Space Details

Key:	GMOxDOC20ja
Name:	Apache Geronimo v2.0 - JA
Description:	
Creator (Creation Date):	hcunico (11 09, 2007)
Last Modifier (Mod. Date):	hcunico (11 09, 2007)

Available Pages

Documentation

This page last changed on 4 28, 2008 by JAGUG.

ここは、Apache Geronimo v2.0 ドキュメントの日本語翻訳のホームページです。

Apahce Geronimo v2.0 ドキュメントへようこそ。この Geronimo のバージョンと同じように、このドキュメントもまだ作成途中です。

Apache Geronimo v2.0 は最初の JEE 5 認証を得たリリースです。現時点では、ウェブ・サービスには CXF を、永続化には OpenJPA を組み込んだ Geronimo-Tomcat において、SUN の Java Enterprise Edition 5.0 Certification Test Suite を通過しています。私たちは他の組み込みソフトウェアでも認証を得ようと努力し続けています。8. RELEASE-NOTES-2.0.2.TXT を参照し、サポートされている機能や既知の問題点や制限事項を確認してください。

以下の文書では、サンプル・アプリケーションで機能をテストしながら、Apahce Geronimo v2.0 を構築し、利用する方法を説明しています。皆様のコメントや投稿は歓迎しています。

1. 管理

This page last changed on 4 20, 2008 by JAGUG.

以下の一連の記事は、管理関連のタスクのすべてを説明しています。これらのタスクを実行するには、Apache Geronimoで提供されているツールを知る必要があるでしょう。これらの記事はApache Geronimoサーバーを管理するために利用可能なすべてのツールとコマンドについて説明します。

現在の記事は以下の通りです。:

1.1. 管理タスク

This page last changed on 4 20, 2008 by JAGUG.

このセクションでは、よく使うものは勿論、あまり使わないものも含めできるだけ多くの管理タスクをご紹介するつもりです。この記事は4つのメイン・セクションに分かれていますが、1.6. Geronimo 管理コンソール を使う上で似たような流れのものをグループにまとめていますので、タスクとコンソールのいずれにも慣れやすいようになっています。これらの記事では管理コンソールだけではなく、一部はコマンド・ライン・オプションについても言及しています。コマンド・ライン・オプションの詳細は 1.9. ツールとコマンド セクションでご説明します。

以下のトピックを扱います。

1.1.1. アプリケーションの管理

This page last changed on 4 20, 2008 by JAGUG.

このセクションではアプリケーションを管理するためのいくつかの方法を紹介します。このセクションではアプリケーションを管理のために 1.6. Geronimo 管理コンソール を使う方法とコマンド・ライン・オプションを使う方法の両方をご説明します。このセクションのいくつかの例ではHelloWorld.war　サンプル・アプリケーションを使いますが、これは5. クイック・スタート - いますぐ始めたい人の Apache Geronimo のセクションで作ったものです。

1.1.1.1. アプリケーションのインストールと除去

This page last changed on 4 25, 2008 by JAGUG.

アプリケーションをインストールしたり除去するには、いくつかの方法があります。

- Geronimo 管理コンソール
- デプロイヤー・ツール (コマンド・ライン)
- ホット・デプロイ (コマンド・ライン)

アプリケーションをパッケージングする際には、デプロイメント・プランをパッケージの中に含めても良いですし、含めなくてもかまいません。Geronimoはパッケージ済みのアプリケーションの WEB-INF ディレクトリーの中にデプロイメント・プラン　geronimo-web.xml および web.xml (またはアプリケーションの種類によっては geronimo-application.xml や geronimo-application-client.xml)があるかどうかを探索します。Geronimoがこれらの記述子を見つけられなかった場合は、デフォルトの設定を使ってアプリケーションをデプロイしようとします。もしデフォルトの設定でデプロイが失敗した場合は、アプリケーションの中のデプロイメント・プランを再びパッケージングするか、または外部からデプロイメント・プランを与える必要があります。以降のセクションでは、このような場合に採りうる選択肢を、更に細かく説明します。

Geronimo管理コンソール

Geronimo管理コンソールを使って新たにアプリケーションを導入する場合には、左側の Console Navigation メニューの Deploy New を選択し、Install New Applications ポートレットを使用します。このポートレットを使えば、デプロイ直後にアプリケーションを自動的に開始させることもできます。

前と同様に、この例でも5. クイック・スタート - いますぐ始めたい人の Apache Geronimo で作成した簡単なJSPのHelloWorldサンプルを使います。但し、あちらでは --inPlace オプションを使いました。つまり、アプリケーションを WAR ファイルにパッケージングする必要はありませんでした。WARファイルを作成するには <app_home> ディレクトリーから下記のコマンドをタイプしてください。

```
jar -cvf HelloWorld.war *
```

このWARには、すでにデプロイメント・プランがパッケージ中に含まれています。

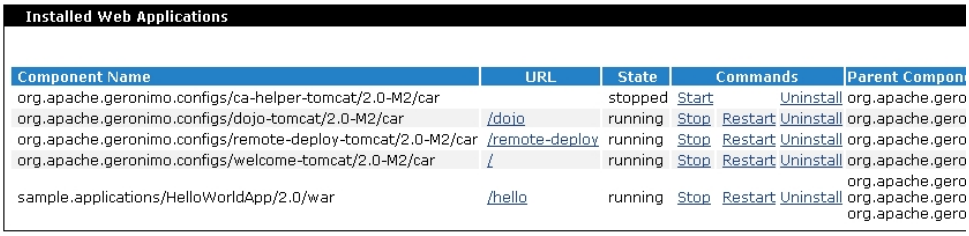
Geronimo管理コンソールに戻り、Install New Applications ポートレットで Browse をクリックし Archive: セクションで `HelloWorld.war` へのパスを指定します。Start app after install チェックボックスがチェックされていることを確認し（デフォルトではチェックされています） Install をクリックします。

ポートレットの先頭に" `The application was successfully deployed.` " と " `The application was successfully started.` " という確認のためのメッセージが表示されるはずです。

アプリケーションが正常にインストールされ、開始したことを確認するもうひとつの方法として、左側の Console Navigation メニューの Web App WARs を選択すると表示される Installed Web Applications ポートレットも利用できます。リスト中の hello アプリケーションのステータスが running になっているのが見えるはずです。

私たちの導入したアプリケーションは HelloWold. WAR なのでこのポートレットを使いましたが、EARをインストールした場合には左側の Console Navigation メニューの Application EARs を選択すると表示される *Installed Application EARs ポートレットも利用できます。インストールの手順はWARとEARアプリケーションで同じです。

Geronimo管理コンソールを使ってアプリケーションを除去する際には、アンインストールするアプリケーションによって、Installed Web Applications か Installed Application EARs のいずれかのポートレットを利用します。



私たちのサンプルでは、Installed Web Applications ポートレットへアクセスして、Component Name のリストで **`sample.applications/HelloWorldApp/2.0/war`** に対して Uninstall をクリックします。その結果、まずアプリケーションが停止され、次にアンインストールが行われます。" `Uninstalled application` " という確認のためのメッセージがポートレットの末尾に表示されます。

以上がGeronimo管理コンソールを用いてアプリケーションをインストール／アンインストールするための基本的な手順です。次の2つのセクションでは、コマンドラインを用いた方法を述べます。

デプロイヤー・ツール

デプロイヤー・ツールを使えばアプリケーションのインストールやアンインストール等々をコマンドラインから行うことができます。このセクションではサンプル・アプリケーションを使いながらインストールとアンインストールのタスクのみを扱います。その他の機能については 1.9.3. Deployer tool - デプロイヤー・ツール のセクションにすべて記載されています。

サンプル・アプリケーション HelloWorld.war をデプロイヤー・ツールを使ってコマンド・ラインからデプロイするには、<geronimo_home>¥binディレクトリーで下記のコマンドをタイプします。

```
deploy --user system --password manager deploy <app_home>/HelloWorld.war
```

もし貴方のアプリケーションの WEB-INF にGeronimoの特定のデプロイメント・プランが含まれていなくても、Geronimo 管理コンソールの場合と同様に、外部から指定することができます。先程のコマンドにデプロイメント・プランのパスとファイル名を追加すればよいだけです。

```
deploy --user system --password manager deploy <app_home>/HelloWorld.war
<deployment_plan_home>/plan.xml
```

外部からデプロイメント・プランを指定する場合には、プランを特定する任意のファイル名を使える点にご注目ください。`geronimo-XYZ.xml` といった書式を使わなければならないわけではありません。私たちの例ではGeronimoの特有のデプロイメント・プランが既にパッケージに含まれているので、更に追加でデプロイメント・プランを指定する必要はありません。

