apache のルートディレクトリ >/conf ディレクトリにある httpd.conf ファイル内です。
- **Netscape/iPlanet** server-http-xxx config ディレクトリにある obj.conf および magnus.conf ファイル内です。

JRun Web サーバーコネクタはこの設定を使用して、JRun サーバーを検出し、どのサーバーに接続すべきかを判断します。次の表では、Web サーバーの設定ファイルに含まれている JRun プロパティを説明します。

プロパティ	説明
bootstrap	JRun サーバーのプロキシサービスがコネクタリクエストをリスンする IP アドレスおよびポート。JRun もこのポート / アドレスの組み合わせをリスンするように設定し、ProxyService をアクティブにする必要があります。
serverstore	ブートストラップクラスタ内にあり JRun サーバーに関する情報が含まれているファイルの名前。コネクタは自動的にこのファイルを作成します。デフォルトは jrunserver.store です。
verbose	JRun サーバーと Web サーバー間の通信に関する、より詳細な Web サーバーのログファイルエントリを作成します。このオプションを有効にすると、Web サーバーのログファイルは急速に大きくなります。true または false を指定します。デフォルトは false です。

プロパティ	説明
scriptpath	IIS のみ適用されます。Web サーバー上の virtual/JRunScripts ディレクトリを指します。
errorurl	(オプション) カスタマイズされたエラーメッセージが含まれているファイルへの URL を指します。デフォルトでは、このプロパティはコメント化されています。
ssl	(オプション) Web サーバーコネクタが SSL を使用して JRun サーバーと通信するかどうかを指定します。true または false を指定します。ほとんどの Web サーバーはファイヤウォール内に存在するので、通常、このプロパティには false を指定します。
apialloc	Web サーバーのアロケータでなくネイティブの OS メモリアロケーションを有効にします。

serverstore プロパティは、クラスタ内のすべて JRun サーバーのマシン名および JRun プロキシポートが含まれているファイル名を指定します。このファイルはデフォルトで jrunserver.store という名前が付いています。**proxyservers** プロパティには、クラスタ内の JRun サーバーがセミコロンで区切られているリストが含まれています。Web サーバーコネクタは、起動時にクラスタ内にブートストラップ JRun サーバーがあることを検出すると、自動的にこのプロパティを挿入します。次の図は、このプロセスを示しています。



詳細については、35 ページの「コネクタベースのクラスタリング」を参照してください。

Web サーバー設定ファイルのサンプル

このセクションでは、Web サーバーと JRun を接続する場合に指定する Web サーバーのプロパティの例を示し、Web サーバーの設定ファイルのパラメータを具体的に説明します。ここで説明する例では、JRun サーバーと Web サーバーが同一のコンピュータ上にあるものとします。

Apache 設定ファイル

Apache Web サーバーと同一のコンピュータに JRun の標準インストールを行った場合、httpd.conf ファイルは次のようになります。

LoadModule ステートメントは、マルチホスティング環境では VirtualHost ディレクティブの外部に記述する必要があります。これは、LoadModule ステートメントのリファレンスはグローバルレベルで 1 回だけ行うようにする必要があるためです。DSO と同様に、Web サーバーコネクタを実行するには LoadModule ステートメントが必要です。

また、Apache 設定ファイルでは、rulespath の代わりに Mappings を使用します。

```
LoadModule jrun_module libexec/mod_jrun.so
<IfModule mod_jrun.c>
    JRunConfig Verbose false
    JRunConfig Apialloc false
    JRunConfig Ssl false
    JRunConfig Serverstore "/usr/apache/conf/jrunserver.store"
    JRunConfig Bootstrap 127.0.0.1:51000
    #JRunConfig Errorurl <オプションで、エラー発生時にこの URL に転送>
</IfModule>
```

IIS 設定ファイル

IIS の場合、JRun では jrun.ini ファイルを使用して jrun.dll フィルタを初期化します。一般的な jrun.ini ファイルは次のようになります。

```
verbose=false
scriptpath=/JRunScripts/jrun.dll
serverstore=C:/JRun4/lib/wsconfig/2/jrunserver.store
bootstrap=127.0.0.1:51000
apialloc=false
ssl=false
#errorurl=<オプションで、エラー発生時にこの URL に転送>
```

Netscape/iPlanet 設定ファイル

Netscape/iPlanet Web サーバーの一般的な obj.conf ファイルは次のようになります。

```
...
<Object name="jrun">
PathCheck fn="jrunfilter"
Service fn="jrunservice"
</Object>
...
```

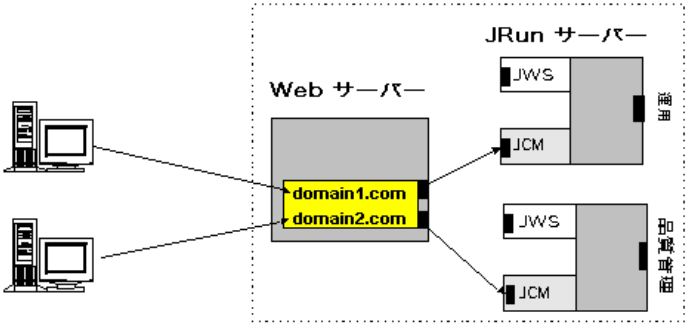
Netscape/iPlanet Web サーバーの一般的な magnus.conf ファイルは次のようになります。

```
...
Init fn="load-modules" shlib="C:/JRun4/lib/wsconfig/1/jrun_nsapi35.dll"
         funcs="jruninit,jrunfilter,jrunservice"
Init fn="jruninit" serverstore="C:/JRun4/lib/wsconfig/1/jrunserver.store"
         bootstrap="127.0.0.1:51000" verbose="false" apialloc="false"
         ssl="false"
```

1 つの Web サーバーと複数の JRun サーバーの接続

1 つの JRun サーバーを 1 つの外部サーバーに接続するのは簡単です。ただし、外部 Web サーバーにさらに別の JRun サーバーを接続する場合は、特別な手順を実行する必要があります。

たとえば、インストールで設定した JRun サーバーの他に、テスト / 品質管理用の JRun サーバーと実際の運用に使用する JRun サーバーが必要であるとします。この場合、複数の JRun サーバーを、同じコンピュータ上の 1 つの Web サーバーに接続して実行します。この例を図に示すと、次のようになります。



このセクションでは、IIS や Apache などの 1 つの Web サーバー、1 つ の JRun プログラム、および複数の JRun サーバーがすべて同一のコンピュータ上にあると想定します。

Web サーバーの接続

このセクションでは、複数の JRun サーバーを同一コンピュータ上の 1 つの Web サーバーに接続する方法について説明します。

複数の JRun サーバーを 1 つの Web サーバーに接続するには

1 Web サーバー固有のユーティリティまたはプロシージャを使用して、JRun サーバー用の仮想 Web サイトを作成します。1 つの JRun サーバーだけがデフォルトの Web サイトを使用できますが、他の JRun サーバーは仮想サイトを使用する必要があります。

2 JMC を使用して各新規 JRun サーバーを作成します。

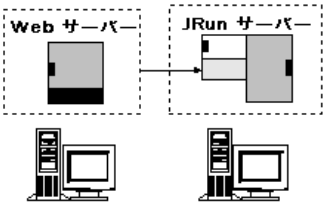
3 JRun サーバーごとに Web サーバー設定ツールを実行し、そのたびに適切な Web サイトを選択します。

メモ : 各 JRun サーバー上の Web アプリケーション、イメージ、および他のリソースの名前が異なる場合は、クラスタリングを使用して複数の JRun サーバーを 1 つの Web サーバーに接続することができます。Web サーバーコネクタは、適切なサーバーにリクエストを送信します。しかし、各リソースに固有の名前が付いていない場合、Web サーバーコネクタは、リソースが要求されると、ロードバランスを実行します。

単純な分散環境での JRun の実行

単純な分散型構成では、1 台のコンピュータを Web サーバー専用とし、もう 1 台を JRun サーバー専用とします。これは、処理の負荷を複数のコンピュータに分散する一般的な方法です。

次の図は、単純な分散環境でのインストールを示しています。



単純な分散型インストール

単純な一対一の分散環境では、次の方法で JRun を設定するようにお勧めします。

JRun を一対一の分散環境にインストールするには

1 Web サーバーのコンピュータに Web サーバー設定ツールをインストールして実行し、リモートホストおよび JRun サーバー名を指定します。

ヒント：カスタムインストールのオプションを使用すると、Web サーバー設定ツールを個別にインストールできます。

2 分散環境で、コンピュータ間のファイルシステムが異なる場合に JSP を使用するときは、仮想パスマッピングプロパティを使用する必要があります。詳細については、19 ページの「分散環境での JSP の使用」を参照してください。

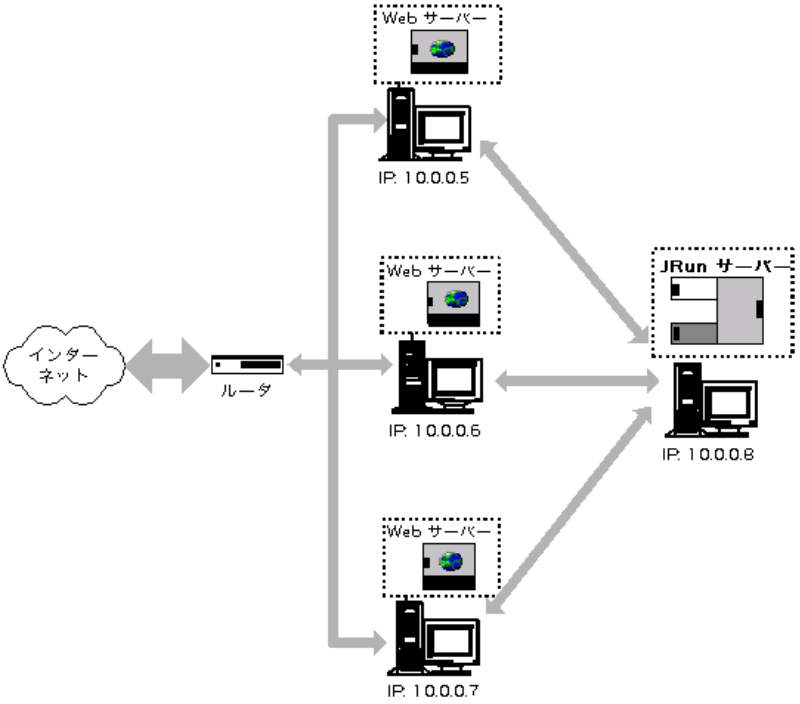
Web サーバーと JRun を異なるコンピュータ上で実行する場合は、jrun.xml ファイルの ProxyService の `interface` 属性と `bindAddress` 属性が、次のように正しく設定されていることを確認します。

- **Interface** JRun に接続できる Web サーバーの IP アドレスを指定します。複数の値がある場合は、パイプ (|) で区切ります。この属性を使用すると、外部の Web サーバーから JRun へのアクセスを制限できます。`interface` 属性のデフォルト値は 127.0.0.1 です。Web サーバー設定ツールは、この属性を Web サーバーコンピュータの IP アドレスに自動的に更新します。
- **BindAddress** 外部 Web サーバーからのリクエストを受信する JRun サーバーの IP アドレスを指定します。

これらの属性は、JMC の [外部 Web サーバー接続設定] パネルを使用して設定することもできます。

複雑な分散環境での JRun の実行

もう 1 つの一般的な事例としては、複数の Web サーバーマシンを使用している環境で、JRun サーバー専用のコンピュータをもう 1 台追加する場合が考えられます。次の図は、この複雑な分散環境を示します。



複雑な分散型インストール

複雑な分散環境では、次の方法で JRun を設定するようにお勧めします。

1 つの JRun サーバーを複数の Web サーバーに接続するには

1 17 ページの「単純な分散環境での JRun の実行」の説明に従って、各 Web サーバーマシンを、JRun と通信するように設定します。

2 ProxyService の `interface` 属性が、17 ページの「単純な分散型インストール」の説明に従って正しく設定されていることを確認します。

分散環境での JSP の使用

複数のコンピュータで異なるファイルシステムを使用することがあります。ユーザーが JSP の URL をリクエストしたとき、Web サーバーの実際のパスは、ページを取得する物理的なパスではありません。Web サーバーの実際のパスを、JRun がリクエストへのレスポンスで使用するパスに変換する必要があります。

JRun の仮想パスマッピングプロパティを使用すると、リクエストを実際のパスにマッピング (パスを変換) することができます。仮想パスマッピングのコードは、次のように、各 Web アプリケーションの WEB-INF/jrun-web.xml ファイル内に作成します。

```
...
<virtual-mapping>
   <resource-path>/images/*</resource-path>
   <system-path>/usr/local/images</system-path>
</virtual-mapping>
...
```

分散 JRun システムの保護

分散環境でセキュリティを実装するときには、注意しなければならないことがあります。このセクションでは、JRun に特有の方法について説明します。

- **admin JWS をシャットダウンします。** デフォルトの JRun インストールでは、admin JRun サーバーなどの JRun サーバーごとに Web サーバーが設定されます。詳細については、20 ページの「JWS をオフにする方法」を参照してください。
- **ホストベース認証を設定します。** JRun には、JRun サーバーと外部 Web サーバー間の通信を他者から防御する基本的なメカニズムが用意されています。この設定については、20 ページの「コネクタのホストベース認証」を参照してください。
- **SSL (Secure sockets layer)** JRun では、コネクタと JRun サーバーの間の接続に SSL を使用できます。この設定については、21 ページの「Web サーバーコネクタでの SSL の使用」を参照してください。

JWS をオフにする方法

デフォルトでは、すべての JRun サーバーで関連している JWS (JRun Web サーバー) が有効に設定されています。各 JWS は、指定されたポート上で HTTP リクエストをリスンします。たとえば、デフォルトの JRun サーバーは、JWS にポート 8100 を使用します。多くのシステム管理者は、ファイアウォールで入力ポートのアクセスを制限しますが、使用していないサービスはオフにすることをお勧めします。このセクションでは、Web サーバーコネクタのみを通じてアクセスされる JRun サーバーのために JWS をオフにする方法について説明します。

使用されていない JRun サーバー用の JWS をオフにするには

1 JRun サーバーの jrun.xml ファイルを開きます。

2 たとえば次のように、`WebService` サービスをコメント化します。

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

または `deactivated` 属性を追加して、true に設定します。

3 JRun サーバーを再起動します。

コネクタのホストベース認証

JRun を実行しているコンピュータと Web サーバーを実行している別のコンピュータ間で接続を確立したら、認証されていないユーザーがネットワーク上の別の場所から JRun サーバーにアクセスできないようにする必要があります。これを行うために、JRun には、JRun コネクタ用のホストベース認証機能が用意されています。これにより、特定のアドレスが設定されているホストだけが JRun サーバーにリクエストを送信できます。

JMC の [外部 Web サーバー接続設定] パネルを使用すると、特定の JRun サーバーと通信可能な IP アドレスを指定できます。[IP フィルタリスト] フィールドを使用して、JRun サーバーがアクセスできる IP アドレスのリストを指定します。これらのマシン上にある Web サーバーのみが JRun サーバーにリクエストを送信できます。「*」を指定すると、すべての Web サーバーが JRun にリクエストを送信できるようになります。

メモ：デフォルトの設定では、JRun サーバーはローカルホストからのリクエストのみを受け付けます。

この設定は、ProxyService を使用して jrun.xml ファイルで行うこともできます。`interface` 属性を使用して IP アドレスを指定します。複数のアドレスを指定する場合は、パイプ (|) でアドレスを区切ります。

メモ：ホストベースの認証による保護では、IP のなりすましや、他の中間一致攻撃 (man-in-the-middle) を防止できません。

Web サーバーコネクタでの SSL の使用

Web サーバーコネクタでは、Web サーバーと JRun サーバーの間で SSL (secure sockets layer) を使用できます。ほとんどの運用構成では、Web サーバーがファイアウォールの内側にあるので、SSL は必要ありません。ただし、セキュリティを最大限強化する場合は、Web サーバーコネクタで SSL を使用できます。

Web サーバーコネクタで SSL を有効にするには、次の手順を実行します。

1 次の Java `keytool` コマンドを使用して、キーストアを生成します。次に例を示します。

```
keytool -genkey -dname "cn=<server name or IP address>, ou=JRunEngineering,
    o=Macromedia, L=Newton, ST=MA, C=US" -keyalg rsa -keystore <keystore
    name>
```

プロンプトが表示されたら、長さが 6 文字以上の適切なパスワードを入力します。

2 `keytool` を再実行して、キーストアに証明を追加します。

メモ：運用環境では通常、証明機関から署名付き証明書を取得します。

3 jrun.xml ファイルを開き、ProxyService の `keyStore` 属性、`keyStorePassword` 属性、および `trustStore` 属性 (オプション) に適切な値を設定します。`keyStore` 属性と `trustStore` 属性は、keystore ファイルと truststore ファイルのパスとファイル名です。

4 OpenSSL をダウンロードしてビルドします。OpenSSL は、http://openssl.org から tar.gz ファイルとして入手できます。配布ファイルをダウンロードしたら、付属の説明に従って、ダウンロードしたファイルをご使用のオペレーティングシステムに合わせてビルドする必要があります。コンパイルした OpenSSL のコードは、<JRun のルートディレクトリ >/servers/lib のような、システムパスのディレクトリに保存します。

5 Web サーバーのコネクタ設定ファイル (jrun.ini、httpd.conf、obj.conf など) を開き、`ssl` プロパティを true に設定します。

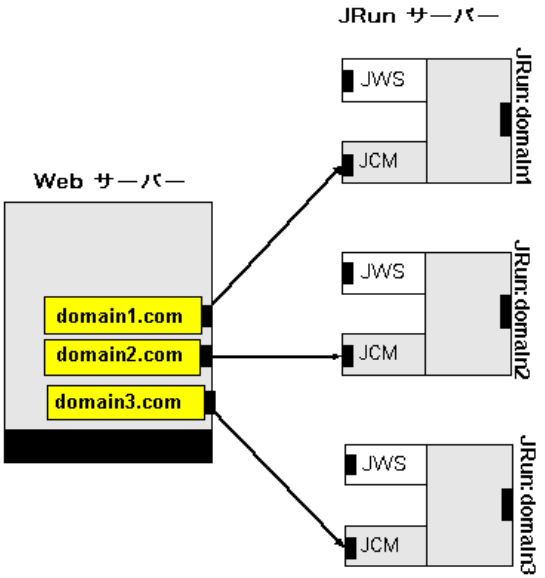
メモ：JWS で SSL を使用するには、SSLService を有効にし、keyStore、keyStorePassword、および trustStore の 3 つの属性に適切な値を設定します。

JRun でのマルチホスティング

「1 つの Web サーバーと複数の JRun サーバーの接続」では、仮想ホストを確立して、1 つの Web サーバーが複数の JRun サーバーと通信できるようにする簡単な事例を説明しました。この事例をより複雑にしたものがマルチホスティングです。マルチホスティングとは、ISP (Internet Service Provider) などが、1 つの Web サーバーの上で複数の仮想ホストを確立することによって別のドメインにアクセスできるようにすることです。通常は、仮想ホストごとに別個の JRun サーバーを作成します。仮想ホストには、次の利点があります。

- 独立した JVM はさまざまな仮想ホストからの Web アプリケーションを処理します。
- 1 つの JRun サーバーが停止しても、すべての Web アプリケーションに支障がないようにします。
- 開発者や顧客が、同じ名前を持つ Web アプリケーションを作成しないようにします (ISP の場合)。

次の図は、1 台のコンピュータで 1 つの Web サーバーを実行する、単純なマルチホスティング構成を示しています。この Web サーバーは 3 つの仮想ホストを持っており、これらの各仮想ホストは独自の JRun サーバーを持っています。



1 つの Web サーバー上に複数の仮想ホストを設定する場合は、特別な手順を実行して JRun をこれらのホストに接続する必要があります。次のセクションでは、一般的な Web サーバーでマルチホスティングを行う方法について説明します。

メモ：これは、マルチホーミングとは異なります。マルチホーミングでは、別の IP アドレスが同じ Web サーバー上の物理的に異なるホストに対応しています。

Apache でのマルチホスティング

Apache 仮想ホストを設定し、各ホストに独自の JRun サーバーを作成するには、Apache の設定ファイル (httpd.conf) の各 VirtualHost ディレクティブ内に JRun 設定ブロックを含めます。

次のリストは、VirtualHost ディレクティブ内の JRun 設定情報の例を示しています。LoadModule ステートメントは、グローバルレベルで 1 回だけ参照できるので、VirtualHost ディレクティブの外部に記述します。

```
LoadModule jrun_module136 "/opt/JRun/connectors/apache/intel-linux/mod_jrun.so"
<VirtualHost 127.0.0.1>
    ServerAdmin webmaster@localhost
    DocumentRoot /usr/local/apache/htdocs/localhost
    ServerName newhost
    ErrorLog logs/newhost-error_log
    CustomLog logs/newhost-access_log common

     # JRun Settings
    <IfModule mod_jrun.c>
       JRunConfig Verbose false
    </IfModule>
</VirtualHost>
```

また、httpd.conf ファイルの編集の他に、各仮想ホストの JRun サーバーを、固有のプロキシポートを使用するように設定する必要があります。

Apache でマルチホスティングを行うには

1 新規に Apache 仮想ホストを作成します。

2 JMC を使用して新規 JRun サーバーを作成します。手順の説明上、新規 JRun サーバーを newserver と呼びます。

各仮想ホストは、VirtualHost ディレクティブ内の ProxyPort プロパティと一致する、固有の JCP ポートを持つ必要があります。JRun ポートの使用方法の詳細については、5 ページの「JRun ポート」を参照してください。

3 local.properties の servlet.services プロパティから Web サービスを削除して、newhost JWS を無効にします。

```
# was:servlet.services=jndi,jdbc,{servlet.webapps},jcp,web
servlet.services=jndi,jdbc,{servlet.webapps},jcp
```

4 Web サーバーがあるマシンで、Web サーバー設定ツールを実行します。[Web サーバー設定の追加] パネルで、上部領域では JRun サーバーを、さらに下部領域では Apache および Apache 設定ディレクトリを選択します。

5 httpd.conf を編集し、適切な VirtualHost ディレクティブ内で JRun 設定ブロックを移動します。必要に応じて、JRun 設定ブロックを複数の VirtualHost ディレクティブと修正した設定にコピーできます。

指定したディレクトリにある Apache コネクタを削除すると、JRun 設定ブロックおよび LoadModule ステートメントのインスタンスがすべて削除されるので、conf ファイルには JRun 設定がなくなります。JRun VirtualHost セクションすべてを削除しない場合は、適切なファイルを手動で削除する必要があります。

IIS でのマルチホスティング

複数の仮想サーバーを設定できるのは Microsoft IIS 4.0 以上だけです。また、Windows プラットホームのすべてに、複数のホストをサポートする IIS のバージョンが提供されているわけではないということに注意してください。

IIS でマルチホスティングを行うには

1. MMC (Microsoft Management Console) の Internet Services Manager を使用し、新規 IIS Web サイト (仮想ホストとも呼ぶ) を作成します。
2. JMC を使用して新規 JRun サーバーを作成します。手順の説明上、新規 JRun サーバーを newserver と呼びます。
3. (オプション) 20 ページの「JWS をオフにする方法」の説明に従って、newserver の JRun Web サーバーを無効にします。
4. JRun サーバーを再起動します。

   Web サーバー設定ツールを実行するには、JRun サーバーを実行する必要があります。
5. Web サーバーがあるマシンで、Web サーバー設定ツールを実行します。[Web サーバー設定の追加] パネルで、上部領域では JRun サーバーを、さらに下部領域では IIS Web サイトを選択します。

   メモ：マルチホスティングをセットアップする際は、JRun フィルタがグローバルにインストールされていないことを確認してください。Internet Services Manager のコントロールパネルを使用して、JRun フィルタがグローバルにインストールされているかどうかを確認します。

Netscape でのマルチホスティング

Netscape では、仮想ホストごとに新規 Web サーバーのインスタンスが必要です。

Netscape でマルチホスティングを行うには

1. 仮想ホストごとに新規 Web サーバーのインスタンスを作成します。
2. JMC を使用して新規 JRun サーバーを作成します。手順の説明上、新規 JRun サーバーを newserver と呼びます。
3. JRun サーバーを起動します。

