JRun 拡張設定ガイド

Windows®、UNIX™、および
Linux™ 用 JRun 3.1

Allaire® Corporation

版権告知

本書とその中に記載されているソフト ウェアは、ライセンス契約のもとに供給され、このライセンスの条項に従ってのみ使用または複製することができます。本書の内容は、情報の提供のみを目的としており、予告なく 変更されることがあります。これについて、Allaire Corporation は一切責任を負いません。Allaire Corporation は、本書の誤りについて一切責任を負いません。

ライセンスによる許可がある場合を除いて、Allaire Corporation の事前の書面による許可なしに、この出版物の一部または全部の複製、検索システムへの保存、あるいは電子的、機械的な記録、または他のいかなる形態や手段による転送を行うことはできません。

ColdFusion および HomeSite は、米国における Allaire Corporation の登録商標です。Allaire、Allaire Spectra、JRun、JRun Studio、ColdFusion Studio、<CF_Anywhere>、ColdFusion ロゴ、JRun ロゴ、および Allaire ロゴは、米国および各国における Allaire Corporation の商標です。MacOS は、Apple Computers Inc. の商標です。Microsoft、Windows、Windows NT、Windows 95、Microsoft Access、および FoxPro は、Microsoft Corporation の登録商標です。Java、JavaBeans、JavaServer、JavaServer Pages、JavaScript、JDK、および Solaris は、Sun Microsystems Inc. の商標です。UNIX は、The Open Group の商標です。PostScript は、Adobe Systems Inc. の商標です。その他の製品および製品名は、各所有者に帰属する商標です。

この製品には RSA Data Security からライセンス供与されたコードが含まれています。このソフトウェアの著作権の一部は、Merant, Inc. に帰属します。1991-2001

部品番号 : AA-JRACG

目次

はじめに

この章では、Web サイト、マニュアル、テクニカル サポートなどの JRun および Allaire リソースにアクセスする場合のガイドラインを説明します。

目次

JRun 製品のラインナップ

JRun は、Sun Microsystems の最新のサーブレット /JSP および EJB 仕様をサポート する Java アプリケーション サーバーです。JRun には、次のエディションがあります。

エディション	説明	価格
Developer 版	Web 開発および EJB/JMS/JTA をサポートしており、JRun JDBC ドライバを含みます。無制限の JVM (Java Virtual Machine) 数、およびサーブレット、JSP、EJB (Enterprise JavaBeans) の 3 つの同時接続数が許可されています。	Web アプリケーションおよび EJB の非営利目的の開発/テスト用のライセンスであり、無償で入手できます。アプリケーションの公開を目的とした使用は許可されていません。
Professional 版	Web 開発のみをサポートしています。JRun Professional 版では、JVM 数、およびサーブレット /JSP (JavaServer Pages) の同時接続数はいずれも無制限です。	営利目的の公開用のライセンスで、CPU 単位のライセンスとなります。
Advanced 版	Allaire ClusterCATS を使用する HTTP ベースのロード バランス機能およびフェイル オーバソフト ウェアが含まれています。Web 開発をサポートしており、JRun JDBC ドライバを含みます。JRun Advanced 版では、JVM 数、およびサーブレット /JSP の同時接続数はいずれも無制限です。	営利目的の公開用のライセンスで、CPU 単位のライセンスとなります。
Enterprise 版	Allaire ClusterCATS を使用する HTTP ベースのロード バランス機能およびフェイルオーバソフト ウェアが含まれています。Web 開発、EJB、JMS (Java Messaging Service)、JTA (Java Transaction API)、および JRun JDBC ドライバをサポートしています。JRun Enterprise 版では、JVM 数、サーブレット、JSP、および EJB の同時接続数はいずれも無制限です。	営利目的の公開用のライセンスで、CPU 単位のライセンスとなります。
Studio 版	HomeSite HTML エディタに基づいた統合 JSP 開発環境です。JRun サーバーは含まれていません。	価格はライセンス単位です。

最新の価格情報については、国内総販売元である (株) アイ・ティ・フロンティアにお問い合わせください (株式会社シリウスは、2001 年 4 月に株式会社アイ・ティ・フロンティアに社名変更いたしました)。

開発者リソース

(株) アイ・ティ・フロンティアでは、開発者の教育、テクニカル サポートなどのサービスにより カスタマ サポート を充実さ せており ます。以下にご紹介する Web サイト では、すべてのオンライン リ ソースにすばやく アクセスできます。次の表に、このよう なオンライン リ ソースにアクセスできる Web サイト の URL を示し ます。

リソース	説明
(株) アイ・ティ・フロンティア JRun のサイト http://cfusion.sirius.co.jp/jrun/	JRun の詳細な製品情報および関連トピック
Allaire 社 Web サイト www.allaire.com	開発元である Allaire 社のサイト
JRun に関する情報 www.allaire.com/products/jrun/	JRun に関する詳細な製品情報と関連トピック
開発者コミュニティ www.allaire.com/developer	JRun による開発に必要な最先端の情報を提供する、オンライン ディスカッション グループ、知識ベース、技術文書などのあらゆるリソース
JRun 開発者センター www.allaire.com/developer/jrunreferencedesk/	サーブレット リソース、開発のヒント、記事、文書、ホワイト ペーパーに関する情報を一括掲載
JRun サポート フォーラム forums.allaire.com/jrunconf	Allaire オンライン フォーラムでは豊かな経験を持つ JRun 開発者と連絡をとり、JRun に関連した数多くのトピックについてメッセージを書き込んだり、回答を得ることができます。

JRun 文書の概要

JRun 文書は、JSP 開発者、サーブレット 開発者、EJB クライアント 開発者、EJB 開発者、システム管理者を含むすべての JRun ユーザ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 オンライン文書には、HTML 形式と Adobe Acrobat ファイル形式があります。

印刷およびオンライン文書セット

JRun の文書セットには、次の文書があります。

文書	説明
『JRun セットアップ ガイド』	JMC を使用した JRun のインストール、設定、および管理について説明します。
『JRun による アプリケーションの開発』	Java サーブレット、JavaServer Pages、および Enterprise JavaBeans から構成される Web アプリケーションの開発方法について 説明します。
『JRun サンプル ガイド』	サーブレット、JavaServer Pages、Enterprise JavaBeans のコード サンプルおよびサンプル アプリケーションを提供します。
『JRun タグ ライブラリ リファレンス』	JRun タグ ライブラリの JavaServer Pages (JSP) カスタム タグについて説明します。
『JRun 拡張設定ガイド』	ISP、ISV、および OEM カスタマ用の JRun のインストール、使用、設定に関する情報があります。
『JRun JSP クイック リファレンス』	JavaServer Pages (JSP) 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構文が記載されています。
『JRun Version 3.1 機能 および移行ガイド』	JRun バージョン 3.1 の機能と、既存のアプリケーションをバージョン 3.1 に移行する方法について説明します。
『JRun タグ ライブラリ クイック リファレンス』	JRun タグ ライブラリの JavaServer Pages (JSP) カスタム タグの簡単な説明と構文について記載されています。

オンライン文書

Allaire 社では、JRun の全文書のオンライン版を Adobe Acrobat (PDF) ファイルで提供しています。PDF ファイルは JRun CD-ROM に含まれ、既定では JRun /docs/pdf 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 ページにある製品の文書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。

また、これらの PDF ファイルは、Allaire 社の Web サイト (www.allaire.com/documents) からダウンロード することも できます。

その他のリソース

本書で扱っ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

サーブレットと JavaServer Pages	
『Java Servlets』	Karl Moss 著、 McGraw Hill 刊、1999 年、 ISBN: 0071351884
『Java Servlet Programming (Second Edition)』	Jason Hunter、William Crawford 著、 O'Reilly & Associates 刊、2001 年、 ISBN: 0596000405
『Core Servlets and Java Server Pages』	Marty Hall 著、 Prentice Hall 刊、2000 年、 ISBN: 0130893404
『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

Enterprise JavaBeans	
『Mastering Enterprise JavaBeans and the Java 2 Platform, Enterprise Edition』	Ed Roman 著、 John Wiley & Sons 刊、1999 年、 ISBN: 0471332291
『Enterprise JavaBeans』	Richard Monson-Haefel 著、 O'Reilly & Associates 刊、2000 年、 ISBN: 1565928695
『Applying Enterprise JavaBeans: Component-Based Development for the J2EE Platform』	Vlada Matena、Beth Stearns 著、 Addison-Wesley Pub Co 刊、2000 年、 ISBN: 0201702673

Enterprise Java プログラミング	
『Professional Java Server Programming J2EE Edition』	Danny Ayers 他著、 Wrox Press 刊、2000 年、 ISBN: 1861004656
『Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition』	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。)
『Building Java Enterprise Systems with J2EE』	Paul Perrone、Venkata S.R. "Krishna" .R. Chaganti 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

オンライン リソース

Java servlet API	java.sun.com/products/servlet
JavaServer Pages API	java.sun.com/products/jsp
Enterprise JavaBeans API	java.sun.com/products/ejb/
Java 2 Standard Edition API	java.sun.com/products/jdk/1.3/docs/api/index.html
Servlet Source	www.servletsource.com
JSP Resource Index	www.jspin.com
Server Side	www.theserverside.com
Dot Com Builder	dcb.sun.com
Servlet Forum	www.servletforum.com

お問い合わせ先

販売元

株式会社アイ・ティ・フロンティア
シリウス事業部

電話 : 03-5562-4099
Fax : 03-5562-4070

http://cfusion.sirius.co.jp/jrun/
E-mail : jrunsales@sirius.co.jp

(株式会社シリウスは、2001 年 4 月に株式会社アイ・ティ・フロンティアに社名変更いたしました)

テクニカル サポート

Allaire 社では、電話および Web による幅広いサポート オプションを提供しています。テクニカル サポート サービスについては、http://www.allaire.com/support/ をご覧ください。

JRun サポート フォーラム (http://forums.allaire.com) へは、いつでも投稿できます。

第 1 章

OEM リソース

この章の説明は、JRun 3.1 用の OEM リソース キットを対象としています。使用している OEM リソースのビルド番号は、global.properties ファイルの jrun.version プロパティで確認してください。

OEM リソースには、OEM 先を対象とした JRun のインストールおよび設定をカスタマイズするファイルが含まれており、OEM 製品と JRun をスムーズに統合できます。OEM リソースを構成するファイル セットは、Allaire 社の Web サイト http://www.allaire.com/partners/oem/jrun/ からダウンロード可能です。この章では、OEM ファイル グループのアーキテクチャと、アプリケーションに JRun を組み込む際に一般的な OEM によって実行される作業について説明します。

メモ

この章の内容は、事前の通知なしに変更する場合があります。JRun OEM キットおよび本書に関する最新のリリース情報については、http://www.allaire.com/partners/oem/jrun/ を参照してください。

目次

OEM リソースについて

OEMリソースは、ご使用のアプリケーションの一部として JRun を統合する際に役立ちます。これらのリソースを使用して、JRun をコンパイルしてアプリケーションにバンドルし、ユーザが実行するインストールおよび設定をカスタマイズできます。Allaire で定義されている OEM には基本的に 2 つの種類があり、ユーザへの対応が次のように異なります。

- JRun をソフトウェアとは別にインストールおよび操作するようユーザに要求する OEM ベンダ
- ユーザに対して透過的に JRun をインストールおよび操作する OEM ベンダ

一般的に、OEM リソースとは後者のことを指します。OEM の顧客が、顧客のアプリケーションとはまったく異なるアプリケーションとして JRun を使用する場合、『JRun セットアップガイド』の手順に従って操作できることが多いためです。このガイドでは、JRun のインストール、設定、および JRun サーバー上での Web アプリケーションの公開について説明しています。

組織によっては、この 2 種類の OEM の特長を併せ持っている場合があります。たとえば、JRun のインストールはユーザに対して透過的に行い、サーブレット エンジンのパフォーマンスの調整や管理については、JRun 管理コンソール (JMC) を使用してユーザ自身が行うように要求することがあります。

OEM ソフトウェアの必要条件

JRun のインストールを変更して独自のインストール ファイルを生成するには、次の表に示す最低限のシステム要件を満たしている必要があります。

プラットフォーム	必要条件	URL
UNIX	GNU make	
Windows	InstallShield 6.2 以降のバージョン	www.installshield.com
	Web (PFW) 3.0 以降のバージョン	www.installshield.com
	のパッケージ	www.cygnus.com
	CYGNUS の Cygwin	

JRun OEM 版と製品版の相違点

JRun 3.0 の製品版と OEM 版では、いく つかの相違点があり ます。OEM 版では省スペースおよび再配布に重点を置いているため、製品版の JRun に含まれている内容の一部が含まれていません。製品版をインストールした場合は、インストールした OEM に製品版から必要なファイルをコピーできます。次の表は、OEM 版と製品版の違いを示し ます。

リソース	OEM 版	製品版
コンパイル済みコネクタ	コネクタのソース コード が含まれていますが、すべてのコンパイル済みコネクタのバイナリ が含まれているわけではありません。	すべてのコンパイル済みコネクタのバイナリが含まれています。サポート されていないプラットフォ ームのコ ネクタ が必要な場合は、ソース コード を使用してコ ネクタをコンパイルできます。お持ち のコ ネクタ のコン パイ ルについては、『JRun セットアップ ガイド』を参照してください。
サンプル	一部のサンプル コードやデモ アプリケーションは含まれていません。	すべてのサンプル コードとデモ アプリケーションが含まれています。
文書	付属の文書が含まれていません。文書は Allaire 社の Web サイト (www.allaire.com/documents) からダウンロード できます。	すべての文書が含まれています。
JRE (Java Runtime Environment)	JRE が含まれていません。JRE の選択およびインスト ールについては、JRun セット アップ ガイド 』を参照してください。	JRE が含まれています (Windows のみ)。

OEM コンポーネント

次の表は、OEM で使用許諾されるコンポーネントの組み合わせとコンポーネントのサイズの概算を示します。

コンポーネント	サイズ (概算)
JSP/サーブレット サーバー	1 MB
EJB トランザクション サーバー (JTA)	300 KB
EJB トランザクション サーバーおよびメッセージ サーバー (JTA/JMS)	500 KB
完全なサーバー (サーブレット /JSP/JTA/JMS)	2 MB
Allaire ClusterCATS	6 MB

上記のサイズは、関連する JAR ファイルの合計に基づく概算です。これには、文書やサンプル アプリケーションは含まれていません。

詳細情報

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 とホスティングについては、xi ページの「お問い合わせ先」を参照してください。

ファイル グループの概要

Allaire 社の OEMリソースでは、makefile スクリプト、/installers ディレクトリ、および /dist ディレクトリが提供されています。これらのリソースは、Windows の場合は jrun-31-win-oem-us.zip、UNIX の場合は jrun-31-unix-oem-us.tgz の各ファイルから解凍されます。

次の表では、OEMリソースに含まれるファイルおよびディレクトリについて説明しています。OEM 契約によっては、この情報が異なる場合があります。

ディレクトリ	ファイル グループ	説明
/	README-KIT.txt	OEMリソースに関する最新の注意事項が含まれています。作業を進める前に、必ずこのファイルをお読みください。
/DIST	/core /core-props /core-unix /core-x86	コンポーネントに依存しない JRun コア ファイルが含まれます。 /core には、JNDI、メール、JDBC JAR ファイルなどの JRun ファイルが含まれます。 /core-props には、JRun サーバーの既定プロパティ ファイルおよび global.properties が含まれます。 /core-unix および /core-x86 には、プラットフォーム固有のファイルが含まれます。 UNIX へのインストールでは、/core、/core-props、および /core-unix を使用します。 Windows へのインストールでは、/core、/core-props、および /core-x86 を使用します。
	/ejipt /ejipt-props /ejipt_samples	JRun の EJB (Enterprise JavaBeans) および JMS (Java Messaging Service) コンポーネントのファイルが含まれます。
	/jrun /jrun-props	サーブレット コンポーネントに加え、プラットフォームに依存しないコンポーネント およびコネクタが含まれます。/jrun-props には、JRun プロパティ ファイルが含まれます。インストール後、これらのプロパティ ファイルに格納された情報がコア ファイル グループに格納されている情報に連結されます。

ディレクトリ	ファイル グループ	説明
	/jrun-aix /jrun-apidoc /jrun-hpux /jrun-solaris /jrun-x86 /jrun-x86-linux	ネイティブ コネクタ、ライブラリ、フィルタなどのプラットフォーム固有のファイルが含まれます。一部の UNIX ファイル グループの場合、該当するプラットフォーム対応のコンパイラが含まれることもあります。
	/jrun-manuals	Windows のみ。空のディレクトリで、PDF 形式の JRun 文書セットを格納するように設定できます。
	/jsp	JSP コンポーネント が含まれます。JSP サポート が不要な場合は、このファイル グループを削除してください。
	/web-apps	JMC および既定の JRun アプリケーション用の WAR ファイルが含まれます。
/installers	/JRun 3 /JRun 3 DemoShield /JRun 3 Web DemoShield	InstallShield を使用して Windows へのインスト ールを行う 場合に必要なファイルがすべて含まれています。

/dist ディレクトリの各ファイルセットには、ディスクに書き込まれるファイルのサブセット が含まれています。 makefile によって、構築中にこれらのツリ ーが 1 つのツリ ーに配置されます。ファイルを追加するには、そのファイルをファイルセット ディレクトリの 1 つに配置するか、makefile に新規のファイルセット ディレクトリを追加します。

製品版 JRun からネイティブコネクタなどの必要な非 Java ファイルを入手し、手作業でこのキットに配置することもできます。

JRun インストールのカスタマイズ

次に示すいくつかの方法で、JRun のコンパイルおよびインストールをカスタマイズしたり、独自の Web アプリケーションをカスタマイズすることができます。

- **makefile のカスタマイズ** makefile を使用すると、より簡単にカスタム アプリケーションを JRun インストールに組み込み、ファイル グループを再構成できるようになります。詳細は、9 ページの「makefile のカスタマイズ」を参照してください。
- **インストール ファイルの変更** JRun では、InstallShield (Windows 用) およびシェル スクリプト (UNIX 用) の 2 種類のインストーラを使用して、ファイルを顧客のコンピュータにコピーします。Linux も含め UNIX の場合はすべて、1 組のシェル スクリプトが使用されます。UNIX および Windows 環境におけるインストール手順については、10 ページの「UNIX JRun インストーラの構築」および 13 ページの「Windows JRun インストーラの構築」を参照してください。
- **JRun プロパティ ファイルの変更** 製品版 JRun のインストールでは、管理者パスワード、ポート番号、JDK/JRE などの設定パラメータに関するプロンプトがユーザに対して表示されます。これらの設定値はすべて JRun プロパティ ファイルに保存されます。ユーザは `PropertyScript` ユーティリティを使用して、顧客に通知せずに JRun プロパティファイル内の JRun の設定を変更できます。
  `PropertyScript` ユーティリティの使用方法については、30 ページの「JRun プロパティのカスタマイズ」を参照してください。
- **外部 Web サーバーへの JRun の接続** 通常、インストール プロセスの最終ステップでは、コネクタを介して Web サーバーと通信できるように JRun サーバーを設定します。製品版 JRun 3.0 のユーザは、JMC のコネクタ ウィザードを使用して、外部 Web サーバー用のコネクタを設定します。OEM の場合は、ユーザには表示されない `ConnectorInstaller` ユーティリティを使用することもできます。詳細は、40 ページの「外部 Web サーバーへの JRun の接続」を参照してください。
- **OEM アプリケーションの公開** インストーラに内蔵されている機能を使用して、Web アプリケーションを JRun インストールに組み込むことができます。詳細は、19 ページの「Web アプリケーションのカスタマイズ」を参照してください。

JRun でユーザ設定をカスタマイズする別の方法については、35 ページの「JRun の組み込み」を参照してください。

makefile のカスタマイズ

Windows と UNIX のどちらで JRun インストーラをコンパイルする場合も、顧客に提供するインストール実行パッケージ (インストーラ) は makefile を使用して生成されます。このインストーラでは、カスタムアプリケーションを含めたり、JRun の不要なコンポーネント (JSP サポート、文書、デモアプリケーションなど) を除外することができます。

makefile によって、/dist ツリーからインストーラにファイルが組み込まれます。UNIX の場合、makefile は mk-sh-installer を使用してインストーラをコンパイルします。Windows の場合、makefile は InstallShield を呼び出して最終的な実行可能モジュールをコンパイルします。いずれの場合も、インストールの構築に使用するファイルソースは /dist ツリーです。/dist ツリーの内容については、6 ページを参照してください。

makefile は次の `make` コマンドを使用して呼び出されます。

```
% make
```

メモ

Windows ベースのシステムで `make` コマンドを使用するには、CYGNUS の Cygwin などの UNIX シェルエミュレータを使用する必要があります。CYGNUS の Cygwin は http://www.cygnus.com からダウンロードできます。

---

makefile をカスタマイズする場合、Web アプリケーションの追加と削除 (19 ページの「Web アプリケーションのカスタマイズ」を参照) や、JRun の主要コンポーネント (EJB/JMS サポートなど) のカスタマイズ (15 ページの「JRun コンポーネントのカスタマイズ」を参照) などの作業を通常行います。また、makefile にコンポーネント、アプリケーション、その他のファイルを追加したり、削除する場合には、インストールスクリプト (UNIX) および InstallShield ファイル (Windows) の更新が必要になることもあります。

UNIX JRun インストーラの構築

UNIX OEMリソースでは、サポートされているすべての UNIX プラットフォームに対して 1 つのインストール スクリプトを使用します。

クイック スタート

このセクションの手順に従って、まったく変更を加えずに UNIX JRun インストーラを構築します。以降のセクションでは、Web アプリケーションやコンポーネントの追加と削除を行ったり、インストール自体を変更できるように JRun をカスタマイズする方法について説明します。

UNIX JRun インストーラを構築するには

1 ディレクトリに jrun-31-unix-oem-us.tgz を解凍します。次に例を示します。

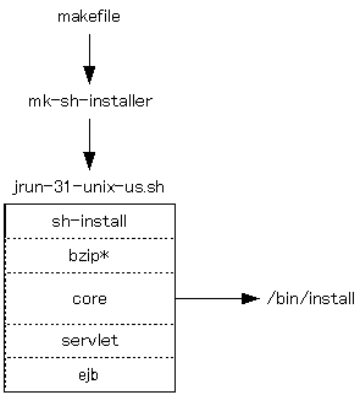
```
% mkdir jrun-kit
% gunzip <jrun-31-unix-oem-us.tgz | (cd jrun-kit; tar xf -)
```

2 make コマンドを実行します。次に例を示します。

```
% cd jrun-kit/dist
% make -i jrun-31-unix-us.sh
```

構築プロセスについて

makefile は、要求された JAR ファイル、WAR ファイル、ライブラリ、および /dist ツリー内のプロパティ ファイルを検証します。次に mk-sh-installer を呼び出し、sh-install および残りのコンポーネントをスクリプト ファイルに挿入して、インストール ファイルを構築します。次の図はこのプロセスを示しています。



ユーザが jrun-31-unix-us.sh ファイルを実行して JRun をインスト ールすると、sh-install は最初のプロンプト を表示し、続いてすべての JRun ファイルをユーザのディスクにコピーします。sh-install は制御を /bin/install に渡し、JRun サーバー、EJB エンジン、JVM 設定などの残りのユーザ プロンプトでインストールおよび設定を完了します。

次の表は、ファイルごとのユーザ プロンプト の一覧です。この一覧を参考にして、さまざまなインストーラ プロンプトの検索、独自のプロンプトの追加、不要なプロンプトの削除、および既存のプロンプトの変更を迅速に行うことができます。

ファイル	ユーザ プロンプト
/installers/sh-install	ライセンス契約の承認 JRun インストール ディレクトリ インストールのタイプ (標準/カスタム) およびインストールするコンポーネント
/dist/core-unix/bin/install	JRE または JDK の選択 JRun ライセンス キー JRun の管理者ユーザ用パスワード JMC Web サーバー ポート ニュースレターまたは通知の受信 (名前および電子メール アドレスの入力を要求)

javadocs の構築時にエラーが発生する場合がありますが、これらのエラーは無視してかまいません。

インストール スクリプトのパッケージ

標準の UNIX インストール スクリプトを構築する際に、makefile は mk-sh-installer を呼び出します。これにより、JRun インスト ール スクリプト (*.sh) が作成されます。makefile を使用せずに mk-sh-installer を呼び出すこともできます。この場合、mk-sh-installer は変更済みファイルを含む JRun インスト ール スクリプト のパッケージを作成します。このスクリプト は /installers ディレクトリ内にあり、jrun-31-unix-us.sh と呼ばれます。

mk-sh-installer スクリプトでは、次の構文が使用されます。

```
mk-sh-installer script-file core-directory servlet-directory
        ejb-directory output-file [compressor] [compressor_path]
```

次に、引数の簡単な説明を示します。

引数	説明	スクリプトに格納される際の表記
script-file	sh アーカイブを解凍するためにシェルスクリプトで使用するテンプレート	INSTALL_SCRIPT
core-directory	/dist/core ディレクトリの位置	CORE_DIRECTORY
servlet-directory	/dist/jrun ディレクトリの位置	SERVLET_DIRECTORY
ejb-directory	/dist/ejipt ディレクトリの位置	EJB_DIRECTORY
output-file	生成された sh アーカイブの名前	OUTPUT_SH
compressor	(オプション) シェル スクリプトを圧縮するためのコマンド。スクリプトは、このコマンドを使用して sh ファイルを構築するときの出力を圧縮します (例 : bzip2、gzip)。 圧縮コマンドを指定しない場合は、mk-sh-installer は UNIX の compress ユーティリティを既定で使用します。	$6
compressor_path	(オプション) アーカイブ ユーティリティへのパス。 このオプションは、汎用 UNIX インストーラを使用するなど、compress ユーティリティを使用しない場合には必須です。	DECOMP_PATH

サイレント インストール オプションの使用

UNIX システムでは、JRun OEM 版のサイレント インストールを実行できます。インストールの処理方法を決める 2 つのスイッチをインストール スクリプト内で設定できます。設定可能なオプションは次のとおりです。

- SILENT

  スクリプトをサイレント モードで実行するように指示します。サイレント モードでは、プロンプトが画面に表示されません。このオプションを設定する場合は、NOPAUSE も設定するか、ファイルからの入力をインストール スクリプトに転送する必要があります。NOPAUSE の設定もインストール スクリプトへの入力も行わないと、インストール スクリプトは最初のプロンプトで停止しますが、ユーザにはスクリプトが入力待ち状態で停止していることがわかりません。

  このオプションを有効にするには、X に設定します。次に例を示します。

  ```
  SILENT = "X"
  ```

- NOPAUSE

  スクリプトを停止せず、ユーザに入力を要求しないように指示します。NOPAUSE を設定すると、JRun のインストール時にデフォルト値が使用されます。プロンプトは画面に表示されますが、ユーザは値を入力できません。SILENT スイッチを設定すると、プロンプトを非表示にすることができます。

  このオプションを有効にするには、X に設定します。次に例を示します。

  ```
  NOPAUSE = "X"
  ```

サイレントインストールの実行時は、ファイル転送を使用してインストールプロンプトに入力できます。このテクニックの例については、43 ページの「ConnectorInstaller とファイル転送の併用」を参照してください。

Windows JRun インストーラの構築

Windows OEM リソースでは、InstallShield を使用して JRun インストーラを構築します。

メモ

JRun OEM リソースは、InstallShield 6.2 および Package for the Web バージョン 3.0 を使用して構築されています。InstallShield 6.0 (および Package for the Web バージョン 2.0) を使用している場合は、アップグレードしてから作業を続行する必要があります。

クイック スタート

このセクションの手順に従って、まったく変更を加えずに Windows JRun インストーラを構築します。以降のセクションでは、Web アプリケーションやコンポーネントの追加と削除を行ったり、インストール自体を変更できるように JRun をカスタマイズする方法について説明します。

メモ

Windows インストーラには、InstallShield、Package For the Web、および make コマンドを使用するための Cygwin などの UNIX エミュレータが必要です。詳細は、3 ページの「OEM ソフトウェアの必要条件」を参照してください。

Windows JRun インストーラをコンパイルするには

1 jrun-31-win-oem-us.zip ファイルを解凍します。次に例を示します。

```
% mkdir jrun-kit
% cd jrun-kit
% jar xf jrun-31-win-oem-us.zip
```