デプロイされると、このような確認用のメッセージが表示されるはずです。

```
D:\geronimo-tomcat6-jee5-2.0-M2\bin>deploy --user system --password manager deploy
\HelloWorld_2.0\HelloWorld.war
Using GERONIMO_BASE:   D:\geronimo-tomcat6-jee5-2.0-M2
Using GERONIMO_HOME:   D:\geronimo-tomcat6-jee5-2.0-M2
Using GERONIMO_TMPDIR: D:\geronimo-tomcat6-jee5-2.0-M2\var\temp
Using JRE_HOME:        C:\Java\jdk1.5.0_06\\jre
Deployed sample.applications/HelloWorldApp/2.0/war @
http://hcunico:8080/hello
```

Geronimo管理コンソールを使った場合と違い、コマンドライン・ツールを使った場合には確認用のメッセージとしてより多くの情報が入手できます。例えば、Component Name（またはモジュールID）や、デプロイされたアプリケーションのコンテキスト・ルートとポート番号を入手できます。この種の設定値はどうしても忘れがちなので、複数のアプリケーションをインストールやテスト、アンインストールする時には便利です。

または、もしアプリケーションをまだパッケージングしていないのであれば、--inPlace オプションを使えば、貴方が開発に使用しているディレクトリーから直接、アプリケーションをデプロイすることもできます。このオプションは 5. クイック・スタート - いますぐ始めたい人の Apache Geronimoの中でサンプル・アプリケーションをデプロイする際に使われています。

この場合には、<geronimo_home>¥binディレクトリーで下記のコマンドを使います。

```
deploy --user system --password manager deploy --inplace <app_home>
```

デプロイヤー・ツールは、インストールされているアプリケーションをリストするために使うこともできます。list-modules コマンドでアプリケーションをリストできますし、started や stopped 状態のアプリケーションのリストに絞り込むこともできます。list-modules コマンドで追加のパラメーターを指定しなかった場合のデフォルトの動きでは、すべてのアプリケーションをリストします。

アプリケーションをアンインストールする場合には最初に正しいComponent Name (またはモジュールID)を入手しなくてはなりません。<geronimo_home>¥binディレクトリーから以下のコマンドを実行してください。

deploy --user system --password manager list-modules

下記の例のリストに似た内容が表示されるはずです。

```
D:\geronimo-tomcat6-jee5-2.0-M2\bin>deploy --user system --password manager list-modules
Using GERONIMO_BASE:   D:\geronimo-tomcat6-jee5-2.0-M2
Using GERONIMO_HOME:   D:\geronimo-tomcat6-jee5-2.0-M2
Using GERONIMO_TMPDIR: D:\geronimo-tomcat6-jee5-2.0-M2\var\temp
Using JRE_HOME:        C:\Java\jdk1.5.0_06\\jre
Found 47 modules
+ console.dbpool/LocalDB/1.0/rar
+ console.dbpool/jdbc%2FTradeDataSource/1.0/rar
+ org.apache.geronimo.configs/activemq/2.0-M2/car
+ org.apache.geronimo.configs/activemq-broker/2.0-M2/car
+ org.apache.geronimo.configs/axis/2.0-M2/car
+ org.apache.geronimo.configs/axis-deployer/2.0-M2/car
+ org.apache.geronimo.configs/axis2/2.0-M2/car
+ org.apache.geronimo.configs/axis2-deployer/2.0-M2/car
+ org.apache.geronimo.configs/client-deployer/2.0-M2/car
+ org.apache.geronimo.configs/connector-deployer/2.0-M2/car
+ org.apache.geronimo.configs/cxf/2.0-M2/car
+ org.apache.geronimo.configs/cxf-deployer/2.0-M2/car
+ org.apache.geronimo.configs/dojo-tomcat/2.0-M2/car @ http://hcunico:8080/dojo
+ org.apache.geronimo.configs/geronimo-gbean-deployer/2.0-M2/car
+ org.apache.geronimo.configs/hot-deployer/2.0-M2/car
+ org.apache.geronimo.configs/j2ee-deployer/2.0-M2/car
+ org.apache.geronimo.configs/j2ee-security/2.0-M2/car
+ org.apache.geronimo.configs/j2ee-server/2.0-M2/car
+ org.apache.geronimo.configs/j2ee-system/2.0-M2/car
+ org.apache.geronimo.configs/javamail/2.0-M2/car
+ org.apache.geronimo.configs/jee-specs/2.0-M2/car
+ org.apache.geronimo.configs/openejb/2.0-M2/car
+ org.apache.geronimo.configs/openejb-deployer/2.0-M2/car
+ org.apache.geronimo.configs/persistence-jpa10-deployer/2.0-M2/car
+ org.apache.geronimo.configs/remote-deploy-tomcat/2.0-M2/car @ http://hcunico:8080/remote-deploy
+ org.apache.geronimo.configs/rmi-naming/2.0-M2/car
+ org.apache.geronimo.configs/sharedlib/2.0-M2/car
+ org.apache.geronimo.configs/system-database/2.0-M2/car
+ org.apache.geronimo.configs/tomcat6/2.0-M2/car
+ org.apache.geronimo.configs/tomcat6-deployer/2.0-M2/car
+ org.apache.geronimo.configs/transaction-jta11/2.0-M2/car
+ org.apache.geronimo.configs/webconsole-tomcat/2.0-M2/car
`-> standard.war @ http://hcunico:8080/console-standard
`-> framework.war @ http://hcunico:8080/console
+ org.apache.geronimo.configs/webservices-common/2.0-M2/car
+ org.apache.geronimo.configs/welcome-tomcat/2.0-M2/car @ http://hcunico:8080/
+ sample.applications/HelloWorldApp/2.0/war @ http://hcunico:8080/hello
org.apache.geronimo.configs/ca-helper-tomcat/2.0-M2/car
org.apache.geronimo.configs/client/2.0-M2/car
org.apache.geronimo.configs/client-security/2.0-M2/car
org.apache.geronimo.configs/client-system/2.0-M2/car
org.apache.geronimo.configs/client-transaction/2.0-M2/car
org.apache.geronimo.configs/directory/2.0-M2/car
org.apache.geronimo.configs/ldap-realm/2.0-M2/car
org.apache.geronimo.configs/online-deployer/2.0-M2/car
org.apache.geronimo.configs/openjpa/2.0-M2/car
org.apache.geronimo.configs/shutdown/2.0-M2/car
org.apache.geronimo.configs/transformer-agent/2.0-M2/car
org.apache.geronimo.configs/uddi-tomcat/2.0-M2/car
```

sample.applications/HelloWorldApp/2.0/war のエントリーを探してください。その値が、アンインストールの際に使う値です。

アプリケーションをアンインストールするには、先のモジュールのリストから特定したコンポーネント名を使って、<geronimo_home>¥binディレクトリーから下記のコマンドを実行します。

deploy --user system --password manager undeploy sample.applications/HelloWorldApp/2.0/war

このコマンドはまずアプリケーションを停止し、次にアンインストールします。下記のようなメッセージが表示されるはずです。

```
D:\geronimo-tomcat6-jee5-2.0-M2\bin>deploy --user system --password manager undeploy
sample.applications/HelloWorldApp/2.0/war
Using GERONIMO_BASE:   D:\geronimo-tomcat6-jee5-2.0-M2
Using GERONIMO_HOME:   D:\geronimo-tomcat6-jee5-2.0-M2
Using GERONIMO_TMPDIR: D:\geronimo-tomcat6-jee5-2.0-M2\var\temp
Using JRE_HOME:        C:\Java\jdk1.5.0_06\\jre
Module sample.applications/HelloWorldApp/2.0/war unloaded.
Module sample.applications/HelloWorldApp/2.0/war uninstalled.

Undeployed sample.applications/HelloWorldApp/2.0/war
```

この例では、同一のツールで3つの異なるコマンドを使ってみました。

- deploy
- list-modules
- undeploy

デプロイヤー・ツールで使えるオプションは他にもあります。更に細かい情報は 1.9. ツールとコマンド セクションを参照してください。

ホット・デプロイ

Apache Geronimoは ホット・デプロイメント をサポートしています。つまり、アプリケーションのJARファイルを <geronimo_home>/deploy ディレクトリーにコピーするだけで、アプリケーションを自動的にデプロイさせることができるということです。また、この方法でデプロイされたアプリケーションをアンインストールしたり更新したりすることもできます。単体のJARファイルをコピーするのではなく、アプリケーション・モジュールを含む（パッケージングされていない） ディレクトリー をそのままコピーしても結構です。