   Web サーバー設定ツールを実行するには、JRun サーバーを実行する必要があります。
4. (オプション) 20 ページの「JWS をオフにする方法」の説明に従って、newserver の JWS を無効にします。
5. Web サーバーがあるマシンで、Web サーバー設定ツールを実行します。[Web サーバー設定の追加] パネルで、上部領域では JRun サーバーを、さらに下部領域では新規 Web サーバーインスタンスを選択します。
6. Netscape サーバーを再起動します。

コマンドラインインターフェイスの使用

Web サーバー設定ツールは、コマンドラインインターフェイスからでも実行できます。コマンドラインインターフェイスを実行するには、コンソールウィンドウを開き、<JRun のルートディレクトリ >/lib ディレクトリに移動した後、次のコマンドラインシンタックスを使用します。

```
java -jar wsconfig.jar [-options]
```

次の表は、オプションを示しています。

オプション	説明
-ws	次の Web サーバーを指定します。 • IIS • Apache • NES • iPlanet • Zeus
-dir < ディレクトリ >	設定ディレクトリを指定します (Apache conf、NES config、または Zeus config)。
-site < サイト名 >	IIS Web サイト名を指定します。
-host	JRun サーバーのアドレスを指定します。デフォルト値は localhost です。
-server	JRun サーバー名を指定します。デフォルトは default です。
-cluster	JRun クラスタ名を指定します。
-l	コネクタの冗長なロギングを有効にします。
-a	ネイティブの OS メモリ割り当てを有効にします。
-s	コネクタと JRun サーバーの間で SSL を有効にします。
-map	IIS アプリケーションマッピングリスト [ext1][,extn] を指定します。
-service	Apache Windows サービス名を指定します。デフォルトは Apache です。
-v	Web サーバー設定ツールからの冗長出力を有効にします。
-list	すべての設定済み Web サーバーをリストします。
-list -host < サーバーホスト >	指定したホスト上にある JRun サーバーをすべてリストします。
-r	設定を削除します。-ws と、-dir または -site のいずれかが必要です。
-u	すべての設定済みコネクタをアンインストールします。
-h	すべてのパラメータをリストします。

第 3 章
クラスタリング

JRun 4.0 は、Web サーバー、Web サーバーの接続、およびサーバーオブジェクトのクラスタリングをサポートしています。

目 次

JRun でのクラスタリングの概要

実際運用されるアプリケーションには、可用性とレスポンスを高いレベルで保証しなくてはいけません。高可用性と、一定のレスポンスタイムを保証する方法の 1 つは、サーバーの**クラスタ**を定義して、Web とエンタープライズアプリケーションを処理することです。クラスタは、単一のシステムイメージを提供するサーバーのセットです。アプリケーションとサーバーの負荷は、クラスタに属するサーバー間で分散されます。クラスタ全体でのロードバランスには、次の要素が関係します。

- **ロードバランス**　ラウンドロビンなどのあらかじめ定義されたアルゴリズムに従って、クラスタ内のサーバーにリクエストを振り分けます。
- **フェイルオーバー**　サーバーがオフラインになったときに、リクエストを自動的に他のサーバーに転送します。

JRun では、次のタイプのクラスタリングをサポートしています。

タイプ	説明	詳細情報
Web サーバーコネクタのクラスタリング	Web サーバーコネクタは、クラスタ内の JRun サーバー間で HTTP リクエストを管理します。Web サーバーコネクタのクラスタリングによって、ロードバランスとフェイルオーバーが提供されます。	35 ページの「コネクタベースのクラスタリング」
オブジェクトのクラスタリング	JRun では、クラスタに含まれる JRun サーバーにある EJB とサービスへのリクエストに対して、ロードバランスとフェイルオーバーを提供しています。	37 ページの「オブジェクトのクラスタリング」
Web サーバーのクラスタリング	Macromedia ClusterCATS は、クラスタ内の Web サーバー間で HTTP リクエストを管理します。	42 ページの「Web サーバーのクラスタリング」

ロードバランスとフェイルオーバーについてのその他の注意事項は、35 ページの「コネクタベースのクラスタリング」と 37 ページの「オブジェクトのクラスタリング」を参照してください。

JRun サーバーのクラスタ

JRun サーバーのクラスタは、JMC を使用して定義します。JRun サーバーのクラスタの定義については、JMC のオンラインヘルプを参照してください。

メモ：JRun 管理サーバーをクラスタに追加しないでください。

クラスタに属した JRun サーバーを起動すると、クラスタマネージャはマルチキャストを使用して、近くのクラスタされた JRun サーバーすべて (つまり、同じサブネットに属するクラスタされたサーバー) に自分自身を登録します。jrun.xml ファイルのクラスタマネージャの `unicastPeer` 属性に、特定のクラスタメンバーを指定することもできます。ユニキャストピアを指定することで、同じサブネットでは実行されていない JRun サーバーもクラスタできます。

JRun では、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 JCA コネクタを、クラスタ内のすべてのサーバーに自動的にデプロイできます。こうすることで、クラスタ内のすべての JRun サーバーでイメージの整合性が保てます。

クラスタされた環境でのセッションパーシスタンス

JRun のセッションパーシンタンス機能によって、サーバーのシャットダウン時に JSP、サーブレット、およびステートフルセッション bean のユーザーセッションが保持されます。クラスタされていない環境では、JRun が次に開始するときにセッションが復元されます。これに対してクラスタされた環境では、セッション情報をクラスタ内の他のサーバーですぐに利用できます。このセクションでは、JRun のセッションパーシスタンスのオプションと、それらのオプションがクラスタリングに適用される方法を説明します。

メモ：ステートフルセッション bean、またはセッションを使用する Web アプリケーションを実行する場合、JRun はセッション類似性 (Session Affinity) を使用して、適切なサーバーにリクエストを転送します。JRun は、Cookie (デフォルト)、URL パラメータ、またはフォーム変数に基づいてセッションを処理します。

クラスタされた環境を有効に活用するには、セッション管理のオプションと、JRun がセッション管理を行う方法を理解する必要があります。Web アプリケーションの場合、JRun は次のいずれかのメカニズムを使用してセッションを継続します。

- **ファイル (デフォルト)** JRun サーバーのシャットダウン時に、すべてのアクティブなセッションは、Web アプリケーションの WEB-INF/sessions ディレクトリにあるファイルに書き込まれます。

  jrun-web.xml の設定を変えることで、JRun が動作していてもセッションを保存するようにすることもできます。ファイルベースのセッションパーシスタンスでは、クラスタされたセッション管理はサポートされません。

- **複製** JRun は、セッションの設定を自動的に、クラスタ内の 1 台または複数の他のサーバーに複製します。このサーバーは**バディーサーバー**と呼びます。JRun サーバーがシャットダウンすると、Web サーバーコネクタはリクエストを自動的にバディーサーバーに転送します。

  JRun は、ステートフルセッション bean についてはメモリ内での複製のみを使用します。EJB メソッドが完了すると、JRun は自動的にセッションステートを、クラスタ内のバディーサーバーにパッシベートします。

- **JDBC** JRun サーバーがシャットダウンすると、すべてのアクティブなセッションは、jrun-web.xml ファイルの設定に従って、リレーショナルデータベースに書き込まれます。この方法を使用すれば、クラスタされたセッションの管理をサポートできます。

  この方法は、ClusterCATS で Web サーバーのロードバランスをサポートする唯一の方法です。詳細については、42 ページの「Web サーバーのクラスタリング」を参照してください。

- **カスタム** JRun サーバーのシャットダウン時には、すべてのアクティブなセッションは、ユーザーの提供したカスタム構築されたセッションストレージマネージャを使用して、カスタムセッションストアに書き込まれます。この方法を使用すれば、クラスタされたセッションの管理をサポートできます。

jrun-web.xml ファイル内にある次の要素を使用して、ファイル、JDBC、およびカスタムセッションパーシスタンスを設定します。

要素	説明
session-config	persistence-config 要素をラップします。
persistence-config	すべての persistence 要素をラップします。
active	セッションパーシスタンスが有効であるかどうかを示します。true または false を指定します。

要素	説明
persistence-type|persistence-class	次のように、persistence-type または persistence-class を指定します。 • persistence-type は、file または jdbc であることが必要です。 • persistence-class は、カスタムセッションパーシスタンスマネージャの完全修飾クラス名を指定します。 persistence-type と persistence-class の両方を指定することはできません。
persistence-syschronized	セッションパーシスタンスの書き込みを同期するかどうかを指定します。true または false を指定します。デフォルトは true です。
class-change-option	クラスローダの変更時に JRun が行うアクションを次のように指定します。 • Reload　すべてのセッションをリロードします。 • Drop　すべてのセッションをドロップします。 • Ignore　現在ロードされているセッションを使用します。
session-swapping	セッションスワップを有効にするかどうかを指定します。true または false を指定します。セッションスワップが有効な場合、JRun は決まった数のセッションだけをメモリに保持し、他の (古い順に) セッションをパーシスタンスストア (ファイル、JDBC、またはカスタムクラス) に保管します。
session-swap-interval	JRun がセッションプールサイズを監視する間隔と、最初に使用されたセッションをパーシスタンスストアに保管する間隔を指定します。デフォルトは 5 秒です。
session-max-resident	常駐セッションの最大数を指定します。デフォルトは 9999999 です。0 を指定すると、すべてのセッションがパーシスタンスストアに書き込まれます。 ClusterCAT を使用しているときには、クラスタ内のすべてのサーバーに JDBC パーシスタンスタイプを使用し、この値を 0 に設定する必要があります。
init-param	セッションパーシスタンスメカニズムの初期化パラメータを指定します。次のサブ要素を使用します。 • param-name • param-value

次のセクションでは、これらの要素の例について説明します。

ファイルベースのセッションパーシスタンスの使用

デフォルトはファイルベースのセッションパーシスタンスです。ほとんどの場合は設定を変更する必要はありません。ただし、session-swapping や session-swap-interval などの設定は変更できます。

また、ファイルベースのセッションパーシスタンスでは、次の init-param を使用します。

- **path** JRun によってセッション情報が書き込まれるディレクトリのパス。デフォルトは、WEB-INF/sessions です。

次の例は、jrun-web.xml ファイル内にあるファイルベースのセッションパーシスタンスを示すものです。

```
...
<session-config>
  <persistence-config>
    <active>true</active>
    <persistence-type>file</persistence-type>
    <persistence-synchronized>true</persistence-synchronized>
    <class-change-option>reload</class-change-option>
    <session-swapping>true</session-swapping>
    <session-swap-interval>5</session-swap-interval>
    <session-max-resident>500</session-max-resident>
  </persistence-config>
...
```

ファイルベースのセッションパーシスタンスを使用するときには、path 属性を init-param として渡すことができます。この要素では、JRun がセッション情報を書き込むディレクトリが指定されます。

これらの設定の詳細については、JRun オンラインドキュメントのホームページからアクセスできる、オンラインディスクリプタについてのドキュメントを参照してください。

JDBC セッション管理の使用

JDBC ベースのセッション管理を使用すると、複数の JRun サーバーでパーシスタンスストアを共有できます。

JDBC ベースのセッション管理を使用するには

1 セッション ID 用の varchar 列 (最低 100 文字) と、セッションデータを保持する BLOB 列を持つデータベーステーブルを定義します。

2 クラスタ内のすべての JRun サーバーで、セッションテーブルを含むデータベースを参照する JRun データソースを定義します。

3 Web アプリケーションの jrun-web.xml ファイルを編集し、persistence-type を JDBC に変更し、次のアイテムについて init-param 要素を指定します。

   - **JDBCConnection (必須)** セッションテーブルを含むデータベースの JRun データソースの JNDI 名です。
   - **JDBCSessionTable** セッション情報を含むテーブル名です。デフォルトの名前は sessions です。
   - **JDBCSessionIDPrefix** すべての JRun セッション ID がクラスタ内で固有であることを保証する接頭辞です。デフォルトは {jrun.server.name}-{application name}- です。

- JDBCSessionIDColumn　セッション ID を含む列の名前です。デフォルトの ID 列名は id です。
- JDBCSessionDataColumn　セッション情報を含む列の名前です。デフォルトのデータ列名は data です。

次の例は、jrun-web.xml ファイル内の JDBC ベースのセッションパーシスタンスを示すものです。

```
...
<session-config>
  <persistence-config>
    <active>true</active>
    <persistence-type>jdbc</persistence-type>
    <persistence-synchronized>true</persistence-synchronized>
    <class-change-option>reload</class-change-option>
    <session-swapping>true</session-swapping>
    <session-swap-interval>5</session-swap-interval>
    <session-max-resident>500</session-max-resident>
    <init-param>
      <param-name>JDBCConnection</param-name>
      <param-value>samples</param-value>
    </init-param>
    <init-param>
      <param-name>JDBCSessionTable</param-name>
      <param-value>sessions</param-value>
    </init-param>
    <init-param>
      <param-name>JDBCSessionIDColumn</param-name>
      <param-value>sessionid</param-value>
    </init-param>
    <init-param>
      <param-name>JDBCSessionDataColumn</param-name>
      <param-value>sessiondata</param-value>
    </init-param>
  </persistence-config>
...
```

これらの設定の詳細については、JRun オンラインドキュメントのホームページからアクセスできる、オンラインディスクリプタについてのドキュメントを参照してください。

セッションレプリケーションの使用

Web アプリケーションでセッションレプリケーションを使用するには、メモリ内のセッション管理を使用するすべての Web アプリケーションについて、jrun-web.xml ファイルの session-config 要素を更新する必要があります。クラスタ内のすべてのサーバーにセッション情報をレプリケーションするには、replication-config 要素を次のように指定します。

```
...
<replication-config>
  <active>true</active>
  <buddy-list>*</buddy-list>
</replication-config>
...
```

ヒント：この変更を JRun サーバー内のすべての Web アプリケーションに適用するには、SERVER-INF/default-web.xml ファイルで replication-config 設定を変更します。

クラスタ内にある 1 台または複数の特定のサーバーを複製するには、replication-config 要素を次のように指定します。

```
...
replication-config>
  <active>true</active>
  <buddy-list>jrunserver2</buddy-list>
  <buddy-list>jrunserver3</buddy-list>
  <buddy-list>jrunserver4</buddy-list>
</replication-config>
...
```

ヒント：* を指定すると、クラスタ内のすべてのサーバーを複製できます。

これらの変更を加えたら、すべての JRun サーバーを再起動します。

カスタムセッションストレージマネージャの実装

JRun 3.x ではセッション情報をデータベースに保持できます。JRun 4 にはこの機能に代わるインターフェイスがあります。このインターフェイスでは、サイト特有のセッションストアにセッション情報を保持できます。このセッションストアはデータベースやその他のデスティネーションです。

1 jrun.servlet.session.SessionStorage を実装する Java ファイルを作成します。

```
...
public class DBMSSessionStorage implements SessionStorage
...
```

2 カスタムのセッションストレージマネージャに必要なパッケージをインポートします。これには次の必須クラスが含まれます。

- jrun.servlet.session.SessionStorage
- java.io.InputStream
- java.util.List
- java.util.Properties

3 init メソッドに渡される init-param プロパティを含む変数を宣言します。

```
String dataSource;
String table;
String idColumn;
String dataColumn;
```

4 init メソッドで変数を初期化します。たとえば、DBMS ベースのカスタムセッションストレージマネージャでは、次の初期化パラメータを使用できます。

```
public void init(Properties props) throws Exception {
  dataSource = props.getProperty("dataSource");
  if (dataSource == null) {
    dataSource = "defaultDataSource";
  }
```

```
    table = props.getProperty("table");
    if (table == null) {
      table = "sessions";
    }

    idColumn = props.getProperty("idColumn");
    if (idColumn == null) {
      idColumn = "sessionid";
    }

    dataColumn = props.getProperty("dataColumn");
    if (dataColumn == null) {
      dataColumn = "sessioninfo";
    }
  } // init を終了します。
```

5 セッションをカスタムのパーシスタンスメカニズムに保管する store メソッドをコーディングします。store メソッドには次のシグネチャがあります。

```
public void store(String id, InputStream in, int len) throws Exception
```

6 カスタムのパーシスタンスメカニズムからセッションにアクセスする、retrieve メソッドをコーディングします。retrieve メソッドには次のシグネチャがあります。

```
public InputStream retrieve(String id) throws Exception
```

7 カスタムのパーシスタンスメカニズムからセッションを削除する、remove メソッドをコーディングします。remove メソッドには次のシグネチャがあります。

```
public void remove(String id) throws Exception
```

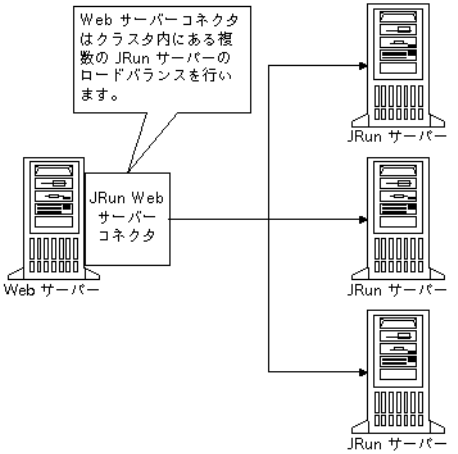
8 サービスのシャットダウン時に必要なクリーンアップを実行する、destroy メソッドをコーディングします。destroy メソッドには次のシグネチャがあります。

```
public void destroy()
```

完全なサンプルプログラムは、Sapmles サーバーの SERVER-INF/lib/jrun/samples/session ディレクトリを参照してください。

コネクタベースのクラスタリング

JRun では、Web サーバーコネクタの後方で JRun サーバーをクラスタすることができます。次の図に示すように、Web サーバーコネクタは、クラスタ内の JRun サーバー間でロードバランスを実行し、フェイルオーバーを管理します。



コネクタベースのクラスタリング

JRun の Web サーバーコネクタは、クラスタ内の JRun サーバーに対してロードバランスとフェイルオーバー機能を提供します。

Web サーバーの設定ファイル (たとえば、Apache では httpd.conf、IIS では jrun.ini) には、serverstore という名前のエントリがあります。このエントリは、jrunserver.store という名前のファイルを指しています。jrunserver.store ファイルには、JRun サーバークラスタ内にあるサーバーに関する情報が含まれています。

Web サーバー設定ツールを実行すると、JRun クラスタが自動的に検出され表示されます。クラスタを選択すると、Web サーバー設定ツールはクラスタメンバーの 1 つをブートストラップサーバーとして選択し、そのサーバーの IP アドレスと Web サーバーコネクタのポート (JCP ポートとも呼ばれます) を、設定ファイルの **bootstrap** プロパティに保存します。また、Web サーバー設定ツールはクラスタ内のすべての JRun サーバーについて、jrun.xml 内 **ProxyService** の **deactivated** 属性を false に設定します。

メモ：Web サーバー設定ツールを実行するには、クラスタ内のすべての JRun サーバーが実行中であることが必要です。実行されていないサーバーがあると、Web サーバー設定ツールは deactivated 属性を変更できません。

Web サーバーコネクタを初めて開始するとき、**bootstrap** プロパティを使用して、JRun サーバーがクラスタ内に存在するかどうかを検証し、クラスタ内の各 JRun サーバーの IP アドレスと JCP ポートを <IP アドレス >:<JCP ポート > の形式で保存します。たとえば、**bootstrap** プロパティが 1.64.22.137:51000 を指していて、JRun サーバーがクラスタ内に存在する場合、コネクタは次のような <IP アドレス >:<JCP ポート設定 > を含む **proxyservers** プロパティを持つ jrunserver.store ファイルを作成します。

```
proxyservers=1.64.22.137:51000;1.64.22.143:51000;1.64.22.126:51000;
        1.64.22.140:51000
```

この例では、すべての JRun サーバーがデフォルトの JCP ポートを使用していることを前提にしています。

jrun.xml の変更

Web サーバー設定ツールを実行する他に、クラスタされた各 JRun サーバーの jrun.xml ファイルの ProxyService に、次の変更を加える必要があります。

- **JNDI バインディング** JRun サーバーがクラスタの変更を Web サーバーコネクタに通知できるようにするには、**ProxyService** の **bindToJNDI** 属性を true に設定します。
- **ロードバランスアルゴリズム** Web サーバーコネクタがリクエストをクラスタ内で分配する方法を指定するには、**LoadBalancingAlgorithm** 属性で次のオプションを選択します。
  - **ラウンドロビン (デフォルト)** Web サーバーコネクタは、リスト内で次にある JRun サーバーに各リクエストを送信します。このオプションを使用するには、**ROUNDROBIN** を指定します。
  - **加重ラウンドロビン** Web サーバーコネクタは、**serverweight** 属性で指定されているより多めのリクエストを特定の JRun サーバーに続けて送信します。このオプションを使用するには、**ROUNDROBIN_WEIGHTED** を指定します。
  - **加重ランダム** Web サーバーコネクタは、**serverweight** 属性で指定されているより多めのリクエストを JRun サーバーにランダムに送信します。このオプションを使用するには、**RANDOM_WEIGHTED** を指定します。
- **サーバーの加重** JRun サーバーの相対的な加重を制御するには、**ServerWeight** 属性を指定します。加重ラウンドロビン、または加重ランダム分配によるロードバランスアルゴリズムには、この属性を使用します。デフォルトの加重は 1 です。1 よりも大きい加重を持つ JRun サーバーは、他よりも多くのリクエストが受信できます。このオプションは、クラスタ内の JRun サーバーの一部に、他のサーバーよりも多くのリソースがある場合に有効です。
- **セッション管理** Web サーバーコネクタが、リクエストに既存のセッションがあるかどうかを検出し、そのリクエストを自動的に元の JRun サーバーに転送するかどうかを指定します。**StickySessions** 属性を指定します。true または false を指定します。デフォルトは false です。Web アプリケーションでセッション管理を使用する場合、この属性を true に設定します。

次の例は、デフォルトのロードバランスアルゴリズムと、StickySessions を使用した jrun.xml スニペットです。

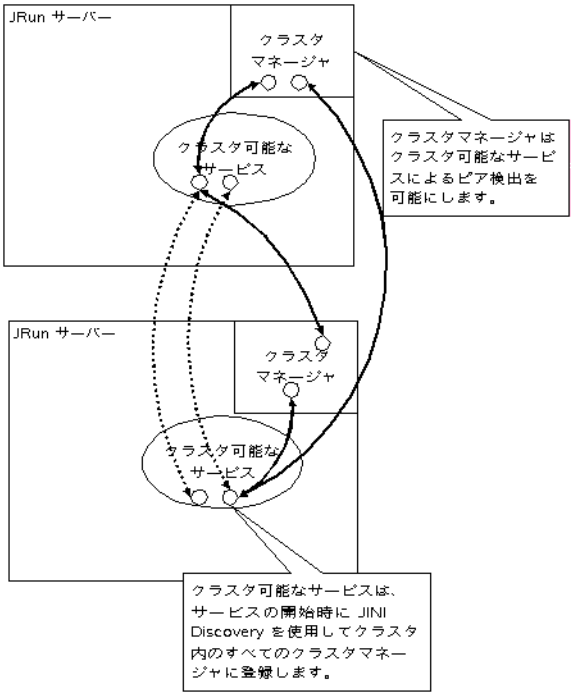
```
<service class="jrun.servlet.jrpp.JRunProxyService" name="ProxyService">
  <attribute name="bindToJNDI">true</attribute>
  <attribute name="port">51010</attribute>
  <attribute name="deactivated">false</attribute>
  <attribute name="StickySessions">true</attribute>
</service>
```

次の例は、加重ラウンドロビンによるロードバランスアルゴリズムと、StickySessions を使用した jrun.xml スニペットです。

```
<service class="jrun.servlet.jrpp.JRunProxyService" name="ProxyService">
  <attribute name="bindToJNDI">true</attribute>
  <attribute name="port">51010</attribute>
  <attribute name="deactivated">false</attribute>
  <attribute name="LoadBalancingAlgorithm">ROUNDROBIN_WEIGHTED</attribute>
  <attribute name="ServerWeight">3</attribute>
  <attribute name="StickySessions">true</attribute>
</service>
```

オブジェクトのクラスタリング

JRun のクラスタリングアーキテクチャは、クラスタリングが有効になっている JRun サーバーで実行されるクラスタマネージャによって制御されます。サーバーの開始時に、クラスタ可能なサービスは、JINI ルックアップサービスを使用して、クラスタ内の各クラスタマネージャに自分自身を登録します。これによって、各クラスタマネージャはクラスタ内にあるすべてのクラスタ可能なサービスを検出することができます。同時にクラスタ可能な各サービスは、クラスタ内でピアを検出できます。次の図は、クラスタマネージャとクラスタ可能なサービスを示したものです。