2 Z ドライブを OEM リソースの場所にマップします。次に例を示します。

```
% subst z: <JRun の OEM ルート>
```

InstallShield は、仮想ドライブ Z を使用してそのファイル グループを見つけます。つまり、Windows NT の `subst` コマンドを使用して、構築する前に Z ドライブを JRun の OEM リソース ルートに割り当てておく必要があります。

メモ

Z などの仮想ドライブから JRun3.ipr を開くと、InstallShield 6.2 によって既定のメディアが再構築されません。InstallShield でプロジェクトを開くときは、仮想ドライブ以外のパスを使用してください。

---

3 makefile 内で InstallShield の変数のパスを正しく設定します。詳細は、15 ページの「InstallShield の変数の更新」を参照してください。

4 `make` コマンドを実行します。次に例を示します。

```
% cd dist
% make jrun-31-win-us.exe
```

問題が発生した場合は、14 ページの「構築プロセスについて」を参照してください。

構築プロセスについて

makefile (`make` コマンドの暗黙引数) は、要求された JAR、WAR、ライブラリ、および /dist ツリー内のプロパティ ファイルを検証してから InstallShield を呼び出し、JRun 実行可能モジュールを作成します。

InstallShield プロジェクト ファイル (jrun 3.ipr) および関連ファイルは /installers/JRun 3 にあります。/dist ツリーのディレクトリは InstallShield プロジェクトのファイル グループに対応しています。

makefile ファイルによって、InstallShield ディレクトリの _isres.dll が /jrun-kit /installers ディレクトリの _isres.dll のコピーで上書きされます。makefile を実行する前に、_isres.dll ファイルのバックアップを作成してください。

次の表では、/installers ディレクトリの内容について説明しています。

ディレクトリ	内容
/installers/ClusterCATS	ClusterCATS PFW ファイル
/installers/JRun 3	InstallShield プロジェクト および PFW ファイル、ファイル グループ定義、およびその他の JRun インスト ール用関連ファイル
/installers/JRun 3 DemoShield	JRun DemoShield プロジェクト ファイル
/installers/JRun 3 Web DemoShield	JRun Demo PFW ファイルおよび関連ファイル
/installers/tools	DemoShield プレーヤー ファイル

InstallShield の変数の更新

makefile 内の次の変数が InstallShield のファイルを参照していることを確認する必要があります。

- `INSTALLSHIELD_DIR` 変数は InstallShield 6.2 ディレクトリを参照している必要があります。次に例を示します。

  ```
  INSTALLSHIELD_DIR=//c/Progra~1/InstallShield/InstallShield
       Professional 6.2
  ```

- `PFTWBIN` 変数は Package for the Web 2.0 ディレクトリを参照している必要があります。次に例を示します。

  ```
  PFTWBIN=//c/Progra~1/InstallShield/PackageForTheWeb 2
  ```

makefile が PFW の正しいバージョンを参照していることを確認します。この参照は、使用している InstallShield のバージョンと一致していなければなりません。

JRun コンポーネントのカスタマイズ

JRun の主要コンポーネントは次の 3 つです。

- サーブレット エンジン (必須)
- JSP エンジン
- EJB/JMS

JSP または EJB/JMS のサポート を削除することによって、JRun インストーラを軽量化することができます。ただし、サーブレットは JRun に必要不可欠なコンポーネントなので削除できません。

UNIX および Windows のいずれの場合も、コンポーネント を削除するには、makefile を編集して適切なファイルまたはディレクトリをスキップするように設定します。Windows では、InstallShield ファイルの変更が必要になる場合もあります。不要なファイル グループを削除します。Windows インスト ール用の InstallShield プロジェクトでは、ユーザがカスタム インスト ールを選択した場合に [コンポーネント] ダイアログ ボックスを編集する必要があります。UNIX インスト ーラでは、sh-install を編集します。

複数の JRun インスタンスのインストール

さまざまな理由から、JRun 3.1 を旧バージョンの JRun とともにインストールすることが必要な場合があります。

- 開発者が Web アプリケーションを 3.0 から 3.1 に移行する場合は、両方のバージョンで作業する必要があります。この変更については、『JRun Version 3.1 機能および移行ガイド』を参照してください。
- 顧客の OEM アプリケーションで旧バージョンの JRun が使用される場合があります。

複数のバージョンの JRun を実行する場合の最も一般的な問題は、すべての JRun サービスに固有のポートを選択することです。このセクションでは、同じマシンで JRun の 2 つのバージョンを実行する方法について説明します。

JRun のインストール

ほとんどの場合、JRun 3.1 と JRun 3.0 を同じマシンにインストールして同時に実行できます。ただし、同じバージョンの JRun の複数のコピーを Windows マシンにインストールすることはできません。

次に、複数のインストール バージョンが確実に共存できるようにする方法について説明します。

- 新バージョンの JRun によって固有のポート番号のみが要求されるように、JRun 3.0 の実行中に JRun 3.1 をインストールします。
- admin JRun サーバーおよび default JRun サーバーの local.properties ファイルを編集します。固有のポート番号を ejb.classserver.port と ejb.homeport に割り当てます。JRun のポートについては、『JRun セット アップ ガイド』を参照してください。
- (Windows のみ) 旧バージョンを NT サービスとして実行している場合は、JRun 3.1 を NT サービスとしてインストールしないでください。サービス名が同じであるため、競合が発生する場合があります。ただし、独自の JRun サーバーを追加し、JRun のバージョンによってサーバーを一意に識別する場合は、両方ともサービスとして実行できます。また、37 ページの「Windows NT サービス名の変更」の指示に従って、default JRun サーバーの NT サービス名を変更できます。

旧バージョンの JRun のインストール

旧バージョンの JRun は、配布 CD のルート レベルに収録されています。また、Allaire 社の Web サイトからもダウンロードできます。ファイル名が異なる点を除き、『JRun セット アップ ガイド』の手順に従ってインストールできます。

JRun のルート ディレクトリの調査

JRun の追加コピーをインスト ールするとき、JRun のルート ディレクトリ の調査が必要になる場合があります。このオプションを使用すると、環境変数で jrun コマンドがどの JRun インスタンスにマッピングされているかにかかわらず、呼び出すインスタンスを指定できます。

-jrundir オプションを使用すると、JRun サーバーの起動時にコマンド ラインで JRun のルート ディレクトリを指定できます。同じマシンにインストールされている複数の JRun インスタンスを区別するには、このオプションを使用します。

次に例を示します。

```
% jrun -jrundir c:¥oemcompany¥servlet_engine start -default
```

インスト ールされている JRun インスタンスが 1 つだけの場合、このオプションは必要ありません。

jrun プロセスでは、次の手順に従ってルート ディレクトリが調べられます。

1 -jrundir "*dir*" を最初の 2 つの引数として受け入れます。

2 JRUN_HOME 環境変数の値を取得します。

3 jrun.exe (Windows) またはシェル スクリプト (UNIX) の場所に基づいてルート ディレクトリを調べます。

  Windows でのインストールの場合、JRun によってレジストリ が使用されることはありません。

バージョン情報の取得

JRun には、インストール済みのバージョンに関する追加情報を取得するための 2 つのコマンド ライン オプションがあります。これらのオプションは、アップグレードのテストやクライアントの問題のデバッグに使用できます。オプションは次の 2 つです。

- `-info`
- `-version`

info

```
jrun -info
```

`-info` オプションでは、現在の JRun インストールに関する情報が返されます。次に例を示します。

```
% bash$ ./jrun -info
JRun 3.1
Version 3.1.12345
Developer Edition
```

Windows で `-info` オプションを使用するには、コマンド ラインから JRun を Java への入力として実行します。次に例を示します。

```
% java JRun -info
```

クラスパスに jrun.jar と servlet.jar を指定する必要があります。

version

```
jrun -version
```

`-version` オプションでは、JRun の現在のバージョンが返されます。

UNIX での例は次のとおりです。

```
% bash$ ./jrun -version
3.1.12345
```

Windows で `-version` オプションを使用するには、コマンド ラインから JRun を Java への入力として実行します。次に例を示します。

```
% java JRun -version
```

クラスパスに jrun.jar と servlet.jar を指定する必要があります。

Web アプリケーションのカスタマイズ

インスト ール時に公開される Web アプリ ケーショ ンへの参照を追加、変更、または削除することによって、インストールされるソフトウェアおよびファイルを追加したり削除できます。WAR ファイルを web-apps ファイルセットに追加し、インストール時にアプリケーションを公開する UNIX インストール スクリプトまたは Windows InstallShield スクリプトのいずれかを変更します。アプリケーションを削除するには、ファイルを削除してスクリプトを変更します。

既定の JRun インストールでは、default と admin の 2 つの JRun サーバーが作成され、次の Web アプリケーションがセットアップされます。

- default-app
- demo-app
- invoice-app
- jmc-app
- web-rds

以降のセクショ ンでは、JRun インスト ール時に公開さ れるアプリ ケーショ ンを追加およ び削除する 方法について説明し ます。

- JRun インストールに含まれている既定のアプリケーションを削除する。詳細は、20 ページの「JRun Web アプリケーションの削除」を参照してください。
- インストール時に OEM アプリ ケーショ ンを公開する。詳細は、21 ページの「Web アプリケーションの追加」を参照してください。
- JRun のインストールおよび設定後に OEM アプリケーションを公開する。詳細は、23 ページの「Web アプリケーションの公開」を参照してください。

さらに、WebAppController インターフェイスでは、これらの作業の一部を実行できます。詳細は、28 ページの「Web アプリケーションの制御」を参照してください。

JRun Web アプリケーションの削除

インストールファイルを小さくする目的で、または既定のアプリケーションを顧客に対して非表示にする目的で、既定のアプリケーションの一部またはすべてを削除することが必要な場合があります。最も一般的には、default-app、demo-app、invoice-app、および rds-app アプリケーションを削除します。顧客が JRun 管理コンソール (JMC) を使用して JRun を設定できるようにする場合は、jmc-app をインストールから削除しないでください。

メモ
JRun インストーラコンパイルに変更を加える前に、すべてのファイルを必ずバックアップしてください。

このセクションでは、InstallShield プロジェクト (Windows) およびインストールスクリプト (UNIX) を編集して、Web アプリケーションを削除する方法について説明します。アプリケーションが実行可能モジュールに組み込まれないように (つまり、必要以上にファイルサイズが大きくならないように) makefile を編集する必要もあります。Windows と UNIX での rds-app の削除例については、21 ページの「例 : web-rds の削除」を参照してください。

Windows でのアプリケーションの削除

Web アプリケーションを削除するには、makefile だけでなく InstallShield プロジェクトも編集する必要があります。アプリケーションを削除するには、InstallShield プロジェクトの jrun.rul ファイル内の undeployApp 関数を使用します。

setup.rul で deploy 関数への呼び出しをコメント化し、インストール時に非表示にするアプリケーションを選んで除外することもできます。

UNIX でのアプリケーションの削除

/dist/core-unix/bin/install ファイルの "#deploy apps" セクションを編集して、インストール時に非表示にするアプリケーションを削除します。アプリケーションの前に # を付けてコメント化するか、行を削除します。

次のコード例は、既定でインストールされる 3 つの JRun アプリケーションを示します。demo-app アプリケーションはコメント化され、無効になっています。

```
# deploy apps
deploy "$JRUNROOTDIR/web-apps/default-app/default-app.war" default
        default-app / "$JRUNROOTDIR/servers/default/default-app"
deploy "$JRUNROOTDIR/web-apps/jmc-app/jmc-app.war" admin jmc-app
        / "$JRUNROOTDIR/servers/admin/jmc-app"
#deploy "$JRUNROOTDIR/web-apps/demo-app/demo-app.war" default demo-app
        #/demo "$JRUNROOTDIR/servers/default/demo-app"
```

例 : web-rds の削除

web-rds アプリケーションは JRun Studio を使用可能にするために使用します。JRun Studio がインストールの一部である場合を除き、このアプリケーションを製品とともに配布する必要はありません。このアプリケーションの自動公開を無効にするには、次のセクションで説明されている手順に従ってください。

Windows で web-rds を無効にするには

1 *JRun キットへのパス*¥installers¥JRun 3¥Script Files¥ComponentEvents.rul ファイルを開きます。

2 `RDSComp_Installed = TRUE` を `RDSComp_Installed = FALSE` に変更します。

3 `RDSComp_Uninstalled = TRUE` を `RDSComp_Uninstalled = FALSE` に変更します。

4 Events.rul ファイルを保存して閉じます。

UNIX で web-rds を無効にするには

1 *JRun キットへのパス*/dist/core-unix/bin/install ファイルを開きます。

2 次の行を削除するか、コメント化します。

```
deploy "$JRUNROOTDIR/web-apps/rds-app/rds-app.war" admin rds-app
       /CFIDE "$JRUNROOTDIR/servers/admin/rds-app"
```

3 install ファイルを保存して閉じます。

Web アプリケーションの追加

アプリケーション ファイルをファイル セットに追加し、インストール スクリプトを変更することにより、インストール時に Web アプリケーションが公開されるように設定できます。このセクションでは、Windows および UNIX での Web アプリケーションの追加方法を説明します。

インストールするアプリケーションが 2 つ以下の場合、お客様のアプリケーションを公開する最も簡単な方法は、OEM リソースに付属する demo-app および default-app インフラストラクチャを再利用することです。これらのアプリケーションの WAR ファイルは /dist/web-apps にあります。その WAR ファイルをいずれかのディレクトリにコピーし、WAR ファイル名を JRun アプリケーションの名前に変更できます。たとえば、YourApp.war を /demo-app ディレクトリにコピーし、YourApp.war を demo-app.war という名前に変更します。

アプリケーションのディレクトリ構造に /demo-app または /default-app という名前を付けない場合、あるいは公開するアプリケーションが 3 つ以上ある場合は、次のセクションの説明に従ってください。

メモ

JRun インストーラ コンパイルに変更を加える前に、すべてのファイルを必ずバックアップしてください。

Windows および UNIX のいずれのインストールの場合も、makefile を編集してアプリケーションを含めて /dist ツリーに追加し、アプリケーションをインストールファイルで使用できるようにします。

Windows での Web アプリケーションの追加

Windows でアプリケーションを追加するには

1 makefile を開きます。

2 大文字と小文字を区別せずに文字列 DEMO を検索し、独自のカスタムコードを追加する場所をすべて見つけます。demo-app 用のコードをコピーして、追加する必要のあるコードを確認します。

3 makefile を保存します。

4 /dist/web-apps ツリーで、makefile に指定したアプリケーション名を持つ別のディレクトリを作成します。これにより、アプリケーションがインストールファイルで使用できるようになります。WAR ファイルを新規ディレクトリにコピーします。

5 InstallShield プロジェクトファイルを編集して Web アプリケーションを含めます。

InstallShield プロジェクトでは、setup.rul から `deployApp()` 関数を呼び出すことによってアプリケーションが公開されます。たとえば、setup.rul の次のコードによって demo-app が公開されます。

```
SdShowMsg ("Web アプリケーションのデモを追加 ...",TRUE );
deployApp(TARGETDIR ^ "demo-app.war", "default", "demo-app", _
   "/demo", TARGETDIR ^ "servers¥¥default¥¥demo-app", TARGETDIR);
DeleteFile(TARGETDIR ^ "demo-app.war");
SdShowMsg ("Web アプリケーションのデモを追加しています。お待ちください ...",
   FALSE);
```

setup.rul ファイルは /installers/JRun 3/Script Files にあります。

6 InstallShield を使用する場合、新規アプリケーションに新規の FileGroup を追加することを忘れないでください。

7 新規の FileGroup を追加した後で、メディアを再構築してプロジェクトを保存する必要があります。

UNIX でのアプリケーションの追加

UNIX で Web アプリケーションを追加するには

1 makefile を開きます。

2 大文字と小文字を区別せずに文字列 DEMO を検索し、独自のカスタム コードを追加する場所をすべて見つけます。demo-app 用のコードをコピーして、追加する必要のあるコードを確認します。

3 makefile を保存します。

4 /dist/web-apps ツリーで、makefile に指定したアプリケーション名を持つ別のディレクトリを作成します。これにより、アプリケーションがインストールファイルで使用できるようになります。WAR ファイルを新規ディレクトリにコピーします。

5 コードを /dist/core-unix/bin/install ファイルに追加してアプリケーションを公開します。

この install ファイルには、公開されるアプリケーションの一覧と `deploy` サブルーチンが含まれています。

Web アプリケーションの公開

JRun には、ユーザが JRun をインストールした後で Web アプリケーションを公開、再公開、削除するための `WarDeploy` ユーティリティがあります。UNIX インストーラでは JRun のインストール時に `WarDeploy` が使用されます。

ユーザはアプリケーションのユーザ インターフェイスを介して `WarDeploy` を呼び出すので、JRun 管理コンソールのインターフェイスを使用してアプリケーションの公開や削除を行う必要はありません。

`WarDeploy` ユーティリティは、JMC の [Web アプリケーションの公開] パネルと同様の機能を果たします。このユーティリティは、アプリケーションの WAR ファイルを既存の JRun サーバーに公開します。WAR ファイルは、JSP、サーブレット、イメージ、および Web アプリケーションのその他の関連コンポーネントで構成されています。これはサーブレット仕様の定義に従って、構造化された階層に配置されます。この構造には、記述子ファイルの web.xml も含まれます。

WarDeploy の使用

`WarDeploy` は jrun.jar ファイル内の allaire.jrun.tools パッケージの一部です。jrun.jar ファイルを使用するには、このファイルをクラスパスに含める必要があります。`WarDeploy` を使用してアプリケーションを公開するには、コマンド ラインで 6 つの必須パラメータを渡すか、`-config` オプションとこれらのパラメータを含むファイルを使用します。

`WarDeploy` を使用してアプリケーションを削除するには、`-remove` オプションを使用してコマンド ラインで 3 つの必須パラメータを渡すか、`-config` オプションを使用してこれらの 3 つのパラメータを含むファイルを参照します。

WarDeploy の構文は次のとおりです。

```
java -cp [classpath] allaire.jrun.tools.WarDeploy
        [-deploy|redeploy]
        [deploy.war.path=value
        deploy.server.name=value
        deploy.webapp.name=value
        deploy.context.path=value
        deploy.webapp.rootdir=value
        deploy.jrun.rootdir=value]
        [-config=property_file_name]
  [-remove]
    [deploy.server.name=value
    deploy.jrun.rootdir=value
        deploy.webapp.name=value]
    [-config=property_file_name]
```

redeploy オプションによって、アプリケーションで remove が呼び出され、次に deploy が呼び出されます。WarDeploy の使用例については、24 ページの「WarDeploy の使用例」を参照してください。次の表では、プロパティについて説明しています。

プロパティ	説明
deploy.war.path	公開する WAR ファイルまたはディレクトリへのファイルシステムパス
deploy.server.name	Web アプリケーションの公開先となる JRun サーバー名
deploy.webapp.name	サーバー内で JRun Web アプリケーションに割り当てる名前。この名前は JRun サーバーとは異なる固有の名前でなければなりません。
deploy.context.path	Web アプリケーションにアクセスするためのコンテキスト パス URI。このパラメータは、JMC の [Web アプリケーションの公開] パネルの [アプリケーションの URI] に対応しています。
deploy.webapp.rootdir	Web アプリケーションの公開先となるルート ディレクトリ。WarDeploy ユーティリティは WAR ファイルまたは WAR ディレクトリの内容をこのディレクトリにコピーします。
deploy.jrun.rootdir	JRun ルート インストール ディレクトリを指すパス。jrun.rootdir システム プロパティが定義されている場合、このプロパティはオプションです。

WarDeploy の使用例

このセクションでは、WarDeploy ユーティリティの使用例を示します。

コマンド ラインでの WarDeploy の使用

次の例では、コマンド ラインでパラメータを渡すことによって nickdanger JRun サーバー上に hws-app を公開します。

```
c:¥>java -cp "c:¥Program Files¥Allaire¥JRun¥lib¥jrun.jar"
        allaire.jrun.tools.WarDeploy -deploy
        deploy.war.path="c:¥temp¥hws.war"
        deploy.server.name=nickdanger
        deploy.webapp.name="hws-app"
        deploy.context.path=/nickshardware
        deploy.webapp.rootdir="c:¥Program Files¥Allaire¥JRun¥servers¥
              nickdanger¥hardware_store"
        deploy.jrun.rootdir="c:¥Program Files¥allaire¥JRun"
```

config ファイルを指定した WarDeploy の使用

次の例では、コマンド ラインにパラメータを渡すのではなく、パラメータの含まれるファイルを指定します。

```
c:¥java -cp "c:¥Program Files¥Allaire¥JRun¥lib¥jrun.jar"
        allaire.jrun.tools.WarDeploy -deploy -config="c:¥temp¥war.txt"
```

war.txt ファイルの内容は次のとおりです。

```
deploy.war.path="c:¥temp¥hws.war" deploy.server.name=nickdanger
        deploy.webapp.name="hws-app" deploy.context.path=/
        nickshardware deploy.webapp.rootdir="c:¥Program
        Files¥Allaire¥JRun¥servers¥nickdanger¥
        hardware_store" deploy.jrun.rootdir="c:¥Program
        Files¥allaire¥JRun"
```

アプリケーション削除時における WarDeploy の使用

次の例では、nickdanger JRun サーバーから hws-app を削除します。

```
c:¥java -cp "c:¥Program Files¥Allaire¥JRun¥lib¥jrun.jar"
        allaire.jrun.tools.WarDeploy -remove
        deploy.server.name=nickdanger
        deploy.webapp.name=hws-app
        deploy.jrun.rootdir="c:¥Program Files¥Allaire¥JRun"
```

JSP ソース コードの非表示

Web アプリ ケーションの一部として JSP ページを公開する場合、顧客にこれらのページのクリア テキスト バージョンを提供しない場合もあります。代わりに、これらのページのバイナリ (.class) バージョンのみを配布できます。

JRun では、JSP ページがクライアントによって初めて要求された場合、または要求された JSP ページが最後の要求後に変更されていた場合に、JSP ページがコンパイルされ、そのページのクラス ファイルが作成されます。ただし、公開されているアプリケーションで、コンパイルを行わずに JRun によってそのページのクラス ファイルが必ず読み込まれるように強制することもできます。

JSP ページのコンパイルを行わない理由としては、次のようなものがあります。

- パフォーマンス : 静的な JSP ページのコンパイルによって、アプリケーションに対する処理オーバーヘッドが最初に増加します。
- セキュリティ : JSP ページは、誰でも読み取りや編集ができるテキスト ファイルです。JSP ページのコンパイルを行わない場合、アプリケーションを構成する JSP ページを出荷する必要はありません。代わりに、対応する Java バイトコード形式のクラス ファイルのみを出荷します。

JSP のコンパイル プロセスと JSP コンパイラの回避による影響については『JRun によるアプリケーションの開発』を参照してください。

JSP ページのコンパイルの無効化

JSP ページのコンパイルを明示的に無効にするには、JMC を使用して JSP ページを処理する JRun サーブレットを変更します。個別の Web アプリケーションに対してコンパイルを有効または無効にすることができるように、この設定は Web アプリケーション レベルで行います。

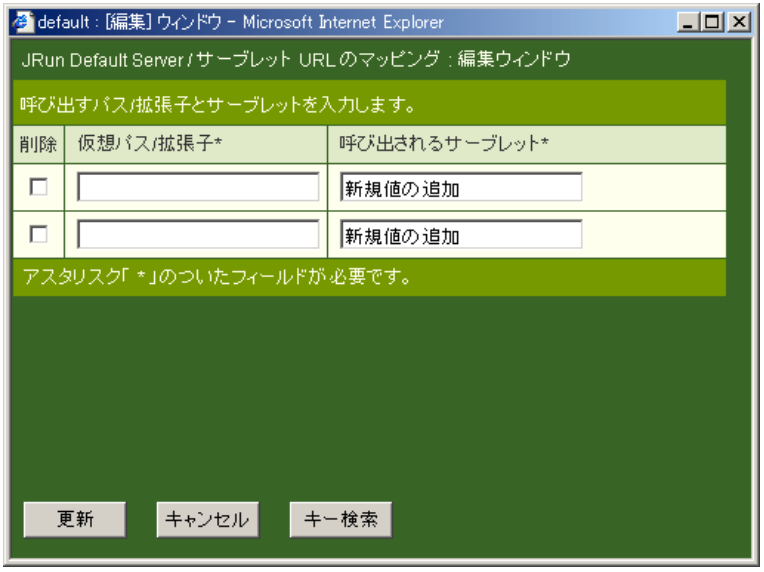
JSP のコンパイルを無効にするには

1 JMC で [*サーバー名*] > **[アプリケーション]** > [*アプリケーション名*] > **[サーブレット URL のマッピング]** を選択します。

[サーブレット URL のマッピング] パネルが JMC の右側に表示されます。

2 [編集] ボタンをクリックします。

サーブレット URL のマッピングの編集ウィンドウが表示されます。



3 [仮想パス / 拡張子] フィールドに「*.jsp」と入力します。

4 [呼び出されるサーブレット] フィールドに「jsprt」と入力します。

5 [更新] ボタンをクリックします。

6 JRun サーバーを再起動します。

これで、JSP ページに対するすべての要求が jsprt サーブレット によって処理されるよう になり ます。このサーブレット は JSP ページの存在をチェック せず、対応するクラス ファイルのみを検索し ます。そのため、公開される Web アプリ ケーション では JSP ページを公開する必要がありません。JSP ファイルはソース コード ファイルであるため、公開されるアプリ ケーション からソース コード を省略できます。

JSP のプレコンパイル

コンパイルをオフにする場合は、すべての JSP ページの対応するクラス ファイルをインストールに含める必要があります。したがって、アプリケーションを公開する前に、すべての JSP ページをプレコンパイルしてクラス ファイルを作成する必要があります。プレコンパイルは次のどちらかの方法で行います。

- JSP ページのコンパイルを有効にして、アプリケーションのすべての JSP ページを要求します。
  JSP ページを要求すると、すべての JSP ページがコンパイルされ、対応するクラス ファイルが確実に作成されます。この方法の短所は、各 JSP ページを手作業で要求しなければならないこと です。要求し ていない JSP ページが 1 つでも あると、クライアント がそのページを要求してそのページがコンパイルされていない場合にエラーが発生し ます。
- JSPC コンパイラを使用して、コマンド ラインから すべての JSP ページをコンパイルします。
  JSPC コンパイラによって、1 つのコマンド でアプリケーション と関連付けられたすべての JSP ページをコンパイルできます。この方法では、すべての JSP ページを確実にコンパイルできます。JSPC コンパイラ の使用方法については、『JRun によるアプリケーションの開発』を参照してください。

Web アプリケーションの制御

JRun には WebAppController インターフェイスがあり、これを使用すると Web アプリケーションの起動、停止、再起動、および再公開を実行できます。WebAppController インターフェイスは allaire.jrun パッケージに含まれています。このセクションでは、このインターフェイスのメソッドを定義し、その使用例を示します。

WebAppController の一部の機能は `WarDeploy` ユーティリティでも利用できます。詳細は、23 ページの「Web アプリケーションの公開」を参照してください。

メソッドの詳細

restartWebapp

```
void restartWebapp() throws ServletException
```

現在の Web アプリケーション サービスを再起動します。サービスの `destroy()` メソッドを呼び出し、サービスを再初期化します。

startWebapp(String serviceName)

```
void startWebapp(String serviceName) throws ServletException
```

Web アプリケーション サービスを起動します。JRun サーバーの local.properties ファイルに新しいマッピングを追加します。この新しいマッピングは、新しい Web アプリケーション名にマップされた Web アプリケーション ルートです (/foo=foo など)。次の表では、パラメータについて説明しています。

パラメータ	説明
serviceName	Web アプリケーションの名前 (例 : demo、admin)。

stopWebapp

```
void stopWebapp() throws ServletException
```

現在の Web アプリケーション サービスを停止します。サービスの `destroy()` メソッドを呼び出します。

redeployWebapp(File warFile)

```
void redeployWebapp(File warFile) throws ServletException
```

WAR ファイルを使用して現在の Web アプリケーションを再公開し、そのアプリケーションを再起動します。次の表では、パラメータについて説明しています。

パラメータ	説明
warFile	再公開する既存の WAR ファイルを参照する java.io.File を取ります。

WebAppController の使用例

次の例では、WebAppControllerインターフェイスが使用されているかどうかがチェックされます。使用されている場合は、Web アプリケーションを再起動します。

```
ServletContext context = getServletContext();
  if (context instanceof WebAppController) {
    WebAppController controller = (WebAppController) context;
    controller.restartWebapp();

  }
```