ホット・デプロイを使う場合は、デプロイメント・プラン をアプリケーションのパッケージに含めねばならない点に注意してください。この方法では、外部からデプロイメント・プランを指定することはサポートされていません。また、1.9.3. Deployer tool - デプロイヤー・ツール や管理コンソールを使ってデプロイされたアプリケーションは <geronimo_home>/deploy ディレクトリー上に現れない点にご注意ください。

あるアプリケーション、たとえばHelloWorld.warを deploy ディレクトリーにコピーすると、アプリケーションがデプロイされたこと、及びそのアプリケーションのコンテキストを示す確認のためのメッセージがGeronimoの稼動中のコンソール上に表示されます。

```
11:45:23,500 INFO  [DirectoryHotDeployer] Deploying HelloWorld.war
11:45:23,953 INFO  [DirectoryHotDeployer]     Deployed sample.applications/HelloWorldApp/2.0/war @
http://hcunico:8080/hello
```

アプリケーションを除去するには、deploy ディレクトリーからWARやEARを削除するだけです。アプリケーションが除去されると、アプリケーションがアンデプロイされたことを示すメッセージがGeronimoの稼動中のコンソール上に表示されます。

```
11:46:17,953 INFO  [DirectoryHotDeployer] Undeploying HelloWorld.war
11:46:18,281 INFO  [DirectoryMonitor] Hot deployer notified that an artifact was removed:
sample.applications/HelloWorldApp/2.0/war
11:46:18,281 INFO  [DirectoryHotDeployer] Undeployed sample.applications/HelloWorldApp/2.0/war
```

サマリー

当セクションではアプリケーションをインストール／アンインストールするための3つの異なる方法をご説明しました。グラフィカルなもの、コマンド・ラインのもの、そしてホット・デプロイメント、好みに応じてコマンド・ラインまたはGUIを選択できるわけです。

利用できるオプションやパラメーターの詳細については以降の 1.9. ツールとコマンド セクションを参照してください。

1.1.1.2. アプリケーション・モジュールの開始と停止

This page last changed on 4 25, 2008 by JAGUG.

アプリケーションのステータスは2つの異なる方法で変更できます。デプロイヤー・ツールかGeronimo管理コンソールです。

前述したとおり、デプロイヤー・ツール にはたくさんのコマンドがあります。1.1.1.1. アプリケーションのインストールと除去セクションでは、主にdeployとundeployを調べ、ごく簡単にlist-modulesに言及しました。このセクションではステータスを変更したいモジュールの名前を入手するためのlist-modulesコマンドについてフォーカスし、さらに新しいコマンド start と stop もご紹介します。後半では、別の方法としてGeronimo管理コンソールにも言及します。

list-modules コマンドは以下のパラメータを受け入れます。

- --all : は他のオプションが指定されなかった時のデフォルトです。利用可能なモジュールをすべてリストします。
- --started : このオプションは現在起動中のモジュールのみをリストします。
- --stopped : このオプションは現在起動していないモジュールのみをリストします。

コマンドの最後に target を指定することもできます。ただ、このオプションはあまり使われることはありません。なぜなら、殆どの場合、個々のサーバーでは構成ストアはひとつしか定義されていないからです。もしtargetが指定されない場合は、すべてのターゲットのすべてのモジュールがリストされます。Targetが指定された場合はそのターゲット上のモジュールのみがリストされます。

この機能は特定のモジュールのステータスやモジュールID自体を特定する際に便利です。これからこのコマンドを使って、HelloWorld アプリケーションを特定し、そのステータスを変更してみます。

<geronimo_home>/binディレクトリーで、すべてのモジュールをリストするための下記のコマンドをコマンド・ライン・ウインドウから入力します。

```
deploy --user system --password manager list-modules
```

そのサーバーにインストールされているすべてのモジュールのリストが入手できます。開始されているモジュール群は左側の ＋ 印で識別され、かつリストの先頭のほうに表示されている点にご注目ください。

```
D:\geronimo-tomcat6-jee5-2.0-M2\bin>deploy --user system --password manager list-modules
Using GERONIMO_BASE:   D:\geronimo-tomcat6-jee5-2.0-M2
Using GERONIMO_HOME:   D:\geronimo-tomcat6-jee5-2.0-M2
Using GERONIMO_TMPDIR: D:\geronimo-tomcat6-jee5-2.0-M2\var\temp
Using JRE_HOME:        C:\Java\jdk1.5.0_06\\jre
Found 47 modules
+ console.dbpool/LocalDB/1.0/rar
+ console.dbpool/jdbc%2FTradeDataSource/1.0/rar
+ org.apache.geronimo.configs/activemq/2.0-M2/car
+ org.apache.geronimo.configs/activemq-broker/2.0-M2/car
+ org.apache.geronimo.configs/axis/2.0-M2/car
+ org.apache.geronimo.configs/axis-deployer/2.0-M2/car
+ org.apache.geronimo.configs/axis2/2.0-M2/car
+ org.apache.geronimo.configs/axis2-deployer/2.0-M2/car
+ org.apache.geronimo.configs/client-deployer/2.0-M2/car
+ org.apache.geronimo.configs/connector-deployer/2.0-M2/car
+ org.apache.geronimo.configs/cxf/2.0-M2/car
+ org.apache.geronimo.configs/cxf-deployer/2.0-M2/car
+ org.apache.geronimo.configs/dojo-tomcat/2.0-M2/car @ http://hcunico:8080/dojo
+ org.apache.geronimo.configs/geronimo-gbean-deployer/2.0-M2/car
+ org.apache.geronimo.configs/hot-deployer/2.0-M2/car
+ org.apache.geronimo.configs/j2ee-deployer/2.0-M2/car
+ org.apache.geronimo.configs/j2ee-security/2.0-M2/car
+ org.apache.geronimo.configs/j2ee-server/2.0-M2/car
+ org.apache.geronimo.configs/j2ee-system/2.0-M2/car
+ org.apache.geronimo.configs/javamail/2.0-M2/car
+ org.apache.geronimo.configs/jee-specs/2.0-M2/car
+ org.apache.geronimo.configs/openejb/2.0-M2/car
+ org.apache.geronimo.configs/openejb-deployer/2.0-M2/car
+ org.apache.geronimo.configs/persistence-jpa10-deployer/2.0-M2/car
+ org.apache.geronimo.configs/remote-deploy-tomcat/2.0-M2/car @ http://hcunico:8080/remote-deploy
+ org.apache.geronimo.configs/rmi-naming/2.0-M2/car
+ org.apache.geronimo.configs/sharedlib/2.0-M2/car
+ org.apache.geronimo.configs/system-database/2.0-M2/car
+ org.apache.geronimo.configs/tomcat6/2.0-M2/car
```

```
+ org.apache.geronimo.configs/tomcat6-deployer/2.0-M2/car
+ org.apache.geronimo.configs/transaction-jta11/2.0-M2/car
+ org.apache.geronimo.configs/webconsole-tomcat/2.0-M2/car
`-> standard.war @ http://hcunico:8080/console-standard
`-> framework.war @ http://hcunico:8080/console
+ org.apache.geronimo.configs/webservices-common/2.0-M2/car
+ org.apache.geronimo.configs/welcome-tomcat/2.0-M2/car @ http://hcunico:8080/
+ sample.applications/HelloWorldApp/2.0/war @ http://hcunico:8080/hello
org.apache.geronimo.configs/ca-helper-tomcat/2.0-M2/car
org.apache.geronimo.configs/client/2.0-M2/car
org.apache.geronimo.configs/client-security/2.0-M2/car
org.apache.geronimo.configs/client-system/2.0-M2/car
org.apache.geronimo.configs/client-transaction/2.0-M2/car
org.apache.geronimo.configs/directory/2.0-M2/car
org.apache.geronimo.configs/ldap-realm/2.0-M2/car
org.apache.geronimo.configs/online-deployer/2.0-M2/car
org.apache.geronimo.configs/openjpa/2.0-M2/car
org.apache.geronimo.configs/shutdown/2.0-M2/car
org.apache.geronimo.configs/transformer-agent/2.0-M2/car
org.apache.geronimo.configs/uddi-tomcat/2.0-M2/car
```