JRun は次のクラスタ可能なサービスを提供しています。

サービス	説明
JNDI コンテキストマネージャ	クライアントが JNDI Lookup を使用してクラスタ可能なオブジェクトをリクエストすると、JRun はクラスタ内にあるすべてのサーバーの情報を持つコンテキストマネージャを返します。JRun サーバーがクラスタに参加していると自動的にフェイルオーバーが有効になり、これらのリソースはいつでも利用できます。また、JRun によってロードバランスが行われることも示しています。
RMI ブローカー	JRun でのすべてのリモートアクセスに対する RMI トランスポート実装です。
セッションレプリケーション	Web アプリケーションにセッションレプリケーションを使用する場合、セッションレプリケーションサービスはクラスタリングを使用して、クラスタ内のバディーサーバーとステートを同期します。詳細については、32 ページの「セッションレプリケーションの使用」を参照してください。

メモ：クラスタ名は固有であることが必要です。また、クラスタ内の JRun サーバーにも固有の名前が必要です。たとえば、複数の開発者が同じサブネット上でデフォルトのクラスタ名を使用してクラスタを定義し、それらのクラスタにデフォルトの JRun サーバーが含まれている場合、問題 (JNDI へのランダムアクセス) が発生します。

JRun のクラスタリングには次の制限があります。

- サーバーは、1 つのクラスタにのみ属することができます。
- JMC をクラスタ全体にデプロイすることはできません。JMC は、管理サーバーでのみ実行します。
- EJB のクラスタリングのトランザクションサポートを有効にするには、クラスタ内のすべてのサーバーの DefaultDomain サービスについて、`clusterEnabled` 属性を true に設定する必要があります。
- JRun はクラスタ全体にわたってトランザクションを実行することができますが、EJB フェイルオーバーに対する、クラスタ全体にわたるトランザクションリカバリはサポートされていません。

JMC 内にクラスタを定義すると、オブジェクトのクラスタリングはほとんどのユーザーに対して行われます。ただし、特定のタイプの開発者は、クラスタリングについて次の点を考慮しなければなりません。

- EJB
- JMS
- サービスの開発と JRun サービスのダイレクト呼び出し

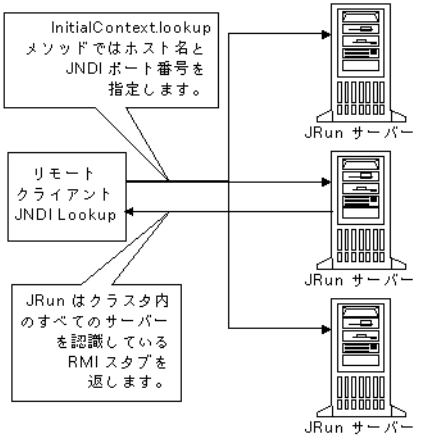
次のセクションでは、これらのトピックについて説明します。

EJB とオブジェクトのクラスタリングの使用

クライアントが JNDI を使用して EJB を参照すると、JRun はクラスタ内のすべてのサーバーの情報を持つコンテキストマネージャを返します。JRun サーバーがクラスタに参加していると自動的にフェイルオーバーが有効になり、これらのリソースはいつでも利用できます。また、次に示すように、異なるロードバランスアルゴリズムを使用して、JRun がクラスタ内でロードバランスを自動的に行います。

- **ステートレスセッション bean**　スティッキーラウンドロビン
- **ステートフルセッション bean のホーム**　スティッキーラウンドロビン
- **ステートフルセッション bean の EJBObject**　バディーアルゴリズム
- **エンティティ bean のホーム**　スティッキーラウンドロビン
- **エンティティ bean EJBObject**　エンティティ bean EJBObject に対するクラスタリングはサポートされていません。

次の図は、EJB のクラスタリングを示すものです。



EJB メソッドの呼び出し中に JRun サーバーがオフラインになった場合、クライアントスタブは自動的にリクエストをクラスタ内の別のサーバーに再転送します。さらに、オブジェクトのクラスタリングによってステートが自動的にクラスタ内のバディーサーバーにパッシベートされ、ステートフルセッション bean が複製されます。

JMC でクラスタを作成する際、EJB のクラスタリングが自動的に有効になります。EJB の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` 要素および `cluster-object` 要素に false を指定します。ローカル EJB はクラスタできません。

クラスタ定義の詳細については、JMC のオンラインヘルプを参照してください。

クラスタされた環境での JMS の使用

JRun JMS は、クラスタされた JRun サーバーの JMS サービスのディスクリプタファイルを変更することによって、クラスタリングをサポートします。

JMS のクラスタリングを有効にするには

1 JMS のデスティネーションと接続ファクトリ (jrun-resources.xml で定義されているもの) ファイルが、クラスタ内のすべての JRun サーバーで同じであることを確認します。

2 クラスタ内の JRun サーバーを 1 台選択し、JMSService を実行します (ターゲットサーバーと呼びます)。クラスタ内の他のすべてのサーバーで、次の手順を実行します。

- 1 台のサーバーを除くすべてのサーバー上の `JRunMQAdapter` サービスをコメントアウトします。
- `JMSServiceWrapper` の `AdapterType` を remote に変更します。
- `JMSServiceWrapper` に、`AdapterServerName` 属性を追加します。`JRunMQAdapter` サービスが有効になっているサーバー名にそれを設定します。
- `JMSServiceWrapper` の `JMSUrl` 属性を、<ターゲットサーバーのサーバー名>:<ターゲットサーバーの JNDI ポート > に変更します。この設定は、ターゲットサーバーの SERVER-INF/jndi.properties ファイルにある、`java.naming.provider.url` プロパティを参照して指定します。

- JMSServiceWrapper の JMSContextFactoryName を、jrun.naming.JRunContextFactory に変更します。この設定は、ターゲットサーバーの SERVER-INF/jndi.properties ファイルにある java.naming.factory.initial プロパティと一致します。

3 クラスタ内にあるすべてのサーバーの SERVER-INF/jrun-resources.xml ファイルを参照し、JMS デスティネーションと JMC 接続ファクトリが、ターゲットサーバーのものと一致していることを確認します。

4 ターゲットサーバーを起動します。

5 クラスタ内の他のサーバーを起動します。

次の例は、ターゲットサーバー以外のすべての JRun サーバーについて JMS のクラスタリングを有効にするために、jrun.xml ファイルに加える変更を示すものです。

```
...
<service class="jrun.jms.JRunJMS" name="JRunJMS">
...
<!-- ========================================================================= -->
<!-- このサービスは、JMS プロバイダを開始します。(JRun ビルトイン ) -->
<!-- ========================================================================= -->
<!--
<service class="jrun.jms.adapter.JRunMQAdapter" name="JMSAdapter">
<attribute name="ConfigFileName">jrun-jms.xml</attribute>
<attribute name="bindToJNDI">true</attribute>
</service>
-->
<!-- ========================================================================= -->
<!--  このサービスは、J2EE クライアントに JMS アクセスを提供します (JRun ビルトイン )。  -->
<!-- ========================================================================= -->
<service class="jrun.jms.wrapper.JRunMQServiceWrapper" name="JMSServiceWrapper">
<attribute name="bindToJNDI">true</attribute>
<attribute name="DefaultQCFName">QueueConnectionFactory</attribute>
<attribute name="DefaultTCFName">TopicConnectionFactory</attribute>
<attribute name="DefaultTransport">RMI</attribute>
<attribute name="JMSUrl">localhost:2918</attribute>
<attribute name="JMSContextFactoryName">jrun.naming.JRunContextFactory
        </attribute>
<!-- AdapterType は、ラッパーがリモートかローカル
     のどちらのアダプタを使用するかを示します。 -->
<attribute name="AdapterType">remote</attribute>
<attribute name="AdapterServerName">fresca</attribute>
</service>
```

JRun サービスとオブジェクトのクラスタリングの使用

JNDI はクラスタ可能なサービスなので、jrun-xml.file 定義で **bindToJNDI** 属性が true に設定されているサービスもすべてクラスタできます。たとえば、クラスタ内の 2 台のサーバーが両方とも、compass という名前の JDBC データソースを定義している場合を考えます。クライアントアプリケーションが、**InitialContext.lookup** によってこのデータソースにアクセスすると、戻り値の javax.sql.DataSource はクラスタ内のいずれの JRun サーバーからも返される可能性があります。

JRun サーバーにデプロイされたリソースのクラスタリングに加えて、サービスもクラスタできます。JRun は、サービスへのリファレンスを、JNDI ツリー内の複数の場所に保存します。クラスタされたエントリは、**jrun:service/**< サービス名 > にあります。ローカル JRun サーバーのエントリは、**jrun:service/**< サーバー名 >/< サービス名 > にあります。JRun での JNDI 使用の詳細については、第 8 章、93 ページの「JNDI に関する考慮」を参照してください。

次のサンプルコードは、JRun の **LoggerService** にアクセスし、クラスタ内にある任意の JRun サーバーへのリファレンスを取得するものです。

```
...
// この例を実行するには、LoggerService（jrun.xml ファイル）内で bindToJNDI 属性を
// true に設定してください。
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.Context;
import jrunx.logger.LoggerService;
...
  try {
    Properties p = new Properties();
    p.put(Context.INITIAL_CONTEXT_FACTORY,
        "jrun.naming.JRunContextFactory");
    p.put(Context.PROVIDER_URL, "localhost:2918");
    InitialContext ctx = new InitialContext(p);
    LoggerService ls = (LoggerService)
        ctx.lookup("jrun:service/LoggerService");
    ls.logInfo("Message logged using LoggerService");
  } // try の終了
  catch (Exception e) {
    System.out.println(e);
    throw new Exception(e);
  }
...
```

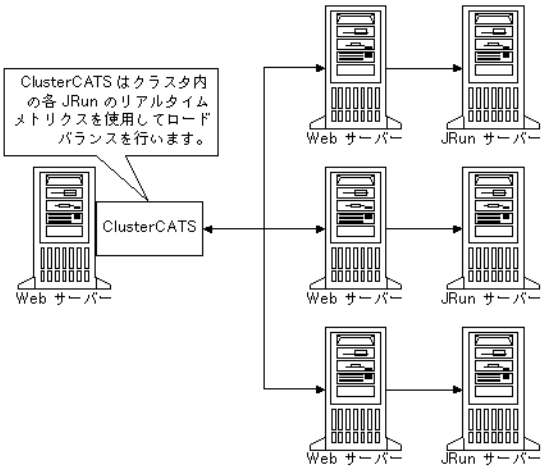
JNDI Lookup が特定の JRun サーバーにリファレンスを返すようにするためには、次の例のように、**lookup** メソッドに渡される文字列にサーバー名を含めます。

```
...
    LoggerService ls = (LoggerService)
        ctx.lookup("jrun:service/samples/LoggerService");
...
```

メモ：JNDI からサービスにアクセスするには、そのサービスの bindToJNDI 属性が true に設定されていなければなりません。詳細については、100 ページの「すべてのサービスに共通の属性」を参照してください。

Web サーバーのクラスタリング

Web サーバーのクラスタリングは、ClusterCATS によって行われます。ClusterCATS は Macromedia のクラスタリングテクノロジーで、ソフトウェアベースのロードバランスとフェイルオーバーサービスを提供し、Web サーバーと、それらの Web サーバーを使用する Web アプリケーションの高可用性を保証するものです。次の図は、サーバー間の関係を示しています。



Web サーバーのクラスタリングの詳細については、『ClusterCATS ユーザーガイド』を参照してください。

第 4 章
ロギングメカニズム

この章では、JRun のロギングメカニズムについて説明します。

目 次

概要

アプリケーションの起動時および実行時には、サーバーのステータス、エラーの状態、パフォーマンス統計などのログメッセージが JRun によって生成されます。ロギングメッセージは JSP、サーブレット、EJB によっても生成されます。これらのメッセージは JRun ログファイルにあります。

さまざまなタイプのメッセージを処理したり、それらのメッセージをさまざまな出力先に記録できるように JRun を設定できます。また、カスタマイズされたフィルタリングや書き込みを実行する固有のログコンポーネントも作成できます。

一般的に、JRun のロギングメカニズムにはほとんどオーバヘッドがありません。したがって、アプリケーションのパフォーマンスに与える影響は最小限に抑えられています。

メモ：デバッグ情報の生成はアプリケーションのパフォーマンスに影響を与えます。このオプションはアプリケーションのデバッグ時のみ使用してください。

この章では、デフォルトのログの設定、その設定の変更手順、その変更方法を示す例など、ロギングメカニズムについて説明します。

メッセージのタイプ

JRun によって次のタイプのメッセージが生成されます。

- 情報メッセージ。起動時に生成されるメッセージおよびメソッドタイミングメッセージがあります。
- 警告メッセージ
- エラーメッセージ
- デバッグ情報
- システムのメトリクスと、Web サーバーと JRun 間の接続についてのメトリクス (第 7 章、89 ページの「接続の監視」を参照)

1 つまたは複数の種類のメッセージを記録するようにロギングメカニズムを設定できます。JMC を使用すると、記録するメッセージのタイプを変更できます。

メッセージの出力先

ロギングメカニズムによって、メッセージを次の出力先に送信できます。

- ログファイル
- コンソール画面

ロギングコンポーネント

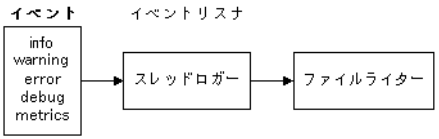
JRun には次のロギングコンポーネントが含まれています。

- **スレッドロガー**　このロガーをメインのキューとして使用します。ログメッセージを作成するコンポーネント (JRun、サーブレット、JSP、EJB) から、メモリ内のキューにメッセージが送信され、制御はすぐにコンポーネントに戻されます。そして、バックグラウンドスレッドによってキューが処理され、ログメッセージは適切なログライターに送信されます。このデザインは、アプリケーション内でのパフォーマンスの管理に役立ちます。
- **ファイルライター**　ロギングイベントを受け取り、そのイベントをファイルに書き込みます。デフォルトでは、スレッドロガーはメッセージからファイルライターへと送信されます。
- **コンソールライター**　ロギングイベントを、画面などのシステムの標準出力デバイスに送信します。

デフォルトのロギング設定

デフォルトのロギング設定を使用していて、ロギングイベントがアプリケーション内で発生した場合、JRun によってイベントがメモリ内のキューに転送され、制御がすみやかに返されます。次に、バックグラウンドスレッドにより、キューからイベントが読み込まれ、適切な出力先に各イベントが転送されます。

次の図は、デフォルト設定のロギングメカニズムです。



次のプロセスは、このデフォルト設定内で実行されます。

1　スレッドロガーがイベントリスナとして機能します。ログイベントが発生すると、そのイベントが記述されてたメッセージがスレッドロガーのキューに書き込まれます。

2　スレッドロガーのバックグラウンドスレッドにより、キューからイベントが読み込まれ、ファイルライターに送信されます。

3　ファイルライターによって、<JRun のルートディレクトリ >/logs/<JRun サーバー > -event.log という名前のファイルにイベントが書き込まれます。

4　ログファイルの最大サイズやローテーションファイルの数を調整するには JMC を使用します。ログファイルのサイズが指定された最大サイズに到達すると、JRun によって現在のログファイルへの書き込みは停止されます。そのログファイルの名前の末尾には続き番号が追加されて名前が変更され、新しいログファイルが作成されます。すべての新しいイベントは、新しいログファイルに書き込まれます。

ロギング設定は JMC で制御し、さらにカスタマイズが必要な場合は jrun.xml ファイルで行います。詳細については、105 ページの「LoggerService」および JMC のオンラインヘルプを参照してください。

ロギング設定のカスタマイズ

JRun では、次にようにロギング設定をカスタマイズおよび拡張できます。

- レベル特有のログファイルの使用
- コンソールライターの使用
- カスタマイズしたロギングメカニズムの定義

レベル特有のログファイルの使用

ファイルライターは、メッセージレベルに基づいて複数のファイルに書き込むことができます。これによって、簡単にメッセージをタイプごとに隔離できます。複数ファイルのロギングを有効にするには、次に示す例のように {log.level} を FileLogEventHandler サービスの filename 属性に追加します。

```
<service class="jrunx.logger.FileLogEventHandler" name="FileLogEventHandler">
  <attribute name="filename">{jrun.rootdir}/logs/{jrun.server.name}-{log.level}
        -event.log</attribute>
  <attribute name="rotationSize">200000</attribute>
  <attribute name="rotationFiles">3</attribute>
  <attribute name="heading"># JRun によって {date MM/dd HH:mm:ss} に作成
        </attribute>
  <attribute name="closeDelay">5000</attribute>
  <attribute name="deleteOnExit">false</attribute>
</service>
```

たとえば、レベル特有のログファイルを使用している場合、デフォルトの JRun サーバーは次のログファイルを書き込みます。

- default-info.log
- default-error.log
- default-warning.log
- default-debug.log
- default-metrics.log

コンソールライターの使用

ログ出力をスクリーンに表示すると、プロジェクトの開発やテスト段階の際に役立ちます。JRun のデバッグメッセージの表示のほかに、ログメソッドをコードに追加し、コンソールにリアルタイムに表示できます。

このセクションでは、コンソールログライターを使用してログメッセージをコンソールウィンドウに出力する方法を説明します。Windows と Unix では手順が異なります。

Windows

コンソールログライターを有効にするには、次の手順を実行します。

1 JRun サーバーを停止します。

コンソールログライターを使用するには、JRun をアプリケーションとして実行する必要があります。コンソールログライターは、JRun を Windows サービスとして実行すると機能しません。

2 コンソールログライターを有効にする JRun サーバーの jrun.xml を開きます。

3 `ConsoleLogEventHandler` サービス内にある `ThreadedLogEventHandler` サービスの行のコメントマークを削除します。

4 jrun.xml ファイルを保存します。

5 JRun サーバーをアプリケーションとして再起動します。

コンソールにログエントリが表示されます。

Unix/Linux

コンソールログライターを有効にするには、次の手順を実行します。

1 JRun サーバーを停止します。

2 コンソールログライターを有効にする JRun サーバーの jrun.xml を開きます。

3 `ThreadedLogEventHandler` サービス内にある `ConsoleLogEventHandler` サービスの行のコメントマークを削除します。

4 jrun.xml ファイルを保存します。

5 JRun サーバーを再起動します。

コンソールにログエントリが表示されます。

カスタマイズしたロギングメカニズムの定義

JRun ロギングアーキテクチャでは、ログイベントハンドラをカスタマイズすることができます。たとえば、サイト特有の形式で、またはデータベースにメッセージを記述するログイベントハンドラを作成できます。また、この機能を利用して、log4j や JDK 1.4 の `java.util.logging` パッケージなどの代替ロギング API でログメッセージを記述することができます。

ログイベントハンドラをカスタマイズするには、次の手順を実行する必要があります。

- MBean インターフェイスのコードを記述します。
- ログイベントハンドラのコードを記述します。
- jrun.xml ファイルで `service` 要素を定義します。

MBean インターフェイスのコードの記述

JRun JMX ベースのアーキテクチャの一部として必要な MBean インターフェイスは、jrun.xml ファイルにあるプロパティと相関関係にあるメソッドを定義します。jrun.xml ファイル内にあるこれらのプロパティに値を指定する際、JRun はサーバーの起動時に自動的にこれらの値を初期化します。ログイベントハンドラである MBean インターフェイスは、`LogEventHandlerMBean` インターフェイスを拡張しており、また次の要素を含んでいる必要があります。

- `jrunx.logger.*` の `import` ステートメント
- `jrunx.kernal.*` の `import` ステートメント
- jrun.xml ファイルにある属性要素を介してログイベントハンドラに渡されるすべてのプロパティに対する set メソッドおよび get メソッドの定義

例

次のコードは、XML ログイベントハンドラの MBean インターフェイスを定義します。

```
import jrunx.logger.*;
import jrunx.kernel.*;

/**
 * XML ログイベントハンドラの MBean インターフェイス
 *
 * 制作 Macromedia, Inc.
 * 日付 2001 年 12 月 5 日
 */
public interface XMLLogEventHandlerMBean
   extends LogEventHandlerMBean {

   /**
    * XML ロギングに使用されるファイルの名前を設定します。
    * パラメータファイル名 filename
    */
   public void setFilename(String filename);

   /**
    * XML ロギングに使用されるファイルの名前を取得します。
    * 戻り値 filename
    */
   public String getFilename();
}
```

ログイベントハンドラのコーディング

ログイベントハンドラとは、次のアイテムが含まれる Java クラスです。

- `jrunx.logger.*` の `import` ステートメント (コンパイルするには、<JRun のルートディレクトリ >/lib/jrun.jar をクラスパスに追加する必要があります。)
- `jrunx.logger.LogEventHandler` を拡張し、MBean インターフェイスを実装するクラス定義
- ログイベントハンドラのプロパティを定義するインスタンス変数
- ログイベントハンドラのプロパティの get メソッドおよび set メソッド
- 次に示すように定義される `logEvent` メソッド

  `public synchronized boolean logEvent(LogEvent event)`

  JRun はログリクエストが発せられると `logEvent` メソッドを呼び出します。このメソッドには、主なロギングロジックが含まれます。
- (オプション) ライフサイクル特有の処理を実行する `init`、`start`、`stop`、および `destroy` メソッドを上書きします。

JRun では、次の順序でイベントハンドラメソッドが呼び出されます。

- `set` メソッド (jrun.xml で定義されている属性を使用)
- `init`
- `start`
- `logEvent` (JRun の実行中は繰り返し呼び出されます。)

- stop
- destroy

例

次のコードは、XML に類似した簡単な形式でログメッセージを書き込むログイベントハンドラを定義します。

```
import java.io.*;
import java.util.*;
import java.text.*;
import jrunx.logger.*;

/**
 * XML 形式でイベントをファイルにログします。
 * これは、このクラスが logEvent メソッドを実装する
 * 必要がある LogEventHandler を拡張します。
 * getFilename および setFilename メソッドを定義する
 *  XMLLogEventHandlerMBean を実装します。
 */
public class XMLLogEventHandler extends LogEventHandler
   implements XMLLogEventHandlerMBean {
    protected String filename;
    protected SimpleDateFormat dateFormat = new SimpleDateFormat
         ("MM/HH hh:mm:ss");
    protected String defaultFormat = null;

    // サーバーの起動時、JRun はライターを初期化します。
    public void init(Properties props) {
        dateFormat = new SimpleDateFormat("MM/HH hh:mm:ss");
    }

   /**
    * XML ロギングに使用されるファイルの名前を設定します。
    * パラメータファイル名  filename
    */
   public void setFilename(String filename) {
        if (filename == null) {
            this.filename = "xml.log";
        }
        else {
         this.filename = filename;
        }
   }

  /**
   * XML ロギングに使用されるファイルの名前を取得します。
   * パラメータファイル名  filename
   */
   public String getFilename() {
      return filename;
   }
```

```
    // ログアクションがリクエストされると、JRun は logEvent メソッドを呼び出します。
    public synchronized boolean logEvent(LogEvent event) {
        PrintWriter out = null;

        try {
            FileOutputStream fos = new FileOutputStream(filename, true);
            out = new PrintWriter(fos, true);

            // 部分的に取得します。
            Properties p = event.getVariables(defaultFormat);
            String level = (String)p.get("log.level");
            String msg = (String)p.get("log.message");
            String exception = (String)p.get("log.exception");
            Date date = new Date();

            String formattedDate;
            synchronized(dateFormat) {
                formattedDate = dateFormat.format(date);
            }

            // 未完成の形式
            out.println("<log>");
             out.println(" <time>" + date + "</time>");
             out.println(" <level>" + level + "</level>");
             out.println(" <message>" + msg + "</message>");
             out.println(" <exception>" + exception + "</exception>");
            out.println("</log>");
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
        finally {
            if (out != null) {
                out.close();
            } // if を終了
        } // finally を終了
       return true;
    }  // logEvent メソッドを終了

    public void flush()
    {
        // No
    }
}
```