JRun プロパティのカスタマイズ

JRun プロパティ ファイルの設定は、OEM リソースに付属の PropertyScript ユーティリティを使用して、変更、追加、および削除できます。PropertyScript では、JRun プロパティ ファイルを変更するディレクティブが含まれている独立したスクリプト ファイルが処理されます。JRun プロパティ ファイルの詳細は、『JRun セットアップ ガイド』を参照してください。

PropertyScript の使用

`PropertyScript` クラスは、install.jar ファイル内の allaire.jrun.install パッケージの一部です。次に例を示します。

```
java -cp [classpath] allaire.jrun.install.PropertyScript script-file
        [property-file]
```

`PropertyScript` を使用するには、クラスパスに install.jar を含める必要があります。

`script-file` オプションは、ディレクティブが含まれているスクリプト ファイルを指すファイル システム パスです。スクリプト ファイルの作成方法については、30 ページの「スクリプト ファイルの作成」を参照してください。

`property-file` オプションでは、global.properties、local.properties、jvms.properties などの JRun プロパティ ファイルを指定します。このオプションは現在のリリースの JRun で推奨されていないため、必要ありません。

次に例を示します。

```
C:¥>java -cp "c:¥program files¥allaire¥jrun¥lib¥install.jar"
        allaire.jrun.install.PropertyScript
        "c:¥program files¥allaire¥jrun¥servers¥default¥script.txt"
```

スクリプト ファイルの作成

`script-file` オプションでは、JRun プロパティ ファイルの変更時に `PropertyScript` によって使用されるすべてのディレクティブが含まれるテキスト ファイルを指定します。スクリプト ファイルの作成時には、少なくとも 1 つの `file` コマンドと `filename` を含める必要があります。各 `file` コマンドの下には、前に示してある `filename` に対して実行するディレクティブおよびオプションを列挙します。

スクリプト ファイルの構文は、次のようになります。

```
file filename
   directive1
   [directive2]
   ...
   [file filename]
   [directive1]
   [directive2]
   ...
```

スクリプト ファイル内のプロパティ ファイルのセクションごとに、新しい `file` コマンドを指定する必要があります。filename (ファイル名) には、プロパティ ファイルへの完全なパスを指定する必要があります。

通常、スクリプト ファイル内の各ディレクティブは、1つのコマンドおよび1組のキーと値から成り立っています。キーは、JRun プロパティ ファイル内のプロパティに対応します。次の例では、`add` がコマンド、`control.endpoint.main.port` がキー、`53000` が値です。

```
add control.endpoint.main.port=53000
```

一部のキーには複数の値を指定できます。この場合、値はスペースまたはカンマで区切られます。

次の表では、スクリプト ファイルのディレクティブおよびそのオプションについて説明します。

ディレクティブ	説明
add *キー 値*	新しいキーと値を、プロパティ ファイルの最後に追加します。既存のキーを追加すると、該当するキーと値が `PropertyScript` によって上書きされます。
replace *キー 値*	指定したキーの値を新しい値に置き換えます。存在しないキーの値を置き換えようとすると、このディレクティブは `PropertyScript` によって `add` として処理されます。
delete *キー*	指定したキーとその値を削除します。
clear *キー*	指定したキーのすべての値を削除しますが、キーは削除しません。
append *キー 値*	指定したキーの最後の値の後に、値を追加します。値の先頭には、区切り文字 (通常はカンマ) を指定する必要があります。 たとえば、`jcp` を `servlet.services` キーに追加する場合は、次のディレクティブを使用します。 `append servlet.services ,jcp` 一部のキーはカンマでなくスペースによって区切られます。この場合は append_space ディレクティブを使用します。
append_space *キー 値*	指定したキーの最後の値の後にスペースを入れてから、値を追加します。スペースを区切り文字として使用する `java.args` キーを変更する場合に使用します。
token_remove *キー 値*	キーの中から指定した値を検索し、その値および後に続く区切り文字 (スペースまたはカンマ) を削除します。

ディレクティブ	説明
ejb_append jrun.services *値*	global.properties ファイル内の `jrun.services` キーを変更する場合にのみ使用します。servlet_services はほかのすべてのサービスが開始してから開始する必要があるので、このメソッドを使用して `{servlet_services}` キーを一覧の最後に表示します。
unplug [servlet|ejb]	サーブレットまたは EJB のいずれかの機能を JRun から削除します。ユーザがコンポーネントを追加および削除できるようにする場合に使用します。また、インストールファイルを編集して、不要なコンポーネントがインストールされないようにすることもできます。 `unplug` によってコンポーネントが無効になりますが、実際にはアンインストールされません。
comment *キー # テキスト文字列*	テキスト文字列で構成されるコメント行を、指定したキーの上の行に追加します。テキスト文字列の前に必ず # を入れてください。
adduser *ユーザ名 パスワード*	新規 JMC ユーザを追加します。`PropertyScript` では、`UnixCrypt` を使用してパスワードが暗号化されます。pass.properties ファイルを変更する場合にのみ使用します。新規 Web アプリケーション ユーザを追加する場合は、『JRun によるアプリケーションの開発』の PropertyFileAuthentication クラスの説明を参照してください。
port *キー 最小値*,*最大値*	インストール時に JRun で設定される、admin サーバーのアクセス可能ポート範囲を設定します。最小値は最小のポート番号、最大値は最大のポート番号です。`PropertyScript` では、この範囲内にあるすべてのポートが試行されます。 ポート ディレクティブは、JRun サーバーの local.properties ファイルごとに指定します。 admin サーバーの既定範囲は 8000 ～ 8099 です。`default` サーバーの既定範囲は 8100 ～ 8199 です。

サンプル スクリプト ファイル

次のスクリプト ファイルの例は、default JRun サーバーの local.properties ファイルに対して、新しい Web アプリケーション "MyStocks" を追加し、demo-app を削除する方法を示します。最後の行は、default JRun サーバーの JRun Web サーバーのポートを、8080 ～ 8089 の範囲で使用可能なポートに変更します。

```
file c:¥Progra~1¥Allaire¥JRun¥servers¥default¥local.properties
   add webapp.mapping./mystocks /mystocks
   add /mystocks.rootdir C:¥¥Mycompany¥¥servers¥¥default¥¥mystocks
   add /mystocks.class {webapp.service-class}
   token_remove servlet.webapps demo-app
   append servlet.webapps ,/mystocks
   comment servlet.webapps #Added mystocks app, removed demo
   delete demo-app.rootdir
   delete demo-app.class
   delete webapp.mapping./demo
   port web.endpoint.main.port 8080,8089
```

サーブレット /JSP での PropertyScript の使用

`PropertyScript` クラスの `main()` メソッドを使用して、サーブレットまたはその他の種類の Java アプリケーション内からプログラムで JRun プロパティ ファイルを変更できます。このメソッドは 1 つの引数 `script_file` を取ります。`script_file` は、`PropertyScript` を実行するディレクティブを含むテキスト ファイルの場所です。`main()` メソッドには文字列の配列が必要であるため、1 つの要素 (`script_file`) を持つ配列を渡します。

`main()` の構文は次のとおりです。

```
main(String script_file)
```

`PropertyScript` を呼び出すには、install パッケージを含める必要があります。次に例を示します。

```
import allaire.jrun.install.*;
```

PropertyScript の使用例

次の Java アプリケーション (PSTest.java) は、`PropertyScript` の使用例を示しています。これにより、次の行が local.properties の最後に追加されます。

```
frogprop=frogvalue
```

propt.txt スクリプト ファイルには次の行が含まれています。

```
file c:¥Progra~1¥Allaire¥JRun¥servers¥default¥local.properties
   add frogprop frogvalue
```

PSTest.java ファイルには次のコードが含まれています。

```
import allaire.jrun.install.*;
import java.io.*;

class PSTest {
  public static void main (String args[]) {
    String result = new String();
    result = PSTest.RunPS();
    System.out.println(result);
  }

  private static String RunPS() {
    String[] sFile = new String[1];
    sFile[0] = "c:¥¥temp3¥¥propt.txt";
    String result = new String();

    PropertyScript ps = new PropertyScript();
    try {
      ps.main(sFile);
      result = "successful";
    }
    catch (IOException ioe) {
      result = "unsuccessful";
    }
    return result;
  }
}
```

JRun の組み込み

JRun にはいくつかの設定方法があるため、JRun はユーザに対して透過的であり、お客様の製品の妨げになることはありません。JRun OEM リソースを使用して製品を設定する場合には、次の点を考慮してください。

設定方法	参照先
JRun の透過的なインスト ール	「JRun インストールのカスタマイズ」8 ページ
コネクタの透過的なインスト ール	「外部 Web サーバーへの JRun の接続」40 ページ
アプリ ケーショ ンの透過的な公開	「Web アプリケーションのカスタマイズ」19 ページ
JRun 管理コンソールの非表示	「JRun 管理コンソールの非表示」35 ページ
JRun サーバーの開始、停止、および再起動	「JRun サーバーの開始、停止、および再起動」36 ページ
デスクトップ上の JRun の非表示	「デスクトップ上の JRun の非表示 (Windows のみ)」36 ページ
ライセンス キーおよびライセンス契約の設定	「JRun ライセンス キーおよびライセンス契約の設定」38 ページ

JRun 管理コンソールの非表示

製品版のユーザは、JRun 管理コンソ ール (JMC) を使用して JRun のインスト ールおよび設定を管理し、スレッド 管理やデータ ソース アクセスなどの設定を調整します。ただし、ユーザが JRun のインターフェイスを使用できないようにする一方、個人設定については独自に変更できるようにする組み込み環境では、JMC は優れたソリューションとは言えません。

JMC は、ユーザが JRun の設定を変更するための主な手段です。ただし、JMC で行う作業のほとんどは JRun プロパティ ファイルの変更です。`PropertyScript` ユーティリティを使用して JRun プロパティ ファイルを変更したり、`PropertyScript` を呼び出す Java アプリケーションを作成することもできます。`PropertyScript` ユーティリティを使用して JRun プロパティ ファイルを変更する方法については、30 ページの「JRun プロパティのカスタマイズ」を参照してください。

たとえば、アプリケーションを JRun Web サーバー上で実行している場合に、default JRun サーバーに対して Web サーバーが受信しているポートをユーザが変更するとします。既定値は 8000 です。この場合、JRun サーバーの local.properties ファイル内の `web.endpoint.main.port` をユーザが変更できるようにするインターフェイスを作成します。

JRun サーバーの開始、停止、および再起動

JRun プロパティ ファイルに変更を加えた場合は、そのたびに JRun サーバーを再起動する必要があります。ただし、users.properties ファイルは唯一の例外です。このファイルは自動的に再ロードされるので、Web アプリケーションのユーザをリアルタイムで追加、変更、または削除できます。

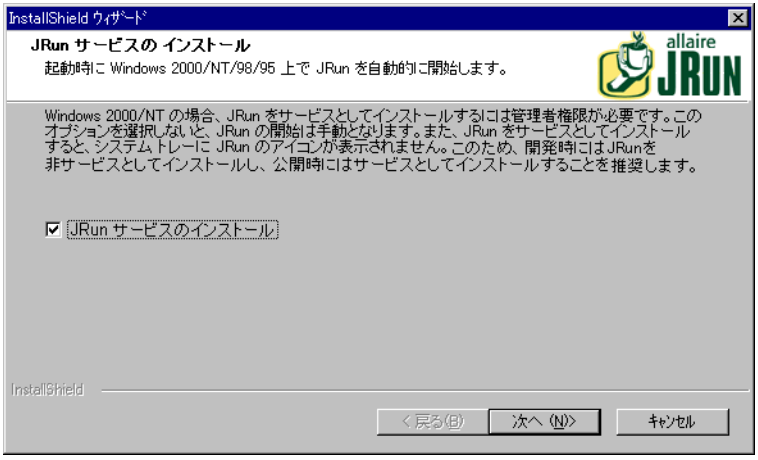
JRun には、JRun サーバーを開始、停止、再起動、インストール、または削除するためのユーティリティがあり、Windows の場合は jrun.exe、UNIX の場合は jrun シェル スクリプトがこれにあたります。このユーティリティは、コマンド ラインまたはスクリプト から呼び出せます。JRun コマンド ライン ユーティリティの使用方法については、『JRun セットアップ ガイド』を参照してください。

デスクトップ上の JRun の非表示 (Windows のみ)

JRun 名が表示される場所は、JRun を Windows のサービスとしてインスト ールするか、またはアプリケーションとしてインストールするかによって異なります。このセクションでは、インストール タイプの選択方法と、JRun の実装を非表示にする方法について説明します。

インストール タイプの選択

製品版のインスト ール時に、JRun のインスト ール タイプを選択するプロンプト が表示されます。次の図は、[JRun サービスのインスト ール] ウィンドウを示します。



ユーザが [JRun サービスのインストール] を選択した場合は、JRun サーバー (admin JRun サーバーや default JRun サーバーなど) が Windows NT サービスとして認識され、コントロール パネルの [サービス] に名前が表示されます。Windows 95/98 では、サーバーは Windows レジストリ でのみ示されます。ユーザが [JRun サービスのインスト ール] を選択しない場合は、JRun サーバーはアプリケーションとして実行され、アイコンがシステム トレイに表示されます。

JRun がアプリケーションとして実行される場合、システムトレイ内のアイコンを非表示にすることはできません。InstallShield プロジェクト ファイルを変更して、JRun のインストール タイプ (サービスまたはアプリケーション) をユーザが選択できないようにすることも可能です。

[JRun サービスのインストール]ウィンドウのコードは、ComponentEvents.rul ファイル内の `Main_Installed2` 関数に表示されます。この関数では、インストールを Windows NT サービスとしてハードコード化できるだけでなく、コントロール パネルの [サービス] に表示される JRun サーバー名を変更できます。

メモ
Windows 2000/NT では、JRun をサービスとしてインストールするためには管理者権限が必要です。

Windows NT サービス名の変更

JRun サーバーは、サーバーの local.properties ファイルで定義されている表示名によって Windows のコントロール パネルの [サービス] に表示されます。たとえば、既定のインストールの JRun サーバーは、"JRun Admin Server" および "JRun Default Server" と表示されます。

これらの名前は、JRun サーバーの local.properties ファイル内の JVM プロパティセクションに次のように格納されます。

```
## jvm properties
jrun.server.displayname=JRun Admin Server
```

JRun インストールでユーザの製品名を使用する場合は、名前の変更が必要なこともあります。

前のセクションで説明した InstallShield ファイル内の JRun サーバー名を変更することも、または `PropertyScript` ユーティリティを使用してインストール後にサーバー名を変更することも可能です。`PropertyScript` ユーティリティの使用方法については、30 ページの「JRun プロパティのカスタマイズ」を参照してください。

設定の一部として新しい JRun サーバーをインストールする場合は、JRun コマンドライン ユーティリティを使用し、`-install` オプションで表示名を設定することもできます。次に例を示します。

```
% jrun -install "Foo Service" foo -quiet
```

`-install` オプションとともに `jrun` コマンドを使用しても、新しい JRun サーバーは作成されないので注意してください。その代わり、既存の JRun サーバーが NT サービスに変更されます。JRun コマンド ライン ユーティリティの使用方法と新規 JRun サーバーの作成方法については、『JRun セットアップ ガイド』を参照してください。

JRun ライセンス キーおよびライセンス契約の設定

JRun インストール コードを使用する場合は、JRun のライセンス契約をお客様自身の契約に置き換えて、シリアル番号検証システムを変更できます。また、ライセンス キーのプロンプトを JRunインスト ールから削除し、ユーザに対して透過的にすることも可能です。

このセクションでは、次の項目について説明します。

ライセンスキーの概要

製品版のインストールでは、ユーザはシリアル番号の入力を要求されます。シリアル番号は、検証された後で、全体のインスト ールの終了時に serial_number.properties ファイルに書き込まれます。JRun サーバーを起動するたびに、このファイル内でシリアル番号が確認されます。このプロパティ ファイルに有効なシリアル番号が格納されていないと、JRun サーバーは、このエディションが Developer 版 (最大同時接続ユーザ数 3) であると見なします。

Windows でのライセンス キーの処理

製品版では、setup.rul によって、インスト ール時にシリ アル番号の入力をユーザに求める [製品シリ アル番号] パネルが作成されます。ユーザが入力したシリ アル番号は `szLicKey` として格納されます。次に、setup.rul ファイルによって、`szLicKey` を検証する license.rul の `isValid30Key` が呼び出されます。ライセンス キーの有効性が確認されると、setup.rul は `writeSerialNumber` 関数を呼び出してシリアル番号を serial_number.properties ファイルに書き込みます。

JRun の OEM 版が製品版と異なる点は、license.rul ファイルに JRun ライセンス キーの検証ロジックでなく検証関数のスタブだけが含まれていることです。つまり、ライセンス キー検証の独自の論理を含めることができるわけです。次に、この関数スタブの一覧を示します。

```
isValid30Key
isValid30UpgradeKey
isValid2xKey
isValidOldKey
```

JRun OEM ライセンス キーを setup.rul 内でハードコード化して serial_number.properties に書き込むことにより、ユーザによる JRun キーの入力を省略できます。次に、setup.rul 内の現在のライセンス キープロンプトと license.rul 内の関数のうち独自の検証関数を使用してキーをチェックします。

有効な JRun のシリアル番号については、OEM/ホスティング関連資料または Jumpstart ライセンス証明書を参照してください。

JRun に含まれているライセンス契約の内容を変更するには、license.txt ファイル (/dist/core/docs) を変更するか、別のファイルに置き換えます。別のライセンス ファイルを使用する場合は、setup.rul 内の次の行を編集します。

```
szLicenseFile = SUPPORTDIR ^ "license.txt";
```

UNIX でのライセンス キーの処理

UNIX では、インストール スクリプトの License Key Handling セクションによって有効なライセンス キーの入力がユーザに要求されます。キーは `$LICENSE` として格納されます。次に、インストールスクリプトによって `LicenseChecker` ユーティリティが呼び出され、`$LICENSE` の有効性がチェックされます。ライセンス キーの有効性が確認されると、`LicenseChecker` によってキーが serial_number.properties ファイルに書き込まれます。

OEM ライセンス キーをハードコード化し、LicenseChecker を使用して serial_number.properties に書き込むことができます。この作業を行うには、次の行を除く License Key Handling セクションをすべてコメント化します。

```
jrun -jrundir $JRUNROOTDIR -classpath "$INSTALL_CLASSPATH" -java
        "$JAVA" allaire.jrun.install.LicenseChecker -i -c "$LICENSE"
```

次に、`$LICENSE` をお客様の OEM シリアル番号に設定します。有効な JRun のシリアル番号については、OEM 契約を参照してください。

ライセンス契約の内容は sh-install ファイルの Display the End User License Agreement セクションに格納されます。テキストは直接変更できます。

外部 Web サーバーへの JRun の接続

JRun のインスト ールと 設定が完了すると 、JRun を外部 Web サーバーに接続できます。たとえば、Apache に対して出された要求は、JRun フィルタによって検証され、処理された後、Apache Web サーバー経由で要求元のクライアントに送信されます。

各 JRun サーバーは、既定では JRun Web サーバー (JWS) のインスタンスに接続されることに注意して ください。ただし、JWS は軽量な Web サーバーです。アプリ ケーション に対するリ モート アクセスの保護や、より 強力な環境内でのアプリ ケーション の実行を計画している場合は、Apache、Netscape iPlanet、Microsoft IIS などの外部 Web サーバーに JRun サーバーを接続する必要があります。1 つの JRun サーバーに対して、任意の数の Web サーバーを接続できます。

接続の概要

JRun サーバーを外部 Web サーバーに接続するには 2 つの方法があります。どちらの方法も Web サーバーのコンフィギュレーション ファイルを変更することにより、Web サーバーと JRun サーバー間の通信を確立します。接続方法は次のとおりです。

- コネクタ ウィザード

  製品版ユーザは、コネクタ ウィザードを使用して JRun サーバーを外部 Web サーバーに接続できます。コネクタ ウィザードを使用するには、ユーザが管理者として JRun 管理コンソールにログインする必要があり ます。詳細は、『JRun セットアップ ガイド』を参照してください。

- ConnectorInstaller

  OEM リソースの allaire.jrun.admin パッケージには `ConnectorInstaller` ユーティリティが含まれています。このユーティリティは、コマンド ラインから呼び出すことができます。

  Microsoft IIS に JRun を接続する場合は、`metaset` (IIS 4.0 の場合) または `regset` (IIS 3.0 の場合) も使用できます。44 ページの「IIS でのコネクタのインストールとアンインストール」で説明して いるよう に、これらのユーティリ ティはメタベースおよび Windows レジストリ の編集に使用されます。`ConnectorInstaller` の代わりにこれらのユーティリティを使用できます。

ConnectorInstaller ユーティリティの使用

`ConnectorInstaller` ユーティリティは、コマンド ライン、Java アプリケーション、またはバッチ ファイルやスクリプトから実行できます。このユーティリティでは、システム固有の設定が必要になり ます。この設定については、コマンド ラインでユーザに対してプロンプトを出力するか、`ConnectorInstaller` が呼び出されたときに読み込むファイルに含めることができます。

ConnectorInstaller クラスは、jmc-app の WEB-INF/lib ディレクトリ 内の jmc-app.jar ファイルに含まれています。

次の手順は、コマンド ラインから ConnectorInstaller を使用する方法を示します。バッチ ファイルから ConnectorInstaller を使用する例については、43 ページの「ConnectorInstaller とファイル転送の併用」を参照してください。

コマンド ラインからの ConnectorInstaller の使用

コマンド ラインから ConnectorInstaller を使用するには

1 外部 Web サーバーを停止します。

2 コマンド ラインから、次のようにして ConnectorInstaller を呼び出します。

```
java -cp [クラスパス] allaire.jrun.admin.ConnectorInstaller
     JRun のルート ディレクトリ
```

クラスパスには、少なくとも次のパッケージを指定する必要があります。

```
jrun.jar
servlet.jar
jmc-app.jar
```

次に例を示します。

```
C:¥>java -cp "c:¥program files¥allaire¥jrun¥servers¥admin¥jmc-app
     ¥web-inf¥lib¥jmc-app.jar;c:¥program files¥allaire¥jrun¥lib¥
     jrun.jar;c:¥program files¥allaire¥jrun¥lib¥ext¥servlet.jar"
     allaire.jrun.admin.ConnectorInstaller
     "c:¥program files¥allaire¥jrun"
```

次のようなメッセージが表示されます。

```
Welcome to JRun 3.0 Connector Wizard.
Please stop your target web server before proceeding.
Press 'Enter' to continue.
```

3 ConnectorInstaller ユーティリティによって、コネクタ ウィザードのフィールドに対応する補足情報の入力が要求されます。次の表を参考にして、必要な値を設定してください。

プロンプト	説明
Available Web Application Containers	外部 Web サーバーに接続する JRun サーバーを選択するか、Enter キーを押して既定値を適用します。
Web Server	JRun サーバーに接続する外部 Web サーバーを選択するか、Enter キーを押して既定値を適用します。
Web Server Version	Web サーバーのバージョンを選択するか、Enter キーを押して既定値を適用します。
Platform	Web サーバーを実行するプラットフォームを選択するか、Enter キーを押して既定値を適用します。

プロンプト	説明
Proxy Host	外部 Web サーバーのホスト の IP アドレスを入力します。JRun サーバーが Web サーバーと 同じ マシン上で実行されている場合は、既定の IP アドレス 127.0.0.1 のままにしておきます。
Proxy Port	JRun 接続モジュール (JCM) が Web サーバーとの通信で使用するポート番号を入力します。 Web サーバーの HTTP ポート と 混同しないでください。『JRun セットアップ ガイド』内の例では 51000 を使用しています。
Config File Directory	外部 Web サーバーのコンフィ ギュレーション ファイル ディレクトリ の絶対パスを入力します。たとえば、IIS の既定値は ¥inetpub¥scripts です。

JRun サーバーを接続している Web サーバーの種類によっては、上記以外にも選択項目があります。表示される 可能性がある その他のプロンプト については、次の表を参照してください。

Web サーバー	プロンプト	説明
Apache	DSO Support	UNIX: JRun モジュールを Apache サーバーにコンパイルしていない場合は、DSO サポートを選択します。 Windows: DSO サポートを選択します。
Microsoft IIS/ PWS	Install as a Global Filter	IIS の場合、[はい] を選択すると、JRun は IIS の該当インスタンス内のすべての仮想サーバーに対して ISAPI フィルタをインスト ールします。単一の仮想サーバーだけにフィ ルタを適用する場合は[いいえ] を選択します。 PWS の場合は [はい] を選択します。
Netscape iPlanet	Use Native or Java Connector	Netscape Web サーバーの場合は、ネイティブ (既定値) または Java コネクタのいずれかを選択します。詳細は、『JRun セット アップ ガイド』を参照してください。

4 Web サーバーを再起動します。

5 JRun サーバーを再起動します。

ConnectorInstaller によって必要なファイルがコピーされ、入力に基づいて Web サーバーのコンフィギュレーション ファイルが編集されます。Web サーバーのコンフィギュレーション ファイルまたは JRun のプロパティに加える変更の詳細は、『JRun セットアップ ガイド』内のコネクタ ウィザードに関する説明を参照してください。

ConnectorInstaller とファイル転送の併用

簡単なプログラミング テクニックを使用すると、ConnectorInstaller を自動化して独自のインストーラに組み込めます。たとえば、プロンプトに対する応答を指定したファイルを作成し、ファイル転送とともに ConnectorInstaller を呼び出して、コマンド ライン インターフェイスの代わりにそのファイルから入力を取得します。

ファイル転送とともに ConnectorInstaller を使用するには

1 コネクタのインストールを実行し、プロンプトごとに入力する応答を記録します。必ず正しい順序で応答を記録してください。

プロンプトは Web サーバーごとに異なるため、コマンド ラインから ConnectorInstaller を何度か実行しなければならない場合があります。詳細は、41 ページの「コマンド ラインからの ConnectorInstaller の使用」を参照してください。

2 ConnectorInstaller のプロンプトに対する応答を指定したテキスト ファイルをプログラムで作成するか、インストール パッケージ内のファイルを編集します。1 行に 1 つずつ応答を指定し、各応答の後で改行します。

たとえば、テキスト ファイルの内容は次のようになります。

```
1
4
2
1
127.0.0.1
8081
c:¥inetpub¥scripts
2
1
```

この場合、最初の行には改行のみが入っているので注意してください。これにより、JRun インストーラで既定値が使用されます。また、このファイルはプレーン テキストとして保存する必要があります。

3 ConnectorInstaller を呼び出し、テキスト ファイルを転送して入力を行います。ConnectorInstaller を呼び出すには、バッチ ファイルを使用するか、Runtime.exec() を呼び出します。

たとえば、テキスト ファイルが answers.txt で、JRun のルート ディレクトリが c:¥Jrun である場合は、バッチ ファイルに次の行を含めます。

```
java allaire.jrun.admin.ConnectorInstaller c:¥jrun < answers.txt
```

"<" 記号に注意してください。この記号は、プログラムの動作を停止してキーボードからの入力を待つのではなく、転送されたファイルから次の行を読み込むように ConnectorInstaller に指示します。

ConnectorInstaller の出力を表示しない方が良い場合があります。UNIX では、> /dev/null を使用して ConnectorInstaller の出力をヌル デバイスに転送できます。Windows では、> /temp/ci.tmp を使用して temp ディレクトリ 内の .tmp ファイルに出力を転送できます。

IIS でのコネクタのインストールとアンインストール

OEM リソースには、Microsoft IIS 3.0 および 4.0 に対応する JRun コネクタをインスト ールまたはアンインスト ールするためのユーティリ ティ が 2 つあり ます。ほとんどの場合は、40 ページの「ConnectorInstaller ユーティリティの使用」で説明した `ConnectorInstaller` ユーティリ ティを使用してコネクタを IIS にインスト ールしますが、Allaire では `metaset` および `regset` という 2 つのユーティリティも提供しています。たとえば、分散環境において JRun をインスト ールする際に、IIS と JRun を別のマシンに接続し、`ConnectorInstaller` ユーティリティがネットワーク間で使用できない場合には、この 2 つのユーティリティを使用します。