私たちの求めているモジュールは"**+ sample.applications/HelloWorldApp/2.0/war @ http://hcunico:8080/hello**" であり、左に + 印があるので、このモジュールのステータスは started 状態です。また、以下のコマンドで、開始されたモジュールのみをリストすることもできます。

deploy --user system --password manager list-modules --started

"**sample.applications/HelloWorldApp/2.0/war** モジュールを停止するには、以下のコマンドを入力します。

deploy --user system --password manager stop sample.applications/HelloWorldApp/2.0/war

そのモジュールが停止したことを示す以下のような確認用のメッセージが表示されます。

```
D:\geronimo-tomcat6-jee5-2.0-M2\bin>deploy --user system --password manager stop
sample.applications/HelloWorldApp/2.0/war
Using GERONIMO_BASE:   D:\geronimo-tomcat6-jee5-2.0-M2
Using GERONIMO_HOME:   D:\geronimo-tomcat6-jee5-2.0-M2
Using GERONIMO_TMPDIR: D:\geronimo-tomcat6-jee5-2.0-M2\var\temp
Using JRE_HOME:        C:\Java\jdk1.5.0_06\\jre

Stopped sample.applications/HelloWorldApp/2.0/war
```

停止されたモジュールをすべてリストした場合には、**sample.applications/HelloWorldApp/2.0/war** がそのリストに含まれていることでしょう。以下の例でコマンドの結果をご覧ください。

```
D:\geronimo-tomcat6-jee5-2.0-M2\bin>deploy --user system --password manager list-modules --stopped
Using GERONIMO_BASE:   D:\geronimo-tomcat6-jee5-2.0-M2
Using GERONIMO_HOME:   D:\geronimo-tomcat6-jee5-2.0-M2
Using GERONIMO_TMPDIR: D:\geronimo-tomcat6-jee5-2.0-M2\var\temp
Using JRE_HOME:        C:\Java\jdk1.5.0_06\\jre
Found 15 modules
org.apache.geronimo.configs/axis2/2.0-M2/car
org.apache.geronimo.configs/ca-helper-tomcat/2.0-M2/car
org.apache.geronimo.configs/client/2.0-M2/car
org.apache.geronimo.configs/client-security/2.0-M2/car
org.apache.geronimo.configs/client-system/2.0-M2/car
org.apache.geronimo.configs/client-transaction/2.0-M2/car
org.apache.geronimo.configs/cxf/2.0-M2/car
org.apache.geronimo.configs/directory/2.0-M2/car
org.apache.geronimo.configs/ldap-realm/2.0-M2/car
org.apache.geronimo.configs/online-deployer/2.0-M2/car
org.apache.geronimo.configs/openjpa/2.0-M2/car
org.apache.geronimo.configs/shutdown/2.0-M2/car
org.apache.geronimo.configs/transformer-agent/2.0-M2/car
org.apache.geronimo.configs/uddi-tomcat/2.0-M2/car
sample.applications/HelloWorldApp/2.0/war
```

モジュールを開始するには、stopのかわりに start を使えばよいだけです。

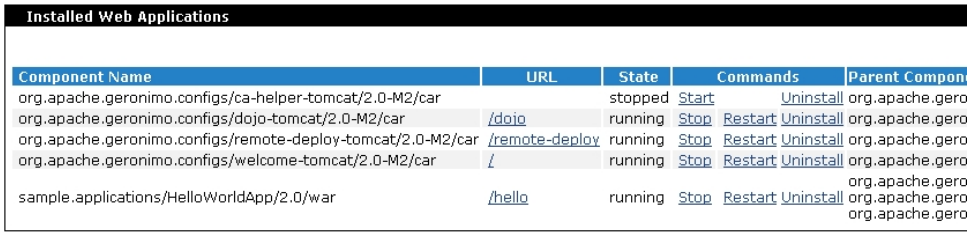
deploy --user system --password manager start sample.applications/HelloWorldApp/2.0/war

そのモジュールが開始したことを示す以下のような確認用のメッセージが表示されます。

```
D:\geronimo-tomcat6-jee5-2.0-M2\bin>deploy --user system --password manager start
sample.applications/HelloWorldApp/2.0/war
Using GERONIMO_BASE:   D:\geronimo-tomcat6-jee5-2.0-M2
Using GERONIMO_HOME:   D:\geronimo-tomcat6-jee5-2.0-M2
Using GERONIMO_TMPDIR: D:\geronimo-tomcat6-jee5-2.0-M2\var\temp
Using JRE_HOME:        C:\Java\jdk1.5.0_06\\jre

Started sample.applications/HelloWorldApp/2.0/war @
http://hcunico:8080/hello
```

モジュールのステータスを変更する際には、コマンド・ライン・オプションの代わりにGeronimo管理コンソールを利用することもできます。Geronimo管理コンソールを開き、左側のConsole Navigationメニューから Applicationsへと辿ります。Application EARs か Web App WARs が見つかるので、ステータスを変更したいアプリケーションの種類によって、いずれかを選択してください。この例では、サンプル・アプリケーションとしてHelloWorld.warを使用しますので、Web App WARs を選びます。



このポートレットは 1.1.1.1. アプリケーションのインストールと除去 セクションでGeronimo管理コンソールを使ってアプリケーションのインストールや除去を行った際に使用したものです。同じポートレットからアプリケーションのステータスを変更できます。使用可能なコマンドの種類は現在のアプリケーションのステータスによって異なります。もしアプリケーションが running ステータス（開始）であれば、Stop と Restart が表示されているはずです。もしアプリケーションが stopped ステータス（停止）であれば、Start のみが表示されています。 Uninstall コマンドはアプリケーションのステータスとは関係なく常に表示されています。アプリケーションのステータスを変更するには、ただ、そのコマンド（StartやStop）をクリックすればよいだけです。

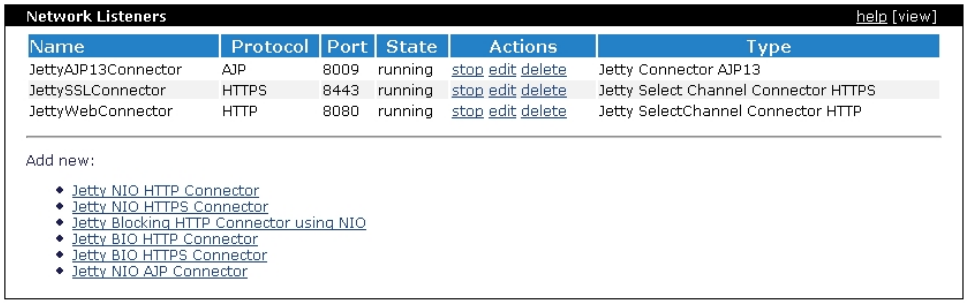
1.1.2. Apache Geronimoサーバーの管理

This page last changed on 4 20, 2008 by JAGUG.

1.1.2.1. Webコンテナーへの新規リスナーの追加

This page last changed on 4 21, 2008 by JAGUG.

ご利用のWebコンテナー(JettyまたはTomcat)に新たにリスナー（HTTP, HTTPS 及び AJP）を構成する場合は、左側の Console Navigation メニューの Web Server を選べば Network Listener ポートレットが利用できます。このポートレットを使えば、新たにリスナーを追加できますし、既存のコネクターのステータス（stop,start,delete）を変更することもできます。



このバージョンのGeronimoでは、特定のコネクター・タイプに対して利用したい実装(JettyではBIOか NIO ,TomcatではBIO, NIO , APR)を指定することができるようになった点にご注目ください。コネクターを構成するに際し、個々のパラメータはどの実装を選んだとしても同じですので、話を簡単にするために、以降、一般的なHTTP,HTTPS,そしてAJPの構成の詳細のみを述べていきます。

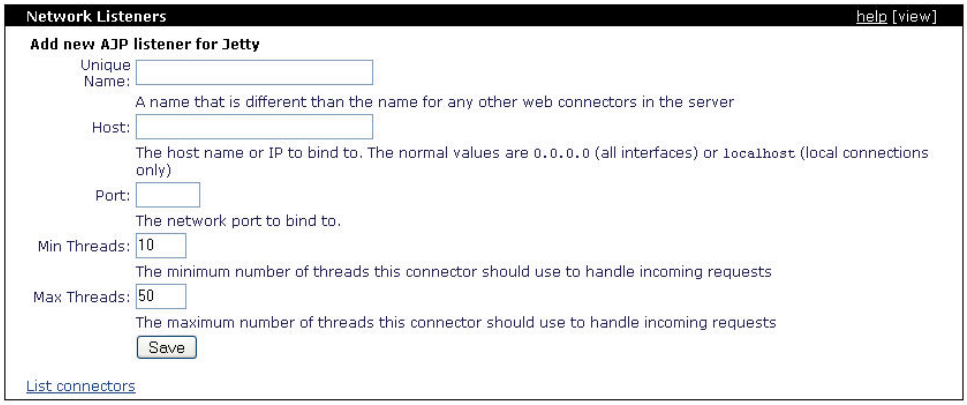
下記リスト中の各々のコネクターの構成オプションはJettyおよびTomcat Webコンテナー間で少しだけ異なっています。例えば両コンテナーで、必須の属性は同じですが、オプションで指定できる属性に関してはTomcatの方が多いです。