サービス要素の定義

XMLLogEventHandler を、jrun.xml ファイルの ThreadedLogEventHandler サービス内のサービスとして定義します。サービスにはファイル名の属性が次のように形式で含まれます。

```
...
<service class="jrunx.logger.ThreadedLogEventHandler"
         name="ThreadedLogEventHandler">

  <service class="jrunx.logger.FileLogEventHandler" name="FileLogEventHandler">
    <attribute name="filename">{jrun.rootdir}/logs/{jrun.server.name}-event.log
         </attribute>
    <attribute name="rotationSize">200000</attribute>
    <attribute name="rotationFiles">3</attribute>
    <attribute name="heading"># JRun によって {date MM/dd HH:mm:ss} に作成
         </attribute>
    <attribute name="closeDelay">5000</attribute>
  </service>

  <service class="XMLLogEventHandler" name="XMLLogEventHandler">
    <attribute name="filename">{jrun.rootdir}/logs/{jrun.server.name}
         -event.xml
         </attribute>
  </service>
</service>
...
```

ログメッセージの形式

デフォルトのログメッセージ形式には、日付と時刻、メッセージの種類、メッセージを生成した JRun サービス名、メッセージのコンテンツが含まれます。メッセージ形式は、JMC または jrun.xml ファイルにある属性を使って修正します。このデフォルトのログメッセージは次の形式で記述されています。

```
{date MM/dd HH:mm:ss} {log.message}{log.exception}
```

設定	説明
{date MM/dd hh:mm:ss}	日付と時刻の形式で、24 時間形式を使用します。
{log.message}	メッセージテキスト
{log.exception}	例外スタックトレース (例外メッセージのみ)

形式は変更できます。また、メッセージに追加情報を含めることもできます。有効な形式コンポーネントのリストについては、105 ページの「LoggerService」を参照してください。

Web サービスロギングの統合

JRun Web サービスエンジンである Apache Axis は、Apache log4j をロギングシステムとして使用します。JRun は、Axis ログメッセージを JRun サーバーのイベントログに出力することができます。各ログメッセージには、ログメッセージを生成した Axis クラスの名前が接頭辞として使用されます。

Axis ログレベルは、<JRun ルートディレクトリ >/servers/< サーバー名 >/SERVER-INF/jrun.xml ファイルで true に設定されている最下位の JRun ログレベルに一致します。INFO は true に設定されている最下位のログレベルであるため、デフォルトでは優先度 INFO 以上の Axis ログメッセージは JRun イベントログにログされます。

ほとんどの Axis ログメッセージの優先度は DEBUG です。jrun.xml ファイルで DEBUG ログレベルを true に設定すると、Axis によって Web サービスリクエストおよびレスポンスが処理される様子を確認することができます。ただし、この設定よってログの出力量が増大します。

Log4j はその設定情報を <JRun のルートディレクトリ >/lib/jrun.jar ファイルの log4j.properties ファイルから読み取ります。log4j.properties ファイルを無視するには、log4j.configuration システムプロパティを他のプロパティファイルに設定し、目的の設定に合った適切な方法で log4j を設定します。JRun 設定は、設定情報がプロパティファイルから読み取られた後に実行されるので、独自の log4j 設定を使用することもできます。

第 5 章
リソース

J2EE アプリケーションは通常、いくつかのリソースを共有しています。このため、アプリケーションサーバーでは、一貫した可用性、命名、およびリソースプールが保証されます。すべてのリソースは、JNDI サブコンテキストに保管されます。JNDI の詳細については、94 ページの「JRun JNDI の使用」を参照してください。

目 次

JDBC

このセクションでは、JRun 管理者のための JDBC に関する注意事項について説明します。JRun 管理者は、JRun サーバーで実行するすべてのアプリケーションについて、JRun データソースが定義されていることを確認する必要があります。実行時に、JRun はこれらのデータソースを作成し、JNDI に保管します。アプリケーションは `InitialContext.lookup` メソッドを使用してそれぞれのデータソースにアクセスします。

JRun データソースには、次の機能があります。

- JDBC 対応ドライバのサポート
- コネクションプール (ドライバがサポートしていない場合も有効)
- XA サポート (ドライバがサポートしていない場合も有効)

JRun には、さまざまなリレーショナルデータベースにアクセスできる Type 4 JDBC ドライバが付属しています。詳細については、JRun ドキュメントに付属の『DataDirect Connect JDBC User's Guide and Reference』を参照してください。

JRun データソースの定義

JRun サーバーの JDBC データソースを定義するには、JMC を使用します。JRun JDBC データソースを定義する場合は、次の項目を設定します。

設定	説明
JNDI 名	アプリケーションがデータソースを検索するときに使用する名前。この名前は、CMP EJB の JRun デプロイメントディスクリプタでも使用されます。この名前にスラッシュを含めることはできますが、/jdbc で始まる必要はありません。
ドライバクラス名	`com.pointbase.jdbc.jdbcUniversalDriver` など、ドライバのクラスファイルの名前。JMS は、よく使用されるほとんどの JDBC ドライバのクラス名を自動的に判定します。
データベース URL	`jdbc:pointbase:server://127.0.0.1:9292/compass` など、データベースを指定する URL。この設定は、DBMS および JDBC ドライバごとに異なります。JMC は、選択された JDBC ドライバーから推測し、一般的な URL を自動的に判定します。

設定	説明
詳細設定	その他の JDBC 設定。次の項目があります。 • native-results　キャッシュできるスクロール可能で更新可能な JRun ResultSet 実装を使用するには true に設定します。false に設定すると、元となる JDBC ドライバによって返される ResultSet が使用されます。 • initial-connections　プールのインスタンス化時に JRun が作成する接続の数を設定します。 • pool-statements　呼び出しのたびに PreparedStatements を再作成せずにプールしておくには、true に設定します。 • minimum-size　JRun がプール内で保持するオブジェクトの最小数を設定します。 • maximum-size　JRun がプール内で保持するオブジェクトの最大数を設定します。 • maximum-soft　プールの最大サイズに到達した際にリクエストがまだオブジェクト上で待機している場合に、コネクションプールが一時的な緊急手段として新しいオブジェクトを作成するには、true に設定します。それによってプールのサイズは一時的に大きくなりますが、プールは自動的に許容可能なサイズに縮小します。オブジェクトが使用可能になるまでリクエストを待機させるには、false に設定します。 • disable-pooling　JDBC コネクションプールを無効化にします。 • connection-timeout　JRun が接続を廃棄するまでに休止状態を維持する時間 (秒) • usertimeout　接続が自動的にプールに戻されるまでにユーザーが接続を維持する時間 (秒) • skimmer-frequency　リープサイクル間で、プールスキマーが待機する時間 (秒)。スキマーは各リープサイクルですべての接続 (チェックインした接続とチェックアウトした接続の両方) を評価して、それらの接続がタイムアウトになった場合に自動的にプールに戻すか、あるいは廃棄するかを決めます。 • shrink-by　1 つのリープサイクルでプールから削除できるオブジェクトの最大数を設定します。JRun は、スキマーによってプールが縮小されるたびにこの値を確認します。そのため、JRun のコネクションプールメカニズムによってピーク時にプールが急激に縮小されるのを防ぎます。 • debugging　冗長なロギング情報を有効にするには、true に設定します。 • cache-enabled　クエリ /ResultSet のキャッシュを有効にするには、true に設定します。 • cache-size　キャッシュすることができるクエリ /ResultSet のペアの最大数を設定します。 • cache-refresh-interval　キャッシュがデータベースからその ResultSet をリロードする間隔 (秒) • remove-on-exceptions　SQLException がある場合にプールから接続を削除するには、true に設定します。

JMC では、多くの一般的な JDBC ドライバのドライバクラス名、URL、ポート、およびその他の設定にドライバ固有のデフォルトを指定します。必要に応じて、これらの設定を変更したり、他の JDBC ドライバを使用したりすることもできます。詳細については、JMC のオンラインヘルプを参照してください。

ドライバクラス名およびデータベース URL の詳細については、JDBC ドライバのドキュメントを参照してください。

JRun では、データソース設定を <JRun のルートディレクトリ >/servers/<JRun のサーバー名 >/SERVER-INF/jrun-resources.xml ファイルに保管します。jrun-resources.xml の設定の詳細については、121 ページの「リソース：jrun-resources.xml ファイル」または JRun オンラインドキュメントのホームページからアクセスできる、オンラインディスクリプタについてのドキュメントを参照してください。

JRun クラスパスの更新

JRun でデータソースを定義したら、JDBC ドライバの JAR ファイルが JRun サーバーのクラスパスにあることを確認する必要があります。簡単な方法として、JAR ファイルを <JRun のルートディレクトリ >/servers/lib ディレクトリにコピーし、すべての JRun サーバーに対して使用可能にする方法があります。JMC の [Java VM 設定] パネルを使用して、個々のサーバーのクラスパスに JAR ファイルを追加することも可能です。

CMP エンティティ bean の JRun EJB デプロイメントディスクリプタは、サーブレットおよび JSP で使用される他、JRun データソースの指定にも使用されます。JRun EJB デプロイメントディスクリプタは、データソースを明示的に指定したり、**defaultdatasource** を指定して、JRun の起動時に最初にバインドされたデータソースを使用したりすることができます。

Java Message Service

このセクションでは、JRun 管理者のための JMS (Java Message Service) に関する注意事項について説明します。デフォルトの JRun JMS プロバイダを使用する方法、代わりに SonicMQ JMS を使用する方法、およびファクトリと送信先を設定する方法について説明します。

JMS プロバイダの選択は、jrun.xml ファイルの設定によって制御されます。このファイルでは、JRun のデフォルトの JMS プロバイダを使用するか、または SonicMQ などのサードパーティの JMS プロバイダを使用するかを指定します。jrun.xml ファイルは、次のサービスを使用して JMS プロバイダを定義します。

- **JMSAdapter** JMS プロバイダの属性を指定します。
- **JMSServiceWrapper** JMS プロバイダ、ファクトリ、および送信先を JNDI にバインドします。

JMSAdapter サービスと JMSServiceWrapper サービスを変更するには、jrun.xml ファイルをテキストエディタで変更します。

メモ: JMS の管理とプログラミングの補足情報については、JMS リソースやこのマニュアルの例を使用する以外に、JMS に関する専門書を使用して取得することもできます。このマニュアルの序章に、これらの書籍のリストが記載されています。

デフォルトの JMS プロバイダの使用

jrun.xml ファイルおよび JMC で設定した内容の他に、SERVER-INF/jrun-jms.xml ファイルを使用して、JRun のデフォルトの JMS プロバイダの動作を制御します。jrun-jms.xml ファイルを変更するにはテキストエディタを使用します。JMC は使用しません。

jrun-jms.xml ファイルには、パーシスタンス設定とサーバー設定があります。

パーシスタンスマネージャ設定

デフォルトの JMS プロバイダは、次のパーシスタンスマネージャをサポートしています。

パーシスタンスタイプ	説明
File (デフォルト)	JRun では、独自のプロトコルを使用してメッセージを SERVER-INF/jms ディレクトリに保管します。
DBMS	JRun では、メッセージテーブル、ハンドルテーブル、コンシューマテーブル、および送信先テーブルを使用して、メッセージをリレーショナルデータベースに保管します。DBMS を使用したパーシスタンスを使用するには、SERVER-INF/jms ディレクトリにあるスクリプトを使用して、サイト固有のデータベースにデータベーステーブルを設定します。さらに、jrun-jms.xml ファイルにデータソースを定義する必要があります。このデータソースには、同じ JRun サーバーにある既存のデータソースと同じ名前を使用することはできません。

パーシスタンスタイプは、jrun-jms.xml ファイルの `active-adapter-type` 要素によって file または rdbm に指定されます。必要に応じ、`cache` 要素を使用して、JMS プロバイダがメッセージをキャッシュするかどうかを制御できます。さらに、`transact` 要素を使用して、DBMS プロバイダがトランザクション機能を使用するかどうかを指定することもできます。これらの要素の他には通常、jrun-jms.xml ファイルの要素を変更することはありません。

JMS サーバー設定

jrun-jms.xml ファイルには、次の要素もあります。

要素	説明
transport	JMS プロバイダで使用するトランスポートメカニズムを指定します。RMI または intraVM です。これらは、JMC が表示するキュー接続ファクトリとトピック接続ファクトリの設定です。
connection-factory	デフォルトの JMS プロバイダのキュー接続ファクトリとトピック接続ファクトリの設定を指定します。
scheduler	内部使用。この要素は変更しないでください。
message-manager	キャッシュ設定とスレッド設定を指定します。
garbage-collection	ガーベッジコレクション設定を、秒単位で指定します。
destinations	システム送信先。この要素は変更しないでください。
JNDI 設定	クライアントが `InitialContext` コンストラクタの `Properties` オブジェクトに渡す必要があるプロパティと値を指定します。JRun サーバーをリモートクライアントで使用している場合は、PROVIDER_URL を localhost や 127.0.0.1 ではなく、その JRun サーバーの実際のホスト名と JNDI ポート番号に変更する必要があります。

JRun での他の JMS プロバイダの使用

JRun では、付属の JMS プロバイダの代わりに、サードパーティの JMS プロバイダを使用できます。JMS プロバイダを切り替えるには、JRun サーバーの jrun.xml ファイルの関連セクションを編集します。

メモ：JRun は、SonicMQ のバージョン 3.5 を使用して検証されています。他のバージョンの SonicMQ はサポートされておらず、SonicMQ 3.5 だけが JRun 4.0 で現在サポートされているサードパーティの JMS プロバイダです。

SonicMQ の使用

ここでは、JRun に組み込まれている JMS プロバイダの代わりに、Progess Software の SonicMQ JMS を使用する手順について説明します。

SonicMQ を JMS プロバイダとして使用するように設定すると、JRun では対象となる SonicMQ の専用インスタンスを自動的に実行します。専用インスタンスの使用するポートと、同じホストで実行中の他のインスタンスの使用するポートが重複しないことだけを注意してください。

SonicMQ をインストールするには、SonicMQ のインストールに関するドキュメントにある手順に従ってください。JRun で使用する場合、SonicMQ の特別なインストール設定は必要ありません。

SonicMQ を使用するように JRun を設定するには、テキストエディタを使用して、JRun に組み込まれているアダプタおよびサービスラッパーの代わりに SonicMQ アダプタおよびサービスラッパーを使用するように、JRun サーバーの jrun.xml ファイルを変更します。デフォルトの jrun.xml ファイルには、SonicMQ アダプタおよびラッパーの正しい設定例がコメントに含まれています。SonicMQ のサーバー固有の設定も行います。

JRun を SonicMQ 対応に設定するには

1 <JRun のルートディレクトリ>/servers/<サーバーのルートディレクトリ>/SERVER-INF ディレクトリにある、JRun サーバーの jrun.xml ファイルを開きます。

2 `<service class="jrun.jms.adapter.JRunMQAdapter" name="JMSAdapter">` で始まるセクションを探し、これをコメントに含めます。

3 `<service class="jrun.jms.wrapper.JRunMQServiceWrapper" name="JMSServiceWrapper">` で始まるセクションを探し、これをコメントに含めます。

4 その次の行のコメントに含まれている、SonicMQ アダプタに関する `JMSAdapter` および `JMSServiceWrapper` の 2 つのセクションを探し、コメントを外して有効にします。これらのセクションはそれぞれ、`<service class="jrun.jms.adapter.SonicMQAdapter" name="JMSAdapter">` および `<service class="jrun.jms.wrapper.SonicMQServiceWrapper" name="JMSServiceWrapper">` という行で始まります。

5 `jrun.jms.adapter.SonicMQAdapter` セクションで、ローカルの SonicMQ インストールを反映するように、次の属性を編集します。

属性	説明
HomeDir	SonicMQ をインストールしたディレクトリへの絶対パスを指定します。例：C:¥Program Files¥SonicSoftware¥SonicMQ
ConfigFileName	Sonic の専用インスタンスを起動するときに JRun が使用する SonicMQ 設定ファイルの名前を指定します。 このファイルは、HomeDir 属性によって指定されたディレクトリにある必要があります。
ClassPath	SonicMQ lib ディレクトリにある client.jar、broker.jar、defdb_server.jar の各 JAR ファイルのエントリを指定します。 これは標準 Java クラスパス指定であるため、プラットフォーム固有の適切な区切り文字およびシンタックスを使用する必要があります。これは、SonicMQ で必要とされる最小のクラスパスです。ローカル設定と SonicMQ の使用によって、この属性に SonicMQ lib ディレクトリにある他の JAR が必要になる場合もあります。
Port	JRun と SonicMQ の間の通信に使用するポートを指定します。 このポートは使用可能であり、同じホストで実行中の他の SonicMQ インスタンスが使用するポートとの重複を避ける必要があります。このポート設定は、JRun と Sonic の間の通信に使用するコネクタであるブローカマネージャが使用します。 SonicMQ がこのブローカマネージャからの接続を受け入れるには、SonicMQ 設定ファイルの `PORT` 行も変更する必要があります。この設定ファイルは、`HomeDir` 属性で指定された SonicMQ インストールのルートディレクトリにあります。デフォルトでは、SonicMQ 設定ファイルの名前は broker.ini です。JRun が別の Sonic 設定ファイルを使用し、このホストで起動する Sonic のデフォルトインスタンスではないインスタンスを起動するには、`ConfigFileName` 属性を使用して、代わりの SonicMQ 設定ファイルの名前を指定します。

6 SonicMQ lib ディレクトリにある SonicMQ の client.jar ファイルおよび broker.jar ファイルを <JRun のルートディレクトリ>/servers/lib ディレクトリにコピーします。

このディレクトリにある JAR ファイルは、実行する JRun サーバーのクラスパスに自動的に追加されます。

7 jrun-resources.xml ファイルにある JMS 接続ファクトリの `transport` 属性が TCPIP に設定されていることを確認します。SonicMQ でサポートされているトランスポートタイプは、TCP/IP だけです。

ファクトリおよび送信先の設定

JRun は、次のタイプの JMS ファクトリおよび送信先を管理します。

- QueueConnection ファクトリ
- TopicConnection ファクトリ
- キュー
- トピック

JMS 接続ファクトリを使用すると、送信者と受信者の両方のクライアントが JMS との接続を確立できます。クライアントは JNDI 検索を使用して JMS 接続ファクトリにアクセスし、このファクトリを使用して接続を確立します。クライアントは、JNDI を介してキューやトピックなどの送信先にもアクセスします。

通常、JMC を使用してファクトリ、送信先、および JNDI でのこれらの場所を定義します。ファクトリおよび送信先の情報は jrun-resources.xml ファイルに保管されます。このファイルは、手動で編集することも可能です。jrun-resources.xml ファイルの詳細については、121 ページの「リソース：jrun-resources.xml ファイル」または JRun オンラインドキュメントのホームページからアクセスできる、オンラインディスクリプタについてのドキュメントを参照してください。

ファクトリおよび送信先の詳細については、JMC のオンラインヘルプを参照してください。

JavaMail

このセクションでは、JRun 管理者のための JavaMail に関する注意事項について説明します。JRun 管理者は、JNDI を介して `javax.mail.Session` オブジェクトにアクセスできるように JRun サーバーの設定が定義されていることを確認する必要があります。

実行時、JavaMail クライアントは JNDI 検索で `Session` オブジェクトを取り出し、これを `Transport` オブジェクトと `Store` オブジェクトのファクトリとして使用します。次の項目を設定する必要があります。

設定	説明
ストアプロトコル	pop または imap
トランスポートプロトコル	常に smtp
デフォルトのメールサーバーホスト名およびユーザー名	デフォルトのメールサーバーおよびユーザー名
デフォルトの smtp メールサーバー名およびユーザー名	電子メールを送信するときのデフォルトのメールサーバーおよびユーザー名
デフォルトの imap メールサーバー名およびユーザー名	電子メールを IMAP で受信するときのデフォルトのメールサーバーおよびユーザー名
デフォルトの pop3 メールサーバー名およびユーザー名	電子メールを POP3 で受信するときのデフォルトのメールサーバーおよびユーザー名
デフォルトの送信者の電子メールアドレス	コードで指定されていない場合の電子メールアドレス

これらは、jrun-resources.xml ファイルをテキストエディタで編集して設定します。jrun-resources.xml ファイルに JavaMail 設定がない場合、JRun では jrun.xml ファイルで指定されているデフォルト設定が使用されます。

JCA (Java Connector Architecture)

JCA (J2EE Connector Architecture) には、EIS (Enterprise Information System) を J2EE アプリケーションに統合するための標準的な方法が用意されています。JCA コネクタ (リソースアダプタとも呼ぶ) は、SAP R3 などの ERP アプリケーション、レガシーシステム、データベースにアクセスする、EIS 固有のコンポーネントです。リソースアダプタは通常、EIS ベンダーによって供給され、サーバーと統合し、セキュリティやトランザクションを実現します。また、クライアントアプリケーションと統合したりするためのサービスとインターフェイスを提供します。

エンタープライズアプリケーションが EAR ファイルにパッケージされているのと同様に、リソースアダプタもリソースアーカイブ (RAR) ファイルにパッケージされています。このファイルには、デプロイメントディスクリプタ (META-INF/ra.xml ファイル内) および JRun 固有のデプロイメントディスクリプタ (META-INF/jrun-ra.xml ファイル内) が含まれています。JCA コネクタの RAR ファイルは、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 と同様にデプロイされます。

実行時、JRun では、`jrun-ra.xml` ファイルの jndi-name 要素として指定されている JNDI にリソースアダプタを保管し、クライアントアプリケーションは、`InitialContext.lookup` メソッドを使用してこれにアクセスします。JDBC、JMS、および JavaMail の `InitialContext.lookup` と同様に、EIS 固有の API へのクライアントアクセスを可能にするファクトリが返されます。オプションで、リソースアダプタによって、CCI (Common Client Interface) を介した EIS へのクライアントアクセスも提供されます。これによって、ベンダー固有の機能へのクライアントアクセスの標準化 API が指定されます。

コネクタは JDBC データソースとして使用できます。blackbox-notx.rar サンプルファイルに含まれているコネクタは、この使用方法を示しています。

blackbox-notx.rar コネクタを使用するには

1. RAR ファイルを解凍します。このディレクトリには、META-INF ディレクトリに blackbox-notx.jar ファイルとデプロイメントディスクリプタがあります。
2. META-INF/jrun-ra.xml ファイルの `connectionURL` プロパティを変更します。値を既存のデータソースの URL に設定します。

   これで、ra.xml ファイルの対応するプロパティが書き換えられます。たとえば、PointBase サンプルデータベースの実行時に、次のサンプルコードに示すように `jdbc:pointbase:server://127.0.0.1:9192/sample` と指定します。

   ```
   ...
   <config-property>
      <config-property-name>ConnectionURL</config-property-name>
      <config-property-type>java.lang.String
         </config-property-type>

         <config-property-value>jdbc:pointbase:server://127.0.0.1:9192/sample</
         config-property-value>
   </config-property>
   ...
   ```
3. jar ユーティリティを使用し、ファイルを再び RAR ファイルに圧縮します。
4. RAR ファイルを JRun にデプロイします。ファイルをサーバーのルートディレクトリにコピーするか (自動デプロイが有効な場合)、または JMC を使用します。

5 他の JDBC データソースへのアクセス方法と同じようにして、このデータソースにアクセスします。`InitialContext.lookup` メソッドを呼び出すには、jrun-ra.xml ファイルの `jndi-name` 要素に指定した値を使用します。

JRun Samples サーバーにはこの RAR ファイルが含まれ、これを使用してサンプルデータベースにアクセスできます。

第 6 章
JRun セキュリティ

多くのアプリケーションにとってセキュリティは非常に重要です。セキュリティによってユーザーが識別され、承認されたユーザーだけがアプリケーションのリソースにアクセスできることが保証されます。

この章では、JRun セキュリティのメカニズムと JRun セキュリティをカスタマイズする方法について説明します。

目 次

JRun セキュリティアーキテクチャ

すべてのアプリケーションにとってセキュリティは重要です。インターネットアプリケーション、リモートアプリケーション、およびエンタープライズ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、J2EE API ではアプリケーションリソースへのユーザーアクセスを管理するために認証および承認メカニズムが定義されています。認証および承認メカニズムは次のように定義されます。

- **認証**　ユーザー ID とパスワードをリクエストし、既存のユーザーストアと照合して、ユーザーを識別します。カスタマイズされた認証メカニズムを使用することができます。
- **承認**　ユーザーがアクセスできるリソースを識別します。J2EE 承認は、各ユーザーに割り当てられているロールに基づいています。安全に保管されたリソースにユーザーがアクセスするには、リソースにアクセスできるロールにユーザーが割り当てられている必要があります。ロールには、manager、developer、customer などが含まれます。

  メモ：ロールはネストできません。つまり、ロールを、他のロールで有効なユーザーに定義できません。

JRun 認証メカニズムはクラスタされたサービスです。これは、あるクラスタ内の 1 つの JRun サーバーで認証されたユーザーは、そのクラスタ内のすべてのサーバーで自動的に認証されることを意味しています。