メモ

`metaset` および `regset` ユーティリ ティを使用してコネクタをインスト ールする場合は、ほかの手順を手作業で実行する必要があります。この手順については、次のセクションで説明します。

IIS 3.0 は Windows のレジストリを使用して設定情報を格納します。コネクタを IIS 3.0 にインストールするには、コマンド ライン ユーティリティの `regset.exe` を使用します。

IIS 4.0 および 5.0 はメタベースを使用して設定情報を格納します。メタベースは Windows レジストリ に類似していますが、アクセスの高速化 (ファイル サイズの縮小) により 重点が置かれています。JRun 接続情報に関して IIS 4.0/5.0 メタベースを更新するには、`metaset.exe` を使用します。

metaset と regset の構文

`regset` および `metaset` ユーティリティの構文は次のとおりです。

```
regset -[i|d] jrun.dll_path [-quiet]
metaset -[i|d] /n jrun.dll_path [-quiet]
```

`metaset` は *JRun のルート ディレクトリ*`¥connectors¥isapi¥intel-win¥metaset.exe` に、`regset` は *JRun のルート ディレクトリ*`¥connectors¥isapi¥intel-win¥regset.exe` にあります。

`-quiet` オプションを使用すると、操作の成否にかかわらず、出力結果が作成されません。

`-i` オプションは、コネクタをインストールするときに使用します。

`-d` オプションは、コネクタをアンインストールまたは削除するときに使用します。`metaset` および `regset` のオプションについては、以降のセクションを参照してください。

metaset および regset を使用した IIS へのコネクタのインストール

regset または metaset を使用してコネクタをインストールする場合、jrun.dll ファイルおよび jrun.ini ファイルを適切な場所 (通常は /inetpub/scripts ディレクトリですが、設定によっては異なる場合もあります) にコピーする必要があります。

jrun.dll は ISAPI フィルタであり、Web サーバーに対する HTTP 要求を中断し、JRun の処理として適切な要求を JRun に渡します。jrun.ini ファイルは jrun.dll を初期化します。

場合によっては、JRun コネクタを正しく機能させるために、jrun.ini ファイル内の一部の設定を変更する必要があります。jrun.ini ファイルのプロパティと分散環境での JRun のインストールについては、『JRun セットアップ ガイド』を参照してください。

メモ
JRun コネクタをインストールするときは、regset または metaset を使用する前に Web サーバーを停止してください。

IIS 3.0 へのコネクタのインストール

regset ユーティリティは、Windows レジストリに JRun コネクタ情報を追加して更新します。IIS 3.0 はマルチホストをサポートしないので、各 Web サーバーインスタンスのスクリプトディレクトリは 1 つだけです。

次に例を示します。

```
regset -i c:¥inetpub¥scripts¥ -quiet
```

IIS 4.0 および 5.0 へのコネクタのインストール

metaset ユーティリティは、メタベースに JRun コネクタ情報を追加して更新します。

次に例を示します。

```
metaset -i /1 c:¥virt2¥inetpub¥scripts -quiet
```

-i オプションは、仮想ホスト番号 n に新しいコネクタを追加します。この場合、n は 1 として処理されます。仮想ホストには明確な番号が付けられませんが、Microsoft 管理コンソールで順序を確認してホストを識別することは可能です。n を 0 に設定すると、JRun はコネクタをグローバルフィルタとしてインストールします。このフィルタは、IIS の該当インスタンス内のすべての仮想ホストに適用されます。

マルチホストで、仮想サーバーが同じ scripts ディレクトリを共有していない場合は、jrun.dll と jrun.ini を必ず正しいディレクトリに入れ、グローバルフィルタを削除してください。

metaset および regset を使用した IIS からのコネクタのアンインストール

metaset および regset ユーティリティには、JRun コネクタを削除するオプションがあります。JRun またはお客様の製品をユーザのマシンからアンインストールするときは、このユーティリティを呼び出します。IIS からコネクタをアンインストールすると、デバッグを簡単に行うことができます。このセクションでは、metaset および regset を使用して IIS から JRun コネクタを削除する方法について説明します。

metaset および regset ユーティリティでは、jrun.dll ファイルや jrun.ini ファイルを実際に削除するのではなく、これらのファイルの登録を Web サーバーから削除します。アンインストールを完全に行うには、これらのファイルを手作業で削除する必要があります。

メモ
JRun コネクタをアンインストールするときは、regset または metaset ユーティリティを使用する前に Web サーバーを停止してください。

IIS 3.0 からのコネクタのアンインストール

regset ユーティリティは、Windows レジストリ内の設定を削除します。

たとえば、コネクタを削除するには、次のコマンドを実行します。

```
regset -d c:¥inetpub¥scripts -quiet
```

IIS 4.0 および 5.0 からのコネクタのアンインストール

metaset ユーティリティは、IIS メタベース内の設定を削除します。

たとえば仮想ホスト番号 1 からコネクタを削除するには、次のコマンドを実行します。

```
metaset -d /1 c:¥virt2¥inetpub¥scripts -quiet
```

グローバル コネクタを削除するには、次のコマンドを実行します。

```
metaset -d /0 c:¥inetpub¥scripts
```

-d オプションは、仮想ホスト番号 n のコネクタを削除します。仮想ホストには明確な番号が付けられませんが、Microsoft 管理コンソールで順序を確認してホストを識別することは可能です。n を 0 に設定すると、グローバル フィルタ (設定されている場合) が削除されます。

JRun サーバーの追加と削除

JRun をインスト ールすると、admin と default の 2 つの JRun サーバーがインスト ールされて起動されます。

JRun で提供されている Server クラスを使用すると、JRun サーバーの追加と削除を行ったり、JRun サーバーが存在するかどうかをプログラムまたはコマンド ラインからテストできます。サーバー設定を最大限に制御するために、このセクションの指示に従って JRun サーバーの追加と削除を手作業で行うこともできます。

JRun には、JMC の [新規サーバーの構成] パネルに独自のフィールドを追加する機能もあります。

このセクションでは、次の項目について説明します。

- 「プログラムによる JRun サーバーの追加と削除」48 ページ
- 「手作業での JRun サーバーの追加と削除」51 ページ
- 「JMC の [新規サーバーの構成] パネルの拡張」54 ページ

新規 JRun サーバーについて

コマンド ラインまたは JMC を使用して新規 JRun サーバーの既定のインストールを実行すると、その新規 JRun サーバーの概要を示す local.properties ファイルが作成されます。このファイルには次のプロパティが含まれています。

- jrun.server.displayname
- jrun.rootdir
- servlet.webapps
- servlet.services
- web.endpoint.main.port
- control.endpoint.main.port
- ejb.services

新規サーバーを Enterprise サーバーに指定した場合、local.properties ファイルには次のプロパティも含まれます。

- ejb.services
- ejb.ejipt.enableMessaging
- ejb.ejipt.classServer.port
- ejb.ejipt.homePort
- ejb.ejipt.classServer.host

プログラムによる JRun サーバーの追加と削除

このセクションでは、allaire.jrun.tools パッケージ内の Server クラスを使用して JRun サーバーの追加と削除を行う方法について説明します。コマンド ラインから Server クラスを使用したり、Java アプリケーション内からそのパブリック メソッドを呼び出すことができます。

Server ユーティリティを呼び出すには、クラスパスに *JRun のルート ディレクトリ*/lib/jrun.jar を含める必要があります。

Server クラスを使用して JRun サーバーを追加すると、次の作業が実行されます。

- 新規サーバーが jvms.properties ファイルに追加されます。
- そのサーバー用のディレクトリが *JRun のルート ディレクトリ*/servers ディレクトリの下に作成されます。
- そのサーバーの概要を示す local.properties ファイルが作成されます。

新規 JRun サーバーを使用するには、そのサーバー上で Web アプリケーションを作成する必要があります。詳細は、『JRun セットアップ ガイド』を参照してください。

コマンド ラインからの Server クラスの使用

コマンド ラインから Server ユーティリティを使用するには、次のコマンドを入力します。

```
% java [-classpath classpath] allaire.jrun.tools.Server
        [add|remove|exists] server [local.properties enterprise]
        jrun-home
```

次に例を示します。

```
% java -classpath c:¥jrun¥lib¥jrun.jar allaire.jrun.tools.Server add
        |new_server c:¥jrun¥
```

add

新規 JRun サーバーを追加します。必要に応じて新規 local.properties ファイルの場所を指定し、add アクションとともに enterprise オプションを設定することもできます。

片方のオプションを指定した場合は、両方とも指定する必要があります。

enterprise オプションはブール値であり、新規 JRun サーバーで EJB および JMS Enterprise 版の機能を有効にするかどうかを決定します。既定値は true です。47 ページの「新規 JRun サーバーについて」で説明したように、このオプションを設定すると Enterprise 関連のプロパティが local.properties ファイルに書き込まれます。

remove

既存の JRun サーバーを削除します。削除する前に JRun サーバーを停止する必要があります。

メモ

admin JRun サーバーを削除しないでください。削除すると JMC にアクセスできなくなります。

exists

サーバーが存在する場合は、"The server *'servername'* exists" というメッセージが返されます。存在しない場合は、"The server '*servername*' does not exist" というメッセージが返されます。このアクションによってサーバーの存在を調べる場合、そのサーバーが実行中である必要はありません。次に例を示します。

```
% java allaire.jrun.tools.Server exists default c:¥jrun¥
The server 'default' exists.

% java allaire.jrun.tools.Server exists nickdanger c:¥jrun¥
The server 'nickdanger' does not exist.
```

Java アプリケーションでの Server クラスの呼び出し

独自の Java アプリケーションから Server クラスのパブリック メソッドにアクセスして、プログラムで JRun サーバーの追加と削除を実行できます。このクラスを使用してサーバーの存在をテストすることもできます。このセクションでは、Server クラスのパブリック メソッドについて説明します。

コンストラクタの詳細

Server

```
public Server(java.lang.String serverName,
              java.lang.String jrunDir)
       throws java.io.IOException,allaire.jrun.tools.ServerException
```

新規サーバーを構築します。次の表では、パラメータについて説明しています。

パラメータ	説明
serverName	新規 JRun サーバーの名前
jrunDir	JRun のルート ディレクトリ

メソッドの詳細

add

```
public void add()
  throws java.io.IOException, allaire.jrun.tools.ServerException
```

新規 JRun サーバーを追加します。local.properties を空のファイルと見なし、サーバー ディレクトリを {*JRun のルート ディレクトリ*}/servers/{ サーバー } と見なして、既定により EJB と JMS を有効にします。

add

```
public void add(allaire.jrun.util.OrderedProperties ordProp)
  throws java.io.IOException, allaire.jrun.tools.ServerException
```

新規 JRun サーバーを追加します。サーバー ディレクトリを {*JRun のルート ディレクトリ*}/servers/{ サーバー } と見なし、EJB と JMS を有効にします。次の表では、パラメータについて説明しています。

パラメータ	説明
ordProp	新規サーバーの local.properties ファイルに保存されるプロパティ

add

```
public void add(allaire.jrun.util.OrderedProperties ordProp,
                boolean enterprise)
  throws java.io.IOException, allaire.jrun.tools.ServerException
```

新規 JRun サーバーを追加します。新規サーバー ディレクトリを {*JRun のルート ディレクトリ*}/servers/{ サーバー } と見なします。次の表では、パラメータについて説明しています。

パラメータ	説明
ordProp	新規サーバーの local.properties ファイルに保存されるプロパティ
enterprise	新規 JRun サーバーの EJB と JMS を有効にします。

add

```
public void add(allaire.jrun.util.OrderedProperties ordProp,
                java.lang.String serverDirNew,
                boolean enterprise)
  throws java.io.IOException, allaire.jrun.tools.ServerException
```

新規 JRun サーバーを追加します。次の表では、パラメータについて説明しています。

パラメータ	説明
ordProp	新規サーバーの local.properties ファイルに保存されるプロパティ
serverDirNew	新規 JRun サーバーのディレクトリの場所
enterprise	新規 JRun サーバーの EJB と JMS を有効にします。

remove

```
public void remove()
  throws java.io.IOException, allaire.jrun.tools.ServerException
```

JRun サーバーを削除します。削除する前に JRun サーバーを停止する必要があります。

exists

```
public boolean exists()
```

サーバーが存在するかどうかをチェックします。

手作業での JRun サーバーの追加と削除

このセクションでは、新規 JRun サーバーの追加と削除を手作業で行う方法について説明します。JMC を使用して新規 JRun サーバーを作成する方法については、『JRun セットアップ ガイド』を参照してください。

手作業での JRun サーバーの追加

新規 JRun サーバーを手作業で最も簡単に作成するには、既定のアプリケーションを含む default JRun サーバーのファイルおよびディレクトリ 構造をコピーし、そのコピーを変更して新規サーバーの内容を反映させます。

JRun サーバーを作成するには

1 新規サーバーの名前を付けたディレクトリ (/servers/foo など) に /servers/default ディレクトリをコピーします。

2 新しいディレクトリ (/servers/foo) 内の local.properties ファイルを開きます。local.properties ファイルでは JRun サーバー固有の設定を定義します。この設定には、名前やポート設定などの基本的なプロパティだけでなく、そのサーバーで公開するアプリケーションも含まれます。

新しい local.properties ファイルを次のように変更します。

a サーバーの新規名 (表示名) を反映するように jrun.server.displayname プロパティを変更します。この名前は JMC に表示されます。コードは次のようになります。

```
## jvm properties
# was:jrun.server.displayname=Default Server
jrun.server.displayname=Foo Server
```

b 「Web Application Settings」セクションから不要なアプリケーションを削除します。default-app を残す場合は、ほかの JRun サーバーの default-app マッピングと競合しないように URL マッピングを必ず変更してください。default サーバーにアプリケーションを追加していない場合、demo-app と invoice-app 以外のアプリケーションを削除する必要はありません。

たとえば、各アプリケーションには次のようなセクションがあります。

```
invoice-app.rootdir=C:¥¥Program Files¥¥Allaire¥¥JRun¥¥
 servers¥¥default¥¥invoice-app
invoice-app.class={webapp.service-class}
webapp.mapping./invoice=invoice-app
```

該当するアプリケーションの設定を削除します。

c default アプリケーションのみが含まれるように servlet.webapps プロパティを再設定します。行は次のようになります。

```
#was:servlet.webapps=default-app, demo-app
servlet.webapps=default-app
```

d ポート設定を固有のポートに変更します (これらの設定はファイル内の別々のセクションにあります)。次に例を示します。

```
web.endpoint.main.port=8101 #was:8100 (Web server port)
control.endpoint.main.port=53001 #was:53000 (Control port)
ejipt.classServer.port=2324 #was: 2323
ejipt.homePort=2334 #was 2333
```

メモ
これらのポート設定を変更しないと、バインドの例外が発生する場合があります。JRun のポートについては、『JRun セットアップ ガイド』を参照してください。

3 local.properties ファイルを保存します。

4 default-app を含むすべてのアプリケーションのディレクトリを /servers/foo から削除します。default サーバーにアプリケーションを追加していない場合、削除する必要があるのは default-app ディレクトリだけです。

5 *JRun のルート ディレクトリ*/lib ディレクトリ内の jvms.properties ファイルを開きます。JRun では jvms.properties ファイルを使用して、インスタンス化する必要があるサーバーが調べられます。新規サーバーの行を追加します。ファイルは次のようになります (NT の場合)。

```
admin=C:/Program Files/Allaire/JRun/servers/admin
default=C:/Program Files/Allaire/JRun/servers/default
foo=C:/Program Files/Allaire/JRun/servers/foo
```

サーバー名が行の先頭にあり、ディレクトリ名が行の末尾にあることに注意してください。

6 新規 JRun サーバーを起動します。

Windows の場合

a */JRun のルート ディレクトリ* /bin 内の jrun-default.bat ファイルを jrun-foo.bat にコピーします。

b jrun-foo.bat を編集して新規サーバーの名前を指定します。

```
@echo off
start jrun -start foo
```

c jrun-foo.bat ファイルを実行します。

d オプションで (NT のみ)、コマンド ライン ユーティリティを使用して新規サーバーを NT サービスとして追加することもできます。

```
% jrun -install "Foo Service" foo -quiet
```

e 次のコマンドを使用してサーバーを起動します。

```
% jrun -start foo
```

UNIX の場合

/jrun/bin 内の jrun コマンド ライン ユーティリティを使用します。

```
% jrun -start foo
```

7 JMC にログインしている場合は、ログアウトしてから admin サーバーを再起動します。

次回のログイン時には、新規 Foo サーバーが JMC の左側ペインに表示されます。Foo サーバーには Web アプリケーションがありません。

8 新規 JRun サーバーを使用するには、そのサーバー上で Web アプリケーションを作成する必要があります。詳細は、『JRun セットアップ ガイド』を参照してください。

手作業での JRun サーバーの削除

JRun サーバーの削除は十分に検討した上で行う必要があります。サーバーを削除する前に、そのサーバー内にある重要なファイルまたはアプリケーションを必ずバックアップしてください。このセクションでは、JRun サーバーを削除する方法について説明します。

注意
admin JRun サーバーまたは default JRun サーバーを削除しないでください。削除すると、これらのサーバー内のアプリケーションが失われます。

JRun サーバーを削除するには

1 削除する JRun サーバーを停止します。

2 そのサーバーのディレクトリとすべてのサブディレクトリを削除します (たとえば、/servers 内の /foo ディレクトリを削除します)。

3 *JRun のルート ディレクトリ*/lib ディレクトリ内の jvms.properties ファイルを開き、該当するサーバーの行を削除します。ファイルは次のようになります (NT の場合)。

```
admin=C:/Program Files/Allaire/JRun/servers/admin
default=C:/Program Files/Allaire/JRun/servers/default
#foo=C:/Program Files/Allaire/JRun/servers/foo
```

4 サーバーの起動スクリプトを削除します。

5 サーバーを NT サービスとして追加した場合は、コマンド ライン ユーティリティを使用してこのサービスを削除します。

```
% jrun -remove "Foo Server" -quiet
```

6 JMC にログインしている場合は、ログアウトしてから admin サーバーを再起動します。

次回のログイン時には、このサーバーが JMC の左側ペインに表示されません。

JMC の [新規サーバーの構成] パネルの拡張

前に説明したように、JMC を使用して JRun サーバーを追加すると、新規 JRun サーバーの概要を示す local.properties ファイルのみが作成されます。JRun では、JMC を使用して新規 JRun サーバーを追加する既定の機能を拡張するための方法がいくつか用意されています。

- JMC の [新規サーバーの構成] パネルにフィールドを追加して、新規 JRun サーバーに関する情報の入力をユーザに要求します。
- 任意の数の既定のプロパティを新規 JRun サーバーに自動的に追加します。

このセクションでは、admin JRun サーバーの local.properties ファイルを変更してこれらの操作を実行する方法について説明します。admin サーバー上で動作している jmc-app によって JMC の内容が制御されるため、local.properties ファイルを編集する必要があります。admin サーバーの local.properties ファイルの編集プロセスは、PropertScript ユーティリティを使用して自動化できます。詳細は、30 ページの「JRun プロパティのカスタマイズ」を参照してください。

[新規サーバーの構成] パネルへのフィールドの追加

JMC の [新規サーバーの構成] パネルにフィールドを追加し、JMC ユーザの応答をプロパティとして新規 JRun サーバーの local.properties ファイルに保存できます。新しいプロパティは、既存の JRun プロパティを置き換えることも、JRun サーバーまたは Web アプリケーションのコンテキストで特別な意味を持つ固有のプロパティにすることもできます。

たとえば、[新規サーバーの構成] パネルに [Logging Level] というフィールドを追加し、そのフィールドを local.properties 内の logging.loglevel プロパティにマッピングし、必要なログ レベル (情報、警告、エラーなど) の入力をユーザに要求できます。

このセクションでは、[Logging Level] フィールド およびプロパティの追加手順を示し、JMC の [新規サーバーの構成] パネルにフィールドを追加する方法について説明します。

[新規サーバーの構成] パネルにフィールドを追加するには

1 *JRun のルート ディレクトリ*/servers/admin/local.properties にある admin JRun サーバーの local.properties ファイルを開きます。

2 AddServer セクションを見つけます。

3 新規フィールドのトークン名を jmcAddServer.items プロパティに追加します。この名前は、local.properties ファイル内の新規プロパティを参照するときに使用されます。

たとえば、新規プロパティの名前を logSetting にする場合は、カンマ区切りリストの最後に logSetting を追加します。

```
jmcAddServer.items=displayName,webPort,controlPort,enterprise,ejbCla
     ssPort,ejbHomePort,logSetting
```

4 次のプロパティと値を追加します。この手順では、logSetting の属性を定義しています。

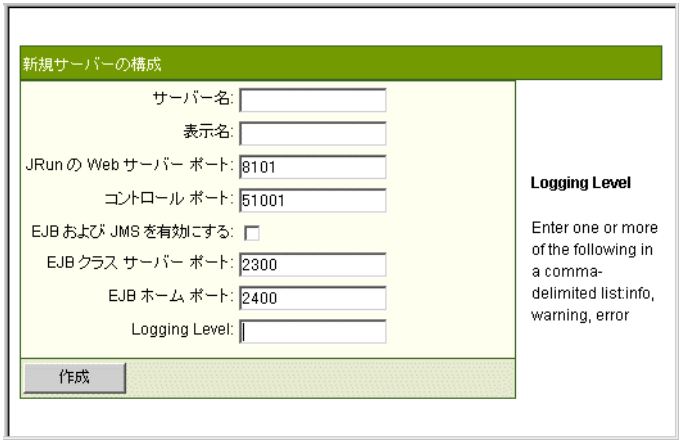
```
jmcAddServer.logSetting.title=Logging Level
jmcAddServer.logSetting.name=logging.loglevel
jmcAddServer.logSetting.type=string
jmcAddServer.logSetting.description=Enter one or more of the
     following in a comma-delimited list:info, warning, error
jmcAddServer.logSetting.optional=false
```

5 admin JRun サーバーの local.properties ファイルを保存して閉じます。

6 admin JRun サーバーを再起動します。

7 JMC で [*マシン名*] を選択します。

[新規サーバーの構成] パネルが新規フィールドとともに表示されます。



ユーザが [Logging Level] フィールドに「info」と入力して [作成] ボタンをクリックすると、既定のプロパティのほかに次の行が新規 JRun サーバーの local.properties ファイルに書き込まれます。

```
logging.loglevel=info
```

新規 JRun サーバーへの既定のプロパティの追加

このセクションの指示に従って、すべての新規 JRun サーバーに追加するプロパティの一覧を作成できます。このプロセスでは、新規フィールドの作成や新規サーバーを作成するユーザに対するプロンプトの表示は行われませんが、代わりにプロパティの一覧がファイルから読み込まれ、新しい local.properties ファイルに書き込まれます。

新規 JRun サーバーに既定のプロパティを追加するには

1 *JRun のルート ディレクトリ*/servers/admin/local.properties にある admin JRun サーバーの local.properties ファイルを開きます。

2 AddServer セクションを見つけます。

3 jmcAddServer.file プロパティでファイルを指定します。既定値は次のとおりです。

```
jmcAddServer.file=/lib/jmcAddServer.file
```

この場所は *JRun のルート ディレクトリ*からの相対パスです。

4 jmcAddServer.file または jmcAddServer.file プロパティで指定したファイルを開きます。このファイルの既定の内容は次のとおりです。

```
# Add here any properties that you wish to be added to a server
# when created in the JRun Management Console.
# See the admin server's local.properties for more information.
```

5 JMC で作成したすべての新規 JRun サーバーに追加するプロパティおよび値を追加します。

たとえば、すべての新規 JRun サーバーにカスタム ロガーを追加するには、jmcAddServer.file に次のプロパティを追加します。

```
logging.threadedlogger.listeners=filelogwriter,mydispatcher
logging.mydispatcher.class=allaire.jrun.logging.DispatchLogger
     logging.mydispatcher.events=allaire.jrun.logging.MetricsLogEve
     nt logging.mydispatcher.destinations=metricswriter
logging.metricswriter.class=allaire.jrun.logging.FileLogWriter
     logging.metricswriter.filename= {jrun.rootdir}/logs/
     {jrun.server.name}-metrics.csv
     logging.metricswriter.heading.line1=# JRun 3.0 Metrics
     logging.metricswriter.heading.line2=# Date, JVM Free
     Memory,Requests,Avg Request Time
     logging.metricswriter.format={date MM/dd
     HH:mm:ss},{log.message}
```

これらのプロパティでは mydispatcher という新しいリスナが追加されます。この DispatchLogger によって MetricsLogEvent タイプのログ イベントが受信され、metricswriter というロガーに転送されます。ほかのすべてのタイプのログ メッセージは無視されます。

新しいログ ライタが設定されてログ イベント が FileLogWriter に server-metrics.csv というファイル名で送信され、出力がフォーマットされます。

メモ

jmcAddServer.file の内容を編集しても、既存の JRun サーバーは影響を受けません。JMC を使用して追加された JRun サーバーのみが影響を受けます。

6 jmcAddServer.file を保存して閉じます。

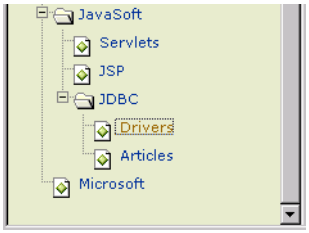
7 admin JRun サーバーを再起動します。

次に JMC の [新規サーバーの構成] パネルを使用して新規 JRun サーバーを作成するとき、jmcAddServer.file 内のすべてのプロパティが新規 JRun サーバーの local.properties ファイルに書き込まれます。

JMC の拡張

JRun を使用すると、JMC の左側パネルにある JRun ナビゲーション ツリーにカスタム リンクを追加できます。追加できるのは、メニュー、サブメニュー、およびオプションです。たとえば、Web サイト 上のテクニカル サポート ページや特別記事へのリンクを含めることができます。数多くのテクニカル サポート 記事にリンクする場合は、トピックごとに記事をグループ化し、サブメニューとしてナビゲーション ツリーに含めることができます。

次の例では、[JavaSoft] がメニューであり、[JDBC] がサブメニューであり、[Servlets]、[JSP]、[Drivers]、および [Articles] がオプションです。



メニュー名またはオプション名をクリックすると、対応するリンクが JMC のメイン ターゲット ウィンドウに表示されます。

JMC 拡張の構文

JMC の拡張は、admin JRun サーバーの local.properties ファイルにプロパティを追加することによって行われます。

トップレベルのオプションまたはメニューをサーバーの項目リストに追加する必要があります。

```
admin.jmcMenu.<server_name>.items=<options_list>
```

server_name は、JMC ナビゲーション ツリーで下にリンクを表示するサーバーの名前です。たとえば、default などです。

options_list は、ナビゲーション ツリーに表示するトップレベル オプションのカンマ区切りリストです。

name および link プロパティを使用して、オプションのリンク先および表示名を追加します。

```
admin.jmcMenu.<server_name>.<options_name>.name=<display_name>
admin.jmcMenu.<server_name>.<options_name>.link=<target_link>
```

option_name は、プロパティ ファイル内でこのリンクを参照するときに使用するリンクの名前です。この名前はスペースや特殊文字を含まない 1 つの単語でなければなりません。たとえば、sun などです。次に、この *option_name* を使用してサブメニュー項目を追加します。

display_name では、JMC ナビゲーション ツリーに表示するリンクの名前を定義します。たとえば、JavaSoft などです。この値にはスペースを含めることができます。

target_link は、ユーザがリンクをクリックしたときに JMC のターゲット ウィンドウに表示される Web ページです。

リンクの項目リストを使用してサブメニューを追加します。

```
admin.jmcMenu.<server_name>.<options_name>.items=<sub-menu_list>
```

sub-menu_list は、*options_name* サブメニューの下に表示されるサブメニュー オプションのカンマ区切りリストです。任意の数のネストされたメニューを作成できます。

これらの変更を有効にするには、admin JRun サーバーを再起動して JMC にログインし直す必要があります。新しいメニューやオプションが表示される JRun サーバーを再起動する必要はありません。

JMC 拡張の追加

JMC ナビゲーション メニューに新しいリンクを追加するには

1 admin JRun サーバーの local.properties ファイルを開きます。

2 項目リストに新しいオプションを追加します。たとえば、sun というオプションを default JRun サーバーに追加するには、次のように入力します。

```
admin.jmcMenu.default.items=sun
```

