JRun サンプル ガイド

Windows®、UNIX™、および
Linux™ 用 JRun 3.1

Allaire® Corporation

版権告知

部品番号 : AA-JJSAM-RK

目次

本書の概要

『JRun サンプル ガイド』の対象読者は、JRun に含まれているコード サンプルおよびサンプル アプリケーションを実行する Java 開発者です。

目次

対象読者

『JRun サンプル ガイド』の対象読者は、JRun に含まれているコード サンプルおよびサンプル アプリケーションを実行する Java 開発者です。

開発者リソース

(株) アイ・ティ・フロンティア (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) 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。次の表に記載されているサイト では、各サービスの内容の詳細が掲載されています。

リソース	説明	URL
(株) アイ・ティ・フロンティア JRun のサイト	JRun の詳細な製品情報および関連トピック	http://cfusion.sirius.co.jp/jrun/
JRun サポート フォーラム	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。	http://forums.allaire.com/jrunconf/
開発者コミュニティ	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	www.allaire.com/developer/
JRun 開発者センター	開発のヒント、記事、文書、ホワイトペーパーに関する情報サイト	www.allaire.com/developer/jrunreferencedesk/

JRun 文書の概要

JRun 文書は、あらゆる関係者をサポートできるように設計されています。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、以下の文書が含まれます。

文書	説明
『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 によるアプリケーションの開発』	Java Servlet、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 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 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 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 セットアップ ガイド』	JRun 管理コンソール (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 タグ ライブラリ クイック リファレンス カード』	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。
『JSP クイック リファレンス カード』	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。
『Allaire ClusterCATS の使用』	マルチサーバーの負荷管理およびサーバー障害の保護を行う ClusterCATS の使用方法に関する情報が記載されています。
『JRun Studio 入門』	JRun Studio を使用した Web コンテンツの構築、テスト、および公開方法について説明します。さまざまなスクリプトおよびマークアップ言語用の組み込みエディタの使用方法についても説明します。

オンライン文書の表示

すべての JRun 文書は、HTML 形式と Adobe Acrobat ファイル形式でオンラインで利用できます。文書を表示するには、JRun を実行している Web サーバーにある URL、*JRun のルート ディレクトリ*/docs/dochome.htm を開きます。

JRun Studio 文書

JRun Studio は、JRun アプリケーションを作成するためのビジュアル開発ツールです。JRun Studio には直観的に操作できる GUI インターフェイスがあり、アプリケーションの構築に必要なツールが利用できます。また、JRun Studio によって、任意の JDBC データベースのデータを選択、挿入、更新、または削除を行う複雑な SQL ステートメントを作成できます。また、HTTP を通じてリモート サーバー上のデータベースに接続することもできます。これは複雑なネットワーク設定を必要としません。

JRun Studio は JRun Developer、JRun Professional、JRun Enterprise とは別売になっています。

JRun 文書の印刷

印刷版の文書を読むには、製品とともにインストールされた Adobe Acrobat PDF ファイルを探します。PDF ファイルからは、優れた印刷出力を得ることができます。文書の全体または一部を印刷できます。

その他の情報リソース

本書で扱っているトピックの詳細については、以下のリソースも参照してください。

書籍

- 『Java Server Pages Application Development』Scott M. Stirling 他著、Sams 刊、2000 年、ISBN: 067231939X
- 『Java Servlets』Karl Moss 著、McGraw Hill 刊、1999 年、ISBN: 0071351884
- 『Java Servlets: By Example』Alan R. Williamson 著、Manning Publications 刊、1998 年、ISBN: 188477766X
- 『Java Servlet Programming』Jason Hunter、William Crawford 著、O'Reilly & Associates 刊、1998 年、ISBN: 156592391X
- 『Developing Java Servlets』James Goodwill 著、Sams 刊、1999 年、ISBN: 0672316005
- 『Inside Servlets: Server-Side Programming for the Java Platform』Dustin R. Callaway 著、Addison-Wesley Pub.Co. 刊、1999 年、ISBN: 0201379635
- 『Mastering Enterprise JavaBeans and the Java 2 Platform, Enterprise Edition』Ed Roman 著、Wiley 刊、ISBN: 0471332291
- 『Enterprise JavaBeans』Richard Monson-Haefel 著、O'Reilly & Associates 刊、ISBN: 1565928695
- 『Enterprise Javabeans: Developing Component-Based Distributed Applications』Thomas C. Valesky 著、Addison Wesley Publishing Company 刊、ISBN: 0201604469

オンライン リソース

- Java servlet API (http://java.sun.com/products/servlet)
- JavaServer Pages (http://java.sun.com/products/jsp)
- Enterprise JavaBeans (http://java.sun.com/products/ejb)
- JSP Resource Index (http://www.jspin.com)
- Servlet Source (http://www.servletsource.com)
- ServerPages.com (http://www.serverpages.com)

疑問の解決方法

プログラミング上の問題を解決する最善の方法の 1 つは、JRun フォーラムで、JRun 開発者コミュニティの幅広い経験に基づいたアドバイスを受けることです。JRun の利用方法についてどのようなことでも、メンバーであるほかの開発者のアドバイスを得ることができます。さらに、検索機能を使用すると、過去 12 か月間のメッセージを呼び出すことができるため、同じ問題をほかの開発者がどのように解決したかを知ることができます。このフォーラムは、JRun の利用方法を知るための優れた情報源であるとともに、JRun 開発者がリアルタイムで活動する様子を知ることのできるすばらしい機会でもあります。

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、いつでも投稿することができます。

第 1 章

はじめに

JRun で提供されるサンプルには、サーバーのさまざまな活用方法が示されています。サーブレットと EJB のサンプルがあります。

目次

JRun サンプルの概要

JRun には、サーブレットの技術 (サーブレット、JSP、カスタム タグ) および EJB の使用方法を示すサンプルが一式付属しています。このサンプルを実行し、それに関連するソース コードを確認することは、JRun を開始する上で役立ちます。

サンプルを実行する前に、インストール作業をすべて終了してください。終了していない場合は、サンプルを実行する前に『JRun セット アップガイド』を参照してください。

JRun バージョン 3.1 のサンプルは、バージョン 3.0 から大幅に変更されています。これらの変更の内容は次のとおりです。

- **新しい Web アプリケーション サンプル** 詳細については、5 ページの「サンプル Web アプリケーションの概要」を参照してください。
- **データベースの統合** JRun 3.1 のタグ ライブラリ および EJB サンプルでは、組み込み型 PointBase データベースを使用して、データベースにアクセスする方法をより包括的に説明します。
- **EJB サンプルの追加、変更、および削除** サンプル 1a、1b、4a、および7a が削除されました。サンプル 9b (サーブレット でのシングル サインオン) が追加されました。
- **すべての EJB サンプルで公開記述子を使用** Bean のプロパティ ファイルは除外されました。

組み込み型データ ベース

JRun バージョン 3.1 には、組み込み型 PointBase DBMS を使用するサンプル データベースが同梱されています。このデータベースは主に、JRun カスタム タグおよび EJB サンプルのサポートを目的としています。このデータベースは、開発データベースまたは公開データベースとしての使用を予定していません。このデータベースは簡単なテストに使用できますが、前もって制限事項について認識しておく必要があります。

制限

PointBase が組み込まれているデータベースは、接続すると自動的に起動されますが、1 つの JVM からの接続に限定されます。これは次のことを意味しています。

- JMC JDBC データソースの [Test] ボタンは、PointBase が組み込まれたデータベースについては無効になっています。
- PointBase Console を使用するには、default JRun サーバーを停止しておく必要があります。
- EJB サンプル 4b および 5a は複数の接続を使用するため、機能しません。

JRun バージョン 3.1 に同梱の組込み型 PointBase データベースのデータは、5 MB に限定されます。

組み込み型データベースの使用

JRun には、サンプル データベース内のデータに追加や変更を行う ユーティリティが同梱されています。これらのユーティリティは、*JRun のルート ディレクトリ*/pointbase ディレクトリにあります。ユーティリティ名は次のとおりです。

- commander (UNIX) および commander.bat (Windows) コマンドライン インターフェイスによって SQL ステートメントを入力できます。
- console (UNIX) および console.bat (Windows)　Swing ベースの GUI から、データベースのテーブルおよび行を変更できます。

メモ
これらのユーティリティは、default JRun サーバーを停止してから実行してください。

詳細については、*JRun のルート ディレクトリ*/pointbase/docs ディレクトリにある PointBase のマニュアルを参照してください。

PointBase コンソールでは、新しいデータベースを作成できます。サーブレット、JSP、および EJB からこのデータベースにアクセスするには、次の設定値を使用してください。

- **ドライバ**　com.pointbase.jdbc.jdbcUniversalDriver
- URL　jdbc:pointbase://embedded/*データベース名*
- **ユーザ名**　PUBLIC (既定値)
- **パスワード**　PUBLIC (既定値)

このサンプル データベースは、ネットワーク バージョンの PointBase DBMS と互換性があります。評価バージョンおよびアップグレード情報については、http://www.pointbase.com を参照してください。

ロック ファイルの削除

JRun の全サーバを停止しても commander または console ユーティリティから PointBase データベースにアクセスできない場合は、*JRun のルート ディレクトリ*/pointbase/databases ディレクトリに *.lck ファイルがないかどうかを確認してください。*.lck ファイルは、PointBase が異常終了した場合に表示されます。これらのファイルを削除できない場合は、次の手順を実行してください。

1　コンピュータを再起動します。

2　再起動で JRun が自動的に実行されたら中止してください。

3　*JRun のルート ディレクトリ*/pointbase/databases ディレクトリから、すべての *.lck ファイルを削除します。

サーブレット技術を示すサンプル

JRun には、JSP、カスタム タグ ライブラリ、サーブレットなどのすべてのタイプのサーブレット技術のサンプルが含まれています。サンプルを見るには、JMC の「ようこそ」ページの [アプリケーションの例] をクリックしてください。

JSP のサンプルの概要

JRun が提供している JSP サンプルは次のとおりです。JSP のサンプルについては、9 ページの第 2 章「JSP のサンプル」を参照してください。

サンプル	説明
Hello World	テキストの大きさを徐々に増減させながら、「Hello World」と表示します。
Color Size Bean	JavaBean の `ColorSizeBean` にあるメソッドとプロパティの使用方法を示します。
JavaScript の例	JSP で (Java の代わりに) JavaScript の使用方法を示します。
Color Size Bean	クエリ文字列からテキストにアクセスする場合の `request.getParameter` メソッドの使用方法を示します。
HTML フォームの例	JSP を使用するフォームのコーディング方法を示します。

カスタム タグ ライブラリのサンプルの概要

JSP 1.1 の仕様には、タグ ライブラリと呼ばれる JSP の拡張メカニズムが含まれています。タグ ライブラリは、ある種の機能を内部に組み込んだ一連のカスタム タグ (アクションとも呼ばれます) を定義したものです。JRun にはカスタム タグ ライブラリが付属しています。JRun の例には、このタグ ライブラリを使用する次のようなサンプルが入っています。カスタム タグ サンプルについては、17 ページの第 3 章「タグ ライブラリのサンプル」を参照してください。

サンプル	説明
sql と sqlparam	`sql` および `sqlparam` タグを使用して、SQL データベースにアクセスする方法を示します。
query2xml	`query2xml` タグを使用して、データベース クエリの結果を XML に変換する方法を示します。
xslt	`query2xml` タグ、埋め込み XML、および外部 XML ファイルを使用して、作成した XML を変換する方法を示します。
sendmsg、msgparam、および getmsg	`sendmsg`、`msgparam`、および `getmsg` タグを使用して、非同期メッセージの送受信方法を示します。

サンプル	説明
sendmail、mailparam、および getmail	sendmail、mailparam、および getmail タグを使用して、電子メールの送受信方法を示します。
servlet と servletparam	サーブレットを呼び出す方法とパラメータを渡す方法を示します。
jndi	jndi タグを使用する方法をいくつか示します。
transaction	トランザクションによって、2 つのデータベースを更新する方法を示します。

サーブレット のサンプルの概要

JRun が提供しているサーブレットのサンプルは次のとおりです。サーブレットのサンプルについては、33 ページの第 4 章「サーブレットのサンプル」を参照してください。

サンプル	説明
SimpleServlet	コンテンツ タイプを設定して、単純なテキスト文字列を含んでいる HTML を返す方法を示します。
DateServlet	現在の日付/時刻を表示する方法と自動的にページをリフレッシュする方法を示します。
CounterServlet	クッキーを使用してページのヒット カウンタを保守する方法を示します。
SnoopServlet	サーブレットと環境情報を取得して、表示する方法を示します。

サンプル Web アプリケーションの概要

JRun では、次のサンプルを含んでいるサンプル Web アプリケーションを提供します。インボイス サンプルについては、41 ページの第 5 章「インボイス Web アプリケーション」を参照してください。

サンプル	説明
Fax Cover Sheet Generator	ユーザの入力を使用してファックスのカバー シートを生成する JSP
Invoice Generator	Invoice Generator には次の 3 バージョンがあります。 • JSP およびスクリプトレットのみ • JavaBean を使用する JSP • カスタム タグを使用する JSP

EJB のサンプル

JRun には、EJB のさまざまなタイプの使用例を示すサンプルが含まれています。このサンプルにアクセスするには、それに関連する指示を読み、サンプル固有の `make` ファイルを実行してください。

EJB サンプルの概要

JRun が提供している EJB サンプルは次のとおりです。

サンプル	説明
サンプル 2 Bean 管理パーシスタンス (第 7 章)	リレーショナル データベースを使用した Bean 管理パーシスタンス (BMP) について説明します。この例では、組み込まれたリレーショナル データベースにアクセスします。
サンプル 3 コンテナ管理パーシスタンス (第 8 章)	リレーショナル データベースを使用したコンテナ管理パーシスタンスについて説明します。
サンプル 4 トランザクション (第 9 章)	複数のサーバー インスタンス間の分散型 2 フェーズ コミット トランザクション管理について示します。
サンプル 5 オブジェクト管理 (第 10 章)	参照されるエンティティ オブジェクトが解放されず、ガーベッジコレクションが行われなくなった場合の、分散ガーベッジ コレクション機能について説明します。また、大量のオブジェクトを作成し、それをクライアントにコレクションとして返す方法や、カスタム RMI オブジェクトを作成する方法についても説明します。
サンプル 6 メッセージ (第 11 章)	ポイントツーポイント (メッセージ キューイング) メカニズムとパブリッシュ /サブスクライブ (ブロードキャスト) メカニズムの両方に対する Java Message Service (JMS) サポートについて説明します。また、JMS と EJB の統合についても説明します。