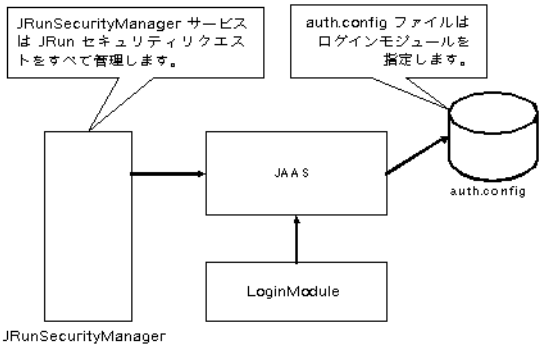
J2EE セキュリティ使用領域

認証および承認システムのすべてを処理するために、J2EE セキュリティは、複数の領域を網羅しています。これらの各領域は、それぞれ異なるロールで処理されます。

- **ロールの定義とセキュリティプログラム**　Web アプリケーションまたは EJB レベルに適用されるロールは、アプリケーションの開発者が定義します。アプリケーション開発者は、アプリケーションのリクエストに基づいてセキュリティプログラムを実装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。ほとんどの J2EE アプリケーションには、セキュリティプログラムよりも宣言セキュリティをお勧めします。
- **クライアントコーディング**　アプリケーションの開発者は、Web アプリケーションと EJB クライアントから必要な証明 (通常はユーザー ID とパスワード) を適切なタイミングで確実に渡すようにコードを記述しなければなりません。Web アプリケーションクライアントによってユーザー ID およびパスワードを入力するように指示されます。EJB クライアントは、呼び出しを行っている Web アプリケーションコンポーネントからの証明を使用するか、ユーザー ID とパスワードをプロパティとして `InitialContext` コンストラクタに渡します。JMS クライアントでは、jrun-resources.xml ファイル内でユーザー ID およびパスワードを指定できます。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。
- **ロールの解決と宣言セキュリティ**　アプリケーションアセンブリは、プログラマーが定義したロールを解決してシステムロールに関連付けます。また、アプリケーションアセンブリは Web デプロイディスクリプタおよび EJB デプロイディスクリプタの宣言セキュリティを指定します。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。
- **セキュリティアーキテクチャとユーザーストア管理**　管理者は、ユーザーストアを管理し、グローバルロールの定義を決め、サイト特有のセキュリティ環境に対応するように JRun セキュリティをカスタマイズします。次のセクションでは、管理者が行うタスクと、セキュリティ管理に関するその他のトピックについて説明します。

JAAS 概要

当初、J2EE 仕様では、アプリケーションを既存のセキュリティシステムおよびユーザーストアに統合する標準メカニズムがありませんでした。この統合メカニズムは各アプリケーションサーバーのベンダーによって実装されていました。J2EE 1.3 では、アプリケーションサーバーは JAAS (Java Authentication and Authorization Service) をセキュリティフレームワークとして使用することになりました。JAAS はさまざまなパッケージがセットになっているもので、ユーザ認証、モジュール式アクセスコントロールを JRun で可能にするものです。



JAAS は、Java 1.3 SDK ではオプションでしたが、Java 1.4 SDK ではパッケージに統合されました。

JAAS では、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できるように、プラグアンドプレイ操作でシステムをカスタマイズすることができます。JAAS の詳細については、http://java.sun.com をご覧ください。

JAAS では、プラグ可能な形式で認証を実行できます。つまり、認証を実行するときに、カスタマイズしたログインモジュールを代用して、JRun でログインモジュールの `login` メソッドを呼び出すことができます。詳細については、68 ページの「JRun セキュリティの拡張」を参照してください。

JAAS のテクノロジーは J2EE だけではありません。JAAS は任意の Java プログラムに使用でき、JAAS 承認はデフォルトでポリシーファイルを使用します。しかし、JRun などの J2EE アプリケーションサーバーのほとんどは、認証にポリシーファイルを使用しません。JRun は JAAS 認証メカニズムを拡張して認証処理を実行します。JRun セキュリティマネージャは、承認リクエストを受け取ると、ログインモジュールの `login` メソッドを呼び出し、ユーザー名と、許可されたロールセットを含む `Collection` オブジェクトを渡します。`login` メソッドは、認証されたユーザーが承認されているロールのメンバーであるかどうかを検証します。

JRun デフォルトセキュリティの実装

JRun セキュリティアーキテクチャは、JAAS ベースのログインモジュールで、ログインモジュールと密接に関連しているユーザーおよびロールストアを使用して認証および承認を行います。ログインモジュールは、`javax.security.auth.spi.LoginModule` インターフェイスを実装します。ログインモジュールでロジックコードを記述すると、JRun では JAAS API を使用してログインモジュールを適切なタイミングで呼び出します。

JRun には、XML 形式のユーザーおよびロールストアにアクセスするデフォルトのログインモジュールが含まれています。ユーザーとロールを管理するユーザーマネージャサービスも含まれています。

JMC はこのシステムを使用してユーザーアクセスを制御します。また、必要に応じて Web アプリケーション、EJB、および他のコンポーネントにセキュリティを使用できます。詳細については、69 ページの「デフォルトの JRun セキュリティメカニズムの使用」を参照してください。

JRun セキュリティの拡張

多くのサイトでは、ユーザーやユーザーのロールを定義するセキュリティインフラストラクチャが既に存在しています。既存のインフラストラクチャと統合するため、次のいずれかのオプションを選択します。

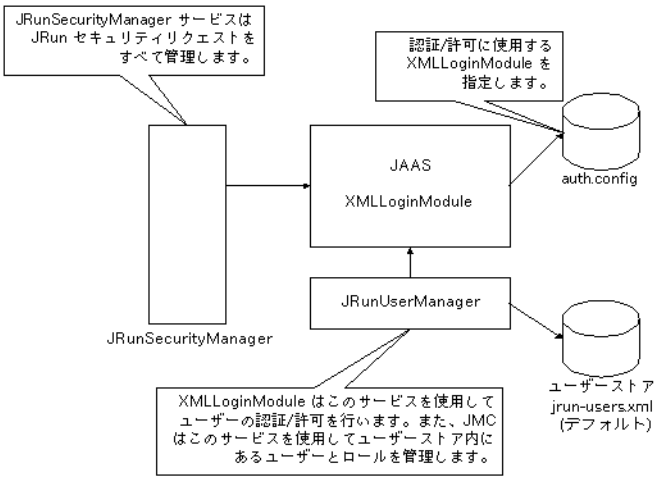
- **JRun で提供するログインモジュール**　JRun に含まれているログインモジュールのいずれかを使用します。詳細については、74 ページの「既存のセキュリティメカニズムとの統合」を参照してください。
- **カスタマイズされたログインモジュール**　サイト固有のログインモジュールを作成して、既存のセキュリティインフラストラクチャを認証や承認に使用することができます。詳細については、76 ページの「カスタムセキュリティの実装」を参照してください。

ユーザーストアをダイナミックに更新できるように、サイト特有のユーザーマネージャを実装することもできます。しかし、ほとんどの場合、カスタマイズされたセキュリティの実装には、この機能は必要ありません。詳細については、87 ページの「カスタムユーザーマネージャの定義」を参照してください。

デフォルトの JRun セキュリティメカニズムの使用

JRun には、テスティング、開発、デプロイに使用できるデフォルトのログインモジュールとユーザーストアが含まれています。このセクションでは、デフォルトのセキュリティメカニズムのアーキテクチャおよび使用方法について説明します。

デフォルトの JRun セキュリティアーキテクチャの概要を次の図に示します。



デフォルトの JRun セキュリティの実装には、次のコンポーネントが含まれています。

- **XMLLoginModule**　`JRunUserManager` サービスを使用して XML 形式のユーザーおよびロールストアからユーザーおよびロール情報にアクセスし、認証および承認を行います。
- **JRunUserManager サービス**　認証および承認のためのメソッドが含まれます。ユーザーストアのユーザーおよびロールをダイナミックに追加、修正、削除するメソッドも含まれています。
- **ユーザーストア**　ユーザー定義、ロール定義、ユーザーロールの割り当てが含まれています。デフォルトのユーザーストアは SERVER-INF/jrun-users.xml です。
- **auth.config ファイル**　JRun サーバーが認証および承認に使用するログインモジュールを指定します。

XMLLoginModule

デフォルトの JRun ログインモジュールである `XMLLoginModule` は JAAS で実行され、`JRunSecurityManager` サービスによって呼び出され、すべての認証および承認が処理されます。このログインモジュールは、JRun サーバーのユーザーストア (デフォルトは SERVER-INF/jrun-users.xml) で指定されている内容に基づいてユーザーを認証および承認します。ログインモジュールは、`JRunUserManager` サービスを使用してユーザーストアにアクセスし、`JRunUserManager` のメソッドを呼び出して情報にアクセスします。これは、リレーショナルデータベースからのユーザーおよびロール情報に SQL を使用してアクセスする JDBC ベースのログインモジュールに似ています。

JRun には、ログインモジュールも含まれています。このログインモジュールでは、リレーショナルデータベース、LDAP ディレクトリ、または Windows に保管されている情報を使用して、認証および承認を行います。詳細については、74 ページの「既存のセキュリティメカニズムとの統合」を参照してください。

ユーザーマネージャ

デフォルトの JRun ログインモジュールである `XMLLoginModule` は、`JRunUserManager` サービスを利用して、JRun 特有の XML ベースのユーザーストアにアクセスし、ユーザーパスワードの認証およびユーザーロールの承認を行います。`JRunUserManager` は、ユーザーストアをダイナミックに更新する機能も持ち合わせています。デフォルトの `JRunUserManager` は、`XMLLoginModule` に特有のサービスです。カスタムログインモジュールを作成する場合、ユーザーストアをダイナミックに更新できるようにサイト特有のユーザーマネージャを作成できます。しかし、これはアプリケーションを考慮した場合に限り、カスタマイズされたセキュリティの実装ではこの機能はほとんど必要ありません。

`JRunUserManager` サービスをカスタマイズする方法については、87 ページの「カスタムユーザーマネージャの定義」を参照してください。

ユーザーストア

デフォルトのセキュリティメカニズムのユーザーストアは、XML 形式のファイルで、次に示すようなユーザーおよびロールの定義に対する要素が含まれます。

XML 要素	説明
jrun-users	ルート要素
encryption	パスワードを暗号化するかどうかを指定します。
user	ユーザー定義をラップします。次のサブ要素が含まれます。 • username • password
role	ロール名、およびロールに割り当てられているユーザーのリストをラップします。次のサブ要素が含まれます。 • rolename • username

ユーザーストアのユーザーやロールを追加、修正、削除するには JMC を使用します。詳細については、JMC のオンラインヘルプを参照してください。

メモ：JMC へのユーザーのアクセスを可能にするには、jmcadmin ロールにこれらを追加します。

デフォルトでは、各 JRun サーバーはそれぞれ別々のユーザーストアを持っています。ユーザーストアのデフォルトの名前や場所は、<JRun サーバー>/SERVER-INF/jrun-users.xml にあります。ユーザーストアは複数の JRun サーバー間で共有することができます。

ユーザー、ロール、ユーザーロールの割り当ては JMC を使って定義できますが、jrun-users.xml ファイルはテキストエディタを使用して変更することもできます。

ユーザーストアの名前および場所は、jrun.xml ファイル内の `JRunUserManager` サービスの `securityStore` 属性によって制御されています。クラスタされた環境では、jrun-users.xml ファイルがクラスタ内のすべての JRun サーバーで複製されなければなりません。

例

次は、jrun-users.xml ファイルの例です。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jrun-users PUBLIC "-//Macromedia Inc.//DTD jrun-users 4.0//EN"
         "http://jrun.macromedia.com/dtds/jrun-users.dtd">
<jrun-users>
     <encryption>false</encryption>
     <user>
       <username>admin</username>
       <password>admin</password>
     </user>
     <user>
        <username>Flash</username>
        <password>Flashpass</password>
      </user>
     <user>
       <username>jmsuser</username>
       <password>jmsuser</password>
     </user>
     <user>
        <username>AxisUser</username>
        <password>AxisPassword</password>
      </user>
     <role>
       <rolename>jmcadmin</rolename>
       <username>admin</username>
     </role>
     <role>
      <rolename>FlashRole</rolename>
      <username>Flash</username>
     </role>
     <role>
      <rolename>AxisRole</rolename>
      <username>AxisUser</username>
     </role>
</jrun-users>
```

auth.config ファイル

JRunUserManager service の属性がユーザーストアを指定するのと同じように、JRunSecurityManager サービスの authConfig 属性がログインモジュールを指定するファイルの名前と場所を定義します。デフォルトでは、これは <JRun サーバー> /SERVER-INF/auth.config です。このファイルには、ユーザーおよびロールのログインモジュールを指定するセクションが含まれています。

このセクションでは、JRun がログインモジュールの initialize メソッドに渡すパラメータも指定します。ただし、最初のパラメータは JAAS が複数のログインモジュールを処理する方法について指定し、このパラメータは initialize メソッドには渡されません。最初のパラメータには、次のいずれかの値を使用します。

- **Requisite** このログインモジュールは、成功するために必須です。成功すると、リスト内の他のログインモジュールについて認証を続行します。エラーが発生すると、制御が JRun セキュリティマネージャに戻り、JAAS はリスト内の他のログインモジュールを呼び出しません。
- **Required** このログインモジュールは、成功するために必要です。成功またはエラーに関係なく、リスト内の他のログインモジュールについて認証を継続します。
- **Sufficient** このログインモジュールは成功に必ずしも必要ではありません。成功すると、JAAS から JRun セキュリティマネージャに制御が戻り、リスト内の他のログインモジュールを呼び出しません。エラーが発生すると、JAAS はリスト内の他のログインモジュールを呼び出します。
- **Optional** このログインモジュールは、成功に必ずしも必要ではありません。成功またはエラーに関係なく、リスト内の他のログインモジュールについて認証を継続します。

認証を成功させるには、Required ログインモジュールおよび Requisite ログインモジュールがすべて成功しなければなりません。Sufficient ログインモジュールを設定して成功した場合、認証を成功させるには、Sufficient ログインモジュールより前に、Required ログインモジュールと Requisite ログインモジュールだけは成功していなければなりません。Required ログインモジュールおよび Requisite ログインモジュールが JRun サーバーについて定義されていない場合は、1 つ以上の Sufficient ログインモジュールか、または Optional ログインモジュールが成功する必要があります。

次のデフォルトの auth.config ファイルで示すように、通常、JRun はユーザーとロール両方の XMLLoginModule を使用します。

```
defaultUser
 {
   jrun.security.XMLLoginModule required userManagerName="JRunUserManager"
         mode="USER";
 };
defaultRole
 {
   jrun.security.XMLLoginModule required userManagerName="JRunUserManager"
         mode="ROLE";
 };
```

暗号化

デフォルトの JRun セキュリティメカニズムでは、暗号化されたパスワードや、暗号化されていないパスワードを使用できます。つまり、すべてのパスワードはプレーンテキストか、暗号化されている必要があります。JMC でパスワードの暗号化を有効にするか、あるいは、jrun-users.xml ファイルで encryption 要素に true を指定します。暗号化を有効にすると、JMC で作成された新しいパスワードは、UNIX のパスワード暗号化を使用して暗号化されます。

また、JRun には、jrun-users.xml ファイル内に保存されているユーザーのパスワードを暗号化するコマンドラインユーティリティもあります。このユーティリティはパスワードを取り、`System.out` に暗号化されたパスワードを返します。暗号化されたパスワードをコピーし、テキストエディタを使用して jrun-users.xml ファイルに貼り付けてください。

メモ：パスワードの長さに制限はありませんが、ユーティリティでは最初の 8 文字だけを暗号化します。

パスワード暗号化ユーティリティを使用するには、クラスパスに <JRun のルートディレクトリ >/lib/jrun.jar が必要です。次のコマンドを使用して、パスワード暗号化ユーティリティを起動します。

```
java jrun.security.JRunCrypterImpl plain-text-password
```

たとえば、次のコマンドは benbob の暗号化パスワードを返します (角括弧をコピーしないでください)。

```
C:¥JRun4¥lib>java -classpath ./jrun.jar jrun.security.JRunCrypterImpl benbob
[benbob] => [hxH15XzpDckzU]
```

メモ：UNIX 暗号化の代わりに Twofish 暗号化メカニズムを使用するには、jrun.xml ファイルの JRunUserManagerService の `encrypterClass` 属性を `jrun.security.JRunCrypterForTwofish` に設定し、`jrun.security.JRunCrypterImpl` の代わりに `jrun.security.JRunCrypterForTwofish` を使用してパスワードを暗号化します。

既存のセキュリティメカニズムとの統合

JRun には、JDBC、LDAP、および Windows のログインモジュールが含まれています。これらのいずれかのログインモジュールを使用するには、JMC を使用してそのモジュールの使用方法を指定し、既存のセキュリティシステムと統合する方法を指定するパラメータを指定します。たとえば、JDBC ログインモジュールを使用するには、ユーザーおよびロール情報についてデータソース、テーブル名、および列を指定します。

メモ：ユーザーとロールに、同じメカニズムを使用する必要はありません。たとえば、リレーショナルデータベース内に既存のユーザーストアがあり、既存のロールストアがない場合は、ユーザーの認証に JDBC ログインモジュールを、承認 (つまり、ロール) に XMLLoginModule をそれぞれ使用できます。

JAAS で使用する他のクラスおよびインターフェイスの情報など、カスタマイズされたログインモジュール作成の情報については、76 ページの「カスタムセキュリティの実装」を参照してください。

JDBC ベースのセキュリティ実装の使用

JRun には、既存のテーブルと JRun セキュリティを統合して認証および承認を行うときに使用できる JDBC ログインモジュールが含まれています。JDBC ログインモジュールでは、ユーザー認証に次の情報が必要です。

- JRun データソースの JNDI 名
- ユーザーおよびパスワードが含まれているテーブル
- ユーザー情報が含まれている列
- パスワード情報が含まれている列
- データベースのユーザー ID (オプション)
- データベースのパスワード (オプション)
- ユーザー情報を返す SQL SELECT ステートメントを指定するクエリ文字列 (テーブルおよび列の情報を指定する代わりに使用します)

JDBC ログインモジュールでは、ユーザーロールの認証に次の情報が必要です。

- JRun データソースの JNDI 名
- ユーザーロールの情報が含まれているテーブル
- ロール情報が含まれている列
- ユーザー情報が含まれている列
- ロール情報を返す SQL SELECT ステートメントを指定するクエリ文字列 (テーブルおよび列の情報を指定する代わりに使用します)

JDBC ログインモジュールの使用は、JMC で行う指定で有効にするか、SERVER-INF/auth.config ファイルを手動で編集して有効にします。詳細については、JMC のオンラインヘルプを参照してください。

LDAP ベースのセキュリティ実装の使用

JRun には、既存の LDAP サーバーと JRun セキュリティを統合してユーザーの認証を行うときに使用する LDAP ログインモジュールが含まれています。LDAP ユーザーログインモジュールでは、ユーザーの認証に次の情報が必要です。

- LDAP ホスト (例：ldap://localhost:389)
- セキュリティタイプ (デフォルトは Simple です)
- 設定データ。通常は、ユーザーの識別名の接頭辞 (例：uid=) と、識別名の接尾辞 (例：ou=People、o=airius.com) を含みます。

メモ：LDAP にはユーザーおよびロールを管理する標準の位置がないため、ユーザーロールを承認するための LDAP ログインモジュールの使用はサポートされていません。

LDAP ユーザーログインモジュールの使用は、JMC で作成される指定で有効にするか、SERVER-INF/auth.config ファイルを手動で編集することによって有効にします。詳細については、JMC のオンラインヘルプを参照してください。

Windows ベースのセキュリティ実装の使用

JRun には、既存の Windows ドメインと JRun セキュリティを統合して認証および承認を行うときに使用できる Windows ログインモジュールが含まれています。このログインモジュールは、グループをロールにマッピングします。Windows ログインモジュールでは、ユーザー認証に次の情報が必要です。

- サーバー名
- ドメイン
- ユーザー情報にアクセスするために必要な情報を指定する、追加パラメータ

Windows ログインモジュールでは、ユーザーロールの認証に次の情報が必要です。

- サーバー名
- ドメイン
- ロール情報へのアクセスに必要な情報を指定する追加パラメータ

メモ：Windows のセキュリティ実装を使用している場合、Windows サーバーに Microsoft セキュリティパッチをすべて適用済みであることを確認してください。

Windows ロールログインモジュールの使用は、JMC で行う指定で有効にするか、SERVER-INF/auth.config ファイルを手動で編集して有効にします。詳細については、JMC のオンラインヘルプを参照してください。

カスタムセキュリティの実装

前のセクションで説明したデフォルトの JRun セキュリティメカニズムは、サイトによっては適用できない場合があります。たとえば、JDBC、LDAP、および Windows ログインモジュールの入力パラメータは、既存のセキュリティシステムでは機能しない場合があります。

このような場合、1 つ以上のカスタマイズされたログインモジュールを作成することによって、JRun セキュリティを必要に応じてカスタマイズして拡張できます。

ログインモジュール

JRun セキュリティをカスタマイズする前に、ログインモジュールが JRun でどのように使用されるかについて理解しておくことが必要です。デフォルトの JAAS 実装では、ログインモジュールは認証のみに使用されます。しかし、JRun では次に示すように認証および承認の両方でログインモジュールを使用します。

- **認証** JRun セキュリティマネージャは、ログインモジュールの `login` メソッドを呼び出し、ユーザー名に `javax.security.auth.callback.NameCallback` オブジェクトを、パスワードに `PasswordCallback` を挿入します。
- **承認** JRun セキュリティマネージャは、ログインモジュールの `login` メソッドを呼び出し、ユーザー名で `jrun.security.RolesCallback` オブジェクトを、また、セキュアリソースにアクセスできるロールのリストが含まれている `Collection` オブジェクトを挿入します。`RolesCallback` は JRun セキュリティ実装に特有のオブジェクトです。JAAS の一部ではありません。

実行する処理は、auth.config ファイルで指定されている `mode` 属性の USER または ROLE を渡して指定します。この属性の処理については、サンプルコードを参照してください。

次のサンプルコードは、ログインモジュールがユーザー名およびパスワードにアクセスする方法を示しています。

```
...
NameCallback n = new NameCallback("User Name - ", "Guest");
PasswordCallback p = new PasswordCallback("Password - ", false);
Callback[] callbacks = {n, p};

try {
  cbHandler.handle(callbacks);
  username = n.getName().trim();
  char[] tmpPassword = p.getPassword();
...
```

次のサンプルコードは、ユーザー名および許可されるロールにアクセスします。

```
...
boolean userRoleFound = false;

RolesCallback rcb = new RolesCallback();
Callback[] callbacks = {rcb};
try {
  // cbHandler は CallbackHandler オブジェクトです。
  cbHandler.handle(callbacks);
  Principal p = rcb.getPrincipal();
  username = p.getName().trim();
  Collection rolesToCheck = rcb.getRoles();
...
```

JDBC を使用するカスタマイズされたログインモジュールの定義

多くのサイトでは、デフォルトの JRun ログインモジュールを使用して認証および承認を行うことができます。ただし、ユーザーのサイトでデフォルトの JRun の JDBC ログインモジュールを使用できない場合は、カスタマイズされたログインモジュールを作成して、ユーザーのサイト固有のリレーショナルデータベースのユーザーおよびロールにアクセスできます。

デフォルトの JDBC ログインモジュールの詳細については、74 ページの「JDBC ベースのセキュリティ実装の使用」を参照してください。

カスタマイズされた JDBC ログインモジュールを使用するには、次の手順を実行します。

- リレーショナルデータベースにアクセスしてユーザーの認証とロールの承認を管理する `login` メソッドが含まれるログインモジュールコードを作成します。例については、samples/SERVER-INF/lib/jrun/samples/security ディレクトリにある `SampleJDBCLoginModule` を参照してください。
- 必要に応じて、JDBC ログインモジュールを指定してパラメータを渡すように SERVER-INF/auth.config ファイルを修正します。例については、samples/SERVER-INF/auth.config を参照してください。このファイルには、`SampleJDBCLoginModule` で使用されるコメント行があります。また、この指定を JMC で行うこともできます。

例

次の例は、カスタマイズされた JDBC ログインモジュールを示しています。