- 1.1.2.1.1. 新規AJPリスナーの追加
- 1.1.2.1.2. 新規HTTPリスナーの追加
- 1.1.2.1.3. 新規HTTPSリスナーの追加

1.1.2.1.1. 新規AJPリスナーの追加

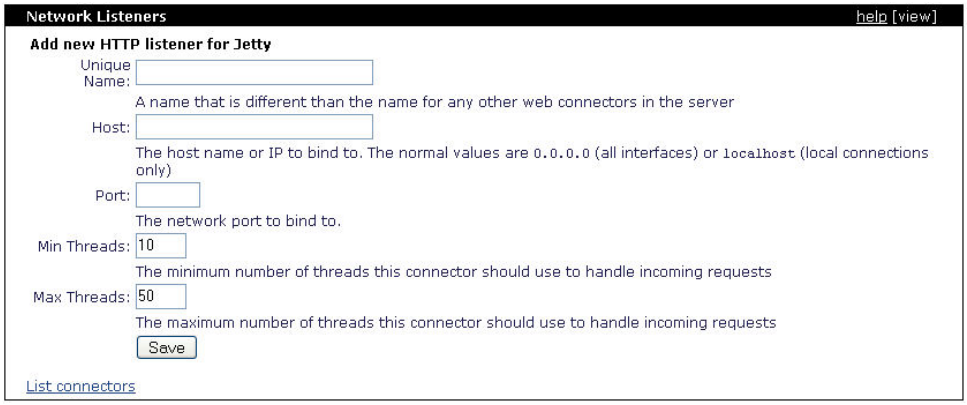
This page last changed on 4 21, 2008 by JAGUG.

新たにAJPリスナーを追加するには、そのリンクをクリックします。以下の図に、新規にAJPリスナーを作成する際に必要な様々なパラメーターを示します。この手順はとてもシンプルであり一目瞭然です。貴方の設定値を入力して Save をクリックしてください。



1.1.2.1.2. 新規HTTPリスナーの追加

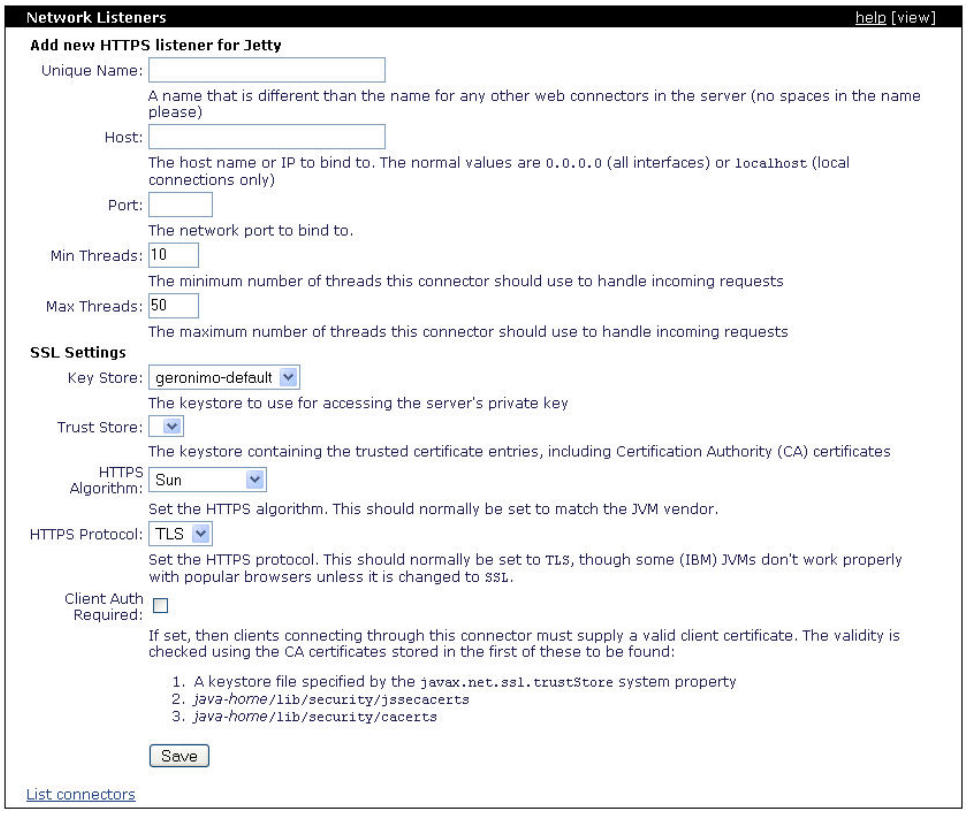
新たにHTTPリスナーを追加するには、そのリンクをクリックします。以下の図に、新規にHTTPリスナーを作成する際に必要な様々なパラメーターを示します。この手順はとてもシンプルであり一目瞭然です。貴方の設定値を入力して Save をクリックしてください。



1.1.2.1.3. 新規HTTPSリスナーの追加

This page last changed on 4 21, 2008 by JAGUG.

新たにHTTPSリスナーを追加するには、そのリンクをクリックします。以下の図に、新規にHTTPSリスナーを作成する際に必要な様々なパラメーターを示します。この手順はとてもシンプルであり一目瞭然です。貴方の設定値を入力して Save をクリックしてください。



1.1.2.2. JAX-WSエンジンの構成

This page last changed on 4 25, 2008 by JAGUG.

Apache Geronimo 2.0は2種類のJAX-WSエンジン、Apache Axis2 および Apache CXF と共に配布されています、Geronimo-Jettyアセンブリーではjax-WSエンジンのデフォルトとしてCXFを使用しています。一方、Geronimo-Tomcatアセンブリーでは、デフォルトはAxis2となっています、

使用したいJAX-WSエンジンを構成するには、二通りの方法があります。

- システム・プロパティの設定
- 構成ファイルの変更

システム・プロパティの設定

GeronimoのJAX-WSエンジンで Axis2 を使うように構成するには、org.apache.geronimo.jaxws.provider システム・プロパティを "axis2" に設定します。以下が例です。

```
export GERONIMO_OPTS="-Dorg.apache.geronimo.jaxws.provider=axis2"
```

GeronimoのJAX-WSエンジンで CXF を使うように構成するには、org.apache.geronimo.jaxws.provider システム・プロパティを "cxf" に設定します。以下が例です。

```
export GERONIMO_OPTS="-Dorg.apache.geronimo.jaxws.provider=cxf"
```

構成ファイルの変更

GeronimoのJAX-WSエンジンを Axis2 を使うように構成するには、$GERONIMO_HOME/var/config/config.xml ファイルを以下のように変更します。

1. org.apache.geronimo.configs/cxf-deployer/2.0/car モジュールの指定箇所で、condition 属性を削除し、load="false" 属性を追加します。
2. org.apache.geronimo.configs/axis2-deployer/2.0/car モジュールの指定箇所で、condition 属性を削除し、load="true" 属性を追加します。

GeronimoのJAX-WSエンジンを CXF を使うように構成するには、$GERONIMO_HOME/var/config/config.xml ファイルを以下のように変更します。

1. org.apache.geronimo.configs/axis2-deployer/2.0/car モジュールの指定箇所で、condition 属性を削除し、load="false" 属性を追加します。
2. org.apache.geronimo.configs/cxf-deployer/2.0/car モジュールの指定箇所で、condition 属性を削除し、load="true" 属性を追加します。

1.1.2.3. ログ・レベルの構成

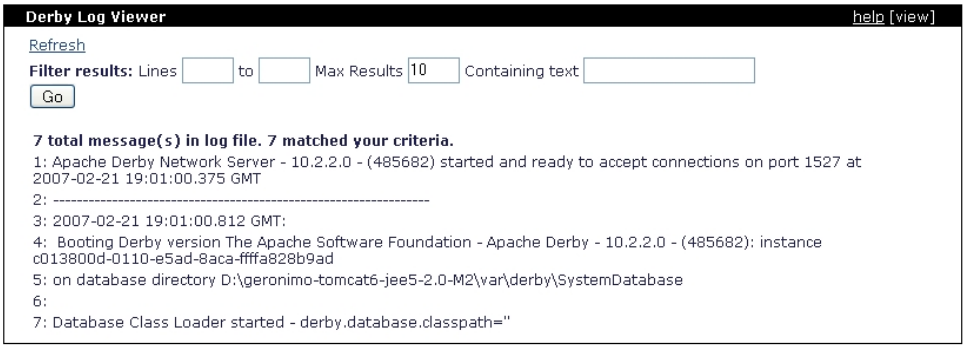
This page last changed on 4 21, 2008 by JAGUG.