サンプル	説明
サンプル 7 高度な Bean (第 12 章)	ビジネス上の複雑な課題を解決するためにともに機能する、さまざまな種類の Bean (エンティティ、ステートフル セッション、およびステートレス セッション) の使用方法について説明します。デッドロック例外の対処方法や自動呼び出し機能についても紹介しています。機能としては、このサンプルは銀行から顧客へのローンの発行シミュレーションを行います。銀行数および利率や収益率は、`deploy.properties` ファイルで設定します。顧客数は、実行時にコマンド ライン引数で定義します。 サンプル 7b は、リレーショナル データベースを通して Bean 管理パーシスタンスを使用しています。 サンプル 7c は、リレーショナル データベースを通してコンテナ管理パーシスタンスを使用しています。
サンプル 9 Servlets (第 13 章)	EJB エンジンでサーブレットおよび JSP を使用する方法について説明します。
サンプル 10 JDK 1.1 クライアント (第 14 章)	JDK 1.1 クライアントで EJB エンジンを使用する方法について説明します。

EJB に関するその他の情報

ここでは、EJB のサンプルの理解に役立つ情報について説明します。

メモ

EJB サンプルは EJB エンジンをスタンドアロン モードで実行するので、Bean の処理をコンソール ウィンドウに表示できます。EJB エンジンは、default JRun サーバーのポート設定を使用して実行されるので、EJB サンプルの実行中は、default JRun サーバーを実行できません。default JRun サーバーを使用して EJB のサンプルを実行する方法については、98 ページの「Make Standalone の使用」を参照してください。

"Bean" は、"Enterprise JavaBeans" のことを表しているものとします。

`instance.store` フラット ファイル データベースは、さまざまなサンプルで使用します。各サンプルを実行した後は `instance.store` をクリアして、次にサンプルを実行するときに前のサンプルのデータが使用されないようにする必要があります。`instance.store` をクリアするには、*`JRun のルート ディレクトリ`*`/servers/default/runtime` ディレクトリに移動して `instance.store` を削除します。

表記法

サンプルをわかりやすくするため、*JRun のルート ディレクトリ*という表記を使用して JRun インストール ディレクトリを表しています。*JRun のルート ディレクトリ*は、実際のインストール環境の適切なパスに置き換えてください。

各サンプルを実行する前に JRun インストール ディレクトリに JRUN_HOME 環境変数を設定する必要があります。*JRun のルート ディレクトリ*と JRUN_HOME は同じものを指します。

メモ

*JRun のルート ディレクトリ*は通常、JRun のインストール ディレクトリを表します。JRUN_HOME は make ファイルおよび makew ファイル (次で説明) で使用する変数で、この変数は JRun インストール ディレクトリを指している必要があります。

コマンド

EJB のサンプルは、サンプルの適切な実行環境を作成するために必要なファイルのコンパイル、パッケージ、および実行を行う make ファイルによって操作されます。いくつかの基本的な make コマンドがありますが、これについては第 6 章で概説し、さらに第 15 章で詳しく述べます。make ファイルに似たバッチファイルが、Windows ユーザのために用意されています。make コマンドを入力する代わりに、makew コマンドを入力してバッチファイルを実行します。make または makew を実行する前に、JRUN_HOME 環境変数を設定してください。

その他のサンプル

定期的に、Allaire 開発者センター (http://www.allaire.com/developer/jrunreferencedesk) の JRun エリアにある追加サンプルを確認してください。

第 2 章

JSP のサンプル

目次

JSP のサンプルの概要

JRun には、JSP コーディングの基本概念の理解に役立つ JSP のサンプルが含まれています。

このサンプルについて十分に理解するために、それらを実行し、View Source リンクをクリックしてそのソース コードを確認し、その上で再度実行してください。サンプルを実行するには、admin JRun サーバーおよび default JRun サーバーが実行されていることを確認し、JMC を起動してから「ようこそ」ページの [アプリケーションの例]をクリックします。JSP のサンプルのソース コードは、*JRun のルート ディレクトリ*`/servers/default/demo-app/jsp` ディレクトリにあります。

Hello World

説明 ポイントの大きさを徐々に増減させながら、「Hello World」と表示します。

ファイル名 `hello.jsp`

確認事項 ポイント サイズの増減を制御する `for` ループと JSP の式を調べます。

Color Size Bean

説明 JavaBean の ColorSizeBean にあるメソッドとプロパティの使用方法を示します。

ファイル名 colorsize.jsp

確認事項 jsp:useBean と jsp:setProperty のステートメントを調べます。また、ColorSizeBean でのメソッドの呼び出し方を調べます。ソースコードは、*JRun のルートディレクトリ*/servers/default/demo-app/WEB-INF/classes の ColorSizeBean.java にあります。

JavaScript の例

説明 JSP で (Java の代わりに) JavaScript の使用方法を示します。

ファイル名 `javascript.jsp`

確認事項 page ディレクティブの中にある `language=javascript` に注意してください。JavaScript `Date` オブジェクトの使用法と `Date` オブジェクトの `toString` メソッドの呼び出し方法にも注意してください。

QueryString の例

説明 `request.getParameter` メソッドを使用して、クエリ文字列からテキストにアクセスする方法を示します。

ファイル名 `qstring.jsp`

確認事項 `request.getParameter` への呼び出しによって、クエリ文字列のサイズと色がどのように返されるかを調べます。

HTML フォームの例

説明 JSP を使用するフォームのコーディング方法を示します。

ファイル名 `form.jsp`

確認事項 フォーム データにアクセスする `request.getParameter` と `request.getParameterValues` の使用方法を確認します。

第 3 章

タグ ライブラリのサンプル

目次

タグ ライブラリのサンプルの概要

JSP 1.1 の仕様には、タグ ライブラリと呼ばれる JSP の拡張メカニズムが含まれています。それぞれのタグ ライブラリは、一連のカスタム タグ (アクションとも呼ぶ) を定義します。カスタム タグは特定のタイプの機能をカプセル化したものです。JRun には JRun タグ ライブラリが含まれており、これを JSP で使用すると、データへのアクセス、フォームの検証、サーブレットへのアクセス、およびその他のタイプの機能を実行できます。

JRun タグ ライブラリのサンプルで、JRun のカスタム タグの使用法を理解してください。JRun カスタム タグ ライブラリの詳細については、『JRun タグ ライブラリ リファレンス』を参照してください。

サンプルを最大限に活用するには、まず、『JRun タグ ライブラリ リファレンス』を参照してください。これは、JRun 文書のページから利用できます。タグ ライブラリの使用法を把握したら、サンプルを実行して JSP ソース コードを確認してください。タグ ライブラリのサンプルを実行するには、JMC を起動して「ようこそ」ページで [アプリケーションの例] をクリックします。タグ ライブラリのサンプルのソース コードは、*JRun のルート ディレクトリ*`/servers/default/demo-app/taglib` にあります。

タグ ライブラリの開発については、『JRun によるアプリケーション開発』の「カスタム タグとタグ ライブラリ」の章を参照してください。

sql と sqlparam

説明 このサンプルでは、`sql` および `sqlparam` タグを使用して SQL データベースにアクセスする方法を示します。

メモ
`sql` タグを使用するには、JDBC ドライバをインストールし、JMC を使用して JDBC クラスパスを JRun サーバーのクラスパスに追加しておく必要があります。

ファイル名 `sql.jsp` と `sqlparam.jsp`

確認事項 構文とサンプル ページを確認します。

query2xml

説明 このサンプルでは、query2xmlタグを使用して、データベースのクエリ 結果を XML に変換する方法を示します。

ファイル名 query2xml.jsp

確認事項 処理済み JSP のソースを表示して、そのページ内の XML の内容を調べます。

xslt

説明 このサンプルでは、`query2xml` タグ、組み込まれている XML、および外部 XML ファイルを使用して作成した XML を変換する方法を示します。

ファイル名 `xslt.jsp`

確認事項 構文およびサンプル ページを確認し、*`JRun` のルート ディレクトリ*`/servers/default/demo-app/taglib` ディレクトリ にある XSL ファイルである `table.xsl` を表示します。

sendmsg、msgparam、および getmsg

説明 このサンプルでは、`sendmsg`、`msgparam`、および `getmsg` タグを使用した非同期メッセージの送受信方法を示します。

ファイル名 `sendmsg.jsp`、`msgparam.jsp`、および `getmsg.jsp`

確認事項 構文とサンプル ページを確認します。

sendmail、mailparam、および getmail

説明 このサンプルでは、sendmail、mailparam、および getmail タグを使用した電子メールの送受信方法を示します。getmail サンプルを実行する場合は、getmail.jsp ファイルを変更する必要があります。使用法については、『JRun タグ ライブラリ リファレンス』を参照してください。

ファイル名 sendmail.jsp、mailparam.jsp、および getmail.jsp

確認事項 構文とサンプル ページを確認します。

servlet と servletparam

説明 このサンプルでは、サーブレットを呼び出す方法とパラメータを渡す方法を示します。

ファイル名 `servlet.jsp` と `servletparam.jsp`

確認事項 `servletparam` タグを使用してパラメータを渡す方法に注意してください。

jndi

説明 このサンプルでは、`jndi` タグを使用する方法をいくつか示します。

ファイル名 `jndi.jsp`

確認事項 構文とサンプル ページを確認します。

transaction

説明 このサンプルでは、トランザクションによって、2 つのデータベースを更新する方法を示します。このサンプルを実行するには、transaction.jsp ファイルを変更して、セットアップを追加する必要があります。使用法については、『JRun タグ ライブラリ リファレンス』を参照してください。

ファイル名 transaction.jsp

確認事項 foreach タグを使用して、Enumeration、または sql、getmail、getmsg のいずれかによって返される テーブルをループ化します。

param

説明 このサンプルでは、JSP スクリプト変数を宣言する方法を示します。

ファイル名 `param.jsp`

確認事項 データ タイプを指定する `type` 属性と、既定値を指定する `default` 属性を確認します。

foreach

説明 このサンプルでは、ループをコーディングする方法を示します。

ファイル名 `foreach.jsp`

確認事項 `Enumeration` または `sql`、`getmail`、`getmsg` タグのいずれかによって返されるテーブルをループ化する場合の `foreach` タグの使用法です。

if

説明 このサンプルでは、実行を条件ブロックする方法を示します。

ファイル名 `if.jsp`

確認事項 構文とサンプル ページを確認します。

switch

説明 このサンプルでは、Java の switch-case 構成のように実行を条件ブロックする方法を示します。

ファイル名 `switch.jsp`

確認事項 構文とサンプル ページを確認します。

case

説明 このサンプルでは、Java の switch-case 構成のように実行を条件ブロックする方法を示します。

ファイル名 `case.jsp`

確認事項 構文とサンプル ページを確認します。

form、input、および select

説明 このサンプルでは、form タグ、input タグ、および select タグを使用して対話機能を拡張する方法を示します。

ファイル名 form.jsp、input.jsp、および select.jsp

確認事項 このタグにより、クライアント側の JavaScript 検証が組み込まれている HTML 形式ファイルを作成できます。input タグと select タグを使用して検証基準とエラー基準を宣言します。これらのタグによって、必要な機能を実行する JavaScript が自動的に生成されます。

第 4 章

サーブレットのサンプル

目次

サーブレットのサンプルの概要

JRun には、サーブレット API を使用したコーディングの基本概念の理解に役立つサーブレットのサンプルが含まれています。

このサンプルについて十分に理解するために、それらを実行して、そのソース コードを確認し、その上で再度実行してください。サーブレットのサンプルを実行するには、JMC を開始してから「ようこそ」ページの [サンプル アプリケーション] をクリックしてください。サーブレットのサンプルのソース コードは、*JRun のルート ディレクトリ*`/servers/default/demo-app/WEB-INF/classes` にあります。

メモ

すべての JRun サンプル サーブレットでは `JRunDemoServlet` クラスが拡張されているので、サンプル サーブレット の一貫性のある見た目と使い心地を実現できます。特に、サンプル サーブレット が `generateDemoPageStart` メソッドと `generateDemoPageEnd` メソッドを呼び出していることに注意してください。これらは、`JRunDemoServlet` で定義されています。

JRunDemoServlet

説明 すべての JRun サンプル サーブレットのベース クラスとして使用される抽象クラス。サンプル サーブレットはこのクラスを拡張しており、`generateDemoPageStart` メソッドと `generateDemoPageEnd` メソッドを呼び出して各ページの始めと終わりに HTML を構築します。

ファイル名 JRunDemoServlet.java

確認事項 `generateDemoPageStart` メソッドと `generateDemoPageEnd` メソッドを確認します。これらのメソッドで `out` 変数を使用して、どのように HTML を返すかを調べます。また、`out.println` メソッドが、引用符で囲まれたテキストとオブジェクト変数 (`ROW_ALT_COLOR`、`TITLE_COLOR` など) を統合する方法にも注意してください。

SimpleServlet

説明 コンテンツ タイプを設定して、単純なテキスト文字列を含んでいる HTML を返す方法を示します。

ファイル名 SimpleServlet.java

確認事項 このサーブレットは、`HttpServletResponse.getWriter` メソッドによって、`PrintWriter` を作成します。サーブレットはこのオブジェクトを使用して HTML をブラウザに返します。また、サーブレットが、`HttpServletResponse.setContentType` メソッドを使用して、コンテンツ タイプを `text/html` に設定していることにも注意してください。

DateServlet

説明 現在の日付/時刻を表示する方法とページを自動的に更新する方法を示します。

ファイル名 DateServlet.java

確認事項 サーブレットは `Date` オブジェクトを作成し、`toString` メソッドを使用して現在の日付および時刻を `String` として返します。また、サーブレットが、`mode` と呼ばれる URL パラメータを使用して、簡単な自動更新機能を実装していることに注意してください。

CounterServlet

説明 クッキーを使用してページのヒット カウンタを保守する方法を示します。

ファイル名 CounterServlet.java

確認事項 サーブレットは、`HttpServletRequest.getCookies` メソッドを呼び出して、すべてのクッキーを配列として取得します。次に、その配列をループ化して、`counter` という名前のクッキーを検索します。`counter` というクッキーが検出されない場合、サーブレットはこのクッキーを作成してその値を 1 に設定します。`counter` クッキーが検出された場合は、現在の値を表示してからそのクッキーの値を増やします。

SnoopServlet

説明 サーブレットと環境情報を取得して、表示する方法を示します。

ファイル名 SnoopServlet.java

確認事項 サーブレットは次のような private メソッドを実装して、サーブレットと環境データを取得します。

- `getInitParameterData`
- `getContextParameterData`
- `getAttributeData`
- `getSessionData`
- `getRequestParameterData`
- `getRequestParametersData`
- `getHeaderData`
- `getCookieData`
- `getRequestData`