```
...
// 必要なインポート
import javax.security.auth.spi.LoginModule;
import javax.security.auth.Subject;
import javax.security.auth.login.LoginException;
import javax.security.auth.login.FailedLoginException;
import javax.security.auth.callback.*;
import java.security.Principal;
// ログインモジュール特有の機能に使用されるインポートです。
// ログインモジュール特有（JDBC）の機能に使用されるインポートです。
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;
import java.util.Map;
import java.util.Collection;
```

```
import java.util.Iterator;
import java.util.ArrayList;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// JRun 特有のインポート
import jrun.security.SimplePrincipal;
import jrun.security.RolesCallback;
// リソースバンドル処理に使用されます。
import jrunx.util.RB;
// ロギングに使用されます。
import jrunx.kernel.JRun;

public class SampleJDBCLoginModule implements LoginModule {
    private Subject subject;
    private CallbackHandler cbHandler;
    private Map sharedState;
    private Map options;

    /**
     * mode = "USER" の
     * SQL ステートメントは、特定のユーザー名のパスワードを照会します。
     * 例："select PasswordColumn from UsersTable where UserName=?"
     *
     * mode = "ROLE" の
     * SQL ステートメントは、特定のユーザー名のロールを照会します。
     * 例："select RoleColumn from RolesTable where UserName=?"
     */
    private String queryString=null;

    /**
     * ユーザーは、ユーザー名と、ユーザー名を入力として扱うパスワードを
     * 取り出す SQL ステートメントを指定するか、テーブル名、ユーザー名、
     * 列、およびパスワード列を指定できます。
     * auth.config でパラメータとして指定されます。
     */
    private String userColumn=null;
    private String passwordColumn=null;
    private String tableName=null;
    private String roleColumn=null;
    /**
     * ユーザーセキュリティ情報が含まれるデータベースにアクセスするユーザー ID とパスワードです。
     * auth.config でパラメータとして指定されます。
     */
    private String dbUserId=null;
    private String dbPassword=null;
```

```
private String username=null;
private char[] password=null;

/**
 * データベースにアクセスするために JNDI 内で検索する JDBC データソースです。
 * これは、JRun データソースとなり得ます。
 * auth.config でパラメータとして指定されます。
 */
String dsJndiName=null;

// 認証ステータス
private boolean succeeded = false;
private boolean commitSucceeded = false;

//認証された principal オブジェクト
private SimplePrincipal userPrincipal;

// ログインモードを "USER" または "ROLE" にして、ユーザー証明を認証するか、
// ユーザーロールを承認するかを指定します。
protected String loginMode = "USER";

/**
 * <p>JRun セキュリティマネージャは、認証または
 * 承認リクエストの開始時に initialize メソッドを
 * 呼び出します。
 */
public void initialize(Subject subj, CallbackHandler cbh, Map sharedState,
     Map options) {
    this.subject = subj;
    this.cbHandler = cbh;
    this.sharedState = sharedState;
    this.options = options;

    loginMode = (String) options.get("mode");
    dbUserId = (String) options.get("databaseUserId");
    dbPassword = (String) options.get("databasePassword");
    dsJndiName = (String) options.get("dataSourceJNDIName");
    queryString = (String) options.get("queryString");
    tableName = (String) options.get("tableName");
    userColumn = (String) options.get("userNameColumn");

    if(loginMode.equals("USER") )
        // ユーザー認証に特有のパラメータ
        passwordColumn = (String) options.get("passwordColumn");
    else // ユーザーロール承認に特有のパラメータ
        roleColumn = (String) options.get("roleColumn");
    JRun.getLogger().logInfo("*** SampleJDBCLoginModule:initialize");
}

/**
 * <p>JRun セキュリティマネージャは、認証または承認リクエスト
```

```
     * に対して initialize メソッドを呼び出した後、login メソッドを
     * 呼び出します。
     */
    public boolean login() throws LoginException {

        if(cbHandler == null) {
            throw new LoginException(RB.getString(SampleJDBCLoginModule.class,
             "SampleJDBCLoginModule.noCallBackHandlerAvailable") );
        }

        JRun.getLogger().logInfo("*** SampleJDBCLoginModule: ログイン。モード：" +
         loginMode);
        if (loginMode.equals("ROLE") ) {
            // 承認のためにこのメソッドを呼び出します。
            return validateRole();
        }
        else {
           // デフォルト
           // 認証のためにこのメソッドを呼び出します。
           return loginUser();
        }
    }

    /**
     * <p> mode=USER のときに呼び出されます。
     */
    protected boolean loginUser() throws LoginException {
        NameCallback n = new NameCallback("User Name - ", "Guest");
        PasswordCallback p = new PasswordCallback("Password - ", false);
        Callback[] callbacks = {n, p};

        try {
            cbHandler.handle(callbacks);
            // 渡されたユーザー名 NameCallback を取得します。
            username = n.getName().trim();
            // デバッグの補助
            JRun.getLogger().logInfo("*** loginUser(): ログインしているユーザー：" +
         username);
            // PasswordCallback から渡されたパスワードを取得します。
            char[] tmpPassword = p.getPassword();
            if( tmpPassword != null ) {
                password = new char[tmpPassword.length];
               System.arraycopy(tmpPassword, 0, password, 0, tmpPassword.length);
                p.clearPassword();
            }

        }
        catch(java.io.IOException e) {
            throw new LoginException(e.toString());
        }
        catch(UnsupportedCallbackException uce) {
```

```
            // 新規 LoginException("サポートされていないコールバック：" +
         uce.getCallback());
            throw new LoginException(RB.getString(SampleJDBCLoginModule.class,
         "SampleJDBCLoginModule.unsupportedCallBack"));
        }

        // コールバックからのパスワードとデータベースのパスワードを検証します。
        String passwordFromDB = getUserPassword().trim();
        JRun.getLogger().logInfo("*** loginUser(): データベースのパスワード :" +
         passwordFromDB);
        succeeded = true;
        char[] pw = passwordFromDB.toCharArray() ;
        if(password.length == pw.length) {
            for(int i = 0; i < pw.length; i++) {
                if (password[i] != pw[i]) {
                    succeeded = false;
                    break;
                } // 内側 if を終了
            } // for を終了
        }
        else
            succeeded = false; // 同じ長さではない
        return succeeded;
    }

    /**
     * <p> mode=ROLE のときに呼び出されます。
     */
    protected boolean validateRole() throws LoginException {
        boolean userRoleFound = false;

        JRun.getLogger().logInfo("*** validateRole()");
        RolesCallback rcb = new RolesCallback();
        Callback[] callbacks = {rcb};
        try {
            cbHandler.handle(callbacks);
            // RolesCallback からユーザー名を取得します。
            Principal p = rcb.getPrincipal();
            username = p.getName().trim();

            // 承認されたロールを RolesCallback から取得します。
            Collection rolesToCheck = rcb.getRoles();
            // 割り当てられているロールをデータベースから取得します。
            ArrayList rolesFromDatabase = getUserRoles();

            //rolesToCheck を繰り返し、rolesFromDataBase からユーザー名を持つものを検索し
         ます。
            Iterator i = rolesToCheck.iterator();
            while(i.hasNext() && !userRoleFound) {
                String thisRoleName = (String) i.next();
                int numberOfRolesFromDB = rolesFromDatabase.size();
                for(int index = 0; index < numberOfRolesFromDB; index++) {
```

```
                String dbRoleName = (String) rolesFromDatabase.get(index);
                if(thisRoleName.equals(dbRoleName.trim()) ) {
                    succeeded = true;
                    userRoleFound = true;
                } // 内側 if を終了
            } // for を終了
        } // while を終了
    }
    catch(java.io.IOException e) {
        throw new LoginException(e.toString());
    }
    catch(UnsupportedCallbackException uce) {
        throw new LoginException(RB.getString(SampleJDBCLoginModule.class,
     "SampleJDBCLoginModule.unsupportedCallBack") );
    }
    catch(Exception e) {
        JRun.getLogger().logError(RB.getString(SampleJDBCLoginModule.class,
     "SampleJDBCLoginModule.errorValidatingRole"), e);
    }
    return userRoleFound;
}

/*
 * <p> データベースを呼び出して、ユーザー名のパスワードを取り出します。
 */
protected String getUserPassword() throws LoginException {
    Connection conn = null;
    PreparedStatement ps = null;
    String passwordFromResult=null;

    try {
        InitialContext ctx = new InitialContext();
        DataSource ds = (DataSource) ctx.lookup(dsJndiName);
        if(dbUserId != null)
            ds.getConnection(dbUserId, dbPassword);
        else
            conn = ds.getConnection();

         // SQL クエリ文字列か、テーブル名、パスワード、およびユーザー名の列を提供できます。
         if(queryString != null) {
            ps = conn.prepareStatement(queryString);
            ps.setString(1, username);
         }
         else {
            queryString = "select " + passwordColumn + " from " +  tableName
     + " where " + userColumn + "=" + username;
             ps = conn.prepareStatement(queryString);
         }
         ResultSet rs = ps.executeQuery();
         if( rs.next() == false )
            throw new
      FailedLoginException(RB.getString(SampleJDBCLoginModule.class,
```

```
                "SampleJDBCLoginModule.userNameNotFound") );

            passwordFromResult = rs.getString(1);
            rs.close();
        }
        catch(NamingException ex) {
            throw new LoginException(ex.toString(true));
        }
        catch(SQLException ex) {
            throw new LoginException(ex.toString());
        }
        finally {
            connCleanup(ps, conn);
        }
        return passwordFromResult;
    }

    /**
     * ユーザー名に割り当てられてたロールを取り出すためにデータベースを呼び出します。
     */
    protected ArrayList getUserRoles() throws LoginException {
        Connection conn = null;
        PreparedStatement ps = null;
        ArrayList rolesFromDB= new ArrayList();
        try {
            InitialContext ctx = new InitialContext();
            DataSource ds = (DataSource) ctx.lookup(dsJndiName);
            if(dbUserId != null)
                ds.getConnection(dbUserId, dbPassword);
            else
                conn = ds.getConnection();

            // auth.config ファイルでは、SQL クエリ文字列、または
            // テーブル名、ロール名、およびユーザー名の列を提供できます。
            if(queryString != null) {
                ps = conn.prepareStatement(queryString);
                ps.setString(1, username);
            }
            else {
                queryString = "select " + roleColumn + " from " +  tableName + "
          where " + userColumn + "=" + username;
                 ps = conn.prepareStatement(queryString);
            }
            ResultSet results = ps.executeQuery();
            String dbRolename=null;
            if(results.getFetchSize() > 0) {
                while(results.next() ) {
                    dbRolename = results.getString(1);
                    rolesFromDB.add(dbRolename);
                } // while を終了
            } // if を終了
            else
```

```
                throw new
         FailedLoginException(RB.getString(SampleJDBCLoginModule.class,
                    "SampleJDBCLoginModule.userNameNotFound") );

            results.close();
        } // try を終了
        catch(NamingException ex) {
            throw new LoginException(ex.toString(true));
        }
        catch(SQLException ex) {
             throw new LoginException(ex.toString());
        }
        finally {
            connCleanup(ps, conn);
        }
        return rolesFromDB;
    }

    protected void connCleanup(PreparedStatement ps, Connection conn) {
        if( ps != null ) {
            try {
               ps.close();
            }
            catch(SQLException e)
            {}
         }
         if( conn != null ) {
            try {
               conn.close();
            }
            catch (SQLException ex)
            {}
         }
    }

    /**
     * <p> 認証および承認が成功したかどうかを示します。
     */
    public boolean commit() throws LoginException {
        if (succeeded == false) {
            return false;
        }
        else {
            // Principal (認証済みの ID) を
            // Subject に追加します。

            // 認証されるユーザーは SimplePrincipal とします。
            userPrincipal = new SimplePrincipal(username);
            if (!subject.getPrincipals().contains(userPrincipal))
            subject.getPrincipals().add(userPrincipal);
```

```
        // いずれの場合も、ステートをクリーンアップします。
        username = null;
        // for (int i = 0; i < password.length; i++)
        //password[i] = ' ';
        password = null;

        commitSucceeded = true;
        return true;
    }
}

/**
 * <p> このメソッドは LoginContext の
 * すべての認証が失敗した場合に呼び出されます。
 * ( 関連する REQUIRED、REQUISITE、SUFFICIENT、および OPTIONAL LoginModules
 * は成功していません。)
 *
 * <p> この LoginModule の認証が成功すると (<code>login</code> および
 * <code>commit</code> メソッドで保存されている プライベートステートを
 *取得することによって確認できます )、このメソッドははじめから
 * 保存されているすべてのステートをクリーンアップします。
 *
 * <p>
 *
 * abort が失敗した場合は例外 LoginException です。
 *
 * この LoginModule のログインまたはコミットが失敗した場合は false が返され、
 * それ以外は true が返されます。
 */
public boolean abort() throws LoginException {
    if (succeeded == false) {
        return false;
    }
    else if (succeeded == true && commitSucceeded == false) {
        // ログインには成功しましたが、全体の認証には失敗しました。
        succeeded = false;
        username = null;
        if (password != null) {
        password = null;
        }
        userPrincipal = null;
    }
    else {
        // 全体の認証およびコミットは成功しましたが、
        // 他のユーザーのコミットには失敗しました。
        logout();
    }
    return true;
}
```

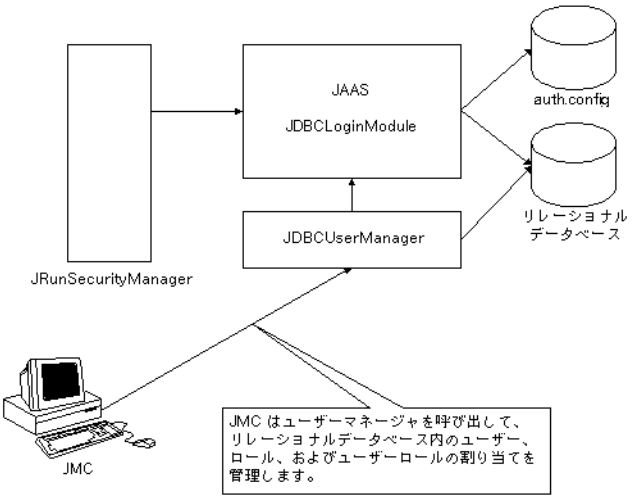
```
    /**
     * ユーザーをログアウトします。
     *
     * <p> このメソッドは、<code>SamplePrincipal</code> を削除します。
     * これは、<code>commit</code> メソッドによって追加されたものです。
     *
     * <p>
     *
     * logout に失敗した場合は例外 LoginException です。
     *
     * この <code>LoginModule</code> は無視できないので
     * すべてのケースで true が返されます。
     */
    public boolean logout() throws LoginException {
        subject.getPrincipals().remove(userPrincipal);
        username = null;
        if (password != null) {
            for (int i = 0; i < password.length; i++)
            password[i] = ' ';
            password = null;
        }
        userPrincipal = null;
        return true;
    }
} // クラスを終了します。
```

カスタムユーザーマネージャの定義

JRun では、次のような目的でデフォルトのユーザーマネージャが使用されます。

- **認証と承認**　デフォルトの JRun セキュリティでは、ユーザーマネージャは認証および承認のデフォルトのログインモジュールである `XMLLoginModule` と密接に関連しています。JDBC データベースなどの他のデータストアのユーザーおよびロールにアクセスするログインモジュールを使用すると、ユーザーマネージャは認証および承認に関連付けられません。
- **ユーザーストア管理**　デフォルトの JRun セキュリティでは、JMC はユーザーマネージャを使用してユーザー、ロール、およびユーザーロールの割り当てを管理します。JDBC データベースなどの他のユーザーストアを使用すると、ユーザー、ロール、およびユーザーロールの割り当ての管理に、通常は JMC 以外のツールが使用されます。ただし、ユーザーマネージャのコードを作成し、JMC でユーザーストアをダイナミックに更新することもできます。

カスタマイズされたユーザーマネージャを使用すると、JMC ユーザーは次の図に示すようにユーザーストアにあるユーザー、ロール、およびロールの割り当てを更新できます。



ユーザーマネージャをカスタマイズするには、次の手順を実行する必要があります。

- ユーザーマネージャのコードを作成します。
- jrun.xml ファイルにある `JRunUserManager` サービスを更新します。

カスタマイズされたユーザーマネージャで、`jrunx.kernal.ServiceAdapter` を拡張し、`jrun.security.JRunUserManager` を実装し、次のメソッドを実装する必要があります。

- `public void setSecurityStore(String storename);`
- `public boolean isUser(String username);`
- `public boolean isRole(String rolename);`
- `public boolean addUser(String username, String password, String description);`
- `public boolean addRole(String roleName, String description);`
- `public boolean addUserToRole(String rolename, String username);`
- `public int addUsersToRole(String rolename, Collection usernames);`

- `public boolean removeUser(String username);`
- `public boolean removeFromRole(String rolename, String username);`
- `public boolean removeRole(String roleName);`
- `public char[] getPassword(String username);`
- `public String getPasswordString(String username);`
- `public boolean changePassword(String username, String oldpassword, String newpassword);`
- `public Collection getUsers(String roleName);` - このロールのユーザー名を含む String の `Collection` を返します。
- `public Collection getRoles(String userName);` - このユーザーが属するロール名を含む String の `Collection` を返します。
- `public boolean isUserInRole(String username, String rolename);`
- `public String getRoleDescription(String roleName);`
- `public String getUserDescription(String userName);`
- `public boolean changeUserDescription(String username, String newdescription);`
- `public boolean changeRoleDescription(String rolename, String newdescription);`
- `public void clearAll();`

第 7 章
接続の監視

JRun が提供する監視メカニズムの 1 つを使用して、Web サーバーと JRun の接続に関する統計情報を取得できます。この統計には、処理されたリクエストの数、リクエストの処理に使用可能なスレッドの数、ヒープメモリの使用状況が含まれます。このメカニズムを使用して、JRun スケジューラの統計を表示することもできます。

この章では、接続管理メカニズムと、このメカニズムの設定、およびこのメカニズムにより収集される情報の制御について説明します。

目 次

Web サーバー接続の監視

JRun は、JRun サーバーと Web サーバー間の接続についてのステータス情報を、JRun ログファイルに書き込むことができます。このステータス情報は、JRun サーバーと、サードパーティ Web サーバーまたは JRun Web サーバー (JWS) 間の接続から取得されます。

JRun が接続ステータス情報を収集するように設定するには、JMC を使用して、または JRun サーバーの jrun.xml ファイルで **LoggerService** の **metricsEnabled** 属性を true に設定して、メトリクスロギングを有効にします。

JRun のロギングメカニズムの詳細については、第 4 章、43 ページの「ロギングメカニズム」を参照してください。JMC の詳細については、JMC のオンラインヘルプを参照してください。

次は、**metricsEnabled** 属性を true に設定した例です。

```
...
<service class="jrunx.logger.LoggerService" name="LoggerService">
  <attribute name="errorEnabled">true</attribute>
  <attribute name="warningEnabled">true</attribute>
  <attribute name="infoEnabled">true</attribute>
  <attribute name="debugEnabled">false</attribute>
 <attribute name="metricsEnabled">true</attribute>
...
```

メトリクスロガーは、**metricsLogFrequency** 属性に指定された間隔で、情報を JRun サーバーのログファイルに書き込みます。この値は秒単位で指定します。デフォルトは 60 秒です。

metricsFormat 属性は、ログファイルに書き込まれる情報を制御します。

JRun Web サーバーのメトリクスを取得するには、**web** 接頭辞を使用します。Web サーバーコネクタの統計を取得するには、**jrpp** 接頭辞を使用します。接頭辞のない統計は、JRun サーバーに関連付けられた JVM のものです。

metricsFormat 属性には、テキストと、次のメトリクス変数のどれか、またはすべてを含めることができます。

メトリクス変数	説明
[web.|jrpp.]listenTh	新規接続をリスンするスレッド
[web.|jrpp.]idleTh	新規リクエストを待つスレッド
[web.|jrpp.]delayTh	実行待ちのスレッド
[web.|jrpp.]busyTh	現在実行中のスレッド
[web.|jrpp.]totalTh	ワーカースレッドの総数
[web.|jrpp.]delayRq	同時処理できずに遅延したリクエスト
[web.|jrpp.]droppedRq	ドロップされたリクエスト
[web.|jrpp.]handledRq	処理されたリクエスト
[web.|jrpp.]handledMS	遅延時間 (delayMs) を除く、リクエストへのサービスに費やされた時間 (ミリ秒単位)
[web.|jrpp.]delayMs	遅延の状態で経過した時間 (ミリ秒単位)

メトリクス変数	説明
[web.|jrpp.]bytesIn	リクエストから読み込まれたバイト数
[web.|jrpp.]bytesOut	レスポンスに書き込まれたバイト数
freeMemory	ヒープ内の空きメモリのキロバイト数
totalMemory	使用されているかどうかに関係なく、ヒープ全体のメモリのキロバイト数
sessions	現在のアクティブセッション数
sessionsInMem	メモリにあるセッション数

次の例は、メトリクスロガーの属性を示したものです。

```
...
<service class="jrunx.logger.LoggerService" name="LoggerService">
  <attribute name="errorEnabled">true</attribute>
  <attribute name="warningEnabled">true</attribute>
  <attribute name="infoEnabled">true</attribute>
  <attribute name="debugEnabled">false</attribute>
  <attribute name="metricsEnabled">true</attribute>
  <attribute name="metricsLogFrequency">10</attribute>
  <attribute name="metricsFormat">Web threads (busy/total):{jrpp.busyTh}/
        {jrpp.totalTh} Sessions:{sessions} Total Memory={totalMemory}
        Free={freeMemory}</attribute>
...
```

次に示すのは、この形式のログメッセージの例です。

```
02/14 16:11:53 metrics Web threads (busy/total):0/2 Sessions:2 Total Memory=7252
        Free=3103
```

監視出力形式の設定

metricsFormat 属性で、メトリクスロガーが表示する情報を指定します。デフォルトでは、metricsFormat は次のように設定されます。

```
Web threads (busy/total):{jrpp.busyTh}/{jrpp.totalTh} Sessions:{sessions} Total
        Memory={totalMemory} Free={freeMemory}
```

テキストと変数の他に、中括弧 {} の中に完全な式を入力することで、変数値を数値演算できます。たとえば、次のように指定すると、JWS と Web サーバーコネクタについて、ワーカースレッドの合計数が表示されます。