新しいオプションを JMC に初めて追加する場合は、行全体を入力する必要があります。それ以外の場合は、カンマ区切りリストに新しいオプションを追加します。

3 新しいオプションの名前およびリンクを追加します。

```
admin.jmcMenu.default.sun.name=JavaSoft
admin.jmcMenu.default.sun.link=http://java.sun.com
```

4 必要な場合は、項目リストを使用してサブメニュー内のオプションを現在のオプションに追加できます。サブメニュー内の jdbc というオプションを sun に追加するには、次のように入力します。

```
admin.jmcMenu.default.sun.items=jdbc
```

5 次に、サブメニュー内の jdbc オプションの名前およびリンクを追加します。

```
admin.jmcMenu.default.jdbc.name=JDBC
admin.jmcMenu.default.jdbc.link=http://java.sun.com/jdbc
```

6 JMC からログアウトします。

7 admin JRun サーバーを再起動します。

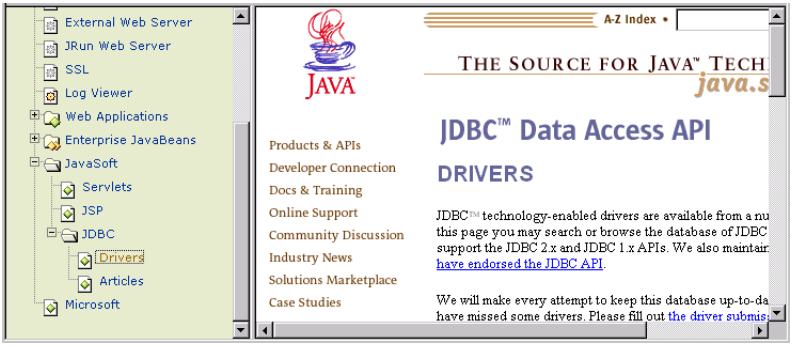
8 JMC にログインし直し、新しいリンクをテストします。

JMC 拡張の例

次の例は、admin JRun サーバーの local.properties ファイルの内容です。これにより、58 ページの画像に示されているメニュー、サブメニュー、およびオプションが表示されます。

```
admin.jmcMenu.default.items=sun,microsoft
admin.jmcMenu.default.sun.items=servlets,jsp,jdbc
admin.jmcMenu.default.sun.name=JavaSoft
admin.jmcMenu.default.sun.link=http://java.sun.com
admin.jmcMenu.default.servlets.name=Servlets
admin.jmcMenu.default.servlets.link=http://java.sun.com/products/
     servlets
admin.jmcMenu.default.jsp.name=JSP
admin.jmcMenu.default.jsp.link=http://java.sun.com/products/jsp
admin.jmcMenu.default.jdbc.items=jdbc_drivers,jdbc_articles
admin.jmcMenu.default.jdbc.name=JDBC
admin.jmcMenu.default.jdbc.link=http://java.sun.com/products/jdbc
admin.jmcMenu.default.jdbc_drivers.name=Drivers
admin.jmcMenu.default.jdbc_drivers.link=http://
     industry.java.sun.com/products/jdbc/drivers
admin.jmcMenu.default.jdbc_articles.name=Articles
admin.jmcMenu.default.jdbc_articles.link=http://java.sun.com/
     products/jdbc/articles.html
admin.jmcMenu.default.microsoft.name=Microsoft
admin.jmcMenu.default.microsoft.link=http://www.microsoft.com
```

次の図は、ナビゲーション要素とそのサブ要素が追加された JMC を示します。



終了ハンドラの使用

JRun でカスタム終了ハンドラを実装することによって、OEM ユーザは JRun の終了動作を制御できます。たとえば、JRun の終了やエラーが発生した場合に、独自のエラーメッセージを表示したり、ファイルを消去できます。

既定では、JRun は次の場合に `System.exit()` を呼び出します。

終了コード名	説明	終了コード値
`RESTART_EXIT`	再起動コマンドによって JRun が終了しました。	戻り値は 5 です。
`FATAL_ERROR_EXIT`	重大なエラーによって JRun が終了しました。	戻り値は 6 です。
`STATUS_RUNNING_EXIT`	ステータス コマンドが受信され、JRun は現在実行されています。	戻り値は 7 です。
`STATUS_NOT_RUNNING_EXIT`	ステータス コマンドが受信され、JRun は現在実行されていません。	戻り値は 8 です。

通常の終了時には、JRun によって `System.exit()` が呼び出されません。jrun.exe (Windows) とシェル スクリプト (UNIX) によってステータス コードがチェックされ、JRun を再起動する必要があるかどうかが判断されます。

OEM ユーザが独自のカスタム終了ハンドラを作成できるように、JRun には OEM ユーザ用の allaire.jrun.JRunExitHandler インターフェイスが含まれています。このインターフェイスには 1 つのメソッドがあります。

```
void exitJRun(int exitCode)
```

JRunExitHandler を実装するクラスの新しいインスタンスを生成した後、その JRun クラス上で新しい終了ハンドラを設定できます。このインスタンスは静的なフィールドに格納されるため、特定の時点でアクティブにできる終了ハンドラは 1 つだけです。

実装例は次のとおりです。

```
import java.io.*;
import allaire.jrun.*;
/**
* ベース JRunExitHandler クラスを拡張して、終了を無視するための
* カスタム実装を行います。
*/

public class MyJRun implements JRunExitHandler
{
   public static void main(String[] args) throws IOException
   {
     JRun.setExitHandler(new MyJRun());
     JRun.main(args);
   }

   public void exitJRun(int exitCode)
   {
     // 終了条件に対応した終了コードを使用して、
     // ここに終了処理コードを記述します。
   }
}
```

上記のサンプル クラスでは JRunExitHandler が実装されます。`main()` メソッドによってクラスがインスタンス化され、JRun 上で終了ハンドラが設定され、次に JRun が呼び出されます。JRun が異常終了するか、再起動要求を受信すると、JRun の既定の `exitJRun()` の代わりに、新規のカスタム `exitJRun()` メソッドが呼び出されます。

第 2 章

ホスティング/ISP

この章には、インターネット サービス プロバイダ (ISP)、Web ホスティング会社、およびその他の上級 JRun ユーザ向けの情報が含まれています。

目次

JRun でのロード バランスとクラスタリング

専用ホスティング環境では、JRun サーバー、Web サーバー、データベース、およびその他のサーバーは異なるマシン上に配置できます。JRun および Web サーバーの複数のインスタンスを使用することにより、複数のプロセッサ間に負荷を分散できます。ロードバランスだけでなく、アラーム通知やフェイルオーバのサポートも利用できます。これらの機能は、JRun の Enterprise 版に含まれる Allaire ClusterCATS によって提供されます。

ClusterCATS は、JRun Enterprise 版に含まれているソフトウェアベースのロード バランスおよびクラスタリング ソリューションです。ClusterCATS のロード バランスおよびクラスタリング ツールを使用するには、クラスタ内のすべてのサーバーに ClusterCATS をインストールする必要があります。

このセクションでは、ClusterCATS で提供されるテクノロジを紹介します。詳細は、『Allaire ClusterCATS の使用』を参照してください。

概要

ClusterCATS を使用すると、Web アプリケーションをサポートする適切なクラスタを簡単に作成、最適化、および保守できます。ClusterCATS は、Windows NT、Solaris、および Linux プラットフォームで動作し、Microsoft IIS、Netscape Enterprise Server、Apache などの主要な基幹業務 Web サーバーと連動します。ClusterCATS は、リモートで管理できます。ClusterCATS には次の強力な機能が含まれています。

- 負荷しきい値と転送しきい値をサーバーごとに設定します。
- アプリケーション対応ロード バランス機能とセッション対応ロード バランス機能を使用してロード バランス方式を最適化します。
- エラーを自動的に検出して復旧します。
- 使用可能なサーバーにトラフィックを自動的に転送します。
- 管理者に問題を自動的に通知します。

次のセクションでは、これらの方法について簡単に説明します。

フェイルオーバ

使用不能になったサーバーを冗長サーバーにフェイルオーバする機能は基幹業務アプリケーションに不可欠であり、これによってアプリケーションの信頼性の高い継続的な動作が保証されます。サーバーのフェイルオーバは、ClusterCATS インストールの実行中に選択するオプションです。

負荷しきい値

ClusterCATS は、クラスタリングされたサーバーに対する HTTP トラフィックの量を識別して管理することによって、顧客の Web アプリケーションが常に最適なパフォーマンスで動作するようにします。クラスタ内の各サーバーで負荷しきい値を設定することによって、Web サイトの可用性およびパフォーマンスを制御し、管理できます。しきい値の設定は、多くの場合、Web サイトのアーキテクチャと一連の処理リソースをどこに割り当てるかによって決まります。

HTTP 転送時に、ClusterCATS は HTTP サーバーのステートと JRun サーバーの負荷に基づいてクラスタのステートを評価します。この方式は、集中型および分散型のいずれの ClusterCATS 構成でも同じです。すべての Web サーバーが 1 つのサイト上にある集中型 ClusterCATS クラスタの場合、ClusterCATS は、サーバーがビジー状態か、または動作が制限されている場合に限り転送します。

クラスタ メンバごとに、2 つの負荷しきい値を設定できます。

- **完全転送しきい値** サーバーが、そのパフォーマンスを著しく低下させない程度に、あるいは使用不能にならない程度に処理できる最大負荷を表します。
- **漸次転送しきい値** サーバーのパフォーマンスが低下しないうちに、あるいは使用不能になる前に、サーバーがクラスタ内の負荷の少ないほかのサーバーに HTTP 要求の転送を開始する時点を表します。

負荷しきい値の設定方法の詳細は、『ClusterCATS の使用』を参照してください。

JRun プローブの使用

JRun プローブは、顧客のクラスタリングされたサーバー上で JRun サーバーが正常に動作していることを確認する ClusterCATS の高可用性機能です。これらは、特定の JRun URL を定期的にテストし、返されるページに含まれるユーザ定義文字列と比較してその妥当性を確認します。

トラフィックの転送

妥当性テストが成功すると、着信 HTTP 要求はプローブが存在するサーバーに引き続き送信されます。ただし、テストが失敗した場合は (URL が見つからない、タイムアウトした、またはアクセスされたページ内のユーザ指定文字列が返されない場合)、ClusterCATS はサーバーを制限し、クラスタ内の使用可能なほかのサーバーに要求を転送します。ClusterCATS は制限されたサーバーのテストを続行し、制限されたサーバーはプローブが有効な値を返すとすぐに使用可能であるとみなされます。

フェイルオーバ

JRun サーバーがハングするか故障した場合、ClusterCATS は障害のあるサービスの復旧を試みます。JRun サービスが復旧すると、プローブは JRun サーバーを再起動し、そのサーバーへの HTTP トラフィックの送信を開始します。

JRun プローブの使用方法の詳細は、『Allaire ClusterCATS の使用』を参照してください。

管理者アラーム通知の使用

ClusterCATS のアラーム通知機能を使うと、クラスタ内で発生する重要なイベントについて即座にフィードバックを得ることができます。イベントによりアラームが発生すると、ClusterCATS が 1 人または複数のユーザに電子メールで通知します。

次の表は、発生する可能性があるイベントと、イベントごとの通知スケジュールの一覧です。

イベント タイプ	通知のタイミング
ディスクの障害	即時
HTTP サーバーの障害	即時
サーバー ビジー警告	24 時間ごと
サーバーへ未着信	即時
Web サーバー フェイルオーバ	即時
JRun プローブの障害	即時

選択したイベントが発生すると、ClusterCATS によって、指定したユーザに電子メールメッセージが送信されます。アラーム通知の使用方法の詳細は、『Allaire ClusterCATS の使用』を参照してください。

JMC へのアクセスの保護

JRun は、JRun 管理コンソール (JMC) を使用して管理されます。JMC は、JRun サーバー、JRun サーバーに関連付けられている JRun Web サーバー、および外部 Web サーバー接続を設定するためのブラウザベースの管理ユーティリティです。

既定では、JMC は、admin JRun サーバー上で、admin JRun サーバーに関連付けられている JRun Web サーバー (JWS) と連動します。外部 Web サーバーとは接続されません。通常、admin サーバーの JWS にはポート 8000 を介してアクセスします。

JMC へのリモート アクセスを許可する場合は、その設定時に予防措置を講じておく必要があります。JMC を保護する方法には、次のものがあります。

- ユーザの管理
- 外部 Web サーバーを介した admin JRun サーバーへの安全なリモート アクセスの提供
- JMC にアクセスする際の、ホストベースの認証の使用

このセクションでは、これらの方法について説明します。

JMC ユーザの管理

新規 JMC ユーザ アカウントを追加する際は、次の点に注意してください。

- Java の設定

  ユーザは、自分の JRun サーバーの [Java の設定] パネルにアクセスできます。これにより、JRun 環境にその他の変更を加えるだけでなく、Java 引数を追加および削除し、それらの JVM のクラスパス設定を変更できます。これは、リソースやセキュリティ上の問題となる可能性があります。[Java 引数] フィールドの使用を制限する、作成したポリシーを強制するか、またはネットワーク OS を使用してファイル権限に制限を加えることを検討してください。

- パスワード

  JMC へのリモート アクセスを許可する場合は、JMC のユーザが必ず推測しにくいパスワードを使用し、頻繁にパスワードを変更するようにしてください。現在の実装では、一定期間後にユーザにパスワードを変更するように警告することも、推測しにくいパスワードを選択するように強制することもありません。`admin` パスワードを推測しにくいものにし、頻繁に変更するようにしてください。

- JWS の制限

  JMC アプリケーションは、既定では、admin JRun サーバー上で実行されます。admin JRun サーバーは、ユーザのニーズを満たすのに十分に強力な Web サーバーではありません。その場合は、admin JRun サーバーを外部の Web サーバーに接続できます。詳細は、68 ページの「外部 Web サーバーを介した JMC へのリモート アクセス」を参照してください。

各顧客に対して、顧客独自の JRun サーバー、および 1 つ以上の JMC ユーザ アカウントを提供できます。顧客は、データ ソースの変更、アプリケーションの追加と削除、およびその他のタスクを行うために、JMC 内の JRun サーバー設定値へのアクセス権が必要となる場合があります。

JRun 管理者は、JMC の [ユーザの管理] パネルを使ってユーザ アカウントを作成し、1 つ以上の JRun サーバーへの顧客のアクセスを制限できます。また、PropertyScript の adduser ディレクティブを使用して、JMC ユーザの追加、変更、および削除プロセスを自動化できます。PropertyScript ユーティリティの使用方法の詳細は、30 ページの「JRun プロパティのカスタマイズ」を参照してください。

外部 Web サーバーを介した JMC へのリモート アクセス

jmc-app を実行する JWS は、ユーザのニーズを満たすことのできる強力な Web サーバーではありません。その場合は、admin JRun サーバーを Apach や IIS などの外部 Web サーバーに接続できます。その後、admin サーバーの JWS を無効にします。これにより、ユーザや顧客は JMC に安全にリモート アクセスを行うことができます。

外部 Web サーバーを介した JMC へのリモート アクセスを設定するには

1 admin JRun サーバーを外部 Web サーバーに接続します。

これらのサーバーが分散環境にある場合は、コネクタ ウィザードを使用して接続するか、『JRun セットアップ ガイド』の説明に従って接続します。

2 接続をテストします。

たとえば、外部の Web サーバーがポート 8001 上で受信している場合、次の URL を開きます。

```
http://www.yourdomain.com:8001/security/login.jsp
```

JMC ログイン ページが表示されます。

3 JMC に接続されている JWS を停止します。これにより、新たに接続された外部 Web サーバーを介してのみ JMC にアクセスできるようになります。これを行うには、admin サーバーの local.properties ファイル内の `servlet.services` プロパティから `web` サービスを削除します。

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

4 admin JRun サーバーを再起動します。

あるいは、admin サーバーを外部 Web サーバーに接続するのではなく、admin サーバーの JWS のクライアント IP フィルタ設定を編集することも可能です。この設定は、JMC の [JRun Web サーバー] パネルで行うことができます。詳細は、69 ページの「JWS へのホストベースの認証の設定」を参照してください。

JWS へのホストベースの認証の設定

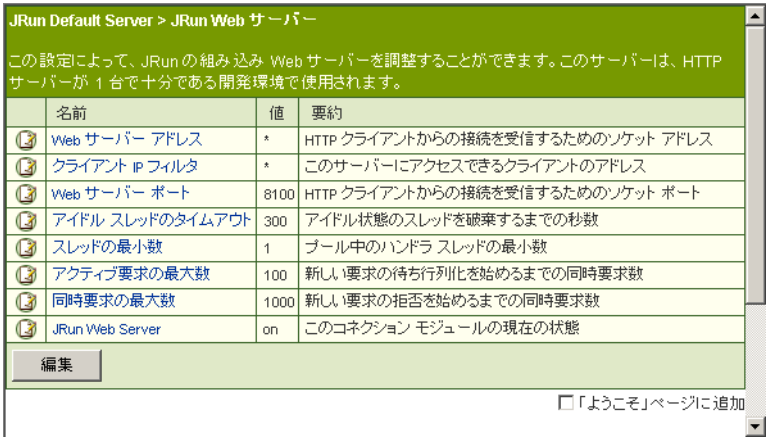
JWS 上で jmc-app を実行する場合 (既定の設定)は、JMC の [JRun Web サーバー]パネルを使用してホストベースの認証を指定できます。クライアント IP フィルタ設定は、JWS にアクセスできる IP アドレスの定義されている一覧と照合することによって、要求元に基づいてアクセスを制御します。

メモ
既定では、JWS はすべてのクライアントからの要求を受け入れます。

JWS のクライアント IP フィルタを編集するには

1 JMC の左側のペインで、[***マシン名***] > [*JRun サーバー名*] > [JRun Web サーバー] を選択します。

[JRun Web サーバー] パネルが表示されます。



2 右側のペインで、[クライアント IP フィルタ] フィールドをクリックします。

JRun Web サーバーの編集ウィンドウが表示されます。

3 IP アドレスを入力します (複数の場合はカンマで区切ります)。

上記の IP アドレスを使用するクライアントのみが、JRun Web サーバーとそのアプリケーションにアクセスできます。アスタリスク (*) は、すべてのクライアント がその Web サーバーにアクセスできることを意味します。

* を、リモート 管理を実行するユーザの IP アドレスに変更できます。クライアント IP フィルタ設定は、ここで定義された IP アドレスと照合することによって、要求元に基づいてアクセスを制御します。

この方法は、ファイアウォールまたはクライアント マシンの真の ID をマスクするほかのサービスを介して JMC にアクセスする場合に問題となることがあります。この場合、68 ページの「外部 Web サーバーを介した JMC へのリモート アクセス」で説明されているように、JWS を無効にし、外部 Web サーバーへの接続を介して jmc-app にアクセスする必要があります。

4 変更を適用するには、[更新] ボタンをクリックします。

5 JRun サーバーを再起動します。

JRun 環境の保護

このセクションでは、共有ホスティング環境での JRun アプリケーション サーバーの保護に関する問題について説明します。

Java アプリケーション サーバーの概要

各 JRun サーバーは、JRun サーバー起動時、サーバー独自の Java Virtual Machine (JVM) にロードされます。その JVM は、JRun サーバーがコンパイルして、実行するサーブレット、EJB、および JSP のみを実行します。これにより、各 JRun サーバーがほかの JRun サーバーの動作に影響を与えることがなくなり、JRun サーバーが保護されます。その結果、default JRun サーバーとその JVM が応答を停止しても、ほかのすべての JRun サーバーとその JVM は問題なく動作します。

次は、JVM およびサーブレット エンジンの特性の一部を示した概要で、JRun サーバーがそれらの関連 JVM とどのように対話するかについて説明します。

- JRun サーバーは、サーブレット、EJB、および JSP スレッドを処理します。
- JVM は、オブジェクト クラス ファイルをロードして検証します。これにより、クラスの不正な動作を防ぐことができます。
- JVM は、アプリケーションのメモリ管理を処理します。JVM メモリ使用率の監視方法の詳細は、83 ページの「リソース使用率の制限」を参照してください。
- JVM はガーベッジ コレクションを実行します。
- Java の例外処理によって、例外は捕捉されるまで、呼び出し元のスタックに伝搬されます。例外が明示的に捕捉されない場合は、JVM がクラッシュすることなく例外を捕捉します。
- JVM 命令セットはメモリを直接示さず、それ自体の入出力命令はありません。その代わり、JVM は、ネイティブ ライブラリを介して下位システムと通信します。これらのネイティブ ライブラリは、Java セキュリティ マネージャでの権限の検証後にのみアクセスできます。Java セキュリティ マネージャの使用方法の詳細は、72 ページの「Java セキュリティについて」を参照してください。

Java セキュリティについて

サーブレット /JSP ホスティングを提供する場合は、Java アプリケーション環境に対して厳密なセキュリティ マネージャおよびポリシー ファイルを設定することでメリットが得られます。これにより、JVM のシャットダウンやシステム ファイルへの書き込みなど、望ましくない結果を生じるサーブレットまたは JSP を顧客が実行できないようにすることができます。

メモ
ただし、ファイアウォール、OS レベルのセキュリティ、およびその他の従来の方法などのセキュリティ方法に代わるものはありません。

このセクションでは、Java セキュリティ マネージャを使用して、代表的な ISP アーキテクチャに対して JRun インストールを設定する方法を説明します。ただし、このトピックは内容が膨大で複雑なため、ほかの参考資料も参照する必要があります。

Java セキュリティ マネージャの使用

Java 言語で構築されたセキュリティ マネージャを使用して、ユーザがアクセスするクラスおよびメソッドを制御できます。これを行うには、java.policy ファイルを作成し、そのポリシー ファイルを JRun サーバーの Java 引数で指定します。

Java セキュリティ マネージャを使用するには

1 目的の権限が含まれている java.policy ファイルを編集します。

2 JRE の java.security ファイルに、この新規 java.policy ファイルを参照する `policy.url` を追加します。

3 JMC の `Java Arguments` プロパティを変更します。

4 JRun サーバーを再起動します。

次のセクションで、手順 1 ～ 3 を詳しく説明します。

java.policy ファイルの編集

既定の java.policy ファイルは、<*jdk*>¥jre¥lib¥security にあります。新規のポリシーファイルを作成し、目的の場所にそのファイルを格納します。ただし、この場合は、java.security ファイル内に新規の場所を指定する必要があります。既定の場所を使用する場合も、java.security ファイル内でその場所を明示的に示す必要があります。後で、Java 引数として java.security ファイルの場所を JMC 内で指定します。

既定のポリシーファイルでは、Java 拡張ディレクトリからロードされたすべてのクラスに対してすべてのアクセス権が与えられます。システム以外のクラスには、リストされているシステムプロパティに対する読み取り権が与えられ、1024 より大きいポート番号を持つソケットを使用して応答できます。その他のクラスはすべて、スレッドの `stop` メソッドを呼び出すことができます。

ポリシーファイルは、手作業で編集するか、Sun の JDK に含まれている `policytool` アプリケーションを使って編集できます。`policytool` 使用方法の詳細は、Sun の Java マニュアルを参照してください。

ポリシー ファイルのサンプル

このセクションに記載するサンプルポリシーファイルには、ISP に適している設定値が一部含まれています。サンプルでは、顧客がサーブレットと JSP の処理に default JRun サーバーを使用することを前提としています。

このポリシーファイルは、JRun の基本機能に必要な権限を提供します。この権限には、JSP の処理、サーブレットの動的な再ロード、システムおよび JRun プロパティの読み取り、ログへの書き込み、サーブレット /JSP での `sendRedirect` および `RequestDispatcher` の使用があります。ただし、複数ユーザのためのポリシー、JDBC ドライバを使用するデータベースへの接続に対するアクセス権、JNI のためのネイティブライブラリのロードなど、より複雑な設定には対応していません。

このファイルを使用するには、自分のファイルシステムと JRun インストールに一致するように、すべてのディレクトリとポートを変更します。このファイルは、admin JRun サーバー (8000)、default JRun サーバー (8100)、および `proxyport` (55555) については既定のポートを想定していますが、設定に適合するようにポートを変更できます。

java.policy ファイルのカスタマイズ、独自の java.policy ファイルの作成、および Java セキュリティアーキテクチャの知識に関する追加情報については、Java のマニュアルを参照してください。

java.policy ファイルのサンプル

```
/* このファイルは例示を目的としており、実際の実装には適していません。*/

grant {
  permission java.io.FilePermission "D:¥¥JRun3", "read";
  permission java.io.FilePermission "D:¥¥JRun3¥¥-", "read";
  permission java.io.FilePermission "D:¥¥JRun3¥¥logs¥¥*", "read,write";
  permission java.io.FilePermission "D:¥¥JRun3¥¥lib¥¥*", "read";

  /* /servlets ディレクトリにサーブレットを格納する旧式メソッドをサポートするには、
         次のアクセス権をコメント解除します。*/
  // permission java.io.FilePermission "D:¥¥JRun3¥¥servlets", "read, write";

  permission java.io.FilePermission "C:¥¥jdk1.2.2¥¥jre¥¥lib¥¥-", "read";
  permission java.io.FilePermission "C:¥¥jdk1.2.2¥¥lib¥¥tools.jar", "read";

  permission java.io.FilePermission
        "D:¥¥JRun3¥¥servers¥¥default¥¥default-app¥¥WEB-INF¥¥-", "read, write";
permission java.io.FilePermission
        "D:¥¥JRun3¥¥servers¥¥default¥¥default-app¥¥WEB-INF¥¥jsp, "delete";
  permission java.io.FilePermission "<<ALL FILES>>", "execute";

  /* JRun を介してネイティブ Web サーバーからドキュメントを使用可能にする場合は、
         次のアクセス権をコメント解除し、Web サーバーのドキュメント ルートに一致させます。*/
  // permission java.io.FilePermission "C:¥¥Inetpub¥¥wwwroot¥¥-", "read";

  permission java.net.SocketPermission "127.0.0.1:8000", "accept,listen,resolve";
  permission java.net.SocketPermission "127.0.0.1:8100", "accept,listen,resolve";
  permission java.net.SocketPermission "127.0.0.1:1024-", "accept,resolve";
  permission java.net.SocketPermission "127.0.0.1:51000", "accept,listen,resolve";
  permission java.net.SocketPermission "*.com", "connect,resolve";
  permission java.net.SocketPermission "*.org", "connect,resolve";
  permission java.net.SocketPermission "*.net", "connect,resolve";
  permission java.util.PropertyPermission "*", "read";
  permission java.lang.RuntimePermission "setIO";
  permission java.lang.RuntimePermission "exitVM";
  permission java.lang.RuntimePermission "stopThread";
  permission java.lang.RuntimePermission "createClassLoader";
}
```

java.security への policy.url の追加

JRE が新規のセキュリティ ファイルを認識する前に、JRE の <*java_home*>/jre/lib/security/java.security ファイル内に、java.policy ファイルの `policy.url` を追加する必要があります。たとえば、ファイルに 次の3 行を追加します。

```
policy.url.1=file:${java.home}/lib/security/java.policy
policy.url.2=file:${user.home}/.java.policy
policy.url.3=file:/D:/path/to/your_jrun.policy
```

既定の java.policy ファイルを変更し、それを既定の場所に配置しておいた場合は、java.security ファイル内に新しい場所を指定する必要はありません。

JMC での Java 引数の変更

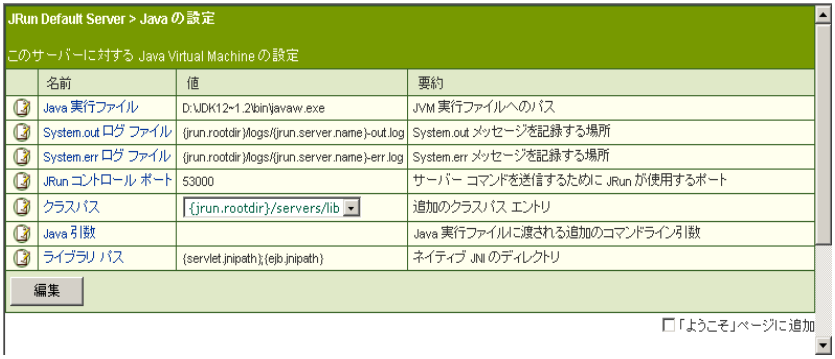
新規のポリシー ファイルを参照する java.security ファイルを JVM が読み込めるように、JVM への引数として `java.security.manager` を追加する必要があります。