これらのメソッドを確認して、サーブレット API メソッドを呼び出してデータを取得する方法を理解してください。

このサーブレットには、折りたたみ式の表を表示する機能があります。これは、`makeTableEntry` メソッドで確認できます。

第 5 章

インボイス Web アプリケーション

目次

インボイス Web アプリケーションの概要

インボイス Web アプリケーションでは、広範な技術を説明し、コメント付きのコードを紹介します。これは、demo-app の Web アプリケーションとは別に invoice-app としてインストールされています。このサンプル アプリケーションには次の URL からアクセスします。

```
http://localhost:8100/invoice/index.jsp
```

インボイス Web アプリケーションの特徴を次に示します。

- 注釈や使用手順の詳細な説明
- ソース コードのカラー表示

この Web アプリケーションには、次の 2 つの主要コンポーネントがあります。

- Fax Cover Sheet Generator フォームを表示し、印刷に適したファックス用のカバー シートを作成する簡単なファックス用カバー シート作成アプリケーションです。この Fax Cover Sheet Generator では、JSP の宣言、スクリプトレット、および式について説明します。
- Invoice Generator インボイス作成アプリケーションには 3 つのバリエーションがあります。Invoice Generator では、次の内容について説明します。
  - JSP スクリプトレット
  - JSP と JavaBeans
  - JSP とカスタム タグ

Fax Cover Sheet Generator

説明 フォームを表示し、印刷に適したファックス用のカバー シートを作成する、簡単なファックス用カバー シート作成アプリケーションです。このアプリケーションでは、JSP の宣言、スクリプトレット、および式について説明します。

ファイル名 `coverSheet.jsp` と `generateCoverSheet.jsp`

確認事項 作成ページでは、スクリプトレットを使用して起動ページに入力されているパラメータを抽出します。ヘッダの作成後、作成ページによってパラメータ値、ボイラープレートの順に、フォーマットされたテーブル内に配置されます。値は、インライン スクリプトレット内で宣言され、初期化された変数を使用して JSP 式として提供されます。

オプションのメッセージが標準情報のテーブルの後に続きます。スクリプトレットによって、オプションでのメッセージの有無を操作します。ここでも、JSP 式によって実際の値が提供されます。

Invoice Generator

説明 Invoice Generator には、3 つのバリエーションのインボイス作成アプリケーションが用意されています。このサンプルでは、次について説明します。

- JSP スクリプトレット
- JSP と JavaBeans
- JSP とカスタム タグ

確認事項 各バリエーションを実行し、基礎となるコードを注意して確認してください。さらに、関連付けられた JavaBeans やカスタム タグ ハンドラのコードも表示してください。

第 6 章

EJB サンプルの実行開始

目次

メモ

本書のこの章以降のサンプルでは `bash` シェルを使用しています。Windows ユーザは、`make` コマンドを `makew` コマンドに置き換えて使用してください。標準 DOS ウィンドウを実行する Windows ユーザは、`export` の代わりに「`set`」と入力し、フォワードスラッシュの代わりに円記号を使用してください。

概要

各サンプルを実行するための基本的なコマンドがいくつかあります。これらのコマンドについては、この章と95 ページの第 15 章「Make ファイル」で詳しく説明しています。ここでは *JRun のルート ディレクトリ*という表記を使用して、JRun インストール ディレクトリを一般的に表しています。*JRun のルート ディレクトリ*は、実際のインストール環境の適切なパスで置き換えてください。サンプルを簡単にするために、UNIX および Linux の場合は `/opt/jrun` の下に JRun がインストールされ、Windows の場合は `C:¥Program Files¥Allaire¥JRun` の下に JRun がインストールされていると仮定します。

JRun を再インストールするたびに EJB サンプルをいくつか実行することをお勧めします。これによって、環境が正しく設定されていることを容易かつ迅速に確認できます。

はじめに

EJB サンプルを実行する前に次のシステム設定を確認し、必要なリソースを確実に利用できるようにしてください。

- JDK 1.2 以上を実行できるシステムであること。JDK 1.1 では EJB エンジンは稼動しません。
- Java コンパイラを含むディレクトリが、システム パスにあること。たとえば、`jdk1.2.2/bin` となります。このディレクトリがシステム パスにないと、`make/makew` コマンドが使用できません。
- `JRUN_HOME` 環境変数を、JRun ルートを指示するように定義する必要があります。

サンプルの実行

ホスト名の設定

EJB エンジンは、クラス サーバーの位置を把握しておく必要があります。そのために、host プロパティを設定します。*JRun_rootdir*/samples/sample2a に移動し、テキスト エディタを使用して deploy.properties ファイルを開きます。ここで、ejipt.classServer.host プロパティが現在 localhost に設定されていることに注意してください。この設定をホスト名またはホストの IP アドレスに変更します。サーバーとクライアントをローカルで実行している場合は、このプロパティをマシン名に設定するか、localhost のままにしておくことができます。変更内容は必ず保存してください。このプロパティを指定しないと、既定では JRun はサーバーが動作しているホストの名前を使用します。

ejipt.classServer.host プロパティで指定したホスト名または IP アドレスを使用して、すべてのクライアントがサーバーにアクセスできる必要があることに注意してください。このことは、ファイアウォールやネットワーク間を経由する場合に特に重要です。

シェル (bash シェル) または DOS ウィンドウを開く

シェルの場合の手順

UNIX、Linux、または Windows (bash シェル使用時) では、コマンド プロンプト ウィンドウを開き、次のコマンドを入力します (opt/jrun は、UNIX および Linux 上の既定の JRun インストール ディレクトリです)。

```
> bash
bash$ export EJIPT_HOME=/opt/jrun
bash$ cd /opt/jrun/samples/sample2a
```

メモ
基本的に、この説明では bash (Bourne-Again Shell) を使用します。bash コマンドを使用すると、make ファイルを実行するための bash シェルを作成できます。

常に、JRUN_HOME 環境変数を設定する必要があります。export コマンドで、環境変数 JRUN_HOME を JRun があるディレクトリ (この場合は /opt/jrun ディレクトリ) に設定します。

bash シェルで作業する場合は、区切り記号としてフォワード スラッシュ (/) を使用してください。必要に応じて別のシェルを使用することもできます。しかし、bash シェルは、make ファイルを 一語一句正確に実行する場合に必要です。

DOS ウィンドウの場合の手順

Windows では、DOS ウィンドウを使用して次のコマンドを入力できます。

```
set JRUN_HOME=c:¥Program Files¥Allaire¥JRun
```

```
cd "c:¥Program Files¥Allaire¥JRun¥samples¥sample2a"
```

常に、JRUN_HOME 環境変数を設定する必要があります。set コマンドで、環境変数 JRUN_HOME を JRun があるディレクトリ (この場合は c:\Program Files\Allaire\JRun ディレクトリ) に設定します。

Bean JAR ファイルの作成

これで Bean およびクライアントの JAR ファイルを作成できるようになりました。次のコマンドを入力します。

```
bash$ make jars
```

make jars コマンドは、EJB ソースファイルをコンパイルして、そのサンプルの EJB JAR ファイル (sample*xx*_ejb.jar) を作成し、それを *JRun のルート ディレクトリ*/servers/default/deploy ディレクトリにコピーします。さらに、このサンプルのクライアント JAR ファイル (sample*xx*_client.jar) を作成し、それを *JRun のルート ディレクトリ*/samples/sample*xx* ディレクトリにコピーします。ここで、*xx* はサンプル番号 (2 など) です。

Bean の展開

次のコマンドを入力して Bean を展開します。

```
bash$ make deploy
```

make deploy コマンドによって、Deploy ツールが実行されます。Deploy ツールは、make jars の手順で作成した EJB JAR ファイルを使用して、ホームおよびオブジェクトのインターフェイス実装を生成します。その結果作成される ejipt_objects.jar ファイルは、*JRun のルート ディレクトリ* /servers/default/deploy ディレクトリに設定されます。この手順ではスタブ クラスも作成され、その結果作成された ejipt_exports.jar ファイルは *JRun のルート ディレクトリ*/servers/default/deploy ディレクトリに再度設定されます。次に、Make deploy によって deploy ディレクトリに deploy.properties がコピーされ、runtime.properties ファイルを作成するためのベースとして使用されます。

メモ

make deploy コマンドは、サンプルのディレクトリから *JRun のルート ディレクトリ*/servers/default/deploy ディレクトリに、サンプル固有の deploy.properties ファイルをコピーします。これにより、個々のサンプルの整合性が保たれます。しかし、EJB で作業を開始したら、*JRun のルート ディレクトリ*/servers/default/deploy ディレクトリにある deploy.properties ファイルでのみ作業する必要があります。

Deploy ツールでは既定で、JDK 1.2 コンパイラを使用して生成されたクラスをコンパイルします。deploy.properties ファイルにさまざまな ejipt.javac.* プロパティを設定することによって、Jikes などの異なるコンパイラを使用できます。

EJB エンジンの起動

Bean が展開されたら、EJB エンジンをスタンドアロン モードで起動します。次のコマンドを入力します。

```
bash$ make standalone
```

処理が終了したら、EJB エンジンの コマンド プロンプトを確認します。

```
Server is running (type h[elp]<ENTER> for help on commands)
>
```

`make standalone` コマンドは、`deploy` ディレクトリ内の JAR ファイルを使用して EJB エンジンを起動します。`deploy` ディレクトリ内の JAR およびプロパティ ファイルは `runtime` ディレクトリにコピーされます。これで、EJB エンジンはいつでもクライアントの要求を受け入れられる状態になりました。

メモ

`make standalone` コマンドは、default JRun サーバーのディレクトリおよびポート設定を使用して EJB エンジンを起動します。default JRun サーバーを停止してから、`make standalone` を発行してください。

クライアントの起動

クライアントを起動するには、別のコマンド プロンプト ウィンドウを開き、次のコマンドを入力します。*JRun のルート ディレクトリ*は、実際の JRun インストール ディレクトリに置き換えてください。

```
C:¥> bash
bash$ export JRUN_HOME=JRun のルート ディレクトリ
bash$ cd JRun のルート ディレクトリ/samples/sample2a
bash$ make run
```

クライアント ログイン ウィンドウが表示されます。説明および使用法については、各サンプルの説明を参照してください。クライアントを停止するには、ログイン ウィンドウの [閉じる] ボタンをクリックします。

EJB エンジンの停止

サンプルを終了する場合、「q」を入力して Enter キーを押し、EJB エンジンを停止します。次のような出力が表示されます。

```
>q
Server stopped
```

クライアント アプリケーションの概要

EJB アプリケーションには、クライアント側とサーバー側があります。各 EJB サンプルでは異なる技術が示されていますが、ほとんどのサンプルでクライアント側と同じコンポーネントが使用されます。

`make run` (Windows では `makew run`) を指定すると、グラフィカルな Java アプリケーションが起動し、ユーザ名 (通常は、chief、saver1、saver2、spender1、spender2 のいずれか) およびパスワード (通常は pass) の入力を要求するプロンプトが表示されます。ログインに成功すると異なるウィンドウが表示されます。このウィンドウでは結果的に EJB を呼び出す処理を実行できます。

クライアント側の処理で使用されるメイン ファイルは、`EJBClient.java` です。このファイルについて検討し、クライアントに追加する必要があるコードについて理解する必要があります。

メモ

サンプル 9a と 9b はサーブレットのサンプルなので、`EJBClient.java` を使用しません。これらのアプリケーションについては、`WEB-INF/classes` にあるファイルを参照してください。その他の例外として、サンプル 4b、7b、および 7c があります。これらは、EJB を別の EJB のクライアントにする方法を示しています。

第 7 章

Bean 管理パーシスタンス

目次

メモ

本書のこの章以降のサンプルでは bash シェルを使用しています。Windows ユーザは、`make` コマンドを `makew` コマンドに置き換えて使用してください。標準 DOS ウィンドウを実行する Windows ユーザは、`export` の代わりに「`set`」と入力し、フォワードスラッシュの代わりに円記号を使用してください。

概要

サンプル 2 では、リレーショナル データベースを使用した Bean 管理パーシスタンス (BMP、bean-managed persistence) について説明します。ユーザ エンティティ Bean は、BMP を使用し、特定の再呼出しメソッドの中に適切なロジック (一般には SQL ステートメント) をコーディングすることによって、パーシスタンスを管理します。

このサンプルには、貯蓄または支出によって残高を調整する機能があります。残高は長期間保存され、エンティティ Bean として表示されます。メソッドを呼び出している間、ロール (役割) を与えるためのユーザの認証と認可が行われます。サンプルは、`deploy.properties` ファイルで定義されているユーザおよびロールにアクセスします。パーシスタンスのために、既定のデータベースが使用されます。

`BalanceBean.java` では、残高を更新するビジネス ロジックが実装されます。公開記述子 (`META-INF/ejb-jar.xml`) を参照して、Balance Bean のプロパティがどのように設定されているかを確認してください。`deploy.properties` ファイルには、ユーザとユーザの各ロールとサーバー名が含まれています。

クライアント側の機能を確認するには、`EjbClient.java` から処理を開始します。クライアントには、プロパティ ファイルや公開記述子は不要であることに注意してください。クライアントは JNDI によって認証されます。

サンプル 2a：既定の認証

サンプル 2a 実行の準備

このサンプルを開始するには、*JRun のルート ディレクトリ*/samples/sample2a/ejbeans ディレクトリに移動し、BalanceBean.java ファイルを確認します。さまざまなメソッドに SQL ステートメントがあります。EJB エンジンは、Bean のライフサイクルの中で一定の回数だけこれらのメソッドを呼び出します。これらのメソッドの中の SQL ステートメントは、Bean のパーシスタンスを実行します。BMP の詳細については、『JRun によるアプリケーションの開発』を参照してください。

次の行を参照してください。これらの各メソッドが示されています。

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("java:comp/env/jdbc/source1");
Connection connection = ds.getConnection();
```

deploy.properties ファイルを開いて、source1 がどのように定義されているかを確認します。次のエントリに注意してください。

```
ejipt.jdbcSources=source1
source1.ejipt.sourceDriverClassName=com.pointbase.jdbc.
jdbcUniversalDriver
source1.ejipt.sourceURL=jdbc:pointbase://embedded/sample
```

ejipt.jdbcSources プロパティには、EJB エンジンで使用できるデータ ソースが定義されています。このプロパティには、カンマで区切られたリストに複数のデータ ソースを含むことができます (たとえば、ejipt.jdbcSources=source1, source2)。jdbcSources の中で指定されている名前を接頭辞として使用することにより、データソース固有のプロパティを指定できます。

source1.ejipt.sourceURL プロパティは、URL を通してデータベースを識別するため、標準 Java JDBC 規則を使用しています。この例では、PointBase を使用して sample という名前のデータベースに接続します。

これ以外のデータベース ドライバを使用する場合は、それを反映するようにプロパティを変更する必要があります。特に、Oracle ドライバを使用している場合は、必ず source1.ejipt.sourceURL に @host を設定してください。

サンプルの deploy.properties ファイルにも、コメント化した JDBC-ODBC ブリッジのプロパティが含まれています。

このサンプルでは、サンプル データベースにあらかじめ定義されているアカウント表を使用します。これには次の列があります。

列名	列のデータ タイプ
id (キー)	INTEGER
value	INTEGER

サンプル 2a の使用

デモを開始するには、コマンド プロンプトからシェルを開きます。

JRun 3.1 にインストールされている PointBase データベース以外のデータベースを使用している場合は、次のコマンドを入力して JDBC ドライバへのパスを定義します。

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

JRUN_HOME は export コマンドを使用して設定し、/jrun/sample2a ディレクトリに移動します。次のコマンドを入力します。