```
Total worker threads={{jrpp.totalTh}+{web.totalTh}}
```

スケジューラの監視

JRun では、スケジューラのステータス情報を JRun ログファイルに書き込むこともできます。スケジューラは、バックグラウンドスレッドで内部タスクをスケジュールおよび実行するときに使用されます。これらのタスクには、セッションタイムアウト、ファイルロギング、メトリクスロギング、JDBC コネクションプールの管理が含まれます。

次の表では、スケジューラ用のメトリクス変数を説明します。

メトリクス変数	説明
scheduler.listenTh	リスン中のスレッド
scheduler.idleTh	新規リクエストを待つスレッド
scheduler.delayTh	実行待ちのスレッド
scheduler.busyTh	現在実行中のスレッド
scheduler.totalTh	ワーカースレッドの総数
scheduler.delayRq	同時処理できずに遅延したリクエスト
scheduler.droppedRq	ドロップされたリクエスト
scheduler.handledRq	処理されたリクエスト
scheduler.handledMS	遅延時間 (delayMs) を除く、リクエストへのサービスに費やされた時間 (ミリ秒単位)
scheduler.delayMs	遅延の状態で経過した時間 (ミリ秒単位)

第 8 章
JNDI に関する考慮

この章では、JRun JNDI の使用方法の概略を説明します。

目 次

JNDI の概要

JNDI (Java Naming and Directory Interface) は、Java プログラムにネーミングとディレクトリサービスを提供するものです。JNDI によって、プログラムは、階層構造体に保管されている、オンラインディレクトリデータベースの Java オブジェクトを含むデータにアクセスできます。

JNDI は重要な J2EE テクノロジーです。EJB を含む多くの J2EE コンポーネントは、Java オブジェクトにアクセスするためのレポジトリとして JNDI を使用します。J2EE コンポーネント使用のサポートに加えて、JRun は JNDI を使用して JRun サービスの情報を保管します。

JNDI の詳細については、http://java.sun.com または専門書を参照してください。

JRun JNDI の使用

JRun には、JRun で使用される JNDI コンテキストの保管と管理を行う JNDI プロバイダが含まれています。JRun JNDI プロバイダはリモートアクセスを提供しています。また、このサービスはクラスタすることが可能です。このため、JNDI サービスに (`bindToJNDI` 属性を true に設定することで) 登録されている JRun サービスはすべてリモートアクセス可能であり、クラスタすることができます。クラスタリングの詳細については、37 ページの「オブジェクトのクラスタリング」を参照してください。

JNDI サービスは、JNDI ポートをリスンします。リモートクライアントは、`InitialContext` コンストラクタの `PROVIDER_URL` としてこのポート番号を指定します。詳細については、102 ページの「NamingService」を参照してください。

保管されるオブジェクト

JRun は、次のタイプのオブジェクトを JNDI に保管します。

オブジェクトまたはサービス	説明
データソース	トップレベルにあり、SERVER-INF/jrun-resources.xml ファイルの `data-source` 要素の `jndi-name` サブ要素によって定義された名前を持ちます。クライアントは、リソースリファレンス (デプロイメントディスクリプタからの)、JNDI 名、または java:<jndi 名 > でデータソースを参照できます。
EJB	トップレベルにあり、jrun-ejb-jar.xml ファイルの `jndi-name` 要素によって定義された名前を持ちます。jrun-ejb-jar.xml ファイルが存在しない場合、デフォルトは ejb-jar.xml ファイルの `ejb-name` 要素に定義された値です。クライアントは、JNDI 名、または java:<jndi 名 > で EJB を参照できます。
DefaultDataSource	起動時に定義された最初の JDBC データソースへのアクセスを提供します。
jms/queue	SERVER-INF/jrun-resources.xml ファイルの `jms-destination` 要素の `jndi-name` サブ要素で定義されたすべての JMS キューが含まれます。

オブジェクトまたはサービス	説明
jms/topic	SERVER-INF/jrun-resources.xml ファイルの **jms-destination** 要素の **jndi-name** サブ要素で定義されたすべての JMS トピックが含まれます。
jms/< 接続ファクトリ >	JMS 接続ファクトリを定義します。デフォルトでは、JRun サーバーは次の JMS 接続ファクトリを使用します。 • jrun-jms.xml 定義されている QueueConnectionFactory • jrun-jms.xml で定義されている TopicConnectionFactory • jrun-resources.xml で定義されている jndi-QueueConnectionFactory
mail/Mail セッション	jrun-resources.xml または jrun-xml で定義されている Mail セッションです。
jms_provider/queue	JRun JMS プロバイダによって作成されたキューの、プロバイダ固有の JNDI バインディングです。
jms_provider/topic	JRun JMS プロバイダによって作成されたトピックの、プロバイダ固有の JNDI バインディングです。
jms_provider/< トランスポート >	JRun JMS プロバイダによって使用される各トランスポートメカニズムの接続ファクトリ設定です。
comp/UserTransaction	ユーザートランザクションです。
comp/env/AxisServer	Web サービスの Axis サーバーです。
DefaultDomain/comp/UserTransaction	デフォルトドメインのユーザートランザクションです。
DefaultDomain/TransactionManager	デフォルトのドメイントランザクションマネージャです。
service/< サービス名 >	クラスタされた JRun サービスです。クラスタされた環境でこの JNDI バインディングを使用してサービスを参照する場合、コンテキストは、クラスタ内の任意の JRun サーバーから返されます。
service/< サーバー名 >/< サービス名 >	指定されたサーバーの JRun サービスです。この JNDI バインディングを使用してサービスを参照する場合、コンテキストは常に指定された JRun サーバーから返されます。

JNDI による JRun サービスへのアクセス

JRun JNDI ツリーのサービスには、**jrun:** という接頭辞が付いています。たとえば、JNDI を使用してロガーサービスにアクセスするには、次のコードを使用します。

```
...
  try {
    Properties p = new Properties();
    p.put(Context.INITIAL_CONTEXT_FACTORY, "jrun.naming.JRunContextFactory");
    p.put(Context.PROVIDER_URL, "localhost:2918");
    InitialContext ctx = new InitialContext(p);
```

```
    LoggerService ls = (LoggerService) ctx.lookup("jrun:service/LoggerService");
    ls.logInfo("Message logged using LoggerService");
...
```

JNDI を使用してクラスタ環境の JRun サービスを検索する方法の詳細については、41 ページの「JRun サービスとオブジェクトのクラスタリングの使用」を参照してください。

ユーティリティメソッド

JRun は、実行時 JNDI ツリーにあるすべてのエントリを取り出すために次のメソッドを提供しています。

- **System.out に出力する**　次のメソッドを使用します。

  `public static void jrun.naming.JRunNamingContext.printTree(Context ctx)`

- **文字列を返す**　次のメソッドを使用します。

  `public static String jrun.naming.JRunNamingContext.treeToString(Context ctx)`

これらのメソッドを使用するには、コンパイルクラスパスに <JRun のルートディレクトリ>/lib/jrun.jar を指定します。JRun は、クラスパスに jrun.jar を自動的に含みます。

環境ネーミングコンテキスト (ENC) の使用

ENC (Environment Naming Context : 環境ネーミングコンテキスト) は、アセンブラと開発者が、コンポーネントのソースコードを変更せずに EJB およびサーブレット機能をカスタマイズできる JNDI の場所です。ENC は、JNDI の java:comp/env にあり、J2EE 仕様によって定義されています。

JRun は、起動時に java:comp/env ネーミングコンテキストを作成します。このコンテキストには、デプロイメントディスクリプタに宣言されている、環境エントリとリソースリファレンスのすべてが含まれています。コンポーネントは、`javax.naming.InitialContext` オブジェクトを作成し、java:comp/env の下の ENC を参照することで、エントリにアクセスします。JRun は、各コンポーネントのネーミング環境を、ENC、あるいはその直接または間接サブコンテキストに保管します。

サードパーティ JNDI プロバイダの使用

サードパーティの JNDI プロバイダを、JRun JNDI プロバイダと統合することができます。これは複雑な操作であり、JMS プロバイダを運用環境で使用する前に十分なテストを行う必要があります。詳細については、JNDI 仕様書と、API のドキュメントを参照してください。

メモ：サードパーティの JNDI プロバイダはクラスタすることはできません。

第 9 章
JRun 管理向けデプロイメントディスクリプタ

J2EE テクノロジーで使用する JRun 固有のデプロイメントディスクリプタの他に、JRun はサーバー管理用のデプロイメントディスクリプタを使用しています。これらのディスクリプタは、XML 形式のファイルで、ポート、サービス、リソースなどサーバー固有の機能の設定を指定します。

目 次

サーバー設定：jrun.xml ファイル

各 JRun サーバーには、SERVER-INF/jrun.xml ファイルがあります。このファイルにはサーバー固有の設定情報が含まれています。通常、jrun.xml ファイルを直接変更する必要はありません。JMC には、JRun の設定とカスタマイズを行うために必要な機能のほとんどが含まれています。ただし、jrun.xml を手動で変更することが必要な場合もあります。たとえば、メソッドタイミングを使用するには、Instrumentation サービスの設定を変更します。メソッドタイミ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。

次の表は、jrun.xml ファイルに定義されているサービスのリストです。

サービス	説明
JRunServer	他のすべての jrun.xml 要素をラップします。
ClusterManager	サービスのクラスタリングを容易にし、JINI ルックアップサービスをカプセル化します。
JRunRMIBroker	すべての JRun リモート通信で使用する共通の RMI ブローカーを作成します。このサービスはクラスタすることができます。
JRunTransactionService	トランザクションマネージャを開始します。
LicenseService	JRun のライセンスを管理します。
MetricsService	メトリクスコレクションとロギングを管理します。
SchedulerService	スケジューラを作成します。
LoggerService	JRun のロギング機能を管理します。次のネストされたサービスが含まれます。 • ThreadedLogEventHandler • FileLogEventHandler • ConsoleLogEventHandler
JRunSecurityManagerService	JRun の実行時セキュリティを管理します。
JRunUserManagerService	ユーザーストアの管理を提供します。
ResourceService	ファイルリソースを扱います。
ServletEngineService	JRun サーブレットエンジンを確立します。SessionIDGeneratorService が含まれます。
JRunJMS	JRun とともに使用される JMS プロバイダの設定を提供します。次のネストされたサービスが含まれます。 • JMSSecurityManager • JMSAdapter • JMSServiceWrapper
MailService	SERVER-INF/jrun-resources.xml のメール設定のデフォルトを指定します。
ResourceDeployer	jrun-resources.xml に定義された内容に従って、データソース、JMS 接続ファクトリ、JMS デスティネーション、およびリソースアダプタをデプロイします。

サービス	説明
XDocletService	EJB インターフェイスと EJB、サーブレット、およびカスタムタグデプロイメントディスクリプタの XDoclet 生成を自動化します。
DeployerService	コンポートネントデプロイの設定です。次のネストされたサービスが含まれます。これらのサービスは、特定のデプロイ条件で使用されます。 • RI エンタープライズアプリケーションファクトリ • JRun 3.x EJB コンテナファクトリ • JRun 3.x エンタープライズアプリケーションファクトリ
WebService	JRun Web サーバー (JWS) を管理します。
SSLService	JWS の SSL を管理します。
ProxyService	Web サーバーコネクタの設定を管理します。
InstrumentationService	メソッドタイミングを管理します。
HTMLAgentService	JMX エージェントの HTML アダプタを制御します。この機能を使用して、ブラウザから JRun サーバーの設定を参照できます。
JRunAdminService	JRun サーバーへのリモートアクセスを容易にします。

次のセクションでは、jrun.xml ファイルの編集の解説に続いて、これらのサービスで使用される要素について説明します。

jrun.xml ファイルの編集

jrun.xml ファイルは単純な XML 形式であり、次の要素を使用します。

- **jrun-server**　トップレベルの要素です。jrun.xml ファイルに含まれる他のすべての要素をラップします。
- **service**　サービスを定義します。次のパラメータを使用できます。
  - class　サービスを管理するときに JRun が起動するクラスの完全修飾名を指定します。
  - name　サービスの管理とアクセスのために JRun と他のサービスが使用する名前を指定します。
  - service　埋め込みサービスを指定します。埋め込みサービスには、class および name パラメータが含まれ、属性を使用できます。埋め込みサービスの例については、jrun.xml ファイルを参照してください。
- **attribute**　サーバーの起動時に JRun が初期化するサービス属性を指定します。

次の例は、JRunSecurityManager サービスの service および attribute 要素を示したものです。

```
...
<service class="jrun.security.JRunSecurityManagerService"
         name="JRunSecurityManager">
  <attribute name="bindToJNDI">true</attribute>
  <attribute name="authConfig">{jrun.server.rootdir}/SERVER-INF/auth.config
         </attribute>
```