JRun サーバーにはそれぞれ独自の Java 引数セット があり 、JVM の初期設定時に JVM に渡されます。この設定値は、JRun サーバーの local.properties ファイル内に `user.javaargs` として格納されています。このセクションでは、JMC を使用して `user.javaargs` プロパティの設定を変更する方法について説明します。

JVM の java.policy ファイルを指定するには

1 JMC の左側のペインで、[***マシン名***] > [*JRun サーバー名*] > [Java の設定] を選択します。

[Java の設定] パネルが表示されます。



2 右側のペインで [編集] をクリックします。

Java 設定値の編集ウィンドウが表示されます。

3 [Java 引数] フィールドで、JRun が JVM を起動するときに JVM 実行可能モジュールに渡す引数を入力します。次に例を示します。

```
-Djava.security.manager
```

4 変更を適用するには、[更新] ボタンをクリックします。

5 JRun サーバーを再起動します。

ポリシー ファイルの保護

ポリシー ファイルを保護するには次のような方法があります。

- JRun プロセスがファイルへの書き込み権限を持っていないファイル システム上に java.policy および java.security ファイルを保存します。
- ユーザがポリシー ファイルを制御できないように、JMC へのアクセス権をユーザに与えないようにします。
- `user.javaargs` プロパティは、JRun サーバーの local.properties ファイルに保存されています。JRun サーバーの local.properties ファイルを読み取り専用にすることによって、ユーザがポリシーファイルを制御できないようにすることができます。この方法を使用すると、ユーザに JMC へのアクセス権を与える一方で、それらの設定値をユーザが変更できないようにすることが可能です。

74 ページのサンプル ポリシーファイルでは、JMC を介して JRun のプロパティ ファイルを変更する権限は設定されていません。ユーザ独自の JRun サーバーの設定値をユーザが変更できるようにするには、JMC へのアクセス権をユーザに与え、プロパティ ファイルのディレクトリ内のファイルへの書き込み権限を追加します。

JMC ユーザの詳細は、67 ページの「JMC ユーザの管理」を参照してください。

JRun システムの保護

分散環境でセキュリティを実装する場合には検討事項がいくつかありますが、このセクションでは、ユーザが実行できる次の JRun 固有のアクションについて説明します。

- **admin サーバーの JWS をオフにします。**既定の 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 サーバーがセットアップされます。JWS のシャットダウンの詳細は、77 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。この設定の詳細は、77 ページの「コネクタのホストベース認証の設定」を参照してください。
- **admin サーバーの JWS を SSL 対応にします。**JRun では、Sun の JDK を使用する場合に JRun Web サーバー (JWS) を拡張するための Secure Socket Layer (SSL) プロトコルの使用がサポートされています。これにより、ユーザが安全なソケットを介してアクセスできるように、admin JRun サーバーに SSL を適用できます。詳細は、『JRun セットアップ ガイド』を参照してください。
- 既定サーブレットを無効にします。

JWS をオフにする方法

JRun をインストールすると、default および admin の 2 つの JRun サーバーと、これらのサーバーに接続された 2 つの all-Java JWS が作成されます。既定では、これらの Web サーバーはそれぞれポート 8000 と 8100 で応答します。ほとんどの ISP では受信ポートアクセスをファイアウォールで制限していますが、使用しないサービスはオフにする必要があります。このセクションでは、JRun サーバーの JWS をオフにする方法について説明します。

メモ
admin サーバーの JWS をオフにすると、JMC を開くことができなくなります。JRun のリモート管理を行うには、admin JRun サーバーをより強力な外部 Web サーバーに接続する必要があります。リモート管理の詳細は、68 ページの「外部 Web サーバーを介した JMC へのリモート アクセス」を参照してください。

JWS をオフにするには

1 JRun サーバーの local.properties ファイルを開きます。このファイルは、*JRun のルート ディレクトリ*/servers/*サーバー名*/local.properties にあります。

2 サーブレット サービスの一覧から web を削除することによって、servlet.services プロパティから web サービスを削除します。次に例を示します。

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

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の設定

JRun を実行するマシンと Web サーバーを実行するもう 1 台のマシン間の接続を作成したら、承認されていないユーザが 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 ベース認証が用意されています。これにより、IP アドレスの定義済みセットのホストだけが、JRun サーバーに要求を送信できるようになります。

JRun 管理コンソール (JMC) の [外部 Web サーバー] 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。現在、外部 Web サーバーと JRun サーバーの間のトラフィックを保護するための SSL やその他の暗号化テクノロジを使用することはできません。

メモ
既定の設定では、JRun サーバーは「すべての」IP アドレスからの要求を受け付けます。

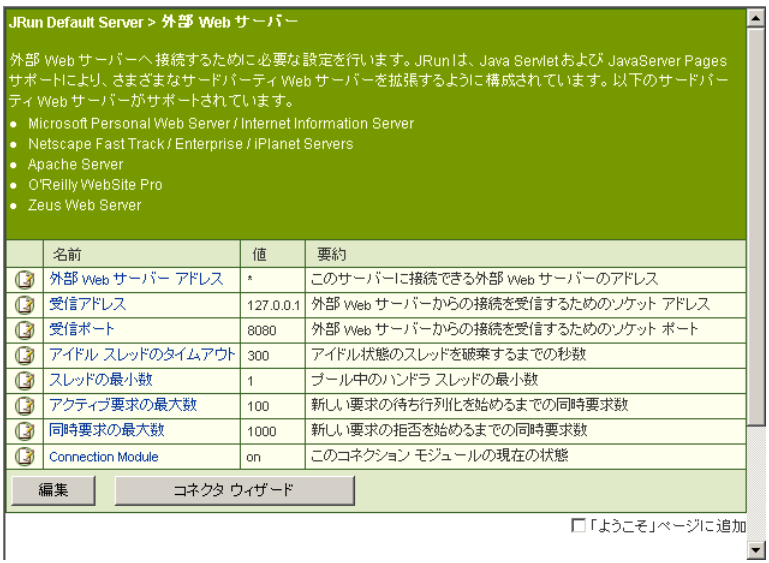
Web サーバーと JRun 間の接続をロックするには

1 JMC の左側のペインで、[**マシン名**] >[*JRun サーバー名*] > [外部 Web サーバー] を選択します。

メモ

外部 Web サーバーに JRun サーバーを接続する際に、コネクタ ウィザードをまだ実行していない場合は、コネクタ ウィザードを実行するように要求されます。

[外部 Web サーバー] パネルが表示されます。



2 右側のペインで、[外部 Web サーバー アドレス] フィールドをクリックします。

外部 Web サーバーの編集ウィンドウが表示されます。

3 IP アドレスを入力します (複数の場合はカンマで区切ります)。JRun サーバーは、それらの IP アドレスを持っている Web サーバーからの要求にのみ応答します。「 *」を入力すると、すべての Web サーバーが JRun に要求を送信できるよう になります。

4 変更を適用するには、[更新] ボタンをクリックします。

5 JRun サーバーを再起動します。

既定のサーブレットの無効化

セキュリティ上の理由から、JRun に付属およびインストールされている既定のサーブレットを無効にできます。これらのサーブレットには次のものがあります。

- allaire.jrun.servlets.JRunStats (global.properties 内でのエイリアスは JRunStats)
- allaire.jrun.servlets.MetricsServlet

これらのサーブレットは、すべての JRun サーバーにインストールされ、アクティブになっています。既定値 "True" を使用して、これらのサーブレットに対して新しい初期化パラメータ “enabled” を指定できます。このパラメータを False に設定すると、サーブレット初期化時に UnavailableException が返されます。これによって、サーブレットの起動が阻止されます。ブラウザでは、このサーブレットにユーザがアクセスしようとすると、内部サーバー エラー (500) が表示されます。

"enabled" パラメータを False に設定するには次の方法があります。

- global.properties または local.properties ファイル内にグローバル サーブレット エイリアスを作成し、"enabled" 初期化パラメータを False に設定します。たとえば、次の行をプロパティ ファイルに追加します。

  ```
  allaire.jrun.servlets.MetricsServlet.class=allaire.jrun.servlets.
       MetricsServlet
  allaire.jrun.servlets.MetricsServlet.enabled=false
  ```

  プロパティ ファイル内でサーブレットの初期化パラメータのエイリアスを作成し、設定するための構文は次のとおりです。

  ```
  [alias].class=[class]
  [alias].[initParam1]=[value1]
  ```

  次に例を示します。

  ```
  file.class=allaire.jrun.file.FileServlet
  file.enabled=false
  ```

  すでにエイリアスが作成されている JRunStats の場合は、次の行のみを追加してください。

  ```
  JRunStats.enabled=false
  ```

  PropertyScript ツールを使用すると、JRun プロパティ ファイルをプログラムで変更できます。詳細は、30 ページの「JRun プロパティのカスタマイズ」を参照してください。

- web.xml ファイル内で初期化パラメータを設定します。たとえば、次の行を web.xml ファイルに追加します。

  ```
  <servlet>
   <servlet-name>allaire.jrun.servlets.MetricsServlet</servlet-name>
   <servlet-class>allaire.jrun.servlets.MetricsServlet</servlet-class>
   <init-param>
    <param-name>enabled</param-name>
    <param-value>false</param-value>
   </init-param>
  </servlet>
  ```

invoker サーブレットの無効化

JRun は、文字列 /サーブレット が含まれている URI を JRun invoker サーブレット に関連付ける、暗黙的なサーブレット マッピング機能を備えています。invoker サーブレットを使用すると、サーブレットのクラス ファイルを、Web アプリケーションのクラスパス内の任意のディレクトリにコピーし、サーブレットを登録せずに参照できます。未登録のサーブレットを参照するには、フォーム内で URL を使用します。

```
http://local_host/app1/servlet/サーブレットのクラス名
```

セキュリティおよびパフォーマンス上の理由から、必ず、すべてのサーブレットについて明示的なマッピングを定義してください。invoker サーブレットに全面的に依存しないでください。運用アプリ ケーション では、次のマッピングを global.properties ファイルからコメント化してください。

```
webapp.servlet-mapping./servlet=invoker
```

JRun でのログ記録

ISP は、JRun を利用する顧客にアクティビティ ログを提供しなければならない場合があります。JRun のログは非常に便利で、特にデバッグや、パフォーマンスおよびトラフィックの測定に役立ちます。ログ ファイルには、次の 3 種類があります。

- Event ログ
- System.out ログ
- System.err ログ

既定では、JRun は、これらの各ログ ファイルを JRun サーバーごとに作成します。ファイルは次の命名構文に従って、*JRun のルート ディレクトリ* /logs に保存されています。

```
{jrun.server.name}-<log_file_type>.log
```

たとえば、JRun の既定のインストールでは、*JRun のルート ディレクトリ* /logs ディレクトリ内に、次のようなログ ファイルが作成されます。

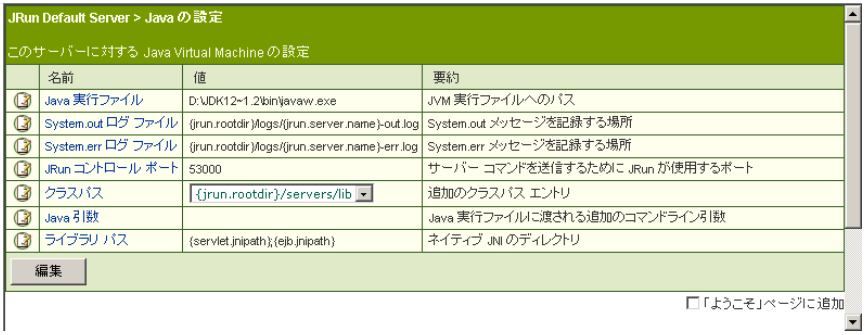
```
admin-err.log
admin-out.log
admin-event.log
default-err.log
default-out.log
default-event.log
```

特に、ログに対する読み取りおよび書き込み権限をすべての顧客に与える場合は、JRun サーバーごとに個別の /log ディレクトリを作成することをお勧めします。

JRun サーバーのログ ファイルを新規ディレクトリに転送するには

1 JMC の左側のペインで、[*マシン名*] > [*JRun サーバー名*] > [Java の設定] を選択します。

[Java の設定] パネルが表示されます。



2 右側のペインで [編集] をクリックします。

Java 設定値の編集ウィンドウが表示されます。

3 Event ログ、System.out ログ、および System.err ログ フィールドの設定値を変更します。

たとえば、Event ログのログ ファイルを転送するには、[イベント ログ] フィールドを {jrun.rootdir}/logs/{jrun.server.name}-event.log から /usr/logs/{jrun.server.name}/{jrun.server.name}-event.log に変更します。これにより、既定サーバー上の JRun は、c:¥logs¥default¥default-event.log ファイルに書き込みを行えるようになります。

4 変更を適用するには、[更新] ボタンをクリックします。

5 JRun サーバーを再起動します。

作業が終了したら、/usr/logs/default ディレクトリのプロパティを編集して、その顧客だけがそのファイルの読み取りおよび書き込みをできるように設定できます。

ログ ファイルおよびログ プロパティの詳細は、『JRun によるアプリケーションの開発』を参照してください。

既定のアプリケーションの設定

local.properties ファイル内の *application_name*.use-webserver-root プロパティは、JRun サーバーの既定の Web アプリケーションを指定します (アプリケーションのコンテキスト パスは / です)。

要求が既存の URL マッピングにマップされない場合、JRun は、その要求を既定アプリケーションに転送します。

use-webserver-root の既定の設定では、default-app は true に設定され、ほかのアプリケーションは false に設定されるか、またはプロパティ ファイルから除外されます。たとえば、default JRun サーバーの local.properties ファイル内には、次のような行が表示されます。

```
default-app.use-webserver-root=true
```

既定の設定を上書きするには、local.properties ファイルを手作業で編集することにより、このプロパティを false に設定します。

JSP では、これを true に設定すると、JRun はアプリケーションのディレクトリではなく、Web サーバーのルート ディレクトリに関連した JSP を検索します。

JRun が要求をサーブレットおよび JSP にどのようにマッピングするかについては、『JRun によるアプリケーションの開発』の第 6 章を参照してください。

リソース使用率の制限

共有ホスティング環境では、各 JRun サーバーとその JVM のリソースの使用率を監視することが重要です。JRun を使用したメモリ リソースの使用状況の監視には次の 2 つの方法があります。

- 各 JRun サーバーと JVM の実際のメモリ使用率を調べます。
- 各 JVM の使用可能なメモリを削減または制限します。

次のセクションでは、これらのオプションについて説明します。

メモリ使用率の判別

JRun にはメトリック ログ機能が用意されています。この機能を使用すると、指定された間隔で JVM ヒープ内の総メモリ容量と空き容量を詳しく記録できます。たとえば、メトリック ログ機能をオンにし、次の行を JRun サーバーの local.properties ファイルに追加すると、このサーバーの JVM の時間の経過に応じたメモリ使用率をログ ファイルに表示できます。

```
monitor.jcp-format=(jcp),{totalMemory},{{totalMemory}-{freeMemory}}
```

これにより、合計メモリ量と使用メモリ量 (合計量から空き容量を引いたもの) を示すカンマ区切りリストが出力されます。

また、スレッドの使用状況を監視して、どの仮想ホストがプロセッサを最も長い時間使用しているかを調べることができます。JRun のログには、次のようなスレッド メトリックを記録できます。

- 現在のスレッド数
- 遅延スレッド数
- アイドル スレッド数
- 合計スレッド数

詳細は、96 ページの「JRun メトリックの使用」を参照してください。

JVM メモリ使用率の制限

JMC の [Java の設定] パネルを使用して、各 JRun サーバーの JVM の初期ヒープ サイズと最大ヒープ サイズを設定できます。

1 JMC の左側のペインで、[*マシン名*] > [*JRun サーバー名*] > [Java の設定] を選択します。

[Java の設定] パネルが表示されます。

2 右側のペインで [編集] をクリックします。

Java 設定値の編集ウィンドウが表示されます。

3 [Java 引数] フィールドで、JRun が JVM を起動するときに JVM 実行可能モジュールに渡す引数を入力します。

次の表は使用可能な引数の一覧です。

引数	説明	最低	既定
-mx<*val*>[k|m]	JVM の最大ヒープ サイズ	1 K	16 MB
-ms<*val*>[k|m]	JVM の初期ヒープ サイズ	1 K	1 MB
-oss<*val*>[k|m]	各 Java スレッド内における Java コードのスタック サイズ	1 K	400 K
-ss<*val*>[k|m]	各 Java スレッド 内における C コード のスタック サイズ	1 K	128 K

上記の各引数では、キロバイトを示すには k を、メガバイトを示すには m を使用してください。値をキロバイト (k) で指定する場合、値は 1024 の倍数である必要があります。これらの引数は、どの JRE または JDK を使用しているかによって異なります。詳細は、JVM のマニュアルを参照してください。

4 変更を適用するには、[更新] ボタンをクリックします。

5 JRun サーバーを再起動します。

JVM セキュリティ

ユーザが JMC を介して JRun サーバーにアクセスできる場合は、[Java の設定] パネルの [Java 引数] フィールドの値を変更できます。これは、リソースやセキュリティ上の問題となる可能性があります。[Java 引数] フィールドの使用を制限する、記述されたポリシーを作成および実行するか、またはネットワーク OS を使用してファイル権限に対して制限を加えることを検討してください。共有ホスティング環境におけるセキュリティの実施については、71 ページの「JRun 環境の保護」を参照してください。

平行処理の設定

JRun と外部 Web サーバー間のコネクタの構成には、並行処理設定の最適化が含まれます。並行処理は、HTTP 要求をプールし、分散する方法を定義します。これらの設定値を変更することによって、各 JRun サーバーによって処理されるスレッドと要求の数を制限できます。実際に、その JRun サーバーのトラフィックを調節できます。

メモ
「同時要求数」と「同時ユーザ数」は別の概念であることを忘れないでください。たとえば、各仮想 Web サイトが 1000 個の同時要求をサポートするとしても、1000 人の同時ユーザによって同時に 50 個の要求しか生成されていない場合があります。JRun によって、要求ごとに 1 つのスレッドが割り当てられます。

JRun には、Web サーバーへの接続に対する並行処理設定を構成する設定が 4 つあります。この設定は次のとおりです。

- アイドル スレッドのタイムアウト
- 最小スレッド カウント
- 最大アクティブ要求数
- 最大同時要求数

顧客の Web サイトでトラフィックの増加が急激に発生する環境では、Web サイトのトラフィックが急激に増加した場合にスレッドのグループを作成しなくても済むように、最小スレッド カウントを高く設定します。また、最小スレッド カウントを、予期される安定した状態の負荷の同時要求数に設定することもできます。たとえば、Web サイトで常時 80 個の同時要求が発生する場合は、最小スレッド カウントは 80 に設定します。

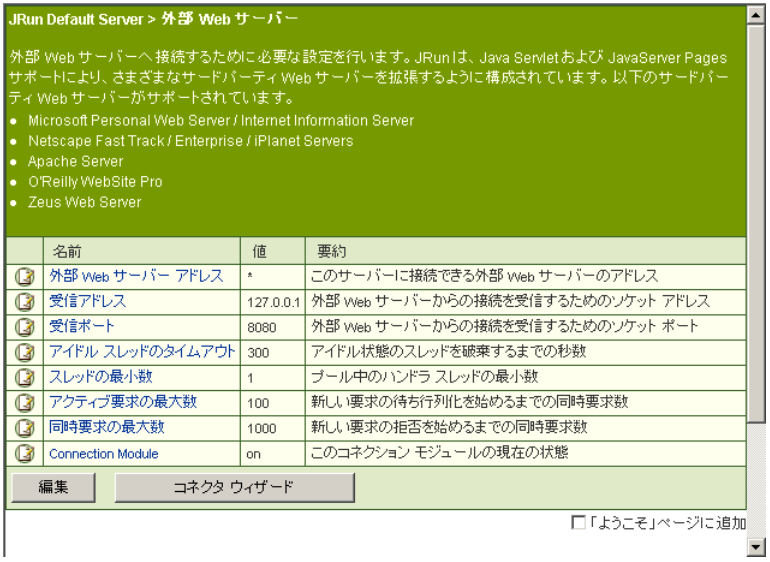
たとえば、3 段階の RMI-CORBA データベース トランザクションを実行しているために、顧客の Web サイトの平均応答時間が長い場合、新しい要求を拒否せずにスループットを保持するには、より多くの要求を待ち状態にしなければならないことがあります。この場合、最大同時要求数を予期される要求数よりも大きい値に設定する必要があります。最大同時要求数の設定値は、リソースの安全弁としての役割を果たします。

メモ
既定の JRun 並行処理設定を変更する前に、トラフィック パターンが実際に監視できることを確認します。設定値を変更してしまうと、リソースを消耗する可能性があります。

外部 Web サーバーの並行処理設定を変更するには

1 JMC の左側のペインで、[*マシン名*] > [*JRun サーバー名*] > [外部 Web サーバー] を選択します。

[外部 Web サーバー] パネルが表示されます。



2 右側のペインで [編集] をクリックします。

外部 Web サーバーの編集ウィンドウが表示されます。

3 次の表の説明に従ってプロパティを編集します。

プロパティ	説明
外部Web サーバー アドレス	IP アドレスを入力します (複数の場合はカンマで区切ります)。この JRun 接続モジュール (JCM) は、それらのアドレスからの要求のみを JRun サーバーに渡します。 既定値 * を使用すると、この JCM がすべての IP アドレスからの要求を受け入れます。
受信アドレス	外部 Web サーバーからの接続を受信するソケットの IP アドレスを入力します。サーバーに複数の IP アドレスがある (マルチホーミング) 場合は、この機能を使用してください。 既定値 * を使用すると、JRun はすべてのサーバー IP アドレスにバインドされます。
受信ポート	この JCM が外部 Web サーバーからの接続の受信に使用する固有のポート番号を入力します。 このポートを外部 Web サーバーの HTTP ポートと混同しないでください。

プロパティ	説明
アイドル スレッドのタイムアウト	JRun によって破棄されるまでのスレッド のアイド ル状態の秒数を入力します。このパラメータは、Web サーバーがビジー状態から休止状態に戻る速さを決定します。スレッドが破棄されるたびに、少量のシステム リソースが解放されます。 JRun は Java スレッド メカニズムを使用して同時要求を処理します。要求ごとに新しいスレッドを作成する代わりに、JRun は新しい要求のために準備されたハンドラ スレッド のプールを保持します。スレッド のこのプールは、Web サーバーでの要求数の変化に応じて拡大または縮小します。Web サーバーのトラフィック負荷と能力のバランスが取れるようにプールのパラメータを設定します。 既定値は 300 秒です。
最小スレッド カウント	初期プールに (開始時に) 作成されるハンドラ スレッドの数を入力します。システム実行中、スレッドは作成されて破棄されますが、プール サイズがこの数より小さくなることはありません。 既定値は 1 です。
最大アクティブ要求数	JRun が処理する同時要求の数の上限を入力します。この上限値を超える数の要求は、ハンドラ スレッド がそれらを処理できるようになるまで遅れます。このパラメータは、外部 Web サーバーでの並行処理を 制限するための JRun の基本的なメカニズムです。 既定値は 100 です。
最大同時要求数	Web サーバーが処理できる要求の最大数を入力します。サーバーは、この絶対最大数を超える要求をすべて破棄します。 既定値は 1000 です。
接続モジュール	この JCM をオンにするには、このチェックボックスをオンにします。この JCM をオフにするには、このチェックボックスをオフにします。 接続モジュールをオフにすると、JRun と外部 Web サーバー間の接続が切断されます。そのため、ユーザが JSP ページまたはサーブレットにアクセスしようとするとエラーが発生することがあります。

4 変更を適用するには、[更新] ボタンをクリックします。

5 JRun サーバーを再起動します。

JWS または外部 Web サーバーの Web サーバー設定値の編集については、『JRun セットアップ ガイド』を参照してください。

接続プール

1 台のコンピュータ上で、または専用の並列配置セットアップの場合でも複数の仮想ホストを実行する場合は、要求数の急激な増加に備える必要があります。数十個、数百個、または数千個の要求が同時に 1 つのサイトで発生すると、設計が不適切なアプリケーションではすぐに対応できなくなります。

データベース接続を作成すると、特に厄介なボトルネックになる可能性があり、最も負荷の大きいデータベース操作の 1 つは初期接続の確立です。使用しているデータベースによっては、プロトコル ハンドシェークの実行、ユーザ情報の確認、ディスク ファイルのオープン、またはメモリ キャッシュの作成が接続に必要となる場合があります。標準的なサーブレットや JSP では、エンド ユーザがこの接続のためのパフォーマンス コストを負担することになります。

プールを使用しない場合は、サーバーによって、新規ユーザごとに新しい接続が生成されます。

帯域幅を使用する上でピーク負荷を処理する方法の 1 つは、ある種のデータベース接続プールを実装すること です。JRun はこのために簡単なメカニズムを提供しています。

接続プールとは

接続にかかる時間を短縮することはできませんが、ユーザの使用に備えて事前に接続のコレクション **(プール)** を設定しておくことが可能です。別個のスレッドでこれらの接続を設定することによって、サーブレットのパフォーマンスの低下を最小限に抑えることができます。サーブレットはプールから接続を取得し、それを使用し、完了後に返します。

これまで、多くの開発者は、独自の接続プールを開発するか、または、市販の接続プールを利用してきました。JRun 製品には強力な接続プールがバンドルされているため、接続プールの開発という作業が不要になりました。接続プールは JMC で簡単に管理できます。

接続プールを使用する理由

接続プールによってパフォーマンスが大幅に向上します。その他にも 多数の利点があります。これらの利点については、次の表で説明します。

利点	説明
拡張性	サーブレット の全体的なパフォーマンスが向上することにより、アプリケーション全体の拡張性も大幅に向上します。
コード の再利用および削減	接続プールを使用すると、記述して維持するコード の量を減らすことができ、また、すでにテストおよび動作確認が行われたコードを再利用できます。
集中管理	サーブレット を変更したり 再コンパイルせずに、接続パラメータ (ユーザ名、パスワード 、カスタム引数など) を 1 か所で変更できます。
セキュリティ	接続を使用するときにサーブレット に必要なのはデータ ソース名のみなので、データ ベースのユーザ名と パスワード が公表されることはありません。
パフォーマンス	パフォーマンスの向上は、サーブレットがデータベースを使用する場合の最も注目すべき最大のメリットの 1 つです。

接続プールの使用

JRun 接続プール メカニズムを使用できるのは、JMC でセットアップされた JDBC データ ソースとともに使用する場合だけです。JDBC の作成方法の詳細は、『JRun によるアプリケーションの開発』を参照してください。[JDBC データ ソース] パネルの使用方法については、『JRun セットアップ ガイド』を参照してください。

使用しているデータ ベース ドライバに独自のネイティブ接続プール メカニズムが含まれている場合は、パフォーマンスを最適化するために、JMC 内の JRun 接続プールをオフにします。

JDBC 接続プールの起動

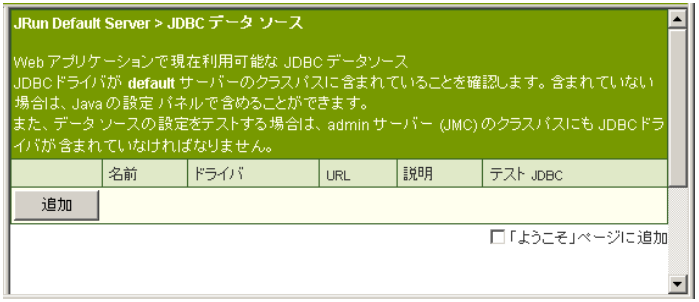
このセクションでは、接続プール メカニズムをオンにする方法について説明します。

JDBC 接続プールを起動するには

1 JRun 管理コンソールを開きます。

2 [*マシン名*] > [*JRun サーバー名*] > [JDBC データ ソース] を選択します。

[JDBC データ ソース] パネルが表示されます。



3 データ ソースの名前をクリックします。

JDBC データ ソースの編集ウィンドウが表示されます。

4 [プール] チェック ボックスをオンにして、この JRun サーバーについて、JRun の接続プール メカニズムをオンにします。

5 JRun サーバーを再起動します。

プールを使用するには、データ ソースについてプールを明示的に有効にする必要があります。既定では、プールは使用可能になっていません。実際の JDBC データベース接続のエイリアスを作成すると、プール機能のない JRun データ ソースでもサーブレットで使用できます。