1.6. Geronimo 管理コンソール のセクションで説明したように、管理コンソールはサーバーのログを構成し、参照するために4つのポートレットを提供しています。それらのポートレットとは、Log Manager, Server Log Viewer, Derby Log Viewer 及び Web Access Log Viewer です。

1.1.2.3.1. Derby ログ・ビューアー

This page last changed on 4 21, 2008 by JAGUG.

下の図にはDerby Log viewer ポートレットのオプションが示されています。このポートレットから、Derbyのサーバー・ログを閲覧できますし、表示される結果を調整するためのフィルターを設定することもできます。



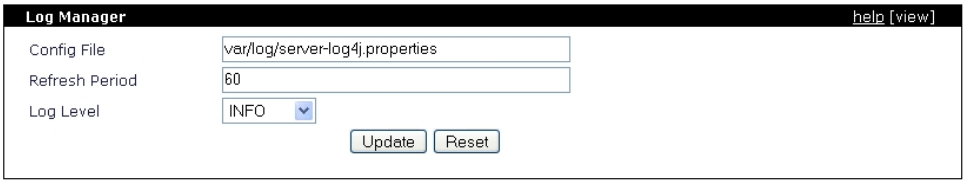
左上隅にある Refresh リンクは、もし 1.6. Geronimo 管理コンソール のデフォルトで指定したフィルター基準があったとしてもそれをリセットし、現在のDerbyサーバー・ログの最新の10行を（もしあれば）表示します。

1.1.2.3.3. サーバー・ログ・ビューアー ポートレットと同様に Filter Results: エリアで検索を更に絞り込むための別のフィルター基準を指定することができます。Lines ..to.. で分析対象のログの行範囲を指定することができます。Max Results では表示する行数に制限をかけられますし、Containing text では特定の文字列でサーバーのログを検索することもできます。

1.1.2.3.2. ログ・マネージャー

This page last changed on 4 21, 2008 by JAGUG.

下の図にはLog Manager ポートレットのオプションが示されています。このポートレットから、ログの構成ファイルの場所を指定することができます。デフォルトでは、この値は **`var/log/server-log4j.properties`** に設定されています。



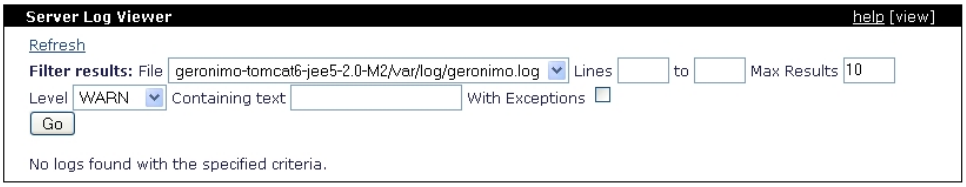
このポートレットで変更可能な別の値は Refresh Period です。この設定値はGeronimoに対し、どのくらいの頻度（秒数）で構成ファイルへの変更有無のチェックを行うべきかを指定するものです。

このポートレットから Log Level を変更することもできます。デフォルトでは INFO に設定されています。選択できる有効な値は All, DEBUG, INFO, WARN, ERROR, FATAL, TRACE 及び OFF です。

1.1.2.3.3. サーバー・ログ・ビューアー

This page last changed on 4 21, 2008 by JAGUG.

下の図にはサーバー・ログ・ビューアー・ポートレットのオプションが示されています。このポートレットからGeronimoのサーバー・ログを閲覧できますし、表示される結果をフィルタリングすることもできます。



左上隅にある Refresh リンクは、もし 1.6. Geronimo 管理コンソール のデフォルトで指定したフィルター基準があったとしてもそれをリセットし、現在の Geronimo サーバー・ログの最新の10行を(もしあれば)表示します。

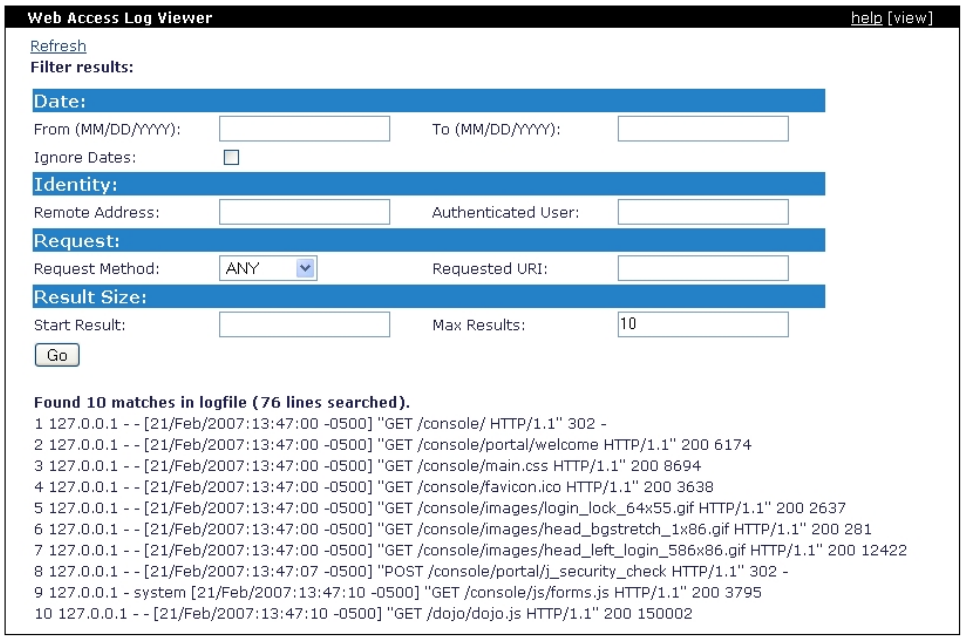
Filter Results: エリアで、検索を更に絞り込むための別のフィルター基準を指定することができます。File プルダウン・メニューで、閲覧するログ・ファイルを選択できます。プルダウン・メニューにリストされるログは 1.1.2.3.2. ログ・マネージャー ポートレットの Configuration File に記述されているものです。(デフォルトでは{{server-log4j.properties}} です) File プルダウン・メニューで選択できるログ・ファイルの数は構成ファイル `server-log4j.properties` に定義されているログ・ファイルの数により決まります。

Lines ..to.. で分析対象のログの行範囲を指定することができます。Max Results では表示する行数に制限をかけられます。Level はそのログ・レベルで発生したエラーのみを表示させることもできます。Containing text では特定の文字列でサーバーのログを検索することもできます。With Exceptions チェックボックスをクリックすればエラーだけではなく例外も表示されます。表示されるスタック・トレースの行数は Max Results で設定した行数までとなります。

1.1.2.3.4. Webアクセス・ログ・ビューアー

This page last changed on 4 21, 2008 by JAGUG.

下の図にはWeb Access Log Viewer ポートレットのオプションが示されています。このポートレットから、Webサーバー・ログを閲覧できますし、表示される結果をフィルタリングすることもできます。



左上隅にある Refresh リンクは、もし 1.6. Geronimo 管理コンソールのデフォルトで指定したフィルター基準があったとしてもそれをリセットします。他のポートレットと異なり、Web Access Log Viewer はログのすべての行を表示し、Webブラウザーが表示できる許容量によってのみ制限を受けます。

Filter Results: エリアで、検索を更に絞り込むための別のフィルター基準を指定することができます。Web Access Log Viewerでは、このエリアは Date , Identity 及び Requests の3つのメイン・グループに分割されています。

- Date:
  日付範囲を指定します。もし Ignore Dates チェックボックスがチェックされた場合は、日付に基づいたフィルタリングは行われません。日付にかかわらず、他のフィルター基準を満たす全てのログが表示されます。
- Identity:
  Remote Address (例. 192.168.0.1)とAuthenticated User (例. system).を指定できます。
- Request:
  Request Method とRequested URIを指定できます。ドロップダウン・メニューから Requested Method を選択でき、有効な値は ANY, POST 及び GET です。Requested URI　でフィルタリングする場合には、例えば http://localhost:8080/console/login.jsp のように、URIを入力するだけです。

1.1.2.4. リモートApache HTTPサーバーの構成

This page last changed on 4 25, 2008 by JAGUG.