```
  <attribute name="securityDomain">defaultUser</attribute>
  <attribute name="roleMappingDomain">defaultRole</attribute>
</service>
...
```

JMC によって、特定の jrun.xml 属性の管理が可能ですが、通常は jrun.xml ファイルの属性をテキストエディタで変更します。HTMLAgentService を使用することもできます。これについては 120 ページの「HTMLAgentService」を参照してください。

次のセクションではさらに、jrun.xml ファイルに定義されるサービスと属性についても解説します。

すべてのサービスに共通の属性

次の表は、すべての JRun サービスで利用できる属性のリストです。

属性	説明
BindToJNDI	サービスが JNDI によってアクセス可能であるかどうかを決定します。
Deactivated	サーバーが実行中であるかどうかを決定します。デフォルトは false です。deactivated を false に設定すると、JRun はサービスの **init** メソッドを呼び出しますが、**start** メソッドは呼び出しません。
DomainName	親サービスを指定します。トップレベルサービスは DefaultDomain にあります。
JRunService	現在のオブジェクトインスタンスへのリファレンスを含んでいます。
Logger	**jrunx.logger.Logger** へのリファレンスを保持します。クライアントはこのサービスを参照して **getLogger** メソッドを呼び出すことで、JRun ロガーへのリファレンスを取得できます。
Name	サービス名を指定します。
ParentService	親サービスのクラス名を指定します。
ServerName	サーバー名を指定します。
Status	サーバーのステータスを指定します。この値に設定する定数については、**jrunx.kernel.Service** JavaDocs を参照してください。

JRunServer

JRunServer サービスは、サーバーの属性を定義し、クラスタリング機能を定義するサブサービスも含まれます。このサービスには次の属性が使用できます。

属性	説明
ProductBuildNumber	ビルド番号を指定します。
ProductName	製品名を指定します。デフォルトは、JRun アプリケーションサーバーです。

属性	説明
ProductVendorName	製造元名を指定します。デフォルトは Macromedia です。
ProductVendorUrl	製造元の URL を指定します。 デフォルトは http://www.macromedia.com です。
ProductVersion	バージョン番号を指定します。
RootDirectory	JRun のルートディレクトリを指定します。
ServerRootDirectory	サーバーのルートディレクトリを指定します。

ClusterManager

ClusterManager サービスは、サービスのクラスタリングを容易にし、JINI ルックアップサービスをカプセル化します。クラスタせずに使用する場合は、enabled 属性を false に設定して、このサービスを無効にします。このサービスはコメントアウトできません。代わりに、enabled 属性を false に設定します。

メモ：クラスタリングを有効にする場合、サーバーコンピュータはネットワークに接続されている必要があります。

このサービスには次の属性が使用できます。

属性	説明
ClusterDomain	クラスタ名を指定します。
ClusteredHostAddresses	クラスタ内で、JRun サーバーを含んだホストのリストを指定します。
ClusteredServerNames	クラスタ内の JRun サーバー名を指定します。
Locators	このサービスのロケータを指定します。
TemporaryDirectory	JRun サーバーのテンポラリディレクトリを指定します。
UnicastPeer	クラスタ内の JRun サーバーで、サブネット外にあるものを指定します。JRun は同じサブネットにあるサーバーを自動的に検出しますが、サブネット外にある JRun サーバーについては、unicastPeer によって指定する必要があります。

ClusterDeployerService

ClusterDeployer サービスは、クラスタ内の JRun サーバーにわたって、自動デプロイの管理を行います。

このサービスには次の属性が使用できます。

属性	説明
ContextRoot	コンテキストルートを指定します。
DeployDirectory	クラスタ内のすべての JRun サーバーにわたって、アプリケーションが自動的にデプロイされるディレクトリの位置を指定します。

属性	説明
HotDeploy	ホットデプロイを有効にするかどうかを指定します。true または false を指定します。デフォルトは false です。
PollFrequency	このサービスが変更を参照する頻度を指定します。値は秒単位で指定します。デフォルトは 10 秒です。

NamingService

NamingService は、クラスタ内の JRun サーバーにわたって JNDI のクラスタリングを管理します。

このサービスには次の属性が使用できます。

属性	説明
JRMPPort	JRMP ポートを指定します。
OrbPort	ORB ポートを指定します。デフォルトは 900 です。
Port	JNDI ポートを指定します。リモートクライアントは、<ホスト名>:<JNDI ポート> の組み合わせを参照して、JRun サーバーにアクセスします。
UsingCORBA	JRun サーバーが CORBA を使用するかどうかを指定します。true または false を指定します。デフォルトは false です。

JRunRMIBroker

JRunRMIBroker サービスは、すべての JRun リモート通信で使用される共通の RMI ブローカーを作成します。このサービスはクラスタすることができます。クラスタリングの詳細については、第 3 章、27 ページの「クラスタリング」を参照してください。

このサービスには次の属性が使用できます。

属性	説明
ClientSocketFactory	クライアントのソケットファクトリを指定します。
ClientSocketFactoryClass	クライアントのソケットファクトリのクラスを指定します。
ClusterAlgorithm	ロードバランスのアルゴリズムを指定します。デフォルトは jrunx.cluster.RoundRobinAlgorithm です。jrunx.cluster.StickyRoundRobinAlgorithm や jrunx.cluster.BuddyAlgorithm を指定することもできます。
Port	このサービスのポートを指定します。通常は 0 を指定します。この場合 JRun は実行時にランダムなポートを選択します。
ServerSocketFactoryClass	サーバーのソケットファクトリのクラスを指定します。

JRunTransactionService

JRunTransactionService は、分散トランザクションマネージャを開始します。

デフォルトのトランザクションマネージャは、`java:/TransactionManager` で参照できます。デフォルトのユーザートランザクションマネージャは、`java:/UserTransaction` で参照できます。ドメイン固有のトランザクションマネージャ、およびユーザートランザクションは、すべて次の JNDI バインディングにあります。

- java:{*txDomainName*}/TransactionManager
- java:[*txDomainName*]/UserTransaction

リソースをドメインに割り当てるには、jrun-resources.xml の `transactionDomain` 要素を使用します。また、jrun-ejb-jar.xml にある同じ要素を使用すると、EJB にドメインが割り当てられます。`transactionDomain` 要素が存在しない場合、JRun はデフォルトのトランザクションドメインを使用します。

このサービスには次の属性が使用できます。

属性	説明
TransactionDomain	トランザクションドメインを指定します。
TransactionManager	トランザクションマネージャのクラスを指定します。デフォルトは、jrunx.tyrex.tm.impl.TransactionManagerImpl です。
UserTransaction	ユーザートランザクションサービスを指定します。

DefaultDomain

DefaultDomain サービスは、デフォルトのトランザクションドメインを管理します。

このサービスには次の属性が使用できます。

属性	説明
ClusterEnabled	クラスタリングのためのトランザクションサポートが有効であるかどうかを指定します。true または false を指定します。デフォルトは false です。トランザクション伝達を有効にするには、true に設定します。
Maximum	トランザクションの最大数を指定します。
TXDomainName	トランザクションドメイン名 (DefaultDomain) を指定します。
Timeout	トランザクションのタイムアウト時間を秒単位で指定します。デフォルトは 30 秒です。
WaitTime	トランザクションの待ち時間を指定します。

LicenseService

LicenseService は JRun のライセンスを管理します。

このサービスには次の属性が使用できます。

属性	説明
Edition	JRun のエディションを指定します。
EvalDays	評価期間の元の日数を指定します。
EvalDaysLeft	評価期間の残りの日数を指定します。
ExpirationDate	有効期限を指定します。
Expired	評価ライセンスの有効期限が経過したかどうかを示します。
LastWarningMessage	最後の警告メッセージを指定します。
LicenseEvaluation	ライセンス評価番号を指定します。
LicenseKey	評価ライセンスキーを指定します。
MajorVersion	メジャーバージョンを指定します。
MaxConcurrency	並行処理の最大数を指定します。
Upgrade	このライセンスがアップグレードライセンスであるかどうかを示します。
Valid	このインストールに、正当なライセンスが使用されているかどうかを示します。

MetricsService

MetricsService は、メトリクスロガーが使用する統計のコレクションを管理します。詳細については、第 7 章、89 ページの「接続の監視」を参照してください。

このサービスには次の属性が使用できます。

属性	説明
HistorySize	メトリクス履歴バッファーのサイズを指定します。
InitialSize	メトリクス配列の初期サイズを指定します。
MetricsService	現在のオブジェクトインスタンスへのリファレンスを含んでいます。

SchedulerService

SchedulerService はスケジュールと、バックグラウンドスレッドで内部タスクを実行するときに使用されます。これらのタスクには、セッションタイムアウト、ファイルロギング、メトリクスロギング、JDBC コネクションプールの管理が含まれます。

このサービスには次の属性が使用できます。

属性	説明
ActiveHandlerThreads	スレッドプールのアクティブハンドラスレッド数を指定します。
MaxHandlerThreads	スレッドプールのハンドラスレッドの最大数を指定します。
MinHandlerThreads	スレッドプールのハンドラスレッドの最少数を指定します。

LoggerService

LoggerService は、すべての JRun のロギング機能を管理します。通常は JMC を使用して、ログの設定を確立します。ただし、jrun.xml ファイルでこれらの属性を手動で設定することもできます。

このセクションでは、次に示す LoggerService のサブサービスを解説します。

- スレッド化されたロガーのサブサービス
- ファイルロガーの属性
- コンソールロガーの属性

このサービスには次の属性が使用できます。

属性	説明
DebugEnabled	デバッグレベルのメッセージをログに記録するかどうかを指定します。
ErrorEnabled	エラーメッセージをログに記録するかどうかを指定します。この属性を有効にすると、大量のログ出力が発生します。業務に使用している JRun サーバーでは、この属性を有効にしないでください。

属性	説明
Format	ロギングメッセージの形式を設定します。メッセージ形式を定義する場合、次のコンポーネントが使用できます。 • {date}　yyyyMMdd の形式で表された現在の日付 • {date <format>}　現在の日付の形式。正当な値の説明については、JavaDocs の `java.text.SimpleDateFormat` を参照してください。 • {day}　01 から 31 までの範囲で日付を表す 2 桁の数値 • {month}　01 から 12 までの範囲で月を表す 2 桁の数値 • {year}　年を表す 4 桁の数値 • {hour}　00 から 23 までの範囲で時間を表す 2 桁の数値 • {julian}　現在の日付をユリウス暦で表したもの • {thread.name}　現在のスレッド名 • {thread.hashcode}　現在のスレッドハッシュコード • {thread.id}　現在のスレッド ID。この ID は、8 文字の 16 進数形式のハッシュコードです。 • {log.level}　ログイベントのタイプ (debug、error、info、warning) • {log.name}　ログメッセージを生成した JRun サービスの名前。デフォルトでは、この名前は括弧で囲まれます。 • {log.message}　ログイベントメッセージ • {log.exception}　例外スタックトレース • {log.logger.className}　ログイベントが発生したクラス名 (パッケージを除く) • {log.logger.fullClassName}　ログイベントが発生した完全クラス名 (パッケージを含む) • {log.logger.methodName}　ログイベントが発生したメソッド名 • {log.logger.sourceFile}　ログイベントが発生したソースファイル名 • {log.logger.line}　ログイベントが発生した行番号
InfoEnabled	情報レベルのメッセージをログに記録するかどうかを指定します。
LoggerService	現在のオブジェクトインスタンスへのリファレンスを含んでいます。
MetricsEnabled	メトリクスメッセージをログに記録するかどうかを指定します。業務に使用している JRun サーバーでは、この属性の使用には注意してください。メトリクスロギングの詳細については、第 7 章、89 ページの「接続の監視」を参照してください。
WarningEnabled	警告レベルのメッセージをログに記録するかどうかを指定します。
MetricsLogFrequency	メトリクスロギングが有効な場合に、JRun が統計をログに書き込む間隔を秒単位で指定します。
MetricsLogFormat	メトリクスログメッセージの形式を指定します。詳細については、91 ページの「監視出力形式の設定」を参照してください。

JRun のロギングの詳細については、第 4 章、43 ページの「ロギングメカニズム」を参照してください。

スレッド化されたロガーのサブサービス

このサービスには、ファイルロガーとコンソールロガーを定義するサブサービスが含まれます。属性はありません。

ファイルロガーの属性

このセクションでは、ファイルライターの設定方法について説明します。ファイルライターはイベントを受けとって、ファイルに書き込みます。通常は、JMC によってこれらの属性を設定します。デフォルトでは、このサービスはスレッド化されたロガーサービス内で定義されます。これによって、ログの処理は別のスレッドで行われ、サービスのパフォーマンスを最大にすることができます。このサービスをロガーサービス内で直接定義することもできますが、この場合スレッド化されたロガーのパフォーマンスの利点は失われます。

このサービスには次の属性が使用できます。

属性	説明
CloseDelay	JRun のシャットダウン時に、ログファイルを閉じる前に JRun が待つ時間を、ミリ秒単位で指定します。JRun は指定された時間内に他のメッセージがファイルに書き込まれないと、ログファイルを閉じます。デフォルトは 5000 (5 秒) です。この属性を 0 に設定すると、書き込み終了のたびにファイルが閉じられます。この設定は推奨しません。
DeleteOnExit	JRun のシャットダウン時に、ログファイルを削除するかどうかを指定します。true または false を指定します。デフォルトは false です。
FileSize	ファイルサイズを指定します。
Filename	このファイルロガーによってすべてのイベントが書き込まれるファイル名。{jrun.rootdir} および {jrun.server.name} 変数を使用して、JRun ルートディレクトリおよび JRun サーバー名を指定できます。デフォルト名は {jrun.rootdir}/logs/{jrun.server.name}-event.log です。
Format	105 ページの「LoggerService」を参照してください。
Heading	JRun により最初にログファイルに書き込みが行われるときに、このファイルの先頭に挿入されるログファイルヘッダー。ヘッダーには、タイムスタンプなどのダイナミックな値を含め、任意のテキストを含めることができます。

属性	説明
RotationFiles	JRun が保持するローテーションファイルの数。たとえば、`rotationFiles` を 2 に設定すると、ロギングメカニズムによってログファイルと、2 つのログローテーションファイルが、ログライターのために保持されます。既に 2 つのログファイルがあるときに、あるイベントによってログファイルがローテーションサイズを越えてしまった場合、古い方のログファイルが削除され、新しいファイルが作成されます。 インストール時のデフォルト値は 3 です。
RotationSize	ファイルがローテーションされるまでの、ログファイルのサイズの最大値 (単位はバイト)。ファイルがローテーションされると、ロギングメカニズムによる現在のログファイルへの書き込みは停止され、新しいログファイルが作成されます。すべての新しいイベントは、新しいログファイルに書き込まれます。この属性によって、ログファイルの最大サイズを制御することができます。 `rotationSize` のデフォルト値は 200000 です。この値は、バイト単位、キロバイト単位 (たとえば、10k)、またはメガバイト単位 (たとえば、10m) で指定できます。

コンソールロガーの属性

このセクションでは、コンソールライターの設定方法について説明します。スクリーンライターは、イベントを取得し、標準の出力に書き込みます。通常、出力先はモニタです。詳細については、46 ページの「コンソールライターの使用」を参照してください。

このサービスには次の属性が使用できます。

属性	説明
Format	ログメッセージの形式です。詳細については、105 ページの「LoggerService」を参照してください。

デフォルトでは、このサービスはスレッド化されたロガー内で定義されます。これによって、ログの処理が別のスレッドで行われ、サービスのパフォーマンスを最大にすることができます。このサービスをロガーサービス内で直接定義することもできますが、この場合スレッド化されたロガーのパフォーマンスの利点は失われます。

JRunSecurityManager

このセクションでは、JRun のセキュリティメカニズムの設定に使用される属性を説明します。

このサービスには次の属性が使用できます。

属性	説明
AuthConfig	セキュリティマネージャによって使用される、auth.config ファイルの名前を指定します。auto.config ファイルは、`securityDomain` および `roleMappingDomain` 属性の指定に従って、セキュリティドメインとロールドメインのログインモジュールと、ログインモジュールのパラメータを定義します。
JRunSecurityManager	現在のオブジェクトインスタンスへのリファレンスを含んでいます。
RoleMappingDomain	ロールのログインモジュールの定義を含む、auth.config セクションの名前を指定します。デフォルトは defaultRole です。
SecurityDomain	ユーザーのログインモジュールの定義を含む、auth.config セクションの名前を指定します。デフォルトは defaultUser です。
TrustedHost	信頼できるホストを指定します。この属性は複数回指定できます。こうすると複数の信頼できるホストが指定できます。

JRunUserManager

このセクションでは、JRun ユーザーマネージャメカニズムの設定に使用される属性を解説します。ユーザーマネージャは、ユーザーとロールのセキュリティストアをダイナミックに更新するメカニズムです。

このサービスには次の属性が使用できます。

属性	説明
SecurityStore	ユーザーおよびロールストアの位置を指定します。デフォルトは SERVER-INF/jrun-users.xml です。

ResourceService

ResourceService は、ファイルリソースを扱います。

このサービスには次の属性が使用できます。

属性	説明
ResourceCacheSize	スタティックリソースのキャッシュに使用される内部バッファーのサイズを指定します。
ResourceCheckInterval	キャッシュされたリソースへの変更をチェックする間隔をミリ秒単位で指定します。

ServletEngineService

ServletEngineService は、JRun サーブレットコンテナの設定を含んでいます。

このサービスには次の属性が使用できます。

属性	説明
DefaultLocale	デフォルトのロケールを指定します。デフォルトは en_US です。
MimeMapping	MIME タイプのマッピングを指定します。
MimeTypesPath	MIME タイプの定義を含むファイルを指定します。この定義は、指定されたファイル拡張子について、どのインターネットメディアタイプがクライアントに送信されるかを制御します。デフォルトの位置は /lib/mime.types です。
ServletCheckInterval	サーブレットのチェック間隔を指定します。デフォルトは 1500 ミリ秒です。
ServletEngine	現在のオブジェクトインスタンスへのリファレンスを含んでいます。

ServletEngineService には、SessionIdGenerator サブサービスも含まれます。このサブサービスはデフォルトの属性のみを受け付けます。

JRunJMS

JRunJMS サービスには、次のサブサービスが含まれます。

- JMSSecurityManager
- JMSAdapter
- JMSServiceWrapper

JMSSecurityManager

JMSSecurityManager サブサービスは、JMS SecurityManager を開始します。

このサブサービスには、デフォルトの属性のみが使用できます。

JMSAdapter

JMSAdaptor サブサービスは、JMS プロバイダを開始します。

このサブサービスには次の属性が使用できます。

属性	説明
ClassPath	JMS アダプタのクラスパスを指定します。
ConfigFileName	JMS プロバイダの設定ファイル名を指定します。デフォルトの JRun JMS アダプタでは、この名前は通常は jrun-jms.xml です。SonicMQ の場合は通常は broker.ini です。
HomeDir	JMS プロバイダのホームディレクトリを指定します。JRun JMS アダプタでは、通常、指定する必要はありません。SonicMQ では、SonicMQ ディレクトリの絶対パスを指定します。

属性	説明
ProviderName	プロバイダ名を指定します。
ProviderVersion	プロバイダのバージョンを指定します。

SonicMQ が JMS プロバイダである場合に使用される属性については、58 ページの「SonicMQ の使用」を参照してください。

JMSServiceWrapper

JMSServiceWrapper サブサービスは、J2EE クライアントに JMS アクセスを提供します。

このサブサービスには次の属性が使用できます。

属性	説明
AdapterServerName	クラスタされた JMS で使用されます。詳細については、39 ページの「クラスタされた環境での JMS の使用」を参照してください。
AdapterType	リモートまたはローカルアダプタが、ラッパーによって使用されるかどうかを指定します。リモートかローカルかを指定します。詳細については、39 ページの「クラスタされた環境での JMS の使用」を参照してください。
DefaultQCFName	デフォルトの QueueConnectionFactory の名前を指定します。デフォルトは QueueConnectionFactory です。
DefaultTCFName	デフォルトの TopicConnectionFactory の名前を指定します。デフォルトは、TopicConnectionFactory です。
DefaultTransport	デフォルトのトランスポートメカニズムを指定します。ビルトイン JRun JMS プロバイダについては、デフォルトは RMI です。
DefaultXAQCFName	XA 準拠の QueueConnectionFactory の名前を指定します。デフォルトはありません。
DefaultXATCFName	XA 準拠の TopicConnectionFactory の名前を指定します。デフォルトはありません。
JMSContextFactoryName	クラスタされた JMS で使用されます。詳細については、39 ページの「クラスタされた環境での JMS の使用」を参照してください。
JMSUrl	クラスタされた JMS で使用されます。詳細については、39 ページの「クラスタされた環境での JMS の使用」を参照してください。

MailService

MailService は、JRun サーバーの jrun-resources.xml ファイルの JavaMail 設定のデフォルト値を指定します。

このサービスには次の属性が使用できます。

属性	説明
MailSessionDefaults	JavaMail 設定のサブ属性が含まれます。各属性の詳細については、61 ページの「JavaMail」を参照してください。

ResourceDeployer

ResourceDeployer サービスは、jrun-resources.xml からデータソース、コネクションプール、および JMS 接続ファクトリをデプロイします。

このサービスには次の属性が使用できます。

属性	説明
ConnectionFactoryNames	接続ファクトリを指定します。
DestinationNames	キューとトピックの名前を指定します。
MailSessionNames	メールセッション名を指定します。

XDocletService

XDoclet サービスは、EJB インターフェイスの自動生成と、サーブレット、カスタムタグ、および EJB のデプロイメントディスクリプタを制御します。

このサービスには次の属性が使用できます。

属性	説明
AntMessageLevel	XDoclet の ant ベースのビルド処理で生成されるメッセージの粒度を指定します。有効な値は次のとおりです。 • quiet (デフォルト) • standard • verbose • debug
EjbSourceFiles	JRun が EJB インターフェイスを生成し、ディスクリプタエントリをデプロイする接尾文字のパターンを指定します。監視される EJB ディレクトリにある、この接尾文字を持つ Java ソースファイルはすべて、コメントを持つエンタープライズ bean とみなされます。XDoclet はこのコメントを使用して、EJBHome、EJBObject、ejb-jar.xml、および jrun-ejb-jar.xml ファイルを作成することができます。デフォルトは *Bean.java です。この要素は、各ディレクトリについて 1 回指定します。
PollFrequency	xdoclet がソースを再調査する間隔です。

属性	説明
WarSourceFileList	WarSourceFiles 属性に指定された WAR ソースファイルのリストを表示します。
WarSourceFiles	JRun が サーブレット、カスタムタグ、およびサーブレットフィルタのデプロイメントディスクリプタエントリを作成する、接尾文字のパターンを指定します。監視される WAR ディレクトリにある、これらの接尾文字の 1 つを持つ Java ソースファイルはすべて、XDoclet が適切なデプロイメントディスクリプタ要素を作成するために使用可能なコメントを含みます。デフォルトは、*Servlet.java、*Tag.java、および *Filter.java です。この要素は各ディレクトリについて 1 回指定します。
WatchedEJBDirectory	JRun が XDoclet スタイルのコメントをベースとして、自動的に 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するディレクトリを指定します。
WatchedEJBDirectoryList	WatchedEJBDirectory 属性に指定された、EJB ディレクトリのリストを表示します。
WatchedWARDirectory	JRun が XDoclet スタイルのコメントをベースとして、自動的にサーブレットとカスタムタグデプロイメントディスクリプタのエントリを生成するディレクトリを指定します。
WatchedWARDirectoryList	WatchedWARDirectory 属性に指定された、WAR ディレクトリのリストを表示します。
WebLogicExport	true を指定すると、WebLogic 固有のデプロイメントディスクリプタが生成されます。
WebSphereExport	true を指定すると、WebSphere 固有のデプロイメントディスクリプタが生成されます。

DeployerService

DeployerService は、すべてのタイプの J2EE コンポーネントのデプロイを処理します。このコンポーネントには、Web アプリケーション、EJB、リソースアダプタ、エンタープライズアプリケーションが含まれます。

このサービスには次の属性が使用できます。

属性	説明
ClientSocketFactory	クライアントのソケットファクトリを指定します。
ClientSocketFactoryClass	クライアントのソケットファクトリのクラスを指定します。
ClusterAlgorithm	ロードバランスのアルゴリズムを指定します。デフォルトは jrunx.cluster.RoundRobinAlgorithm です。jrunx.cluster.StickyRoundRobinAlgorithm や jrunx.cluster.BuddyAlgorithm を指定することもできます。
DeployDirectory	ホットデプロイが有効な場合に、JRun が自動的にアプリケーションをデプロイするディレクトリを指定します。この要素は何回でも指定することができます。
DeployDirectoryList	JRun サーバーのデプロイディレクトリを指定します。

属性	説明
DeployedURLs	JRun サーバー上にデプロイされたアプリケーションの URL をリストします。
EAR	JRun サーバー上にデプロイされたエンタープライズアプリケーションをリストします。
EJB	JRun サーバー上にデプロイされた EJB をリストします。
File	システムの起動時に、JRun がデプロイする WAR、EAR、JAR、および RAR ファイルを指定します。この要素は何回でも指定することができます。
HotDeploy	ホットデプロイを有効にするかどうかを指定します。true または false を指定します。デフォルトは true です。
PersistXML	JRun デプロイヤが、自動的に JRun 固有のデプロイメントディスクリプタ (jrun-ejb-jar.xml など) を作成するかどうかを指定します。true または false を指定します。デフォルトは false です。
PollFrequency	このサービスが変更を参照する頻度を指定します。デフォルトは 5 秒です。
RARs	JRun サーバー上にデプロイされるリソースアダプタをリストします。
TemporaryDirectory	JRun がアプリケーションをデプロイするテンポラリディレクトリを指定します。
Url	URL のリストを指定します。
ValidateXML	デプロイヤが、デプロイメントディスクリプタに含まれる XML を検証するかどうかを指定します。
WAR	JRun サーバー上にデプロイされる Web アプリケーションをリストします。

サーバー固有のデプロイヤ

デフォルトの DeployerService の他に、JRun は各種のサーバー固有のデプロイヤを提供しています。これらのデプロイヤによって、他の J2EE アプリケーションサーバー用に作成されたアプリケーションを JRun にデプロイすることができます。JRun は、次のアプリケーションサーバー固有のデプロイヤをサポートしています。

- J2EE リファレンス実装デプロイヤ (EJB、WAR、および EAR)
- JRun 3.1 デプロイヤ (EJB、WAR、および EAR)

デプロイヤは、サーバー固有のデプロイファイルをスキャンして、サーバー固有のアプリケーションを認識します。たとえば、デプロイヤが {warFileName}.runtime.xml ファイルを見つけると、J2EE リファレンス実装 Web アプリケーションであると想定し、対応するデプロイヤを使用します。デプロイヤが認識するには、アプリケーションに META-INF/ejb-jar.xml (mandatoryFingerPrint として参照されます) などの必須ファイルと、{earFileName}_ejb.runtime.xml (potentialFingerPrint として参照されます) など最低 1 つの他のディスクリプタが含まれている必要があります。

`PersistXML` 属性を true に設定した場合、デプロイヤはサーバー固有のデプロイディスクリプタを、JRun 固有のデプロイメントディスクリプタに変換します。

サーバー固有のデプロイヤは次の属性を使用できます。

属性	説明
IIOP	IIOP を有効にするかどうかを指定します。true または false を指定します。デフォルトは false です。
MandatoryFingerPrint	アプリケーションの識別に必要なファイルの名前を指定します。`MandatoryFingerPrint` と `PotentialFingerPrint` の組み合わせによって、アプリケーションサーバーが識別されます。
MandatoryFingerPrintList	すべての MandatoryFingerPrint のリストを指定します。
PersistXML	JRun デプロイヤが、自動的に JRun 固有のデプロイメントディスクリプタ (jrun-ejb-jar.xml など) を作成するかどうかを指定します。true または false を指定します。デフォルトは false です。
PotentialFingerPrint	アプリケーションの識別に必要なファイルの名前を指定します。通常はこの要素を複数回指定します。JRun はアプリケーションサーバーを識別するためにその 1 つを見つける必要があります。`MandatoryFingerPrint` と `PotentialFingerPrint` の組み合わせによって、アプリケーションサーバーが識別されます。
PotentialFingerPrintList	すべての PotiotialFingerPrint のリストを指定します。
TemporaryDirectory	JRun がアプリケーションをデプロイするテンポラリディレクトリを指定します。
ValidateXML	デプロイヤが、デプロイメントディスクリプタに含まれる XML を検証するかどうかを指定します。

WebService

WebService は、JRun Web サーバー (JWS) を制御します。

メモ: 運用環境で JWS を使用しない場合は、deactivated 属性を true に設定してこのサービスを無効にする必要があります。

このサービスには次の属性が使用できます。

属性	説明
ActiveHandlerThreads	アクティブハンドラスレッドの現在の数を示します。
Backlog	接続待ちキューの最大長を指定します。この値が SOMAXCONN (Winsock では 5、Winsock 2 では 2^31) である場合、ソケットを担当する下層のサービスプロバイダは、バックログをプロバイダ固有のデフォルト最大値に設定します。
BindAddress	JWS のバインドアドレスを指定します。デフォルトは * です。
Interface	JMC の [外部 Web サーバーアドレス] フィールドに対応します。このプロパティを使用すると、JRun に接続可能な Web サーバーを制限できます。デフォルトは、すべての Web サーバー (*) です。JRun サーバーは、インターフェイス属性に一致する Web サーバーからのみリクエストを受け付けます。複数の Web サーバーを指定する場合は、"|" で区切ります (例：10.64.7.106|10.64.7.102)。
KeepAlive	JWS が Keep-Alive HTTP ヘッダーを設定するかどうかを指定します。true または false を指定します。デフォルトは false です。
MaxHandlerThreads	アクティブハンドラスレッドの最大数を指定します。
MinHandlerThreads	アクティブハンドラスレッドの最小数を指定します。
Port	JWS によって使用されるポートを指定します。
SocketFactoryName	ソケットファクトリのクラスを指定します。
Timeout	JWS のタイムアウト時間を、秒単位で指定します。デフォルトは 300 秒です。

SSLService

SSLService は、JWS が HTTPS によって SSL を使用できるようにします。SSL の詳細については、21 ページの「Web サーバーコネクタでの SSL の使用」を参照してください。

このサービスには次の属性が使用できます。

属性	説明
KeyStore	JRun キーストアの名前を指定します。デフォルトは {jrun.rootdir}/lib/keystore です。
KeyStorePassword	キーストアのパスワードを指定します。パスワードは 6 文字以上にする必要があり、大文字と小文字を区別します。
Port	JWS の SSL ポートを指定します。
SocketFactoryName	JWS SSL のソケットファクトリ名を指定します。デフォルトは jrun.servlet.http.JRunSSLServerSocketFactory です。
TrustStore	JRun トラストストアの名前を指定します。デフォルトは {jrun.rootdir}/lib/truststore です。

ProxyService

各 JRun サーバーの jrun.xml ファイルにある ProxyService セクションには、JRun Web サーバーコネクタを制御する属性が含まれます。これらの属性は通常は JMC によって管理します。次の表は、これらの属性の説明です。

属性	説明
ActiveHandlerThreads	アクティブハンドラスレッドの現在の数を示します。
Backlog	接続待ちキューの最大長を指定します。この値が SOMAXCONN (Winsock では 5、Winsock 2 では 2^31) である場合、ソケットを担当する下層のサービスプロバイダは、backlog をプロバイダ固有のデフォルト最大値に設定します。
bindaddress	JMC の [Web サーバー IP] フィールドに対応します。この属性は、JRun サーバーが外部 Web サーバーからリスンするアドレスを指定します。デフォルトは * です。JRun と Web サーバーが同じコンピュータに存在する場合、この値はデフォルトに設定されます。
interface	JMC の [IP フィルタリスト] フィールドに対応します。このプロパティを使用すると、JRun に接続可能な Web サーバーを制限できます。デフォルトは 127.0.0.1 です。JRun サーバーは、インターフェイス属性に一致する Web サーバーからのみリクエストを受け付けます。複数の Web サーバーを指定する場合は、"|" で区切ります (例 : 10.64.7.106|10.64.7.102)。* を使用すると、すべての Web サーバーからのリクエストを受け付けます。
KeyStore	キーストアの位置を指定します。
KeyStorePassword	キーストアのパスワードを指定します。
LoadBalancingAlgorithm	ロードバランスのアルゴリズムを指定します。

属性	説明
MaxHandlerThreads	アクティブハンドラスレッドの最大数を指定します。
MinHandlerThreads	アクティブハンドラスレッドの最小数を指定します。
Port	JMC のリスニングポートに対応します。このプロパティは、JRun サーバーが外部 Web サーバーからの接続を受信するポートを指定します。
ServerWeight	加重ラウンドロビン、または加重ランダムロードバランスアルゴリズムを使用する場合の、JRun サーバーの相対的な加重の量を指定します。
SocketFactoryName	ソケットファクトリのクラスを指定します。
StickySessions	Web サーバーコネクタが、リクエストに既存のセッションが含まれるかどうかを検出し、そのリクエストを元の JRun サーバーに自動的に転送するかどうかを示します。
Timeout	ソケット読み取りのタイムアウトを指定します。
TrustStore	トラストストアの位置を指定します。

メモ: SocketFactoryName が jrun.servlet.jrpp.JRunProxySSLServerSocketFactory に設定され、KeyStore、KeyStorePassword、および TrustStore の各属性が設定されている場合、ProxyService が SSL モードで実行されていることを示します。この場合、外部 Web サーバーの設定ファイルで ssl=true を指定する必要があります。

InstrumentationService

InstrumentationService は、JRun メソッドのタイミング機能を制御します。メソッドのタイミ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。

このサービスには MethodInstrumentor サービスが含まれます。次の属性を受け付けます。

属性	説明
ClassName	装置へのクラス名を指定します。この属性には、クラス名の最後に単一のワイルドカード (*) を含めることができます。
ClassNameList	タイム測定が行われているクラスのリストを表示します。
DirectSubclasses	className 属性に指定されたクラスの直接の子について、タイム測定を行うかどうかを示します。true または false を指定します。デフォルトは true です。
ExcludeCallsTo	タイム測定を行わないパッケージを指定します。デフォルトでは、java.*、javax.*、および sun.* がタイム測定から除外されます。
ExcludeCallsToList	除外されるクラスのリストを表示します。
InstrumentCalls	メソッド呼び出しインストルメンテーションを有効にするかどうかを示します。この指定には、メソッドインストルメンテーションと、インストルメントされた各メソッド内のすべてのメソッド呼び出しが含まれます。true または false を指定します。デフォルトは false です。

属性	説明
InstrumentCallsTo	メソッドインストルメンテーションを使用している場合に、どのメソッド呼び出しがインストルメントされているかを指定します。* を指定すると、className 属性に指定されたクラスのすべてのメソッドで実行されるすべての呼び出しについて、タイム測定が行われます。
InstrumentCallsToList	タイム測定が行われる呼び出しのリストを表示します。
InstrumentMethod	タイム測定を行うメソッドの名前を指定します。* を指定すると、className 属性に指定されたクラスのすべてのメソッドについて、タイム測定が行われます。
InstrumentMethodList	タイム測定されるメソッドのリストを表示します。
InstrumentMethods	className 属性に指定されたクラスのメソッドについて、タイム測定を行うかどうかを指定します。true または false を指定します。デフォルトは false です。
LoggingCallEnterLabel	呼び出し開始メッセージをログファイルに書き込むときに使用する接頭辞を指定します。デフォルトは CALL ENTER です。
LoggingCallExitLabel	呼び出し終了メッセージをログファイルに書き込むときに使用する接頭辞を指定します。デフォルトは CALL EXIT です。
LoggingDelimiter	メソッドタイミングメッセージのコンポーネント間に使用される区切り記号を指定します。デフォルトの区切り記号はスペースです。
LoggingMethodEnterLabel	メソッド入力メッセージをログファイルに書き込むときに使用する接頭辞を指定します。デフォルトは METHOD ENTER です。
LoggingMethodExitLabel	メソッド終了メッセージをログファイルに書き込むときに使用される接頭辞を指定します。デフォルトは METHOD EXIT です。
OutputToRequestThread	メソッドタイミング情報をスレッドローカルストレージに書き込み、インライン表示でタイミングフィルタを使用するかどうかを指定します。true または false を指定します。デフォルトは true です。タイミングフィルタの詳細については、JRun プログラマーガイドを参照してください。
OutputToStandardLogger	メソッドタイミング情報をログファイルに書き込むかどうかを指定します。true または false を指定します。デフォルトは true です。

HTMLAgentService

JMX HTMLAgentService は、実行中の JRun サーバーのすべての属性を表示可能にできるサービスです。このサービスを使用するには、サービス要素のコメントを解除し、JRun サーバーを再起動し、ブラウザを開いて < ホスト名 >:<HTML エージェントのポート番号 > の URL を参照します。ユーザー ID とパスワードは、`adminUsers` 属性のものを使用します。

このサービスには次の属性が使用できます。

属性	説明
AdminUsers	HTMLAgentService にアクセスするために、カンマで区切られた < ユーザー ID>:< パスワード > のリストを指定します。
MaxActiveClients	クライアント接続の最大数を指定します。デフォルトの設定は 10 です。
Port	HTMLAgentService のポート番号を指定します。サーバー設定をブラウザで参照するには、URL にこのポート番号を使用します。

JRunAdminService

JRunAdminService は、JRun サーバーへのリモートアクセスを可能にします。

リソース：jrun-resources.xml ファイル

jrun_server/SERVER-INF/jrun-resources.xml ファイルには、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、J2EE リソースの定義が含まれます。これらの設定は JMC を使用して管理します。また、テキストエディタで編集することもできます。詳細については、第 5 章、53 ページの「リソース」と、オンライン HTML ディスクリプタに関するドキュメントを参照してください。

JMS：jrun-jms.xml ファイル

jrun_server/SERVER-INF/jrun-jms.xml ファイルには、ビルトイン JRun JMS プロバイダの設定が含まれます。このファイルの詳細については、57 ページの「Java Message Service」と、オンラインの HTML ディスクリプタに関するドキュメントを参照してください。

索引