接続プール プロセス

次に、接続プール プロセスの手順について説明します。

1 JNDI InitialContext が作成されます。JNDI InitialContext は、名前を指定したルックアップの実行に使用するコンテキスト です。JNDI によって、名前でオブジェクトをコンテキストにバインドできます。このようにして、JRun データ ソースにアクセスします。

2 JDBC データ ソース オブジェクトが JNDI コンテキストから取り出されます。このオブジェクトは、JDBC 2.0 データ ソースの JRun 実装で、指定したデータ ソース名のプールにあるすべての接続を保持します。検索では、次の形式によるデータ ソース名が必要になります。

```
java:comp/env/jdbc/{Data Source name}
```

3 JDBC 接続がデータ ソースから取得されます。

4 データベースは接続を介して操作されます。

5 すべての操作は、"try/catch/finally" ブロックでラップされます。最終ブロックは、例外が送出された場合でも必ず実行されます。

6 最終ブロック内で接続を閉じます。これにより、接続は解放されてプールに戻されます。JDBC ステートメントまたは結果セットを開いたら、結果セット、ステートメント、および接続の順で閉じる必要があります。

データ ソースを介してプールから接続を要求する際に利用できる接続がない場合は、JRun によって新しい接続が作成されて返されます (プールの初期サイズ、つまり プール内の接続数はゼロであることに注意してください)。利用できる接続がある場合は、直ちにその接続が返されます。

接続を一度使用したら、それを閉じる必要があります。JRun では、接続を閉じる代わりにそれをプールに返す接続プロキシ オブジェクトが用意されています。プールは、指定されたデータ ソース名に対して、同時ユーザの数だけ大きくなります。接続がタイムアウトになるとプールは縮小します。

JSP を作成する場合は、JRun カスタム タグ ライブラリの使用を検討してください。このライブラリには、JRun データ ソースを使用する jrun:sql タグが含まれています。

接続プールの例

このセクションでは、サーブレット、EJB、および JSP での接続プールの使用方法について説明します。

メモ

JRun 接続プールを使用する前に、『 JRun セット アップ ガイド 』の説明に従って JRun で JDBC データ ソースをセットアップする必要があります。

JRun は、標準 API メカニズム、特に JNDI および JDBC 2.0 を介してプールを提供します。たとえば、次のサンプル コードでは、JRun JDBC データ ソースを使用したデータベース接続が確立されます。

```
import java.sql.*;
import javax.naming.*;
import javax.sql.*;
...
InitialContext context = null;
Connection con = null;
try {
    // JNDI コンテキストを取得します。
    context = new InitialContext();

    // データ ソースのルックアップを行います。
    DataSource ds = (DataSource) context.lookup("java:comp/env/jdbc/" +
        dsn);

    // データ ソースから接続を入手します。
    con = ds.getConnection();
    ...
}
catch (Exception ex) {
    throw new ServletException(ex.getMessage(), ex);
}
finally {
    if (con != null) {
        // 必ず接続を閉じます。
```

```
            try {
                con.close();
            }
            catch (SQLException ex) {
                // 閉じる際のエラーは無視します。
            }
        }
        if (context != null) {
            // 必ずコンテキストを閉じます。
            try {
                context.close();
            }
            catch (NamingException ex) {
                // 閉じる際のエラーは無視します。
            }
        }
}
```

JRun データ ソースを利用するには JDK 1.2 以降を使用する必要があります。これは、この機能が JNDI および JDBC 2.0 に依存しているためです。ただし、jndi.jar および jdbc.jar を JRun のクラスパスに追加することによって JDK 1.1.x も使用できるようになります。これらの jar ファイルは、JRun 3.0 とともに *JRun のルート ディレクトリ*/lib/ext ディレクトリにインストールされます。

EJB での接続プール

JRun 3.0 は、現在、データ ソースを指定するために 2 つの異なる場所を提供しています。1 つはサーブレット および JSP 用、もう 1 つは EJB 用です。EJB 内から データ ソースを使用する場合は、EJB の deploy.properties ファイル内でデータ ソースを定義する必要があります。この方法の例については、『JRun サンプル ガイド』を参照してください。

JRun 接続プールを使用しない場合

使用している JDBC ドライバがすでに JDBC 2.0 接続プールをサポートしている場合、そのドライバには JRun 接続プール メカニズムを使用しないことをお勧めします。ドライバのベンダは、一般の接続プールよりも効率的で強力なデータベース専用の機能を採用しています。

また、サーブレット コンテナ間のサーブレット相互運用性について不安がある場合は、接続プールを使用しないことをお勧めします。JDBC データ ソースを取得するための JNDI メカニズムは J2EE 仕様の一部であるため、この機能を使用するサーブレットは J2EE 互換のサーブレット コンテナに移植可能になります。あいにく、一部のサーブレット コンテナではこの機能がサポートされていない場合があります。したがって、サーブレットで接続プールを使用する前に、サーブレットが公開されるサーバーが JDBC データ ソースをサポートしていることを確認する必要があります。

Active Server Pages と JRun の統合 (IIS のみ)

JRun は、顧客に提供するアプリケーション サーバー スイートの一部として組み込むことができます。ほかのテクノロジと併せてよく提供されるものの 1 つに、Microsoft Active Server Pages (ASP) があります。

残念なことに、ASP ページを default-app ではなく JRun 3.0 Web アプリケーションに配置すると、IIS は ASP ページを処理できなくなります。このセクションでは、この制限の回避策について説明します。

JRun と ASP を統合するには

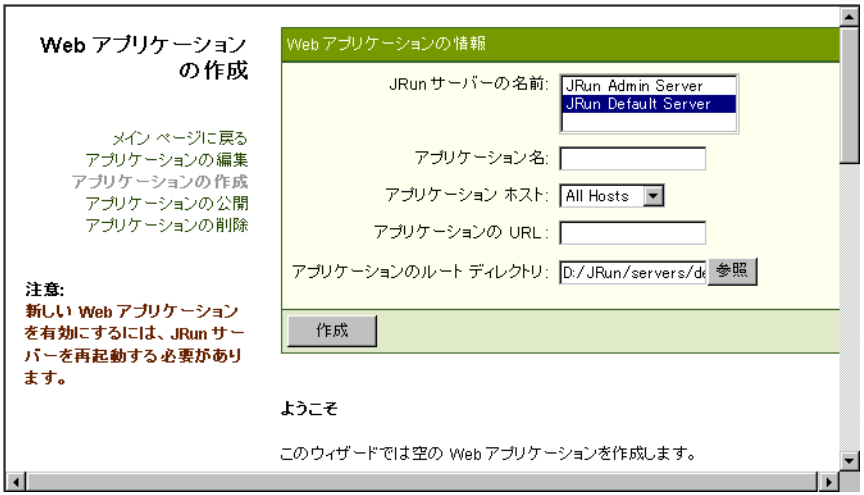
1 IIS を介してアプリケーションを作成します。物理ディレクトリは c:¥inetpub¥wwwroot¥*appname* にする必要があります。これは、Microsoft Interdev を使用して行うことも可能です。

2 JRun 管理コンソールを開き、[**マシン名] > [*JRun サーバー名*] > [Web アプリケーション]** を選択します。

[アプリケーション] パネルが表示されます。

3 [アプリケーションの作成] リンクをクリックします。

[Web アプリケーションの作成] パネルが表示されます。



4 [アプリケーションの URL] フィールドに「`/appname`」と入力します。URL は、IIS アプリケーションの名前と同じである必要があります。

5 [アプリケーションのルート ディレクトリ] フィールドで、「`c:¥inetpub¥wwwroot¥appname`」と入力します。

6 [作成] をクリックします。

JRun によってアプリケーションが追加され、
c:¥inetpub¥wwwroot¥appname¥WEB-INF ディレクトリが作成されます。

7 JRun サーバーの local.properties ファイルを開きます。

8 local.properties ファイルの最後に次の行を追加します。

```
JrunAppName.use-webserver-root=true
```

ファイルにはすでに次のプロパティが含まれています。

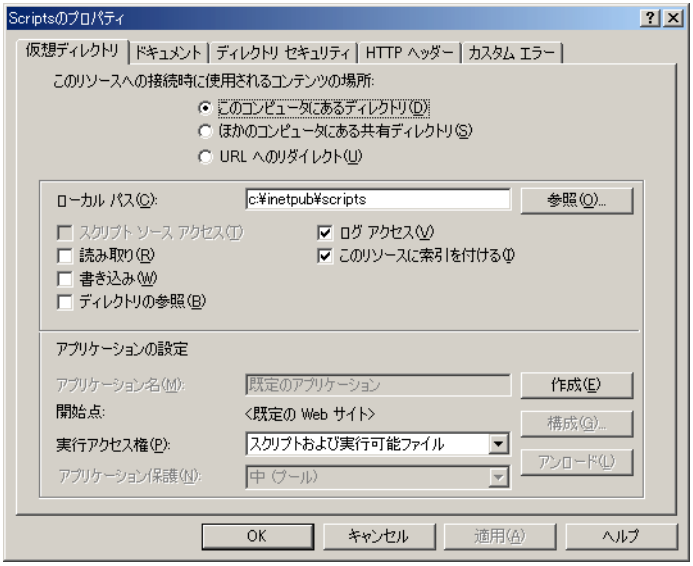
```
JrunAppName.rootdir=c:¥¥inetpub¥¥wwwroot¥¥appname
JrunAppName.class={webapp.service-class}
```

9 local.properties ファイルを保存して閉じます。

10 Microsoft 管理コンソール (MMC) を開きます。

11 手順 5 で作成した /appname/WEB-INF ディレクトリを選択し、[プロパティ] を右クリックします。

[プロパティ] ダイアログ ボックスが表示されます。



12 [読み取り] チェックボックスをオフにし、アクセス権を [なし] に設定します。

このディレクトリには JRun コンフィギュレーション ファイルが格納されていますが、表示することはできません。

13 [OK] をクリックし、MMC を閉じて変更を保存します。

14 JRun サーバーを再起動します。

`use-webserver-root=true` フラグが使用されていることに注意してください。これにより、JRun がすべてのアプリケーション要求を処理しないようになります。default-app はまた、このフラグを使用して Web サーバーのパスを把握します。これにより、JRun は、明示的に定義されたサーブレット マッピングを使用せずに、すべての Web アプリケーション要求に対する制御を Web サーバーに渡して処理します。

また、このフラグを True に設定すると、パスを静的ファイルに変換する際に、Web サーバーのマッピング ルールは、Web アプリケーションのルート ディレクトリ内のパス設定値に上書きされます。

JRun メトリックの使用

JRun には、メトリック サービスが用意されています。メトリック サービスを使用すると、JWS または JCP サービス、メモリの使用状況、スレッド、セッション、およびその他のデータに関する情報を効率よく柔軟に収集できます。メトリック サービスを使用して収集できる情報の種類を把握したら、メトリック サービスをログ サービスに組み込むことによって、独自のレポートを生成できます。

このセクションでは、メトリック サービスの概要を述べ、一部のサンプルの使用例を記載します。詳細は、『JRun によるアプリケーションの開発』を参照してください。

メトリックの設定

メトリック情報のログ機能を有効にするには

1 JMC の左側のペインで、[**マシン名] > [*JRun サーバー*] > [ログ ファイルの設定**] を選択します。

[ログ ファイルの設定] パネルが表示されます。

2 右側のペインで [編集] をクリックします。

[ログ ファイルの設定] 編集ウィンドウが表示されます。

3 [Logging Level] フィールドで、[Metrics] チェックボックスをオンにします。

これにより、この JRun サーバーの local.properties ファイル内の `logging.loglevel` プロパティにメトリックが追加されます。

```
logging.loglevel=info,warning,error,metrics
```

メトリック サービスをすべてのサーバーが使用できるように、メトリックを global.properties ファイルに手作業で追加できます。

4 JRun サーバーを再起動します。

5 JRun サーバーのログ ファイル (*JRun のルート ディレクトリ*/logs 内) を調べます。次のようなエントリがあります。

```
10/02 12:48:04 metrics (JRun) (jcp+web) Heap=4263KB Listen=2
    Idle=0 Queued=0 Busy=0 Total=2
    Requests (count/total ms)=3/594 Delayed=0
    TotalDelay=0 BytesIn=778 BytesOut=539
    Sessions (active/in memory)=0/0
```

既定では、メトリック ログ エントリは、JRun サーバーの *<server>*-event.log ファイル内に表示されます。すべての情報は同じ行に表示されます。独自のログ ファイルの作成方法については、『JRun によるアプリケーションの開発』を参照してください。

既定のメトリック ログ メッセージには、JRun Web サーバー (web サービス) とサードパーティ Web サーバーへの接続 (jcp サービス) の両方のメトリックが含まれています。ほかの JRun ログ メッセージと同様に、使用可能なメトリック変数を追加または削除することによって、メトリックの形式を制御できます。使用可能なメトリック変数についての説明は、97 ページの「ログ形式について」に記載されています。

ログ形式について

メトリック サービスは複数のメトリック変数を使用します。メトリック変数には、JVM および JRun サーバー専用のものと、JRun Web サーバー (JWS) または外部 Web サーバー接続専用のものがあります。次のセクションでは、これらの変数について説明します。

既定のメトリック形式

`monitor.format` プロパティは、メトリック サービスのログ メッセージの形式を指定します。このプロパティは、あらかじめ定義された形式に指定するか、または独自の形式を作成できます。次に例を示します。

```
monitor.format={monitor.jcp-format}
```

JRun には次のあらかじめ定義された形式が含まれています。

```
monitor.web-format
monitor.jcp-format
monitor.combined-format
```

これらの形式の構成を調べるには、global.properties ファイルを参照してください。たとえば、既定の `jcp-format` は、このプロパティ ファイルでは、次のメッセージから構成されています。

```
monitor.jcp-format=(jcp) Heap={totalMemory}KB Listen={jcp.listenTh}
        Idle={jcp.idleTh} Queued={jcp.delayTh} Busy={jcp.busyTh}
        Total={jcp.totalTh} Requests (count/total ms)={jcp.handledRq}/
        {jcp.handledMs} Delayed={jcp.delayRq} TotalDelay={jcp.delayMs}
        BytesIn={jcp.bytesIn} BytesOut={jcp.bytesOut} Sessions (active/
        in memory)={sessions}/{sessionsInMem}
```

形式フィールドはすべて同じ行にある必要があります。

カスタム メトリック形式

JRun は、ログ メッセージを生成するときにすべての変数値をリセットします。したがって、ログ内の各ログ メッセージ内には、直前の時間間隔において発生したイベントを蓄積できます。

カスタム レポート では、`monitor.interval` プロパティが非常に重要です。このプロパティを秒単位で設定することによって、分、時、日、またはどのような間隔でも統計を追跡できます。これは、課金モデルのサポートに役立ちます。このセクションで取り上げられているサンプル データは 10 秒間隔で追跡されています。

たとえば、JVM 空きメモリ 容量、処理する要求の数、JRun Web サーバーのクライアントに書き戻される総バイト数のみを追跡する場合は、`monitor.format` プロパティは次のように設定してください。

```
monitor.format=Free={freeMemory}KB Requests={web.handledRq}
        Out={web.bytesOut}
```

ログ ファイル内のメトリック ログ メッセージは次のように表示されます。

```
10/02 13:16:11 metrics (JRun) Free=4271KB Requests=9 Out=1613
```

ログ ファイルの作成時は次の点に注意してください。

- すべてのメトリック変数を中カッコ { } で囲みます。
- サーバー固有の各メトリックの前に `jcp.`(外部 Web サーバーの場合) または `web.`(JWS の場合)を付けます。
- (オプション) 演算を中カッコで囲む場合は、メトリック データに対して基本的な数学演算を行います。たとえば、`{{totalMemory} - {freeMemory}}` は、2 つのメトリック間の差を返します。

次の 2 つのセクションでは、メトリック サービスが利用できる変数について説明します。

JVM 別および JRun サーバー別メトリック

JRun には、JVM および JRun サーバー固有のメトリック変数がいくつか用意されています。これらの変数をモニタ形式に追加すると、JVM または JRun サーバーに関する特殊情報を記録できます。次の表でこれらのメトリックを説明します。

変数	説明
freeMemory	JVM ヒープ内の空きメモリのキロバイト数
totalMemory	JVM ヒープの総キロバイト数 (空きメモリおよび使用メモリ)
sessions	アクティブ セッションの現在の数
sessionsInMem	メモリ内のアクティブ セッションの現在の数。セッションの永続性は、JRun の構成により、セッション レポジトリまで維持できます。

JRun Web サーバーまたはサードパーティ Web サーバーのメトリック

また JRun には、JRun Web サーバー (JWS) および外部 Web サーバーに関する特殊情報を反映するメトリックも用意されています。これらの変数をモニタ形式に追加すると、JWS または JRun サーバーの外部 Web サーバーへの接続に関する特殊情報を記録できます。

これらの変数の前には jcp.(外部 Web ブラウザの場合) または web.(JWS の場合)を付ける必要があります。次に例を示します。

```
{web.bytesIn}
{jcp.bytesIn}
```

次の表でこれらのメトリックを説明します。

変数	説明
[jcp|web].busyTh	現在動作中のスレッドの数
[jcp|web].delayTh	実行を待っているスレッドの数
[jcp|web].idleTh	新しい要求を待っているスレッドの数
[jcp|web].listenTh	新しい接続に応答するスレッドの数
[jcp|web].totalTh	ワーカ スレッドの総数
[jcp|web].delayRq	高並列処理による、遅延した要求の数
[jcp|web].droppedRq	破棄された要求の数
[jcp|web].handledRq	処理された要求の数
[jcp|web].handledMs	要求の処理に要したミリ秒数
[jcp|web].delayMs	遅延ステートで要したミリ秒数
[jcp|web].bytesIn	すべての要求から読み出されたバイトの数
[jcp|web].bytesOut	すべての応答のために書き込まれたバイトの数

メトリック データを使用したレポートの生成

JRun のメトリック ログ メカニズムを使用すると、データを出力して、パフォーマンス解析またはトラフィック レポートなどのさまざまな種類のデータを生成できます。

このセクションでは、JRun ログ情報を使用して生成できる、簡単なカスタム レポートについて説明します。

CSV ファイルの作成

メトリック データの一般的な用途は、スプレッドシート アプリケーションに直接インポートできる CSV (Comma Separated Value) ファイルの出力です。CSV 出力を生成するには、メトリック変数および定数間にカンマが挿入されるようにモニタ形式を変更します。次の例のように、必ずカンマで開始して日付 / 時刻を区切ります。

```
monitor.jcp-format= ,{jcp.bytesIn},{jcp.bytesOut},{totalMemory},
        {freeMemory},{{totalMemory}-{freeMemory}}
```

この例では、次のような JRun サーバーのログ ファイルが作成されます。

```
11/01 13:34:51 metrics (JRun) ,0,0,3191,944,2247
11/01 13:35:01 metrics (JRun) ,0,0,3191,1239,1952
11/01 13:35:11 metrics (JRun) ,0,0,3191,1180,2011
```

モニタ形式変更後に JRun サーバーを再起動して変更を有効にする必要があります。

次のセクションでは一般的なモニタ形式について説明します。また、データを例示するサンプル グラフも記載します。

メモリの使用状況の追跡

JRun サーバーの JVM ヒープ統計は、`totalMemory` および `freeMemory` メトリックを使用することによって記録できます。仮に `monitor.format` プロパティ が `jcp-format` を参照する場合は、次の行を JRun サーバーの local.properties ファイルに追加します。

```
monitor.jcp-format=,{totalMemory},{{totalMemory}-{freeMemory}}
```

このサンプルには、日付 / 時刻の列のほかに次の 2 つの列があります。

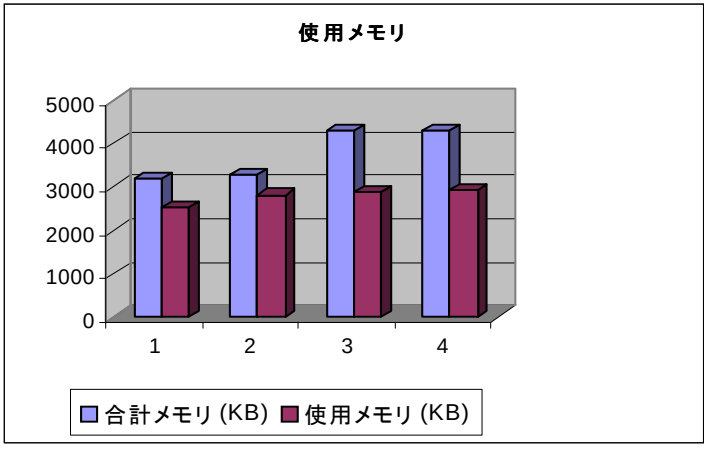
- `totalMemory`: JVM ヒープの総キロバイト 数(空きメモリ および使用メモリ)を表します。
- `freeMemory`: JVM ヒープ内の空きメモリ のキロバイト 数を表します。

ログ エントリのサンプル

```
10/27 15:29:55 metrics (JRun) ,3191,2518
10/27 15:30:05 metrics (JRun) ,3267,2811
10/27 15:30:15 metrics (JRun) ,4291,2866
10/27 15:30:25 metrics (JRun) ,4291,2912
```

サンプル グラフ

次のサンプル グラフは、使用メモリが一定の場合に、ヒープ サイズが増加していることを示します。



転送量の追跡

転送量の追跡は、非常に重要なレポート要素です。トラフィック量の周囲に収益モデルを設計したり、JRun サーバーの使用量のレポートをベースにトラフィックを制限できます。仮に `monitor.format` プロパティが `jcp-format` を参照する場合は、次の行を JRun サーバーの local.properties ファイルに追加します。

```
monitor.jcp-format=,{jcp.bytesIn},{jcp.bytesOut}
```

このサンプルには、日付 / 時刻の列のほかに次の 2 つの列があります。

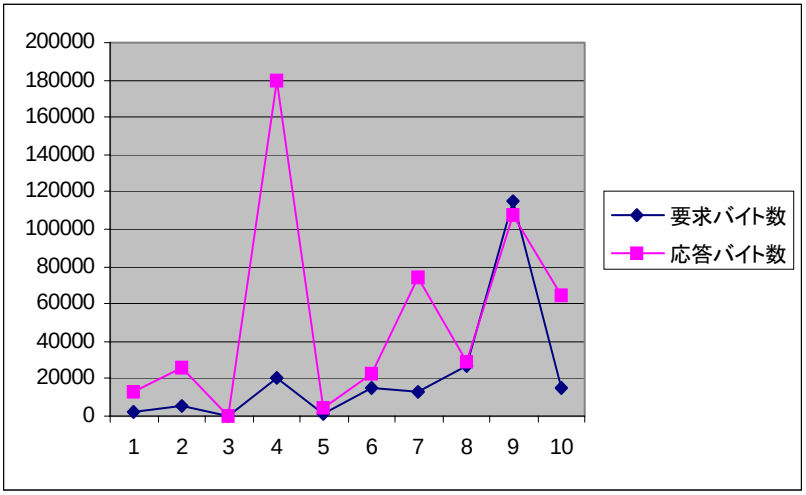
- `bytesIn`: すべての要求から読み出されたバイトの数
- `bytesOut`: すべての応答のために書き込まれたバイトの数

ログ エントリのサンプル

```
10/27 16:41:46 metrics (JRun) ,1925,12596
10/27 16:41:56 metrics (JRun) ,5320,25503
10/27 16:42:06 metrics (JRun) ,0,0
10/27 16:42:16 metrics (JRun) ,20841,179626
10/27 16:42:26 metrics (JRun) ,1491,4017
10/27 16:42:36 metrics (JRun) ,14580,23002
10/27 16:42:46 metrics (JRun) ,13113,74097
10/27 16:42:56 metrics (JRun) ,26674,29309
10/27 16:43:06 metrics (JRun) ,115106,107670
10/27 16:43:16 metrics (JRun) ,14529,64132
```

サンプル グラフ

次のサンプル グラフは、一般的にトラフィックが増大傾向にある場合の負荷量の急激な増加をバイト単位で示します。



要求の追跡

もう 1 つのトラフィック解析方法は要求を利用したものです。この場合、JCP 要求を追跡する際に、Web サーバーと JRun サーバー間の通信をチェックします。仮に `monitor.format` プロパティが `jcp-format` を参照する場合は、次の行を JRun サーバーの local.properties ファイルに追加します。

```
monitor.jcp-format=,{jcp.handledRq},{jcp.handledMs},{jcp.bytesOut}
```

このサンプルには、日付 / 時刻の列のほかに次の 3 つの列があります。

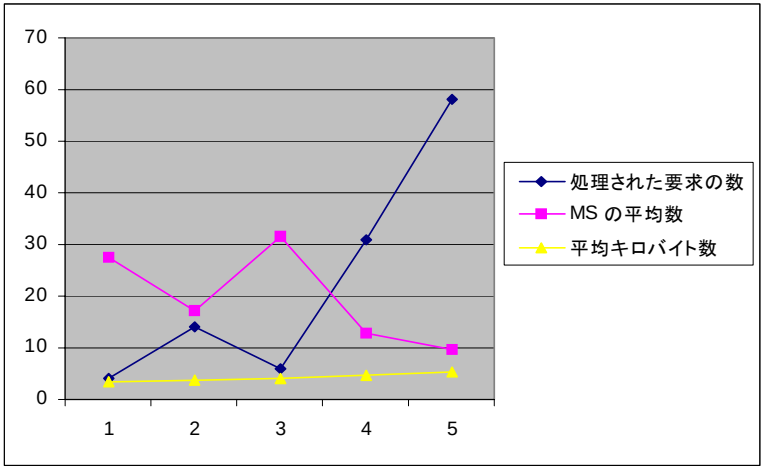
- `handledRq`: 処理された要求の数
- `handledMs`: 要求の処理に要したミリ秒数
- `bytesOut`: すべての応答のために書き込まれたバイトの数

ログ エントリのサンプル

```
10/27 16:38:08 metrics (JRun) ,4,110,13440
10/27 16:38:18 metrics (JRun) ,14,241,52907
10/27 16:38:28 metrics (JRun) ,6,190,23828
10/27 16:38:38 metrics (JRun) ,31,401,143053
10/27 16:38:48 metrics (JRun) ,58,560,302296
```

サンプル グラフ

次のサンプル グラフは、この要求について、各要求の平均キロバイト数が一定であることを示します。要求を処理する MS の平均数は実際には、要求数が増えるにつれて小さくなります。



NOC との統合

このセクションで説明してきたレポートのほかに、次の JRun メトリック変数にも留意してください。これらの変数を使用すると、カスタム レポートを生成したり、Netcool や BMC Software の PATROL などの NOC に対して、カスタム データを提供できます。

- `delayTh`
- `delayRq`
- `droppedRq`
- `delayMS`
- `busyTh`

たとえば、`droppedRq` (破棄された要求) をログに含め、`monitor.interval` を 60 に設定します。PATROL ナレッジ モジュール エージェントにカスタム ログ ファイルを解析させ、`droppedRq` をチェックさせます。エージェントのルールの一環として、1 分間に破棄された要求の数が 5 を超えたかどうかをチェックします (エラーに備えて少し余裕を残しておきます)。JRun サーバーが要求の破棄を開始すると、NOC によってユーザに通知されます。

第 3 章

EJB エンジンの設定

EJB エンジンには多くの機能があり、高度な技術を持つ開発者は、これらの機能を使用して基本機能をカスタマイズおよび拡張できます。これらの機能を利用すると、エンドツーエンド ソリューションを提供しながら、同時に EJB プログラミングを完全に活用することができるようになります。

目次

フェイルセーフ モード

フェイルセーフ モードでの実行時は、JRun がリモート起動による自動復旧機能を提供します。サーバーが停止している場合にクライアントが接続を試みると、RMID が自動的にそのサーバーを再起動します。フェイルセーフ モードでサーバーを起動する前に、RMID を起動しておく必要があります。RMID を起動せずにサーバーを起動しようとすると、エラー メッセージが表示され、サーバーは起動されません。