```
bash$ make jars
bash$ make deploy
bash$ make standalone
```

次に、別のコマンド プロンプト ウィンドウでクライアントを起動し、JRUN_HOME を設定して次のコマンドを入力します。

```
bash$ make run
```

サーバーとクライアント の両方が起動し ていれば、サンプルを実行できます。クライアントの [Server] テキスト フィールドにホスト名を入力します。ホスト名は、deploy.properties ファイルの ejipt.classServer.host プロパティに設定した値と同じです。次に、ユーザに「server1」、パスワードに「pass」と入力します。[Login] ボタンをクリックします。

[Amount]フィールドと [Repeat]フィールド がある新しい画面が表示されます。[Amount] フィールドに値を入力し、[Repeat] フィールドに反復回数を入力します。[Save] ボタンをクリックします。サーバー ウィンドウの残高が変更されます。貯蓄者としてログインしており、支出できないので、[Spend] ボタンをクリックします。

次に [Logout] ボタンをクリックし、「spender1」および「pass」と入力して再度ログインします。今回は支出者になったので、貯蓄できません。「chief」および「pass」と入力してログインすると、貯蓄も支出も許可されます。/jrun/samples/sample2a/deploy.properties ファイルでは、ユーザおよびロールを定義する次のエントリを参照できます。

```
ejipt.users=spender1:pass;spender2:pass;saver1:pass;saver2:pass;
        chief:pass
ejipt.roles=spender:spender1,spender2,chief;saver:saver1,saver2,chief
```

ユーザ名、パスワード、およびロールを変更できます。変更したら、次のコマンドを実行して変更内容が反映されていることを確認してください。

```
bash$ make deploy
bash$ make standalone
bash$ make run
```

別のコマンド プロンプト ウィンドウを開き、次の 4 つのコマンドを入力して、クライアントを再起動します。*JRun のルート ディレクトリ*は、実際の JRun インストール ディレクトリに置き換えてください。

```
C:¥> bash
bash$ export JRUN_HOME=JRun のルート ディレクトリ
```

```
bash$ cd JRun のルート ディレクトリ/samples/sample2a
bash$ make run
```

サーバーを停止してから再起動すると、残高がそのまま保持されていることがわかります。JRun では、パーシスタンス オブジェクトにリレーショナル データベースを使用します。

複数のクライアント

追加のクライアントを作成するには、別のコマンド プロンプト ウィンドウを開き、次の 4 つのコマンドを入力します。*JRun のルート ディレクトリ*は、実際の JRun インストール ディレクトリに置き換えてください。

```
C:¥> bash
bash$ export JRUN_HOME=JRun のルート ディレクトリ
bash$ cd JRun のルート ディレクトリ/samples/sample2a
bash$ make run
```

動的な Bean のロード

動的に Bean がロードされるように、サンプル 2a の動作を変更できます。変更するには、`/jrun/samples/sample2a/ejbeans/BalanceBean.java` ファイルを開き、ログ メッセージにアスタリスクを付けるなどの変更を行います。

```
public void save(final int value)
        throws RemoteException {
    _value += value;
    // アスタリスクを付けます。
    ResourceManager.getLogger().logMessage("***saving, balance is:" +
        _value);
}

public void spend(final int value)
        throws RemoteException {
    _value -= value;
    // アスタリスクを付けます。
    ResourceManager.getLogger().logMessage("***spending, balance is:" +
        _value);
}
```

`BalanceBean.java` をコンパイルしたら、新しいコマンド ウィンドウを開いて、`JRUN_HOME` の設定、サンプル 2a ディレクトリへの変更、および次のコマンドの入力を行って、コンパイルした BalanceBean.java を `classes` ディレクトリに配置します。

```
bash$ make classes
```

サーバー ウィンドウに戻り、次のコマンドを入力します。

```
> load
```

最後に、クライアントで `save` 要求と `spend` 要求を再度発行します。EJB エンジンが変更された Bean を使用していることを示す、変更済みのメッセージが表示されます。

サンプル 2b : カスタム認証

このサンプルでは、カスタム Bean を使用してユーザ ID の認証を行います。`LoginSessionBean`、`UserBean`、および `RoleBean` が `ejbeans` ディレクトリに追加されています。これらを参照して、認証と承認がどのように行われるのかを確認してください。ここで、`deploy.properties` ファイルにユーザ ID とパスワードが存在していないことに注意してください。

`UserBean` および `RoleBean` を変更することによって、ユーザ名、パスワード、および役割を変更できます。実行時システムにユーザおよびロールを定義するための `UserBean` および `RoleBean` `setEntityContext` メソッドによる、`UserManager` メソッドの呼び出し方法に注意してください。`LoginSessionBean`、`UserBean`、および `RoleBean` の変更および拡張を行うことによって、データベースまたはディレクトリサービスの規則を利用して、セキュリティ サービスと認証サービスを提供できます。

JRun 3.1 統合認証システムの詳細については、『JRun Version 3.1 機能および移行ガイド』を参照してください。

サンプルを開始するには、次のコマンドを入力します。*JRun のルート ディレクトリ* は、実際の JRun インストール ディレクトリに置き換えてください。

```
bash$ export JRUN_HOME=JRun のルート ディレクトリ
bash$ cd /jrun/samples/sample2b
bash$ make jars
bash$ make deploy
bash$ make standalone
```

別のコマンド プロンプト ウィンドウで、クライアントを起動します。

```
bash$ make run
```

クライアント画面が表示されたら、サンプル 2a で説明したようにログインします。データベース上の残高がコンソールに表示されている残高と同じであることがわかります。

第 8 章

コンテナ管理パーシスタンス

目次

メモ

本書のこの章以降のサンプルでは bash シェルを使用しています。Windows ユーザは、make コマンドを makew コマンドに置き換えて使用してください。標準 DOS ウィンドウを実行する Windows ユーザは、export の代わりに「set」と入力し、フォワードスラッシュの代わりに円記号を使用してください。

概要

サンプル 3 では、リレーショナル データベースを使用したコンテナ管理パーシスタンス (CMP) について説明します。EJB エンジンは CMP によって、Bean プロパティ ファイルまたは公開記述子に定義されているデータ ソースおよび SQL ステートメントを使用して、エンティティ Bean パーシスタンスを管理します。

このサンプルの機能はサンプル 2 と同じです。

Sample 3a：既定の認証

このサンプルでは、サンプル 2a と同じデータ テーブルを使用します。

JRun のルート ディレクトリ`/samples/sample3a/` ディレクトリにある `deploy.properties` ファイルを確認します。JDBC ソースを定義するプロパティがあることに注意してください。

`ejipt.logSQLRequests=true` と設定すると、すべての SQL 呼び出しがログ ウィンドウに表示されます。この機能はデバッグには便利ですが、実際の運用環境では無効に設定しておく必要があります。

JRun のルート ディレクトリ`/samples/sample3a/META-INF` ディレクトリの `ejb-jar.xml` ファイルを参照します。このファイルには CMP の仕様が含まれています。`_id` および `_value` の `cmp-field` 要素に注意してください。これらは、`_id` および `_value` フィールドをコンテナで管理するように EJB エンジンに指示します。また、`persistance-type` 要素が `Container` に設定されることにも注意してください。これによって、コンテナによって管理されるフィールドがあることを EJB エンジンに示します。

`env-entry` 要素にある `ejipt.*SQL*` 仕様では、EJB エンジンが Bean 内にデータを格納する方法と Bean からデータを取得する方法を定義します。このサンプルでは、`findByPrimaryKey` (`ejipt.findByPrimaryKeySQL`)、`create` (`ejipt.createSQL`)、`load` (`ejipt.loadSQL`)、`remove` (`ejipt.removeSQL`)、および `store` (`ejipt.storeSQL`) のプロパティがあります。これらの仕様は、Bean インスタンスの状態を適切に管理するためにコンテナによって使用されます。

メモ

JRun のルート ディレクトリ`/samples` ディレクトリには、ユーザの DBMS に必要なエンティティの定義に使用できるファイルが含まれています。`sqlserver.sql` ファイルを使用して、SQL サーバーのエンティティを定義し、`oracle.sql` ファイルを使用して、Oracle のエンティティを定義します。

SQL プロパティの詳細な説明は、『JRun によるアプリケーションの開発』のコンテナ管理パーシスタンスの解説にあります。*JRun のルート ディレクトリ*`/samples/sample3a/ejbeans` の `BalanceBean.java` を確認します。対応するサンプル 2a の `BalanceBean.java` ファイルほどコードは多くありません。特に、`ejbLoad` メソッドおよび `ejbStore` メソッドからすべての SQL 関連の参照が削除されています。

ここで、次のコマンドを入力してサンプルを実行します。*JRun のルート ディレクトリ*は、正しいディレクトリに置き換えてください。

```
bash$ export JRUN_HOME=JRun のルート ディレクトリ
bash$ cd JRun のルート ディレクトリ/samples/sample3a
```

JRun サンプルとともにインストールされている PointBase ドライバ以外の JDBC ドライバを使用する場合は、次のコマンドを入力します。ドライバへの正しいパスを入力してください。

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

次のコマンドを入力します。

```
bash$ make jars
bash$ make deploy
bash$ make standalone
```

次に、2 番目のコマンド プロンプト ウィンドウでクライアントを起動します。*JRun のルート ディレクトリ*は、実際の JRun インスト ール ディレクトリ に置き換えてください。

```
bash$ export JRUN_HOME=JRun のルート ディレクトリ
bash$ cd JRun のルート ディレクトリ/samples/sample3a
bash$ make run
```

サンプルを使用すると、セッション間で値が維持されていることがわかります。

第 9 章

トランザクション

目次

メモ

本書のこの章以降のサンプルでは `bash` シェルを使用しています。Windows ユーザは、`make` コマンドを `makew` コマンドに置き換えて使用してください。標準 DOS ウィンドウを実行する Windows ユーザは、`export` の代わりに「`set`」と入力し、フォワードスラッシュの代わりに円記号を使用してください。

サンプル 4b：分散型トランザクションと CMP

サンプル 4b は、server1 と server2 を使用した分散型 2 フェーズ コミット トランザクション管理の使用法を示しています。また、このサンプルは、EjbClient.java および BalanceBean.java に含まれている save および spend メソッドの実装で示されているとおり、クライアント区分トランザクションとコンテナ管理トランザクションの両方を使用します。

メモ

このサンプルはリレーショナル データベースを使用しますが、独創的な動作はしません (スキーマについては、第 7 章の「サンプル 2」セクションを参照してください)。サンプル 4b を実行するには、複数の接続をサポートしているデータベースとドライバを使用する必要があります。特定のデータ ソース フォーマットに従って、使用しているデータソース情報を deploy.properties ファイルに追加します。

最初に META-INF/ejb-jar.xml ファイルを開き、次の env-entry 要素を調べます。

- save.ejb.transactionAttribute が mandatory に設定されていること。これは、save メソッドが呼び出されたときにトランザクションが実行されていなければならないことを示します。このサンプルでは、EjbClient.java プログラムの save メソッドがトランザクションを管理します。
- spend.ejb.transactionAttribute が required に設定されていること。これは、spend メソッドが呼び出されたときにトランザクションがないと、EJB エンジンがトランザクションを開始することを示します。

JRun のルート ディレクトリ/samples/sample4b ディレクトリにある deploy2.properties ファイルを調べます。deploy2.properties ファイルは、server2 により使用されます。次のエントリに注意してください。

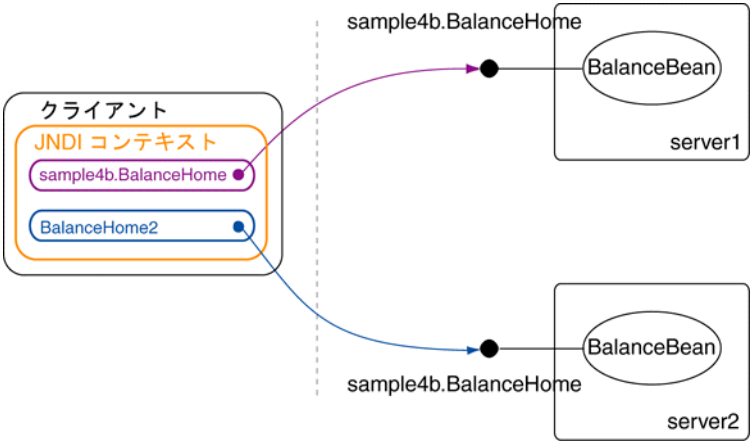
```
sample4b.BalanceHome.maxValue=1000
sample4b.BalanceHome.minValue=-1000
```

これらの 2 つのプロパティは、残高の有効範囲を指定します。server2 における最小値は -1000、最大値は 1000 です。最小値または最大値に達すると、例外となります。Server2 の BalanceBean は、残高をプロパティ ファイルに設定されている値と照合してチェックします。server2 で制限を超えると例外となり、server1 でもトランザクションがロールバックされます。

詳細については、『JRun によるアプリケーションの開発』を参照してください。

JNDI コンテキスト のカスタマイズ

まず、呼び出しをより単純に、また理解しやすくするために JNDI コンテキストをカスタマイズします。*JRun のルート ディレクトリ*/samples/sample4b/client ディレクトリに移動して EjbClient.java ファイルを開きます。login メソッドにより、クライアントがポート 2323 (server1) でリスニングしているサーバーへの context 参照を設定した場所が確認できます。次に、クライアントは、ポート 2324 (server2) で受信しているサーバーに対する context 参照を設定し、sample4b.BalanceHome を BalanceHome2 に結合します。次の図に示すように、これによって "server1" の sample4b.BalanceHome と "server2" の sample4b.BalanceHome を区別します。



クライアント 区分ト ランザクショ ン

EjbClient.java の save メソッド に注目してください。server1 および server2 の残高は save メソッド によって更新されます。save は 2 つの Balance インスタンスを作成します。これらは各サーバーにそれぞれ関連付けられます。transaction.begin を実行します。これは公開記述子の save.ejb.transactionAttribute 設定に必要です。

次にEjbClient.saveメソッドは、Balanceの両方のインスタンスに対して save(amount) を呼び出します。いずれかの呼び出しで例外が発生した場合、トランザクションがロールバックされ、両方のサーバーの残高が更新されません。例外が発生しない場合は、transaction.commit が呼び出されます。そのため、server2 で残高が最小値または最大値に達すると、transaction.rollback が呼び出されて server1 と server2 の同期を取ります。

暗黙トランザクション

EjbClient.java の spend メソッドを調べます。EjbClient.java の save メソッドはサーバーごとに BalanceBean.save を呼び出したのに対し、EjbClient.java の spend メソッドは BalanceBean.spend を一度だけ呼び出します。BalanceBean.spend は、登録されている別の BalanceBean を検索するためです。見つかると、BalanceBean.spend メソッドが呼び出されます。

次は、*JRun のルート ディレクトリ*/samples/sample4b/ejbeans/BalanceBean.java ファイルの spend メソッドに注目してください。spend メソッドは自らの残高を更新しますが、次のコードの抜粋のとおり、server2 の残高も更新します。