当記事の執筆時点では、geronimo管理コンソールで提供される *Apache mod_jk Configuration* ポートレットには、既にインストール済のWebアプリケーションを認識する際に多少具合の悪い箇所があります。回避策として、当記事はリモートApache HTTPサーバーを構成するための2つの代替案をご紹介します。

Apache Geronimoは、利用したいWebコンテナーにより2種類のフレーバーで提供されています。TomcatかJettyのどちらかお好きなディストリビューションを選べます。もしもGeronimoの前面にHTTPdを設置したいのなら、クライアントからのリクエストがGeronimoに到達できるようにHTTPdを構成しなくてはなりません。

既に述べましたが、当記事では、クライアントからのリクエストをGeronimoに送り届けるためにApache HTTPdをどのように構成すればよいかについて、2つの方法をご紹介します。それは、Apacheに同梱されている mod_proxy モジュールを使ってApacheをリバース・プロキシーにする方法と Apache Tomcat のソースと一緒に入手できる Jakarta Connector mod_jk を使用する方法です。

Apache HTTPdでリバース・プロキシーを構成する方法の場合は、Geronimo側での追加の構成をしなくとも(TomcatまたはJettyの）いずれのディストリビューションの場合であってもきちんと動きます。

Jakarta Tomcat Connectorを構成する方法の場合は、更に追加の構成作業が必要になります。必要な場面ではいくつかの新しい単語が出てきますが、この記事で扱うスコープはリモートのHTTPdを使った2層シナリオでの構成の方法である点には注意してください。

上の2つの方法に基づいて、当記事は以下の2つのセクションから成り立っています。

この構成例での共通の前提として、下記がテストで利用したバーションのリストです。

- Apache Geronimo v1.2
- Apache HTTPd v2.2.3
- mod_jk-apache-2.2.3.so
- Windows XP SP2

1.1.2.4.1. リバース・プロキシー(mod_proxy)としてのApache HTTPd の構成

This page last changed on 4 25, 2008 by JAGUG.

以下がリクエストをApache HTTPd からGeroimoサーバー(またはお手元のどのようなサーバーであれ)へリルートする一番簡単な方法です。この機能を使うには特定のモジュールを有効にし、HTTPdの構成に数行を追加する必要があります。手順は以下の通りです。

1. <httpd_home>¥conf ディレクトリー上の **httpd.conf** をオープンします。
2. 以下のLoadModuleディレクティブを探し、行の先頭の # を除去して、コメントではないようにします。
   - **LoadModule proxy_module modules/mod_proxy.so**
   - **LoadModule proxy_http_module modules/mod_proxy_http.so**
3. リルートを有効にするために、httpd.confの末尾に以下の数行を追加します。
   - **ProxyPass /console http://localhost:8080/console**
   - **ProxyPass /images http://localhost:8080/images**
   - **ProxyPassReverse / http://localhost:8080/**

アプリケーションによっては、更にいくつかの ProxyPass ディレクティブを追加する必要があるかもしれません。末尾の ProxyPassReverse ディレクティブがGeronimoサーバーからのレスポンスを受け取り、Apache HTTPdから直接レスポンスが返るようにURLをマスクしますので、Geronimoサーバーを特定できる情報やロケーションを隠蔽できます。

この例ではデモの目的で console を有効化しています。本番環境では、他のネットワーク(通常はインターネット)から console へアクセスできるようにしたいとは思わないでしょう。console をネットからアクセス可能とすることは、セキュリティ面で重要な情報を露出することになりますから。

全ては制限されたアクセスであるべきであり、通常はHTTPとアプリケーション・サーバー間には(トポロジーにもよりますが)ファイアー・ウオールが設置されているべきでしょう。そして、ネット側から貴方のアプリケーションを操作するために必要な最低限のリソースのみをマップするようにすべきです。

もし HTTPd と Geronimo サーバーが同一のマシン上にあるなら、リダイレクトのために localhost を利用することができます。もしサーバーが異なるマシン上に存在する場合には、Geronimo サーバー用に URL を指定する必要があります。

上記のような構成をした後、ブラウザーで http://locahost/console を指定すれば、そのリクエストは http://localhost:8080/console へリダイレクトされることになります。この方法でURLとポートをリルートすることが可能になります。

1.1.2.4.2. Apache HTTPdとJakarta Tomcat コネクター(mod_jk)の構成

This page last changed on 4 25, 2008 by JAGUG.

HTTP サーバーと Geronimo の間の通信は AJP コネクター経由でも可能です。デフォルトでは (Jetty と Tomcat) いずれのディストリビューションに於いてもポート 8009 をリスンする AJP13 コネクターが事前に定義済です。

Jakarta Tomcat コネクター mod_jk モジュールは Apache Tomcat ソースのコネクターとして提供されています。Jetty は(そして当然ながらTomcat も)このコネクターと互換性があります。このモジュールは Tomcat のソースをダウンロードすれば入手できますが、貴方のシステムに適合したものを下記の URL からダウンロードして入手することもできます。

http://tomcat.apache.org/download-connectors.cgi

mod_jk 以外に workers.properties も必要となりますが、Tomcat のソース・ディストリビューションから同様に入手することができます。このプロパティ・ファイルは mod_jk プラグインに対し、どのように Geronimo と接続するのかを伝えるためのものです。Jakarta Tomcat コネクターを構成するために利用可能なすべてのオプションについての詳細な説明は、下記の URL を参照してください。

http://tomcat.apache.org/connectors-doc/config/workers.html

皆様に実際に使っていただけるように、こちらから Windows 用の mod_jk-apache-2.2.3.so モジュールと workers.properties がダウンロードできます。

Back to Top

Apache HTTPd の構成

Tomcat のウェブ・サイトから貴方の利用しているプラットフォームに合った適切な mod_jk をダウンロードします。当サンプルでは、ダウンロードしたmod_jkを mod_jk.so にリネームして <httpd_home>¥modules ディレクトリーにコピーします。次に、Apache Tomcatソース（またはこの記事の Attachment セクション）から workers.properties をダウンロードして、<httpd_home>¥conf ディレクトリーに解凍します。

Jakarta Tomcat_コネクター mod_jk モジュールをロードするために、<httpd_home>¥conf ディレクトリー上の httpd.conf を編集します。以下の行を httpd.conf の末尾に追加してください。

Excerpt from httpd.conf httpd.congの抜粋

```
LoadModule jk_module modules/mod_jk.so
# Loads the Jakarta Tomcat Connector module

JkWorkersFile <httpd_home>\conf\workers.properties
# Tells the module the location of the workers.properties file

JkLogFile     <httpd_home>\logs\mod_jk.log
# Specifies the location for this module's specific log file

JkLogLevel    info
# Sets the module's log level to info

JkMount /console/* ajp13
# Sets a mount point from a context to a Tomcat worker. In this case will allow access (forward the
request) to the console.
```

JkMount は /console/ 以下のすべてをワーカー ajp13 にマップします。ajp13 という名前は、次に説明する workers.properties ファイルの中で定義されたものです。他にもリモートの HTTPd 経由でアクセスされるアプリケーションがあれば、JkMount ディレクティブを更に追加する必要があります。

この例ではデモの目的で console を有効化しています。本番環境では、他のネットワーク(通常はインターネット)から console へアクセスできるようにしたいとは思わないでしょう。console をネットからアクセス可能とすることは、セキュリティ面で重要な情報を露出することになりますから。

全ては制限されたアクセスであるべきであり、通常はHTTPとアプリケーション・サーバー間には（トポロジーにもよりますが）ファイアー・ウオールが設置されているべきでしょう。そして、ネット側から貴方のアプリケーションを操作するために必要な最低限のリソースのみをマップするようにすべきです。

Back to Top

workers.propertiesの構成

workers.properties はHTTPdに対し、Geronimoサーバーがどこにあるのか、どのバージョンのAJPを利用すべきなのか、Geronimoがリスンしているポート番号はいくつなのか、などを伝えるためのものです。

<httpd_home>¥conf ディレクトリー上に存在する workers.properties ファイルを貴方の環境にマッチするように編集してください。以下のworkers.propertiesの例では、貴方が特に意識する必要がある変数のみを抜粋してあります。