RMID

RMID は、Java 2 プラット フォームのリ モート 起動エージェント であり 、EJB エンジンは RMID を使用してフェイルセーフ操作を提供します。特定のサーバー上でフェイルセーフ モード で実行している JRun のすべてのインスタンスが、RMID を使用して管理されます。JRun のあるインスタンスが停止した場合、次のクライアント がそのインスタンスに接続しようとしていることを RMID が検知すると、自動的にそのインスタンスを再起動します。

RMID は、実行ディレクトリのすぐ下に log ディレクトリを作成します。このディレクトリには、RMID が JRun のインスタンスを再起動する際に使用する情報が含まれています。開発および テスト 時は RMID を毎回クリ ーンな状態で起動することが望ましいため、RMID を再起動する前に log ディレクトリを削除してください。詳細は、JDK のマニュアルまたは http://java.sun.com/products/jdk/rmi を参照してください。

JRun をフェイルセーフ モード で起動する前に、RMID を起動する必要があります。RMID を起動するには、コマンド ウィンドウを開いて作業ディレクトリ に移動し 、次の手順を実行します。

Solaris 上の RMID

Solaris 上の RMID を起動するには

- 次のコマンドを入力します。

```
% cd /tmp
% rmid
```

既定では、[rmid log] ウィンドウに出力メッセージが表示されます。

Solaris 上の RMID を停止するには

- 次のコマンドを入力します。

```
% rmid -stop
```

Windows 上の RMID

Windows 上の RMID を起動するには

1 コマンド プロンプト ウィンドウを開きます。

2 次のコマンドを入力します。

```
C:¥> cd ¥temp
C:¥temp> start rmid
```

rmid.exe というタイトルの新しいウィンドウが表示されます。既定では、この新しいウィンドウに出力メッセージが表示されます。このウィンドウは、RMID が停止されるまで開いたままになります。

Windows 上の RMID を停止するには

- 次のコマンドを入力します。

```
C:¥temp> rmid -stop
```

サーバー ツール

サーバー ツールを使用して、EJB エンジンをフェイルセーフ モードで起動および停止します。-start オプションを使用すると、/deploy ディレクトリのファイルを使ってサーバーが起動されます。このとき、Deploy ツールが以前に処理した Bean Jars が、生成された runtime.properties ファイルとともに /deploy ディレクトリから /runtime ディレクトリにコピーされます。また、ejipt_objects.jar および ejipt_exports.jar ファイルもコピーされます。そして、EJB エンジンを起動します。

EJB エンジンをフェイルセーフ モードで起動するには

- 次のコマンドを入力します。

```
% cd /jrun
% java -Djava.security.policy=lib/jrun.policy -classpath
         lib/ejipt_tools.jar allaire.ejipt.tools.Server -start
```

-restart オプションを使用すると、jar ファイル、および /runtime ディレクトリにコピー済みの runtime.properties ファイルを使ってサーバーが起動されます。

事前に準備した runtime ファイルを使用して EJB エンジンを再起動するには

- 次のコマンドを入力します。

```
% cd /jrun
% java -Djava.security.policy=jrun.policy -classpath
         lib/ejipt_tools.jar allaire.ejipt.tools.Server -restart
```

-stop オプションを使用すると、サーバーが停止します。

EJB エンジンを停止するには

- 次のコマンドを入力します。

```
% cd /jrun
% java -Djava.security.policy=lib/jrun.policy -classpath
         lib/ejipt_tools.jar allaire.ejipt.tools.Server -stop
```

EJB エンジンのフェイルセーフ モードでの起動および停止に関する追加情報については、JRun 文書のページで提供されている allaire.ejipt.tools.Server API JavaDocs を参照してください。

EJB エンジンの埋め込み

EJB エンジンは、アプリケーション内でクラスとしてインスタンス化することにより、インプロセスで実行できます。次のサンプル コードはその例です。

```
// これは JRunInstallDirectory/lib/ejipt.jar にあります。
import allaire.ejipt.*;

Ejipt.prepareEnvironment(true);
Ejipt.prepareProperties(null);
final Ejipt ejipt = new Ejipt(true);
ejipt.start();
ejipt.export(0);
```

`Ejipt.prepareEnvironment(true)` ステート メント により 、スタブやプロパティなどの必須ファイルが、/deploy サブディレクトリ から /runtime サブディレクトリ にコピー されます。`Ejipt.prepareProperties(null)` ステート メント により 、さまざまなプロパティ ファイルからプロパティがロード されます。

`final Ejipt ejipt = new Ejipt(true)` ステート メント により 、EJB エンジンのインスタンスが作成されます。`ejipt.start()` ステートメントでリモート オブジェクト およびホーム オブジェクト がロード され、`ejipt.export(0)` ステート メント で EJB エンジンのポートが設定されます。`0` 引数により 、未指定の空きポートを使用するようにサーバーに指示が与えられます。

自動復旧は、このモード では使用できません。

完全なサンプルについては、『JRun サンプル ガイド 』のサンプル 1a を参照してください。

EJB エンジンのサブクラス化

これは、EJB エンジンをスタンド アロン モード で実行し ているとき、および EJB エンジンを JRun サービス として実行しているときに機能します。

カスタム サーバーのコーディング

allaire.ejipt.Ejipt クラスをサブクラス化し、選択したメソッドを上書きするカスタム サーバーのコードを作成できます。この場合も、EJB エンジンはスタンドアロン モード で実行されています。

allaire.ejipt.Ejipt 内のメソッドに関する追加情報については、JRun 文書のページで提供されている allaire.ejipt.Ejipt JavaDocs を参照してください。

スタンドアロン EJB エンジンとしてのカスタム サーバーの使用

このサンプルでは、サブクラス化した EJB エンジンの名前を MyServer とします。また、MyServer.class は、現在の作業ディレクトリに含まれているものとします。/jrun を、適切なホーム ディレクトリのパスに置き換えます。

```
java -Djava.security.policy=policy file
  -Dejipt.home=/jrun -classpath ".;/jrun/lib/ejipt.jar" MyServer
```

メモ

このサンプルを使用するには、jms.jar、ejb.jar、jta.jar、および jndi.jar ファイルがシステムのクラスパスに対して定義されている必要があります。または、これらのファイルを `-classpath` 引数で指定します。

JRun サービスとしてのカスタム サーバーの使用

allaire.ejipt.Ejipt を拡張して、それを allaire.ejipt.Ejipt の代わりに使用して JRun サーバーで実行する場合は、global.properties ファイルを開き、ejb.ejipt.classname プロパティを、allaire.ejipt.Ejipt を拡張するクラスの名前に変更します。

また、クラスが含まれている JAR ファイルを追加して ejb.classpath プロパティを更新します。

サードパーティ JDBC ドライバの使用

サードパーティ JDBC ドライバを使用する場合は、これをサーバー マシンにインストールする必要があります。サーバーを起動する前に、ドライバの JAR ファイルをクラスパスに含める必要があります。EJB サンプルを make ファイルを使用して実行する場合は、次の行を入力します。その際、ドライバの正しいパスを指定してください。

```
bash$ export JDBC_DRIVERS=/path/ドライバ名
```

また、deploy.properties ファイル内の ejipt.sourceDriverClassName も必ず設定してください。

デバッグ モードでの実行

EJB エンジンは、Java Debugger を使用して実行できます。たとえば、JRun サンプルを Java Debugger で実行できます。EJB エンジンをこのように実行することにより、Bean の手順を追って示し、デバッグ処理を非常に単純かつ明瞭なものにできます。

Java Debugger でサーバーを実行するには

1 次のコマンドを入力します ("java" ではなく "jdb" であることに注意してください)。

```
jdb -Djava.security.policy=/jrun/lib/jrun.policy -Dejipt.home=/jrun
       -classpath ".;/jrun/lib/ejipt.jar" Server
```

メモ

このサンプルを使用するには、jms.jar、ejb.jar、jta.jar、および jndi.jar ファイルをシステムのクラスパスに含める必要があります。または、これらのファイルを -classpath 引数で指定します。Windows NT では、-classpath 引数に JAR ファイルを指定する場合、スペースが足りなければファイルの短縮名を使用します (たとえば、Program Files の代わりに Progra~1 を使用します)。

2 プロンプトに、次のコマンドを入力します。

```
% stop at ejbeans.BalanceBean:39
% run
```

3 別のコマンド プロンプト ウィンドウを開き、クライアントを起動します (Windows では必ず makew を使用してください)。

```
C:¥> bash
bash$ export JRUN_HOME=/jrun
bash$ cd /jrun/samples/sample2a
bash$ make run
```

4 ログインしてトランザクションを実行します。

サーバーは、BalanceBean の 40 行目で停止します。

5 プロンプトに「cont」と入力して、処理を続行します。

追加情報およびコマンドについては、「help」と入力するか、または、Java Debugger のマニュアルを参照してください。

/deploy および /runtime ディレクトリの移動

/deploy および /runtime ディレクトリは、公開する Bean だけでなく、JRun が生成したホームおよびオブジェクトの実装が含まれているペアです。そのペアの共通の親ディレクトリは、ejipt.ejbDirectory と呼ばれます。この親ディレクトリは、JRun サーバーごとに *JRun のルート ディレクトリ*/servers/*サーバー名*というディレクトリに設定されます。ただし、次のように ejipt.ejbDirectory プロパティを変更すると、/deploy および /runtime ディレクトリを別の場所から使用できます。

- **EJB エンジンを JRun サービスとして実行する場合** JRun サーバーの local.properties ファイルを開き、-Dejipt.ejbDirectory を、{jrun.server.rootdir} から、/deploy および /runtime ディレクトリが含まれているディレクトリの完全指定パスに変更することによって、ejb.javaargs プロパティを変更します。
- **スタンドアロン EJB エンジンを実行する場合** Java コマンドライン パラメータとして -Dejipt.ejbDirectory="full directory path" を追加します。同じホスト上で複数の EJB エンジンを実行する場合は、-Dejipt.classServer.port=*classserverport* および -Dejipt.homePort=*homeport* という Java コマンドライン パラメータを使用して、エンジンごとに個別のポート番号を指定する必要があります。

ステージングと運用環境

EJB のテストが可能な状態になったら、ステージング領域を設定して実行時環境をシミュレートできます。このステージング領域を使用してシステム テストを行い、runtime 実行可能モジュールを作成します。

最終的な運用環境では、/deploy ディレクトリはオプションです。テストを終了したら、/runtime ディレクトリを圧縮し、運用環境で /runtime ディレクトリを複製できます。ステージング領域では、javac またはその他の Java コンパイラを使用できるようにしておく必要があります。

作成と作成後

必要に応じ、ejbPostCreate() メソッド は、実際の作成を実行し ます。そのため、インスタンス フィ ールド の必須検証を行う には通常、ejbCreate() メソッド が使用されます。ejbCreate() および ejbPostCreate() メソッド の引数リストは同じである必要があります。

次の ejbPostCreate() メソッド の実装ではまず、データ ベース への接続を取得し、そのプライマリ キーによって表されるオブジェクト がデータベース内に存在するかどう かが調べら れます。存在する場合は、Bean インスタンスのステート が設定されます。Select が結果を返さない場合は、オブジェクト が作成されてデータベースに挿入されます。

```
public void ejbPostCreate(final int accountId)
   throws CreateException, RemoteException{
 try{
  final Statement statement =
        ResourceManager.getConnection(“source1”).createStatement();
  final ResultSet results = statement.executeQuery("SELECT value FROM
        account WHERE id = " + _context.getPrimaryKey());
  if (!results.next()){
    statement.executeUpdate("INSERT INTO account (id, value) VALUES
        ("+_context.getPrimaryKey() +"," + _value +")");
  }else{
    _value = results.getInt(1);
    results.close();
  }
 }
 catch (final Exception exception){
  exception.printStackTrace();
  throw new RemoteException("create failed");
 }
}
```

コンテナは、createSQL ステート メントを実行することによって作成引数の妥当性をチェックします。createSQL は、プライマリ キー フィールドが含まれている結果セットを返し ます。結果が返されない場合は、CreateException が返され、作成引数が無効であることを示します。

結果が返された場合は、コンテナにより、返されたフィールドを基にしてプライマリ キーが生成されます。さらにコンテナにより、そのプライマリ キーを持つオブジェクトがサーバー内に存在するかどうかが調べられます。

サーバー内にプライマリ キーが存在し、ejipt.isCreateSilent が true に設定されなかった場合は、javax.ejb.DuplicateKeyException が返されます。

コンテキスト ファクトリ

ホーム オブジェクトの LDAP ディレクトリへのバインディングを可能にする JNDI `Referenceable` のサポートが提供されています。すべてのホーム参照は、LDAP サービス プロバイダによって供給されているものを含む、任意の JNDI コンテキストに含めることができます。

コンテキスト ファクトリを作成することによって、カスタム JNDIコンテキストを使用できます。カスタム コンテキスト またはサード パーティ コンテキスト のいずれかのインスタンスへのアクセスが一度確立すると、これに応じて参照がホーム オブジェクトにバインド または再バインド されます。ホーム オブジェクト 参照は、通常の検索メカニズムを使用して既定の EJB エンジン コンテキストから取り出されます。

JNDIを使用してネーム サーバーまたはディレクトリ サーバーのサービスにアクセスする場合はまず、JNDI コンテキストのインスタンスを作成します。JNDI には、このコンテキスト 作成プロセスを標準化するための `InitialContext` API が用意されています。JNDI によって作成されるコンテキストのタイプは、`InitialContext` API を呼び出す前にサービス プロバイダ固有のコンテキスト ファクトリを指定することによって制御します。

JRun には、JNDI コンテキストを作成するためのコンテキスト ファクトリが含まれています。これらのコンテキストは、標準 `bind` メカニズムを使用して、EJB エンジンによって取り込まれた後にクライアントに割り当てられます。

ディレクトリ サーバー (LDAP) へのアクセスを提供するオプションなど、JNDI コンテキストをカスタマイズするためのオプションが多数あるため、EJB エンジンは、ホーム オブジェクト をサード パーティ コンテキスト にはバインド しません。これにより、拡大し続けるカスタマイゼーション オプション リストを制御するための専用のプロパティ セットを指定する必要はありません。

カスタム コンテキスト ファクトリを作成するには、`java.naming.spi.InitialContextFactory` インターフェイスを、1 つのメソッド `getInitialContext(...)` を持つクラス内に組み込む必要があります。このメソッド内で、EJB エンジンの初期コンテキスト を作成し、任意のカスタム オブジェクト をこのコンテキスト にバインド します。また、クライアント コード内では、カスタム コンテキスト ファクトリの名前とともに `allaire.ejipt.ContextFactory` `INITIAL_CONTEXT_FACTORY` プロパティも必ず設定してください。

EJB エンジンは、JRun によってエクスポートされたホーム オブジェクトを持つサード パーティ JNDI コンテキストをカスタマイズするのに、標準コンテキスト ファクトリ メカニズムに依存します。LDAP ディレクトリ などのサード パーティ コンテキスト からホーム オブジェクトにアクセスする場合はまず、コンテキスト ファクトリを作成します。さらに、このコンテキスト ファクトリは、JRun およびサードパーティ ディレクトリ サーバーの両方に接続し、選択されているカスタマイゼーション オプションを使用してホーム オブジェクトをバインドします。

プロパティの操作

このセクションには、『JRun によるアプリケーションの開発』で扱われていない EJB プロパティの操作方法に関する情報が含まれています。

コンテナのプロパティ

EJB エンジンによって作成されたすべてのコンテナには独自のプロパティ リストがあります。このプロパティ リストは、既定では、サーバーのプロパティ リストに設定されます。JAR のトップレベルの `default.properties` ファイルは、このプロパティ リストにロードされ、以前にロードされたサーバー プロパティに上書きされます。

公開されたすべての Bean は、すべてのシステム プロパティ、ejipt.properties、コマンド ライン プロパティ、および local.properties への読み取り 専用のアクセス権限を持っています。下位レベルのプロパティにアクセスすると、上位レベルでは常にプロパティが上書きされます。

Bean プロパティ

このセクションには、『JRun によるアプリケーションの開発』で扱われていない Bean プロパティに関する情報が含まれています。

命名 Bean

命名 Bean またはそれらのインターフェイスには必要条件も 制限事項も あり せん。命名規則では一切仮説を 立てら れません。したがって、Bean 開発者は、Bean のホーム インターフェイス、Bean のリモート インターフェイス、および Bean の実装の名前を Bean のプロパティ ファイル内で指定することが必要です。たとえば、次のエントリ (推奨規則) を含めることができます。

```
ejb.homeInterfaceClassName=ejbeans.CustomerHome
ejb.remoteInterfaceClassName=ejbeans.Customer
ejb.enterpriseBeanClassName=ejbeans.CustomerBean
#次のエントリを含めることもできます。ただし、推奨はしません。
#ejb.homeInterfaceClassName=ejbeans.Abc
#ejb.remoteInterfaceClassName=ejbeans.Xyz
#ejb.enterpriseBeanClassName=ejbeans.SomeBean
```

エンティティ Bean のプライマリ キー クラス タイプを指定する必要があり ます。このプライマリ キー クラス タイプは、Deploy ツールがクラス 実装を作成する際に使用したり、コンテナがコンテナ管理パーシスタンスを使用する際に使用します。

```
#プライマリ キー クラス タイプ
ejb.primaryKeyClassName=ejbeans.PK
```

Home 名

開発者は、JNDI ネーム空間内の Bean に関連付けられている Home 名を指定する必要があります。このプロパティは、JNDI コンテキスト内のホーム オブジェクトをバインドするのに使用されます。

```
ejb.beanHomeName=sample2a.BalanceHome
```

ステート管理

`ejb.stateManagementType` プロパティは、セッション Bean のステートの管理方法を指定します。有効な値は、`stateful_session` と `stateless_session` です。指定しない場合、EJB エンジンは、Bean をエンティティ Bean と想定します。

```
ejb.stateManagementType=stateful_session
```

許容 ID

開発者は、`allowedIdentities` プロパティを使用して、Bean のすべてのメソッドまたは特定のメソッドを呼び出す権限を与える ID またはロールの一覧を指定することによって、ロールベースのセキュリティを実装できます。指定しない場合、プロパティは、デフォルトでは、すべての認証されたユーザに設定されます。メソッドレベルのセキュリティを指定するには、メソッド名を接頭辞として使用します。

特殊な値 `system` を指定すると、`system` ID を使用した呼び出しのみが、メソッドを実行できます。

```
remove.ejb.allowedIdentities=system
```

特殊な値 `all` は、認証に関係なくすべてのユーザを示します。次の例では、すべてのユーザが `create` および `getValue` メソッドを使用できますが、`save` メソッドは貯蓄者ロール内のユーザしか使用できず、`spend` メソッドは支出者ロール内のユーザしか使用できないように制限します。

```
create.ejb.allowedIdentities=all
getValue.ejb.allowedIdentities=all
save.ejb.allowedIdentities=saver
spend.ejb.allowedIdentities=spender
```

オブジェクトのタイムアウト

Bean は、`ejb.sessionTimeout` を使用して、セッション オブジェクトのタイムアウトまでの秒数を指定できます。指定しない場合、既定では、900 (15 分) に設定されます。

```
ejb.sessionTimeout=300
```

`ejipt.isTimeoutFromCreate` は、セッション オブジェクトのタイムアウトを、オブジェクト作成直後から始めるか、または直前のアクセスから始めるかを指定するためのプロパティです。このプロパティを指定しない場合、既定では、直前のアクセスに設定されます。

```
ejipt.isTimeoutFromCreate=true
```

Bean には、追加のプロパティを指定できます。利用可能なプロパティの詳細な一覧については、JRun JavaDocs の API マニュアルの `EjiptProperties` を参照してください。

既定のプロパティ

default.properties ファイルは通常、コンテナ レベルのプロパティを指定します。コンテナ内のすべての Bean は、このファイル内で設定されているプロパティにアクセスできます。たとえば、すべての Bean が利用できるコンテキストの数を設定したり、特定の Bean についてコンテキストの数を制限できます。また、各 Bean のプロパティ ファイル内でプロパティを指定するのではなく、JAR 内のすべての Bean に適用されるプロパティの指定もできます。

```
ejipt.maxContexts=100
BigBean.ejipt.maxContext=5
```

default.properties ファイル内で設定されるプロパティは、そのコンテナ内の Bean にのみ影響を与えます。プロパティの先頭に Bean の名前を付けると、その Bean のプロパティのみが設定されます。

マニフェスト

記述子ファイルを使用しない場合は、JAR ファイル内の Bean を識別する manifest ファイルを含める必要があります。

公開する各 Bean のプロパティ ファイルは、'Enterprise-Bean : True' というエントリとともに manifest ファイル内にリストされている必要があります。サブディレクトリ /ejbenas 内の EJB ごとに、次の 2 つのエントリがあります。

```
Name:ejbeans/Customer.properties
Enterprise-Bean:True
```

プロパティ ファイルのパスは、スラッシュを除き、manifest ファイル内にリストされているパスと同じである必要があります。このスラッシュには、manifest ファイルによって常にフォワード スラッシュが含まれます。また、name と Enterprise-Bean のペアは空白行で区切る必要があります。

公開プロパティ

deploy.properties ファイルには通常、サーバーレベルのプロパティが含まれています。このファイルは、公開する Bean、ホスト名、データ ソース、および接続制限を指定する際に Deploy ツールによって使用されます。deploy.properties ファイルは、/deploy ディレクトリ 内に保存します。次の例は、簡単な deploy.properties ファイルを示します (接頭辞として # が付いているプロパティはコメントです)。

```
ejipt.classServer.host=localhost
ejipt.ejbJars=sample_ejb.jar
ejipt.jdbcSurces=source1
source1.ejipt.sourceURL=jdbc:odbc:sampledb
#source1.ejipt.sourceUser=xyz
#source1.ejipt.sourcePassword=pass
ejipt.logStackTrace=true
ejipt.userHomeName=sample.CustomerHome
ejipt.roleHomeName=ejipt.RoleHome
ejipt.loginSessionHomeName=sample.CustomerSessionHome
ejipt.storeName=default
```

ejipt.classServer.host はホスト 名を識別し ます。ローカルで実行する場合は、値を localhost のままにできます。ただし、リ モート クライアント が接続する場合は、値をサーバーのホスト 名に設定する必要があり ます。値を指定しない場合、既定では、プロパティは現在のホストの名前に設定されます。

公開する JAR ファイルの一覧を指定するには、ejipt.ejbJars プロパティを使用します。プロパティの値は、公開する JAR ファイルをカンマ区切りのリストにする必要があります。リストした JAR ファイルは、/deploy ディレクトリ内に格納されている必要があります。指定しない場合、既定では、プロパティは /deploy ディレクトリ内のすべての JAR ファイルに設定されます。各 JAR ファイルは、サーバー内に各 JAR ファイル独自のコンテナを持っています。

ejipt.jdbcSources および source1.ejipt.source... プロパティは、データベース情報を指定します。詳細は、『JRun によるアプリケーションの開発』を参照してください。

ejipt.logStackTrace=true は、すべてのスタック トレースが詳しく 記録されるよう に指定します。

ejipt.userHomeName、ejipt.roleHomeName、および ejipt.loginSessionHomeName はすべて、ユーザ認証とセキュリティを参照します。詳細は、『JRun によるアプリケーションの開発』を参照してください。

instance.store は、サーバー内のすべてのコンテナによって使用される既定のストアです。instance.store の固有の名前は、ejipt.storeName プロパティを設定すること によって指定できます。また、次のように、Bean の名前をプロパティの前に付けることによって、その Bean のインスタンス ストアも指定できます。

```
Customer.ejipt.storeName=Customer.store
```

データ ソースの定義

deploy.properties ファイルには、データ ソースを定義するためのプロパティも 含まれています。次のスニペットは、source1、すなわちデータ ソースの定義を示します。

```
ejipt.jdbcSources=source1
source1.ejipt.sourceURL=jdbc:odbc:sample
source1.ejipt.sourceUser=xyz
source1.ejipt.sourcePassword=pass
```

ejipt.jdbcSource プロパティはソースに名前を付けます。ejipt.sourceURL は標準 Java URL 定義です。この場合、標準 JDBC/ODBCドライバを使用して sample という 名前のデータ ベースに接続します。ejipt.sourceUser および ejipt.sourcePassword プロパティには、データベースのユーザの名前とパスワードが含まれています。

サードパーティ JDBC ドライバを使用する場合は必ず、ejipt.sourceDriverClassName および ejipt.sourceURL プロパティを設定してください。

```
source1.ejipt.sourceDriverClassName=oracle.jdbc.driver.OracleDriver
source1.ejipt.sourceURL=jdbc:oracle:thin:@host:1521:orcl
```

正しいプロパティ値は、特定のドライバに固有のものでなければなりません。必ず、ドライバに付属しているマニュアルを読んでください。

これらのプロパティの詳細は、『JRun によるアプリケーションの開発』を参照してください。

コンテナ管理パーシスタンスの使用

EJB が CMP を使用することを示すには、ejb.containerManagedFields プロパティ内のコンテナによって管理されるフィールドを指定します。Bean のプロパティ ファイル内に ejb.containerManagedFields が存在すると、CMP の使用がトリガされます。ejb.containerManagedFields は、CMP を instance.store と併用する場合に設定する唯一のプロパティです。次は、id はプライマリ キー フィールドで、value は保存するデータ フィールドである例を示します。

```
ejb.containerManagedFields= id,value
```

インスタンス ストア

instance.store を使用する場合は、Bean に `finder` メソッドを完全に実装する必要があります。

すべてのコンテナ管理フィールドは、ejb.containerManagedFields リスト内に表示され、instance.store を使用する場合は直列化可能である必要があります。

すべてのコンテナ管理フィールドは、公開記述子 ejb.containerManagedFields リスト内の <cmp-fields> 要素内に表示され、instance.store を使用する際は直列化可能である必要があります。

また、Bean は、SQL クエリで使用するデータなどの一時データの保存に使用できる追加のインスタンス変数も定義できます。

AutoCaller メソッド

EJB エンジンは、EJB オブジェクトに対するメソッド呼び出しのスケジューリングをサポートしています。このタスクは、`ResourceManager.createAutoCaller()` および `ResourceManager.removeAutoCaller()` メソッドを使用して管理します。

AutoCaller を作成するには、`createAutoCaller()` メソッドを、オブジェクト、メソッド、パラメータ、および間隔(ミリ秒)とともに呼び出します。`createAutoCaller()`メソッドは、AutoCaller の固有の ID を返します。次に、サーバーは、AutoCaller が削除されるか、またはサーバーが停止するまで、指定されたメソッドを指定された間隔で呼び出します。

AutoCaller を削除するには、`ResourceManager.removeAutoCaller()` メソッドを ID とともに呼び出します。AutoCaller メソッドの永続性は、サーバーがシャットダウンしている間は失われます。

EJB サンプル 7C の次の例は、`createAutoCaller()` メソッドを呼び出します。

```
...
try {
  Method method = Customer.class.getMethod("updateWorth", new Class[]
        {});
  long interval =
    Long.parseLong(getEnvironment().getProperty("updateInterval",
      "1000"));
  _callerId = ResourceManager.createAutoCaller(customer, method,
    new Object[] {}, interval);
}
catch (Exception exception) {
  exception.printStackTrace();
}
...
```

詳細は、JRun JavaDocs ファイルに付属の API マニュアル `ResourceManager` を参照してください。

詳細情報

このセクションには、『JRun によるアプリケーションの開発』で扱われていない高度な EJB 機能に関する詳細情報が記載されています。

トランザクション属性

Bean レベルのトランザクション属性を設定するには、次の例のように、`ejb.transactionAttribute` env-entry を使用します。

```
ejb.transactionAttribute=Supports
save.ejb.transactionAttribute=Mandatory
spend.ejb.transactionAttribute=Required
```

トランザクションの管理

Bean 管理トランザクション (BMT) は、セッション Bean にのみ使用できます。エンティティ Bean はこの機能を使用できません。BMT を使用する場合は、Bean のプロパティファイル内のトランザクション属性を次のように設定する必要があります。

```
spend.ejb.transactionAttribute=BeanManaged
```

セキュリティの無効化

EJB セキュリティチェックを無効にするには、Bean のプロパティに `ejb.allowedIdentities=all` プロパティを追加します。EJB セキュリティを使用しない場合は、必ずセキュリティチェックを完全に無効にするようにプロパティを指定してください。以上のプロパティを使用すると、すべての呼び出し側は Bean のすべてのメソッドにアクセスできます。プロパティの前にメソッドの名前を付けた場合は、`all` によって、そのメソッドのみを呼び出すことができます。