```
if (_balance2 != null)
{
   _balance2.spend(value);
}
```

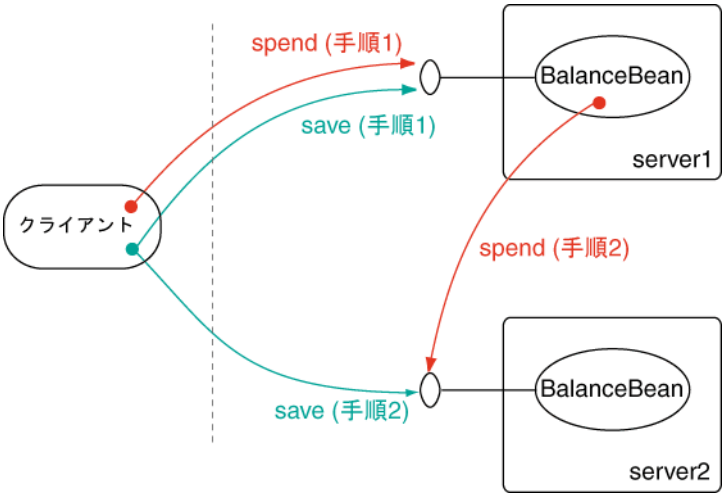
server1 がどのように server2 を認識するかを理解するには、BalanceBean.connect メソッドに注目します。次のコードが表示されます。

```
...
try
  {
    Properties environment = _context.getEnvironment();
    String host =
      environment.getProperty(EjiptProperties.CLASS_SERVER_HOST);
    int port =
      Integer.parseInt(environment.getProperty("balance2Port"));
    Properties properties = new Properties();
    properties.setProperty(Context.INITIAL_CONTEXT_FACTORY,
      "allaire.ejipt.ContextFactory");
    properties.setProperty(Context.PROVIDER_URL, "ejipt://" + host +
      ":" + port);
    BalanceHome home =
      (BalanceHome)(new InitialContext(properties)).lookup
        ("sample4b.BalanceHome");
    _balance2 = createBalance(home, 123);
    }
    catch (NumberFormatException format)
    {
      // ポート情報なし、接続は不要です。
    }
    catch (Exception exception)
    {
      ResourceManager.getLogger().logException
        ("Failed to contact other server", exception);
    }
...
```

このコードは、関連する deploy.properties ファイルで、balance2Port という名前のプロパティをチェックします。*JRun のルート ディレクトリ*/samples/sample4b にある deploy.properties ファイルには、次のエントリが含まれています。

```
sample4b.BalanceHome.balance2Port=2324
```

コードで createBalance を呼び出します。これによって server2 のリモート インターフェイスへの参照が返されます。BalanceBean.getBalance2 メソッドは、ポート 2324 で受信しているサーバー (以前 server2 として参照された) の BalanceHome に対する参照を単に取得します。



複数のサーバー インスタンス

make ファイルは、ejipt.ejbDirectory プロパティを使用して、第 2 のサーバー インスタンスの /deploy および /runtime ディレクトリを指定します。

サンプルを実行するには、コマンド プロンプトを開いて RMID のシェルを起動します (このサンプルはフェイルセーフ モード で実行するので RMID が必要です)。環境変数を設定し、次のコマンドを入力して RMID を開始します。

```
bash$ make rmid
```

別のコマンド プロンプトとシェルを開きます。次のコマンドを入力します。

```
bash$ make jars
bash$ make deploy2
bash$ make start2.
```

make deploy2 および make start2 コマンドは、両方のサーバーを展開または起動します。次に make run を実行してクライアントを起動します。サンプルが終了したら、make stop2 を実行して両方のサーバーを停止してください。

第 10 章

オブジェクト管理

目次

メモ

本書のこの章以降のサンプルでは `bash` シェルを使用しています。Windows ユーザは、`make` コマンドを `makew` コマンドに置き換えて使用してください。標準 DOS ウィンドウを実行する Windows ユーザは、`export` の代わりに「`set`」と入力し、フォワードスラッシュの代わりに円記号を使用してください。

概要

サンプル 5 の焦点は、サーバーの拡張性、カスタマイズ可能な状況、大量の EJB オブジェクトの処理です。

メモ
お使いの JDK、HotSpot によってパフォーマンス特性やガーベッジ コレクション (GC) 特性が異なるため、次のサンプルの結果も異なります。

サンプル 5a : ダイナミック オブジェクト リリース

サンプル 5a は、競売の実例を基にして、エンティティ オブジェクトのダイナミック リリースと全体のリソース管理を示します。普通の競売と同じように、このサンプルにも商品に対して入札を行う入札者が登場します。

商品は常に最良の入札と関連付けられます。最低 4 つの入札があった後は、商品は最良 (最高) の入札価格で売却できるようになります。入札者は最良の取引、つまり最安値で購入した商品の推移を見ます。しかし、商品選択や金額はランダムに生成されるため、すべての入札者が商品の購入に成功するという保証はありません。

サンプルを開始すると、サーバーにより 10,000 人の入札者と 10,000 個の商品が作成されます。`META-INF/ejb-jar.xml` ファイル内の次の `env-entry` 要素を変更することにより入札者数と製品数を簡単にカスタマイズできます。

```
<env-entry>
  <env-entry-name>ejipt.env_entries</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>numProducts,numBidders</env-entry-value>
</env-entry>
```

クライアントは、商品に対してランダム入札の生成を開始します。入札のバッチが生成されると、それらのバッチはサーバーに送信されて処理されます。

バッチ内の入札は管理者によって処理されます。管理者はバッチごとにバッチ内の各入札を関連商品に適用して、全商品の中から十分な入札数 (4 つ以上) があり売却可能な商品を判別します。入札数が 4 つ以上ある商品は売却されます。

このロジックを実行するコードを表示するには、次のファイルをテキスト エディタで開きます。

- *JRun のルート ディレクトリ*`/samples/sample5a/ejbeans/ManagerBean.java`
- *JRun のルート ディレクトリ*`/samples/sample5a/ejbeans/BidBean.java`
- *JRun のルート ディレクトリ*`/samples/sample5a/ejbeans/BidderBean.java`
- *JRun のルート ディレクトリ*`/samples/sample5a/ejbeans/ProductBean.java`

サンプルを開始します。また、deploy.properties ファイルで必要なホスト情報を変更します。次のコマンドを入力します。*JRun のルート ディレクトリ*および*ホスト名*の部分は環境に合わせて置き換えてください。*JRun のルート ディレクトリ*は、JRun インストール ディレクトリに置き換えてください。

```
bash$ export JRUN_HOME=JRun のルート ディレクトリ
bash$ cd JRun のルート ディレクトリ/samples/sample5a
```

次のコマンドを実行して Bean を公開し、サーバーを起動します。

```
bash$ make jars
bash$ make deploy
bash$ make standalone
```

次に、別のコマンド プロンプト ウィンドウでクライアントを起動します。

```
bash$ make go host=ホスト名 size=1000
```

パラメータ size=1000 は、バッチ サイズ、つまり 1 つのバッチで処理する入札数を示します。このパラメータにより、特定の競売に関してバッチを処理する頻度を変更できます。

10,000 の入札者と 10,000 の商品が関係したサンプルを実行すると、サーバーが処理する入札の数は約 50,000、処理するエンティティ オブジェクトの合計数は約 70,000 になります。すべての商品が売却されると、管理者はすべての入札者を閲覧して、最良の入札の時刻を判別します。最良の入札がない場合は、購入がない入札者のカウントが 1 つ増分されます。

参照解除された EJB オブジェクト (Bid) は、この実行中、継続的にガーベッジ コレクションに入れられます。アクティブとして残っている Bid は、購入済みの商品に関して入札者から参照されている Bid であると見なされます。サンプルのレポート作成段階では、これらのアクティブな Bid の再ロードを省けるため、必要な ejbLoads() の数を最小にすることができます。

サンプルでは持続性を持たせるためのデータ ベースを使用していません。CMP または BMP のいずれかを使用して持続性を持たせるのは簡単なことです。Bid はかなり頻繁に生成されるため、Bid テーブルのインデックスを設定しないことをお勧めします。このような制限があるため、ejbLoads() を最小に保つことがより重要となります。

サンプルは、クライアントを再起動するようには設計されていません。サンプルを再実行する場合は、サーバーを再起動して、クリーンな環境を整える必要があります。

サンプル 5b : RMI ソケットのカスタマイズ

サンプル 5b は、ServerSocket にカスタマイズ済みの backlog パラメータを指定するという、RMI ソケットの標準的なカスタマイズ方法を示しています。このサンプルは、カスタム ストリームおよび SSL ソケットを提供するように簡単に拡張できます。サードパーティの標準カスタマイズや製品も使用できます。

サンプルを開始する前に deploy.properties ファイルで必要なホスト情報を変更します。*JRun のルート ディレクトリ*は、JRun インストール ディレクトリに置き換えてください。

```
bash$ export JRUN_HOME=JRun のルート ディレクトリ
bash$ cd JRun のルート ディレクトリ/samples/sample5b
```

次のコマンドを入力します。

```
bash$ make jars
bash$ make deploy
bash$ make standalone
```

次に、別のコマンド プロンプト ウィンドウでクライアントを起動します。

```
bash$ make go host=ホスト名 count=10
```

ServerSockets に対する backlog パラメータの既定値は 50 です。キューが満杯になったときに接続指示が出されると、その接続は拒否されます。count=10 は、サーバーへの同時接続数を指定します。

backlog キューが受け入れ可能な同時接続数を超えてサンプルを再実行すると、エラーが発生します。

```
bash$ make go host=ホスト名 count=150
```

このエラーが発生しないようにするには、META-INF/ejb-jar.xml ファイルの次の env-entry 要素を削除します。

```
<env-entry>
<env-entry-name>ejipt.homeSocketFactory</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-value>ejbeans.SocketFactory</env-entry-value>
</env-entry>
<env-entry>
<env-entry-name>ejipt.objectSocketFactory</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-value>ejbeans.SocketFactory</env-entry-value>
</env-entry>
```

次のコマンドを入力します。

```
bash$ make jars
bash$ make deploy
bash$ make standalone
```

別のプロンプト ウィンドウでクライアントを起動します。

```
bash$ make go host=ホスト名 count=350
```

backlog キューが 350 個の同時接続を処理するように拡張されます。現実的には、350 個もの接続要求がサーバーに対して同時に出されるという事態はあまり起きませんが、活動がピークに達して接続が拒否されるような場合は、この種のカスタマイズが有効です。

サンプル 5c : 大容量の一覧表

このサンプルは、Bid の大容量 `Enumeration` をクライアントに返す方法を示します。クライアントは入札を繰り返すことによって金額を取得します。このサンプルは、クライアントが参照を解除したオブジェクトがどのようにガーベッジ コレクションに入れられるかを示しています。

`BidBean.java` 内の `ejbFindAll` メソッドと `KeyEnumerator` 内部クラスを確認します。また、`EjbClient.java` クラスを調べると、EJB の `FindAll` メソッドを呼び出して Bid の `Enumeration` を返すコードが `run` メソッドにどのように含まれているかがわかります。

サンプルを開始する前に、`deploy.properties` ファイルで必要なホスト情報を変更します。*JRun のルート ディレクトリ*は実際に使用している JRun インストール ディレクトリに、*ホスト名*は適切なホスト名に置き換えます。

```
bash$ export JRUN_HOME=JRun のルート ディレクトリ
bash$ cd JRun のルート ディレクトリ/samples/sample5c
```

次のコマンドを入力します。

```
bash$ make jars
bash$ make deploy
bash$ make standalone
```

別のプロンプト ウィンドウでクライアントを起動します。

```
bash$ make go host=ホスト名 size=1000
```

パラメータ `size=1000` は、サーバーからクライアントに返される `Enumeration` のサイズを指定しています。

第 11 章

メッセージ

目次

メモ

本書のこの章以降のサンプルでは `bash` シェルを使用しています。Windows ユーザは、`make` コマンドを `makew` コマンドに置き換えて使用してください。標準 DOS ウィンドウを実行する Windows ユーザは、`export` の代わりに「`set`」と入力し、フォワードスラッシュの代わりに円記号を使用してください。

概要

サンプル 6 は、Java Message Service (JMS) の機能を紹介しています。ポイントツーポイント (キュー) およびパブリッシュ / サブスクライブ (トピック) メッセージ機能について説明します。

メモ
Multicast Time-To-Live プロパティ (jms.multicast.ttl) はゼロに設定されています。これにより、UDP パケットはリモート クライアントに転送されません。リモート クライアントに対して次のサンプルを使用するには、deploy.properties ファイルの jms.multicast.ttl、jms.multicast.port、および jms.multicast.groupAddress プロパティを、実際の環境に合わせて設定してください。

これらのサンプルでは、instance.store を使用してメッセージを持続させます。各サンプルを実行する前に /runtime ディレクトリから instance.store を削除する必要があります。

JMS の詳細については、『JRun によるアプリケーション開発』を参照してください。

サンプル 6a：ポイントツーポイント

サンプル 6a では、ポイントツーポイントまたはキューベースのメッセージ サービスを使用します。サンプルでは、メッセージがキューに書き込まれる同期メッセージ機能が示されます。この後、キューからメッセージがポーリングされます。また、メッセージを自動的に受信するためにリスナが登録する非同期メッセージも扱われます。

/sample6a/deploy.properties ファイルを確認することから始めます。メッセージを有効にするには、ejipt.enableMessaging プロパティを true に設定します。この設定は、メッセージ関連の Bean を読み込むように EJB エンジンに通知します。このプロパティが true に設定されていないと、メッセージ機能は働きません。

次のプロパティ設定にも注意してください。

```
default.MessageQueueHome.ejb.enterpriseBeanClassName=ejbeans.QueueBean
```

この設定は、default.MessageQueueBean ではなく ejbeans.QueueBean を使用してメッセージを持続させるように JRun に指示します。/sample6a/ejbeans/QueueBean.java ファイルを見ると、onAdding および onRemoved メソッドが実装されていることがわかります。onAdding メソッドはメッセージをキューに入れる直前に呼び出されますが、onRemoved メソッドはメッセージをキューから削除した直後に呼び出されます。この方法によって、サンプルは JRun ログ ファイルにエンティティを書き込めるようになります。ユーザ アプリケーションは、必要に応じて類似した機能を実装するために MessageQueueBean を拡張できますが、通常は必要ありません。

クライアント側アプリケーションには、Sender (送信元) と Receiver (受信側) の 2 つがあります。/sample6a/client/Sender.java ファイルを開いて Sender を確認します。Sender は、次の表が示すように 3 つの引数を受け入れます。

パラメータ	値
host	サーバーのホスト名、あるいはサーバーと Sender がローカルで稼動している場合は localhost を指定します。
queue name	メッセージの送信先のキューを識別します。
mode	Sender が一連のメッセージを自動生成するかどうかを指定します。 値は、manual または auto です。
name	送信元を識別するための名前を指定します。

Sender.java ファイルでは、Sender では最初に QueueConnectionFactory の参照を取得する必要があります。これを行うと QueueConnection を取得できます。接続が確立されると、QueueSession を作成してメッセージの送信を開始できます。さらに、Sender は、queue name パラメータを使用して実際のメッセージ キューを作成します。

実際のメッセージを生成して送信するには、Sender はテキストを指定して Message.setText を呼び出し、次にメッセージ、送信モード、優先順位、および有効期限を指定して QueueSender.send を呼び出します。

/sample6a/client/Receiver.java ファイルを開いて、メッセージを取得する方法について調べます。Receiver は、次の表が示すとおり 3 つの引数を受け入れます。

パラメータ	値
host	サーバーのホスト名、あるいはサーバーと Receiver がローカルで稼動している場合は localhost を指定します。
queue name	メッセージを取得する元のキューを識別します。
mode	受信側がキューからメッセージを手動で取得するか (同期)、あるいはメッセージを自動受信するために受信側をリスナとして登録するか (非同期) を指定します。 値は、manual または auto です。

Receiver も QueueConnectionFactory に対する参照を取得し、その参照を使用して QueueConnection を取得し、QueueSession を作成する必要があります。ここでは、Sender ではなく QueueReceiver が作成されます。auto モードの場合、Receiver はキューの listener として登録されます (非同期)。それ以外の場合は、Enter キーを押すたびにキューがチェックされます (同期)。

サンプルを実行するには、次のコマンドを入力してサーバーを起動します。「No bean found in jar(s)」というメッセージは無視します。

```
bash$ make jars
bash$ make deploy
bash$ make standalone
```

次に新しいシェルを開き、次のコマンドを入力して Sender を開始します。

```
bash$ make sender host=localhost queue=cat mode=manual name=fluff
```

次のような出力が表示されます。

```
Type message to send or 'quit' to exit, then press <ENTER>
```

任意のテキストを入力し、Enter キーを押すと、次のように表示されます。

```
Sending:[delivery:non-persistent, priority:default, from:fluff]
Content:<入力したテキスト>
Type message to send or 'quit' to exit, then press <ENTER>
```

メッセージを送信する場合に、メッセージに接頭辞 :dp を付けると、DeliveryMode が持続するように指定できます。

```
:dp メッセージのテキスト
```

この場合、サーバーがシャットダウンしてもメッセージは持続します。メッセージは、instance.store を使用して QueueBean により持続されます。モードが持続しているかどうかをテストするには、メッセージを送信して、サーバーを停止してから起動し、次に Receiver を起動します。Sender ではメッセージの有効期限が 5 分に設定されているため、送信する前にメッセージが期限切れになることがあります。

メッセージに接頭辞 :pX を付けることによって優先順位を設定することもできます。ここで、X は 0 ～ 9 で、9 が最高の優先順位を表します。優先順位が高いメッセージは、優先順位が低いメッセージより先に送信されます。

```
:p9 メッセージのテキスト
```

試しに優先順位の異なるいくつかのメッセージを送信し、連続して受信してみてください。優先順位が高い順にメッセージを受信するはずです。

mode=auto にして Sender を起動すると、一連のメッセージが自動的に生成されて送信されます。

```
bash$ make sender host=localhost queue=cat mode=auto name=fluff
```

ここで新しいシェルを開き、次のコマンドを入力して Receiver を起動します。

```
bash$ make receiver host=localhost queue=cat mode=manual
```

次のような出力が表示されます。

```
Press <ENTER> to receive message or enter 'quit' to exit
```

Enter キーを押してメッセージを取得します。

```
Received:[delivery:non-persistent, priority:-1, from:fluff]
Content:<your text here>
Press <ENTER> to receive message or enter 'quit' to exit
```

Receiver に対してコマンド ラインに「:wXXX」と入力できます。XXX は、受信側が新しいメッセージを受信するのに待機する秒数を表します。Receiver は、指定された秒数が経過するか、またはメッセージを受信するまで待機します。

mode=auto にして Receiver を起動すると、非同期メッセージを受信するのでポーリングは不要です。その場合、Receiver はキューの listener として登録されていなければなりません。Receiver をリスナとして登録した時点でキューにすでに入っているメッセージがあったとしても、それらのメッセージは Receiver に送信されません。メッセージが生成されたときに Receiver が listener として登録されていなかったためです。

サンプル 6b：パブリッシュ、サブスクライブ

サンプル 6b では、パブリッシュ / サブスクライブ (トピックベース) のメッセージ サポートを使用します。トピックを使用すると、登録されているすべての Subscribers (サブスクライバ) は、期限切れでないメッセージを受信します。サンプル 6a で説明した接頭辞と同じ接頭辞を使用して、メッセージの優先順位や持続性を指定できます。

/sample6b/client/Publisher.java ファイルの run メソッドを調べて、サンプルがトピックにメッセージを発行したときの publish メソッドの使用方法を理解します。/sample6b/client/Subscriber.java ファイルの onMessage メソッドを調べて、サンプルが受信メッセージを表示する方法を理解します。

サンプルを実行するには、次のコマンドを入力してサーバーを起動してください。「No bean found in jar(s)」というメッセージは無視します。

```
bash$ make jars
bash$ make deploy
bash$ make standalone
```

Publisher (パブリッシャ) がメッセージの送信を始める前にリスナとして登録できるように、ここでいくつかの Subscribers を起動する必要があります。登録されたリスナは、トピックに対して新しいメッセージだけを受信します。トピックにすでに含まれているメッセージは、新しく登録されたリスナには送信されません。

新しいシェルを開き、複数の Subscribers に対して次のコマンドを入力します。

```
bash$ make subscriber host=localhost topic=dog mode=auto
```

次に新しいシェルを開き、次のコマンドを入力して Publisher を開始します。

```
bash$ make publisher host=localhost topic=dog name=spot
```

メッセージを送信すると、Subscribers によって自動的にメッセージが受信されます。オートモードの場合、Subscribers はサブスクライバがアクティブなときに発行されたメッセージだけを受信します。

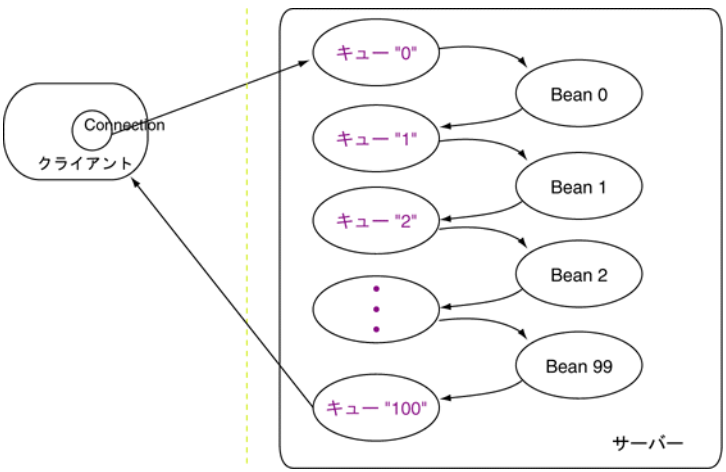
Subscriber をマニュアル モードで起動した場合は、メッセージ バッファに現在入っている任意のメッセージを取得できます。バッファ サイズを指定するには、jms.messageCapacity プロパティを設定します。

トピックを階層型にすることができます。その様子が `/sample6b/ejbeans/TopicBean.java` ファイルに示されています。`ejbFindSuperTopics` メソッドは、同じ接頭辞を持つほかのトピックを検索します。たとえば、dog というトピックと dog.lab というトピックがある場合、dog は上位トピック、dog.lab はサブトピックになります。そのため、dog のサブスクライバは dog.lab メッセージを受信しますが、dog.lab のサブスクライバは dog.lab メッセージだけを受信します。ただし、dog.lab のサブスクライバは、dog.lab.black のメッセージを受信します。`TopicBean.java` は、実際の環境のトピック規則に合わせて簡単にカスタマイズできます。

サンプル 6c：EJB 統合

サンプル 6c では、JRun の JMS の実装と EJB がどのように関係し、またやり取りするのかを示します。クライアントはメッセージのチェーンが開始する最初のメッセージを生成します。このチェーンはサーバー上の Bean によって消費されます。最後のメッセージは元のクライアントによって消費されます。

この仕組みを見るには、まず `/sample6c/Client.java` ファイルを調べます。コンストラクタでは、接続とセッションが確立すると、クライアントは "1" から "100" の名前が付いた 100 個のキューを作成します。さらに `ListenerBean` エンティティ Bean のインスタンスを 100 個作成します。最後に、クライアントは "100" という名前が付けられたキューのリスナとして自らを登録します。



`Client.run` メソッドでは、メッセージがキュー "0" に送信されます。このメソッドは、`onMessage` メソッドが実行されるまで待機します。`Client` は `MessageListener` インターフェイスを実装し、またそのために `onMessage` メソッドも実装しなければならない点に注意してください。メッセージをキュー "100" に転送するとき、`onMessage` メソッドが呼び出されます。

チェーンがどのように実装されるかについては、Listener を調べます。まず /sample6c/ejbeans/Listener.properties ファイルを開きます。ejipt.maxContexts プロパティが 10 に設定されていることに注意してください。これは、アクティブな Bean インスタンスの数が 10 を超えないことを意味します。このため、サンプルの実行時に、インスタンスが必要に応じて有効または無効な状態になります。持続性を設定するために instance.store を使用します。

次に、ListenerBean.java を確認します。setEntityContext メソッドを使用して、現在の Listener オブジェクトが最初の Listener インスタンスであるかどうかをチェックします。最初のインスタンスであれば、Listener のすべてのインスタンスで同じ接続を使用できるように、メソッドは JNDI コンテキストの接続をバインドします。

Listener オブジェクト (Listener.java) によって MessageListener が拡張されるので、ListenerBean.java で onMessage メソッドを実装を提供する必要があります。このメソッドはメッセージを次のキューに転送するためのものです。

サンプルを実行します。次のコマンドを入力してサーバーを起動します。

```
bash$ make jars
bash$ make deploy
bash$ make standalone
```

次のコマンドを入力してクライアントを起動します。

```
bash$ make go host=localhost
```

次のような出力がサーバーに表示されます。

```
>[object:0] forwarding message...
>[object:1] forwarding message...
>[object:2] forwarding message...
        .
        .
        .
        >[object:99] forwarding message...
        >
```

次の出力がクライアントに表示されます。

```
Received: 100
```

チェーンが完了すると、Client.run メソッドは instance.store から Listener インスタンスを削除し、Receiver、Sender、Session、および Connection を閉じます。

第 12 章

高度な Bean

目次

メモ

本書のこの章以降のサンプルでは `bash` シェルを使用しています。Windows ユーザは、`make` コマンドを `makew` コマンドに置き換えて使用してください。標準 DOS ウィンドウを実行する Windows ユーザは、`export` の代わりに「`set`」と入力し、フォワードスラッシュの代わりに円記号を使用してください。

概要

サンプル 7 は、共同で機能するエンティティ Bean、ステートフル セッション Bean、およびステートレス セッション Bean について説明します。またこのサンプルでは、デッドロック例外の対処方法や自動呼び出し機能についても紹介しています。

サンプル 7 ではコマンド ライン入力を使用してクライアント プロセスを起動します。このクライアント プロセスでは、情報をプロパティおよびコマンド ライン引数により実行時に設定し、コンソール ウィンドウに出力を書き込み、リレーショナル データベースにデータを保持します。このアプリケーションは、実際の分散アプリケーションのフレームワークとして簡単に使用できます。

ビジネス ロジックの機能は実に簡単明瞭です。銀行は特定の顧客に融資します。この顧客は、あらかじめ決められた期間内にローンを返済しなければなりません。期間内にローンが返済されない場合は、ローン返済不履行となります。

銀行は、顧客のローンに利息を付けることで収益を得ます。銀行には、慎重な銀行からリスクを恐れない銀行まで、さまざまなタイプがあります。顧客は銀行に基本資本 (相続財産) を預けており、時間が経つと、この資本に収益としての利息が付きます。収益は現在の現財産の一部です。顧客は一時的に資金を増やすため、銀行から資金を追加融資してもらうことはできますが、いかなる場合でもローンの未払いは一度しか許されません。顧客のタイプも、資金をほとんど借りない慎重な顧客から、リスクを恐れず大金を借りる顧客までさまざまです。顧客には信用の格付けもあります。最初は中立的な格付けから始まり、ローンの返済または不履行のいずれかに応じて調整が行われます。

`Web` エンティティ Bean は世界共通の情報源です。Web では、現在の利率や収益率だけでなく、銀行の一覧も参照できます。顧客は `Web` を使用して銀行を探します。銀行は `Web` を使用して現在の利率を決定します。利率と収益率は公開記述子に設定されています。

`CustomerSession` は、ステートフル セッション Bean で、`Customer` (顧客) エンティティ Bean ごとにセッションを管理し、指定された期間ごとに顧客の資産を更新します。`Loan` (ローン) は、各ローンの発生期間を表すステートフル セッション Bean です。`Loan` インスタンスが返済されたり、不履行になると、有効期限が切れ、削除されます。`Calculator` (計算機) は、特定の `Loan` に対する分割払い額を計算するのに使用するステートレス セッションです。

Bean

次の表は、このサンプルで使用されている Bean の一覧です。

Bean	タイプ
Bank	エンティティ
Customer	エンティティ
Web	エンティティ
Loan	ステートフル セッション
CustomerSession	ステートフル セッション
Calculator	ステートレス セッション

プロセス

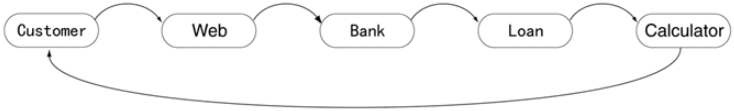
サーバーを起動すると、`deploy.properties` ファイルで定義されている Bank (銀行) の数およびタイプが作成されます。またサーバーは、現在の利率と収益率により `Web` を初期化します。

クライアントを起動すると、ホスト名およびポートを指定する引数、そしてそれぞれの基本資金とともに `Customer` 数が渡されます。`Customer` はランダムに Loan を求め始め、`Web` は `Loan` を提供できる Bank をランダムに選択します。

次に `Customer` が `Bank` に `Loan` を要求すると、`Bank` はその `Customer` に対して融資するかどうかを決定します。決定基準は、`Bank` の資金の有効性と `Customer` の信用格付けに基づいています。`Bank` が `Loan` を行うと、`Customer` は Calculator を使用して分割払い額を決定するための計算を行います。

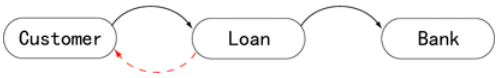
`Loan` の申し込みが認められると、Customer は `Loan` の返済を開始する必要があります。

ローン開始プロセス



Customer は Loan に対して支払いを行います。Loan は支払額を Bank に渡し、Bank の利用可能な資金に追加します。Loan ごとに Customer は 10 回払いで返済するようにスケジュールされています。Loan が不履行になると、Customer の信用格付けは、Loan が削除される前に Loan によって調整されます。

ローン支払いプロセス



デッドロック

サンプル 7 では、デッドロックを起こしやすい状況と、EJB エンジンがデッドロックを検出し、解除する方法について説明しています。Loan は、ある固定期間の間だけ存在します。Customer が期間内に Loan を支払わない場合は、Customer がまだ分割で支払いを行っている場合でも Loan は不履行になります。Loan が期限までにすべて返済されない場合、Loan は Loan.ejbRemove において Customer の信用格付けを格下げします。しかし、Customer によって Loan の支払いが行われると同時に Loan による信用格付けの格下げが実行されると、デッドロック状況が発生します。

サンプルでは、デッドロックを管理する場所が 2 か所あります。

- DeadlockExceptions は LoanBean.payInstallment メソッドの throws 節でチェック済みの例外として宣言されます。これによって、EJB エンジンはデッドロック例外を変更しないで処理を行います。これらの宣言を使用しない場合、例外は java.rmi.RemoteExceptions にラップされます。
- DeadlockExceptions は、CustomerBean.payInstallment メソッドで調べられ、無視されます。結果として、Customer の信用格付けを更新するための呼び出しが最終的に実行され、Loan を不履行にして削除できます。次回 Customer が Loan を支払うときに、NoSuchObjectException が検出されます。ここで、Customer は、Loan の期限が切れていることを知らされ、支払いが停止されます。

サンプル 7b：BMP を使用した複雑な処理

サンプル 7b では、サンプル データベースと Bean 管理パーシスタンスを併用します。

次の表は、サンプル 7b で使用されるスキーマの定義です。

テーブル	フィールド	JDBC の種類	規則
account	id	INTEGER	NOT NULL
	value	INTEGER	NOT NULL
bank	name	VARCHAR (16)	NOT NULL
	type	INTEGER	NOT NULL
	funds	FLOAT	NOT NULL
	loans	FLOAT	NOT NULL
customer	name	VARCHAR (16)	NOT NULL
	password	VARCHAR (16)	NOT NULL
	type	INTEGER	NOT NULL
	rating	INTEGER	NOT NULL DEFAULT 0
	worth	FLOAT	NOT NULL
	handle	VARBINARY (128)	NULL DEFAULT NULL
	installment	FLOAT	NOT NULL

/sample7b/ejbeans ディレクトリにある BankBean.java ファイルを確認します。多くのメソッドに SQL 関連の参照があります。

サンプルを開始します。次のコマンドを入力します。*JRun のルート ディレクトリ*は、JRun インストール ディレクトリに置き換えてください。

```
bash$ export JRUN_HOME=JRun のルート ディレクトリ
bash$ cd JRun のルート ディレクトリ/samples/sample7b
```

既定の PointBase ドライバ以外の JDBC ドライバを使用している場合は、次のコマンドを入力します。ドライバの正しいパスを入力してください。

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

次のコマンドを入力します。

```
bash$ make jars
bash$ make deploy
bash$ make standalone
```

別のコマンド プロンプト ウィンドウで、クライアントを起動します。

```
bash$ make simulate host=ホスト名 first=0 last=10 inheritance=1000
```

このコマンドを実行すると、サーバーによって 0 から 10 までの ID を持った 11 の同時実行 Customer が作成されます。サーバー ポートを 2323 以外 (2324 など) に変更した場合は、次のコマンドを入力します。

```
bash$ make simulate host=ホスト名:2324 first=0 last=10 inheritance=1000
```

もう一度シミュレーションを実行します。別の Customer を入力するか、または inheritance の値を変更します。

サンプルを 100 の Customer で実行すると、1000 のローンと約 10,000 の支払いが発生することになりますが、正確な数は `Loan.properties` 内のローン期間に対する ejb.sessionTimeout 設定によって異なります。1000 の Customer でサンプルを実行します。

別のコマンド ウィンドウを開き、次のコマンドを入力して追加クライアントを実行します。*JRun のルート ディレクトリ*は、実際の JRun インストール ディレクトリに置き換えてください。

```
bash$ export JRUN_HOME=JRun のルート ディレクトリ
bash$ cd JRun のルート ディレクトリ/samples/sample7b
bash$ make simulate host=localhost first=200 last=210
inheritance=1000
```

サンプル 7c：CMP を使用した複雑な処理

サンプル 7c はサンプル 7 で使用されているアプリケーションを利用し、リレーショナル データベースにコンテナ管理パーシスタンスを追加します。

サンプル 7c は、サンプル 7b と同じデータベース スキーマを使用します。

`META-INF/ejb-jar.xml` ファイルを開いて `BankBean` の CMP 設定を調べます。`cmp-field` 要素によって、コンテナが管理するフィールドを指定します。`CustomerBean` にも CMP の指定が含まれています。

`ejb-jar.xml` ファイルの `ejipt.*SQL*` `env-entries` によって、Bean のデータの保存方法と検索方法を定義します。このサンプルには、`load`、`store`、`findByPrimaryKey`、および `findAllBanks` のプロパティがあります。これらのプロパティは、Bean インスタンスの状態を適切に管理する、対応した EJB メソッドによって使用されます。

`/sample7c/ejbeans` ディレクトリにある `BankBean.java` ファイルを確認します。対応するサンプル 7b の `BankBean.java` ほどコードは多くありません。特に、`ejbLoad` メソッドと `ejbStore` メソッドのコードは少なくなっています。

サンプルを開始するには、次のコマンドを入力します。*`JRun` のルート ディレクトリ*および*ホスト名*の部分は環境に合わせて置き換えてください。

```
bash$ export JRUN_HOME=JRun のルート ディレクトリ
bash$ cd JRun のルート ディレクトリ/samples/sample7c
```

JRun に同梱されている PointBase ドライバ以外の JDBC ドライバを使用している場合は、次のコマンドを入力します。ドライバへの正しいパスを入力してください。

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

次のコマンドを入力します。

```
bash$ make jars
bash$ make deploy
bash$ make standalone
```

別のプロンプト ウィンドウでクライアントを起動します。

```
bash$ make simulate host=ホスト名 first=0 last=10 inheritance=1000
```

プリペアド ステートメント

サンプル 7c はプリペアド ステートメントを使用してデータベースにアクセスします。MS SQL Server など標準的な JDBC/ODBC ドライバを使用する一部のデータベースでは、`invalid ResultSet` エラーが発生します。このエラーが発生しないようにするには、次のステートメントを `deploy.properties` ファイルに追加します。

```
ejipt.disableStmtPool=true
```

第 13 章

サーブレットでの EJB の使用

目次

メモ

本書のこの章以降のサンプルでは `bash` シェルを使用しています。Windows ユーザは、`make` コマンドを `makew` コマンドに置き換えて使用してください。標準 DOS ウィンドウを実行する Windows ユーザは、`export` の代わりに「`set`」と入力し、フォワードスラッシュの代わりに円記号を使用してください。

概要

サーブレットは、クライアントがファイアウォールの外側にいるクライアントと通信するための効率的な方法です。このサンプルではサーブレットから Bean にアクセスする方法を示します。

サンプル 9a : サーブレットの呼び出しと EJB

開始する前に、`sample9a/webapp/WEB-INF/classes` ディレクトリにある `.java` ファイルを確認して、EJB と通信するためにサーブレットに追加するコードについて理解してください。

この例を実行するには、次の `make` (または `makew`) オプションを使用します。

- `make jars`　EJB ファイルをコンパイルし、JAR ファイルを生成します。
- `make deploy`　EJB を公開します。
- `make war`　サーブレットをコンパイルし、WAR ファイルを作成します。
- `make wardeploy`　WAR ファイルを公開します。
- `make startup`　default JRun サーバーを起動します。

サンプル 9a を実行するには、コマンド プロンプト ウィンドウを開いて次のコマンドを入力し、環境に合わせて *`JRun` のルート ディレクトリ*を置き換えます。

```
bash$ export JRUN_HOME=JRun のルート ディレクトリ
bash$ cd JRun のルート ディレクトリ/samples/sample9a
```

次のコマンドを入力します。

```
bash$ make jars
bash$ make deploy
bash$ make war
bash$ make wardeploy
bash$ make startup
```

Web ブラウザを開き、ブラウザがクッキーを受け入れるように設定されていることを確認します。ここで、`http://` *ホスト名* `:` *ポート番号* `/sample9a` を指定します。ほかのサンプルで表示される Java アプリケーションに似たログイン画面が表示されるはずです。chief/pass、saver1/pass、または spender1/pass を使用してログインし、トランザクションを開始します。このサンプルは、出力を `default-event.log` ファイルに書き込みます。

サーブレットはクライアントごとにクッキーを 1 つ作成するため、同じマシン上でもう 1 つブラウザを起動した場合は同じ ID が付けられます。

サーバーを停止するには、`CTRL + C` を押します。

サンプル 9b：シングル サインオン

サンプル 9b の Web アプリケーションはフォームベースの認証を使用します。これは、JRun ではアプリケーションに用意されているログイン ページを使用して自動的にユーザが認証されることを意味します (この場合は `login.jsp`)。JRun 3.1 のシングル サインオン機能を使用すると、フォームベースの認証を介してアクセスしたログイン情報を EJB エンジンでも使用できます。サンプル 9b では `LoginServlet` は不要です。

シングル サインオンの詳細については、『JRun Version 3.1 機能および移行ガイド』を参照してください。

開始する前に、`WEB-INF` ディレクトリにある `web.xml` ファイルを確認します。また、`sample9b/webapp/WEB-INF/classes` ディレクトリにある `BalanceServlet.java` ファイルも確認します。これらのファイルをサンプル 9a と比較し、シングル サインオン機能を使用するときのコーディングの違いを理解してください。

この例を実行するには、次の `make` (または `makew`) オプションを使用します。

- `make jars` EJB ファイルをコンパイルし、JAR ファイルを作成します。
- `make deploy` EJB を公開します。
- `make war` サーブレットをコンパイルし、WAR ファイルを作成します。
- `make wardeploy` WAR ファイルを公開します。
- `make startup` default JRun サーバーを起動します。

サンプル 9b を実行するには、コマンド プロンプトを開いて次のコマンドを入力し、環境に合わせて *`JRun_rootdir`* を置き換えます。

```
bash$ export JRUN_HOME=JRun のルート ディレクトリ
bash$ cd JRun のルート ディレクトリ/samples/sample9b
```

まだユーザを追加していない場合は、次のコマンドを発行して `users.properties` ファイルに EJB ユーザを追加します。

```
bash$ make users
```

次のコマンドを入力します。

```
bash$ make jars
bash$ make deploy
bash$ make war
bash$ make wardeploy
bash$ make startup
```

お気に入り のブラウザを開き、ブラウザがクッキーを受け入れるよう に設定されていることを確認してください。ここで、`http://`*ホスト名*`:`*ポート番号*`/sample9b` を指定します。ほかのサンプルで表示される Java アプリケーションに似たログイン画面が表示されるはずです。chief/pass、saver1/pass、または spender1/pass を使用してログインし、トランザクションを開始します。このサンプルは、出力を `default-event.log` ファイルに書き込みます。

サーブレットはクライアントごとにクッキーを 1 つ作成します。そのため、同じマシン上でもう 1 つブラウザを起動すると同じ ID が付けられます。

サーバーを停止するには、CTRL + C を押します。

「makew clean-users」と入力することによりオプションで users.properties ファイルから EJB ユーザを削除できます。

第 14 章

JDK 1.1 クライアント

目次

メモ

本書のこの章以降のサンプルでは `bash` シェルを使用しています。Windows ユーザは、`make` コマンドを `makew` コマンドに置き換えて使用してください。標準 DOS ウィンドウを実行する Windows ユーザは、`export` の代わりに「`set`」と入力し、フォワードスラッシュの代わりに円記号を使用してください。

サンプル 10a：JDK1.1 クライアントの使用

サンプル 10a は、JDK 1.1 と Java 2 クライアントによる EJB エンジンの使用を並行して示します。JDK 1.1 は複数 URL からの RMI クラスのロードをサポートしていないため、JDK 1.1 クライアントを使用する場合は、通常のステップ以外にいくつかの特別なステップがあります。

まず、サンプル 10a 用の `deploy.properties` を見つけ、ホスト名にサーバー名を設定します。`ejipt.isCompatible=true` プロパティに注意してください。このプロパティは、JDK 1.1 クライアント用の RMI スケルトンを生成する EJB エンジンを設定します。

コマンド プロンプトを開いてシェルを開始します。`JRUN_HOME` 環境変数を設定してから、`/sample10a` サブディレクトリに移動して次のコマンドを入力します。

```
bash$ make jars
bash$ make deploy
bash$ make standalone
```

ここでクライアント マシンに接続し、`/sample10a` という名前のディレクトリを作成します。次のファイルをサーバーからクライアント上の `/sample10a` ディレクトリにコピーします。

- *JRun のルート ディレクトリ*`/samples/sample10a/sample10a_client.jar`
- *JRun のルート ディレクトリ*`/servers/default/runtime/ejipt_exports.jar` スタブを含みます。
- *JRun のルート ディレクトリ*`/lib/ejipt_client.jar` JRun 標準拡張
- *JRun のルート ディレクトリ*`/lib/ext/ejb.jar` Java 標準拡張
- *JRun のルート ディレクトリ*`/lib/ext/jta.jar` Java 標準拡張
- *JRun のルート ディレクトリ*`/lib/ext/jndi.jar` Java 標準拡張

クライアントでコマンド プロンプトを開き、作成したばかりの `/sample10a` ディレクトリに移動します。クラスパスを次のように設定してください。

```
> set CLASSPATH=sample10a_client.jar;ejipt_exports.jar;
  ejipt_client.jar; ejb.jar;jta.jar;jndi.jar;C:¥jdk¥lib¥classes.zip
```

次のコマンドを使用してクライアント アプリケーションを起動します。`host` は、必ず `deploy.properties` で指定した名前に設定してください。

```
> java Client1_1 host chief pass
```

saver1 または spender1 ユーザ ID でクライアントを実行すると、追加の認証処理が示されます。クライアントはサーバーに接続し、100 を保存するために `save` メソッドを呼び出し (繰り返し回数 10)、100 を使用するために `save` メソッドを呼び出します (繰り返し回数 10)。`make run` を使用して Java 2 ベース クライアント 用のアプリ ケーションを起動することもできます。

第 15 章

Make ファイル

目次

Make ファイルの使用

この章では、サンプルとともに提供されている Make ファイルについて詳しく説明します。例としてサンプル 2a を使用します。

必要に応じて、make ファイルを使用しないで、コマンド プロンプト ウィンドウにコマンドを直接入力することもできます。コマンドを確認するには、『JRun によるアプリケーションの開発』の「Bean の公開」セクションを参照してください。

Make および makew

JRun には UNIX および Windows 用に個別の make ファイルがあります。UNIX 用の make ファイルは make と呼ばれ、GNU make ユーティリティを使用します。一方、Windows 用 make ファイルの名前は makew で、ファイル形式は BAT です。

メモ
本書で make コマンドを使用する場合、Windows ユーザはそれを makew に置き換える必要があります。

make ファイルについて

このセクションでは、UNIX でサンプルを実行する際に使用する make ファイルの内容を説明します。Windows での (makew の代わりの) make ファイルの実行については、104 ページの「Cygnus ユーザへの注意事項」を参照してください。

Bean のホームおよびリ モート のインターフェイス名、およびその実装は、*JRun のルート ディレクトリ*/samples/sample2a/ejbeans/Makefile に示されている方法で指定する必要があります。すべての Bean ファイルを、次の形式でインクルードしてください。

```
sources = $(addprefix ejbeans/, ¥
RemoteName.java¥
BeanName.java¥
HomeName.java¥
)
```

クライアント関連 Java ファイルを指定します。たとえば、*JRun のルート ディレクトリ*/samples/sample2a/client には、クライアント関連ファイルの名前がすべて記述されています。次の形式でクライアント ファイルを追加します。

```
sources = $(addprefix client/, ¥
ClientUI.java¥
MainPanel.java¥
)
```

JRun のルート ディレクトリ/samples/sample2a/Makefileファイルに説明されているように、Bean のホームおよびリモート インターフェイス クラスも指定する必要があります。次の形式で、Bean のホームおよびリモート インターフェイスをインクルードしてください。

```
ejb_clients = ¥
ejbeans/RemoteName.class¥
ejbeans/HomeName.class
```

makew ファイルについて

このセクションでは、Windows で EJB サンプルを実行する際に使用する makew ファイルの内容を説明します。

JRun のルート ディレクトリ/samples/sample2a/ejbeans/makew.bat ファイルに説明されているように、Bean のホームおよびリモート インターフェイス名、およびその実装を指定する必要があります。すべての Bean ファイルを次の形式でインクルードしてください。

```
@set sources=ejbeans¥Balance.java ejbeans¥BalanceBean.java
        ejbeans¥BalanceHome.java
```

JRun のルート ディレクトリ/samples/sample2a/client/makew.bat に示されているとおりに、クライアント関連 Java ファイルを指定します。次の形式でクライアント ファイルを追加します。

```
@set sources=client¥ClientUI.java client¥EjbClient.java
        client¥LoginEvent.java client¥LoginPanel.java
        client¥MainPanel.java client¥Request.java
```

JRun のルート ディレクトリ/samples/sample2a/make1.bat に示されている方法で、Bean のホームおよびリモート インターフェイス クラスも指定する必要があります。次の形式で、Bean のホームおよびリモート インターフェイスをインクルードしてください。この例では、Balance.class がリモート インターフェイスで、BalanceHome.class がホーム インターフェイスです。

```
@set ejb_clients=ejbeans¥Balance.class ejbeans¥BalanceHome.class
```

Make Jars の使用

make ファイルを使用するには、コマンド プロンプトを開き、シェル (bash または DOS) を起動します。JRUN_HOME を JRun のインストール ディレクトリに設定してから、作業ディレクトリに移動して次のように make jars を実行します。

```
bash$ export JRUN_HOME=JRun のルート ディレクトリ
bash$ cd JRun のルート ディレクトリ/プロジェクトパス
bash$ make jars
```

Windows の場合、DOS シェルを起動して次のように makew jars を実行します。

```
set JRUN_HOME=JRun のルート ディレクトリ
cd JRun のルート ディレクトリ/プロジェクトパス
makew jars
```

Make Deploy の使用

Deploy ツールによってホームおよびリモート実装が生成されます。このツールは JDK コンパイラを使用しますが、`deploy.properties` ファイルの `ejipt.javac` プロパティを設定すれば変更できます。

Bean の JAR ファイルを作成すると、`make deploy` を使用して Bean を公開できます。この場合、すべての Bean の実装は再生成されます。

```
bash$ make deploy
```

次のような出力が表示されます。

```
Generating BalanceHomeObject...
Generating BalanceObject...
Compiling files...
Generating BalanceHomeObject_Stub...
Generating BalanceObject_Stub...
Compiling files...
```

Make Redeploy の使用

`make redeploy` コマンドは、`-redeploy` オプションを指定して Deploy ツールを呼び出します。`-redeploy` オプションを指定すると、新しい Bean についてのみ、または Deploy ツールを前回実行した以後に更新された Bean についてのみ実装生成するという指示が、Deploy ツールに対して出されます。

```
bash$ make redeploy
```

Make Standalone の使用

`make standalone` コマンドは、`/deploy` ディレクトリの JAR ファイルを使用して EJB エンジンをスタンドアロン モードで起動します。`/deploy` ディレクトリ内の JAR およびプロパティ ファイルは、`/runtime` ディレクトリにコピーされます。

```
bash$ make standalone
```

EJB エンジンは、起動が完了すると、要求を処理する準備が整います。その後は、クライアントを起動して、サーバーに接続できます。

メモ

`make standalone` コマンドは、default JRun サーバーのディレクトリおよびポート設定を使用して EJB エンジンを起動します。ユーザが `make standalone` を出している間、default JRun サーバーは実行できません。

EJB エンジンをスタンドアロン モードで実行すると、Bean の処理をコンソール ウィンドウに表示できます。make standalone に代わる方法としては、make deploy を実行した後で default JRun サーバーを再起動し、クライアント処理のために make コマンドを出し、クライアント アプリケーションを実行してから default JRun サーバーのログ ファイル (*JRun のルート ディレクトリ*/logs/default-event.log) でその結果を調べます。

Make Run の使用

make run コマンドは EJB サンプルのクライアント部分を起動します。このコマンドを使用してクライアント アプリケーションを起動します。

```
bash$ make run
```

Make Users の使用

make users コマンドは EJB サンプル ユーザを、サンプル 9b と併用する users.properties ファイルに追加します。または、makew clean-users コマンドを使用して users.properties から EJB サンプル ユーザを削除することもできます。このコマンドを使用してサンプル ユーザを追加します。

```
bash$ make users
```

このコマンドを使用して users.properties ファイルから EJB サンプル ユーザを削除します。

```
bash$ make clean-users
```

Make Classes の使用

make classes コマンドは、変更された Bean の実装を /runtime/classes ディレクトリにコンパイルします。このコマンドを実行した後に、load コマンドを使用して、/runtime/classes ディレクトリから Bean を再ロードする必要があります。このコマンドを使用して動的な Bean をロードします。

```
bash$ make classes
```

Make Start の使用

`make start` コマンドは、`/deploy` ディレクトリの JAR ファイルを使用して EJB エンジンをフェイルセーフ モードで起動します。`/deploy` ディレクトリ内の JAR およびプロパティ ファイルは、`/runtime` ディレクトリにコピーされます。

```
bash$ make start
```

サーバーが起動すれば、要求を処理する準備が整います。その後は、クライアントを起動して、サーバーに接続できます。

Make Restart の使用

`make restart` コマンドは、`/runtime` ディレクトリに以前にコピーされた JAR ファイルを使用して EJB エンジンをフェイルセーフ モードで起動します。

```
bash$ make restart
```

サーバーが起動すれば、要求を処理する準備が整います。その後は、クライアントを起動して、サーバーに接続できます。

適切な Make ファイルの選択

最も一般的な make コマンドをどのような状況で使用するかについては、次のガイドラインを参照してください。

Makefile	使用時期
make jars makew jars	Bean の実装またはインターフェイスが変更されたときに使用します。Bean のプロパティが変更された場合や、default.properties ファイルが変更された場合も使用します。
make deploy makew deploy	すべての Bean のオブジェクト実装を生成するときに使用します。既定値では、Deploy ツールはすべての Bean の実装を生成します。
make run makew run	EJB サンプルのクライアント 側アプリケーションを起動します。
make users makew users	users.properties ファイルに EJB サンプル ユーザを追加します。
make clean-users makew clean-users	users.properties ファイルから EJB サンプル ユーザを削除します。
make redeploy makew redeploy	このコマンドにより、Deploy ツールが新しい Bean または更新された Bean についてのみオブジェクト実装を生成します。
make standalone makew standalone	/deploy ディレクトリ内の JAR ファイルを使用して EJB エンジンをスタンドアロン モードで起動するときに使用します。このコマンドは、Deploy ツールが以前に処理した Bean の JAR ファイルを、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します。また、オブジェクト実装 (ejipt_objects.jar) とスタブ (ejipt_exports.jar) JAR もコピーします。
make start makew start	/deploy ディレクトリ内の JAR ファイルを使用して EJB エンジンをフェイルセーフ モードで起動するときに使用します。このコマンドは、Deploy ツールが以前に処理した Bean の JAR ファイルを、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します。また、オブジェクト実装 (ejipt_objects.jar) とスタブ (ejipt_exports.jar) JAR もコピーします。make start コマンドを実行する前に RMID を起動しておく必要があります。
make restart makew restart	runtime ディレクトリに以前コピーされた JAR および runtime.properties ファイルを使用して EJB エンジンを起動するときに使用します。そのため、EJB エンジンは再起動する前に起動しておく必要があります。make restart コマンドを実行する前に RMID を起動しておく必要があります。

Makefile	使用時期
`make classes` `makew classes`	変更された Bean を `/runtime/classes` ディレクトリにコンパイルするときに使用します。
`make allclean` `makew allclean`	このコマンドは、`/ejbeans` および `/client` ディレクトリから、生成されたファイルをすべて削除するときに使用します。

フェイルセーフ モードの使用

特定の場合を除いて、EJB のサンプルは RMID を使用してスタンドアロン モードでもフェイルセーフ モードでも実行できます。サンプルの説明には、スタンドアロン モードに関する手順が含まれています。フェイルセーフ モードで実行する場合は、次の手順に従って RMID を起動してください。このとき、`make standalone` を `make start` に置き換えます。`make` ファイルではなくコマンド ラインで入力する場合は、次のコマンドを入力します。

```
> cd JRun のルート ディレクトリ; java -Djava.security.policy=jrun.policy
        -classpath
  lib/ejipt_tools.jar allaire.ejipt.tools.Server -start
```

フェイルセーフ モードで実行しているサーバーを停止するには、「`make stop`」と入力するか、または次のコマンドを入力します。

```
> cd JRun のルート ディレクトリ; java -Djava.security.policy=jrun.policy
        -classpath
  lib/ejipt_tools.jar allaire.ejipt.tools.Server -stop
```

JRun によってすぐにシャットダウンされ、同時にクリーン アップが行われて、トランザクションが終了します。サーバーを停止したら、次の手順に従って RMID サービスもシャット ダウンしてください。

RMID の使用

RMID ではサーバーをリモートからアクティブにすることができます。クライアントがサーバーに接続しようとすると、RMID は必要に応じてサーバーを起動することにより、サーバーを確実に使用可能にします。RMID を起動するには、環境に応じて適切な手順を使用してください。

Solaris および Linux 上での RMID

Solaris または Linux で 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 ログ ウィンドウに出力メッセージが表示されます。

Solaris で RMID を停止するには、次のコマンドを入力します。

```
% rmid -stop
```

Windows 上での RMID

Windows で RMID を起動するには、コマンド プロンプト ウィンドウを開いて次のコマンドを入力します。

```
> cd ¥temp
> start rmid
```

rmid.exe というタイトルが付いた新しいウィンドウが表示されます。既定では、出力メッセージは新しいウィンドウに表示されます。RMID が停止するまで、ウィンドウは開いたままになります。

Windows で RMID を停止するには、次のコマンドを入力します。

```
> rmid -stop
```

RMID のトラブルシューティング

RMID は、ログ ファイルを生成します。これは、サーバーを自動的に再起動するために使用されます。サーバーの新しいインスタンスを起動するには、最初にログ ファイルを削除しておく必要があります。`/log` ディレクトリは、RMID を起動した時点でアクティブなディレクトリに作成されています。

RMID を先に起動しないでサーバーをフェイルセーフ モードで起動しようとすると、サーバーは正常に起動しません。このような問題が発生した場合、または「ロックを取得することができませんでした。動作状態は終了しました」というメッセージによる例外を受け取った場合は、環境をリセットする必要があります。環境をリセットして、サーバーを起動するには、次の手順を実行します。

1 `make stop` か、またはこれと同等のコマンドを発行します。

2 `/runtime` ディレクトリにアクセスし、`*.id` ファイルと `*.lock` ファイルを削除します。

3 RMID の前のインスタンスの `/log` ディレクトリはすべて削除してください。

4 `start rmid` コマンドを発行します。

5 `make start` か、またはこれと同等のコマンドを発行します。

Cygnus ユーザへの注意事項

必要に応じて、GNU make ユーティリティの Windows バージョンを使用することもできます。Cygnus ツールを使用している場合は、次の環境変数を設定しておく必要があります。

```
MAKE_MODE=UNIX
```

JRun を Program Files ディレクトリにインストールしており、bash シェルを使用している場合は、bash シェルで作業するときに、エスケープ文字を使用して名前のスペースを処理する必要があります。コマンドは次のとおりです。

```
bash$ export JRUN_HOME="/Program¥ Files/Allaire/JRun のルート ディレクトリ"
bash$ cd /Program¥ Files/Allaire/JRun のルート ディレクトリ/samples/
        sample2a
```

JRun を C: ドライブ以外のドライブ上にインストールした場合、bash シェルで作業しているときに入力するコマンドは若干異なります。たとえば、D:¥jrun にインストールした場合は、コマンドは次のとおりです。

```
bash$ export JRUN_HOME=d:/jrun
bash$ cd //d/jrun/samples/sample2a
```

索引