Minimum requirements for the workers.properties workers.propertiesで最低限必要な定義

```
worker.ajp13.type=ajp13
# Sets the version of AJP used. The AJP listeners defined in Geronimo are AJP v13.

worker.ajp13.host=localhost
# Specifies the location of the Geronimo server. Use default localhost for single-tier scenarios.
Specify the hostname of the Geronimo server for multi-tier environments.

worker.ajp13.port=8009
# Both Tomcat and Jetty come with a predefined AJP13 listener on port 8009
```

この例でどのようにワーカーの名前が定義されているかにご注目ください。worker.ajp13.* という変数を定義している部分の ajp13 という名前は、前項で httpd.conf に記述した名前です。

最後の手順として、Apache HTTPdを再起動し、上記の変更が確実にロードされることを確認します。

Back to Top

テスト

設定した構成で Geronimo と HTTPd の両方が正常に稼動していることを以下のようにテストします。

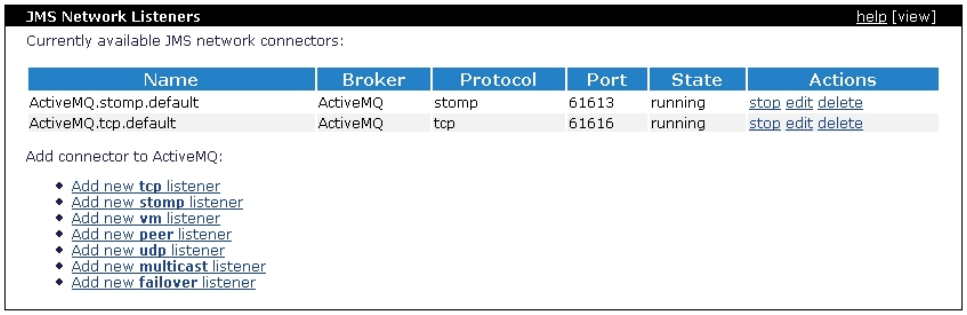
1. http://localhost:8080/console にアクセスしてGeronimo が接続可能であることを確認します。正常ならGeronimo管理コンソールが表示されるはずです。
2. http://localhostにアクセスしてHTTPdが接続可能であることを確認します。正常ならApache HTTPd が表示されるはずです。
3. http://localhost/console/ へアクセスしてHTTPd - Geronimo間でリクエストが転送されていることを確認します。正常ならGeronimo管理コンソールが表示されるはずです。URLの末尾に"/"を指定することをお忘れなく。この / を付け忘れた場合はJakarta Tomcat コネクター・モジュールが Not Found エラーを返します。

Back to Top

1.1.2.5. JMSサーバーの構成

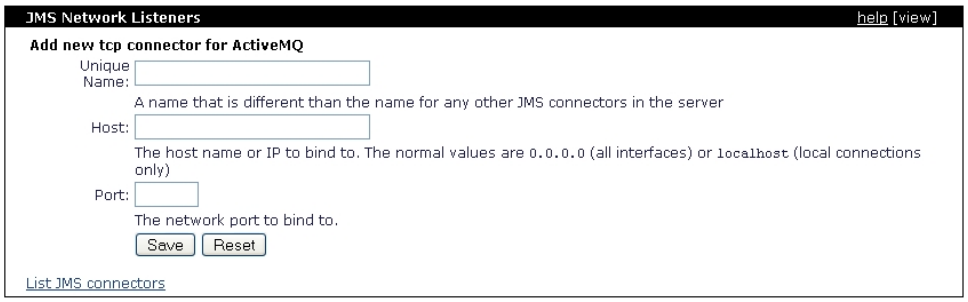
This page last changed on 4 21, 2008 by JAGUG.

JMSサーバーを構成する場合は、左側の Console Navigation メニューの JMS Server を選べば JMS Network Listener ポートレットが利用できます。JMS Server をクリックすると JMS Server Manager ポートレット上に、サーバー上のJMSブローカーとその状況が表示されます。下の図はこれらの2つのポートレットとそのオプションを示しています。



JMS Network Listeners ポートレットから新規にリスナーを定義したり既存のリスナーのステータス(stop,start,delete)を変更することができます。図中に、ActiveMQにコネクターを追加する際に利用可能な選択肢が示されています。

ActiveMQへ新規にコネクターを追加する場合には、該当のリンクをクリックします。下の図はこのコネクターを作成する際に必要となる様々なパラメーターを示しています。この作業はとても簡単でパラメーターは自明ですので、貴方の環境にあった設定値を入力して Save をクリックしてください。



追加したいリスナー毎に、同様のオプションが表示される点にご注意ください。

コネクション・ファクトリー、キュー、トピック、宛先(destination)等を構成するための方法の詳細は、次のセクション 1.1.4.3. JMSの構成でカバーされています。

1.1.2.6. JVM情報の表示

This page last changed on 4 21, 2008 by JAGUG.

System Property values for the Server JVM (または単に JVM) ポートレットは、導入済でGeronimoサーバーが使用しているJava ランタイムに関する詳細情報を表示します。情報は下記の6つのメイン・セクションにグループ分けして表示されます。

- Java
- Virtual Machine
- Operating System
- JVM Vendor
- User
- Etc

下記のように表示されます。

System Property values for the Server JVM

Java	
java.awt.graphicsenv	sun.awt.Win32GraphicsEnvironment
java.awt.printerjob	sun.awt.windows.WPrinterJob
java.class.path	D:\geronimo-tomcat6-jee5-2.0-M2\bin\server.jar
java.class.version	49.0
java.endorsed.dirs	C:\Java\jdk1.5.0_06\\jre\lib\endorsed D:\geronimo-tomcat6-jee5-2.0-M2\lib\endorsed
java.ext.dirs	C:\Java\jdk1.5.0_06\\jre/lib/ext D:\geronimo-tomcat6-jee5-2.0-M2/lib/ext
java.home	C:\Java\jdk1.5.0_06\jre
java.io.tmpdir	D:\geronimo-tomcat6-jee5-2.0-M2\var\temp

...

java.runtime.name	Java(TM) 2 Runtime Environment, Standard Edition
java.runtime.version	1.5.0_06-b05
java.specification.name	Java Platform API Specification
java.specification.vendor	Sun Microsystems Inc.
java.specification.version	1.5
java.util.prefs.PreferencesFactory	
java.vendor-	Sun Microsystems Inc.
java.vendor.url	http://java.sun.com/
java.vendor.url.bug	http://java.sun.com/cgi-bin/bugreport.cgi
java.version-	1.5.0_06
Virtual Machine	
java.vm.info	mixed mode
java.vm.name	Java HotSpot(TM) Client VM
java.vm.specification.name	Java Virtual Machine Specification
java.vm.specification.vendor	Sun Microsystems Inc.
java.vm.specification.version	1.0
java.vm.vendor	Sun Microsystems Inc.
java.vm.version	1.5.0_06-b05
Operating System	
os.arch	x86
os.name	Windows XP
os.version	5.1
Sun	
sun.arch.data.model	32
sun.boot.class.path	D:\geronimo-tomcat6-jee5-2.0-M2\lib\endorsed\xercesImpl-2.8.1.jar D:\geronimo-tomcat6-jee5-2.0-M2\lib\endorsed\yoko-rmi-spec-1.0-incubating-M2-486709.jar D:\geronimo-tomcat6-jee5-2.0-M2\lib\endorsed\yoko-spec-corba-1.0-incubating-M2-486709.jar C:\Java\jdk1.5.0_06\jre\lib\rt.jar C:\Java\jdk1.5.0_06\jre\lib\i18n.jar C:\Java\jdk1.5.0_06\jre\lib\sunrsasign.jar C:\Java\jdk1.5.0_06\jre\lib\jsse.jar C:\Java\jdk1.5.0_06\jre\lib\jce.jar C:\Java\jdk1.5.0_06\jre\lib\charsets.jar C:\Java\jdk1.5.0_06\jre\classes
sun.boot.library.path	C:\Java\jdk1.5.0_06\jre\bin
sun.cpu.endian	little
sun.cpu.isalist	pentium_pro+mmx pentium_pro pentium+mmx pentium i486 i386 i86
sun.io.unicode.encoding	UnicodeLittle
sun.java2d.fontpath	
sun.os.patch.level	Service Pack 2
User	
user.country	US
user.dir	D:\geronimo-tomcat6-jee5-2.0-M2\bin
user.home	C:\Documents and Settings\hcunico
user.language	en
user.name	hcunico
user.timezone	America/New_York
user.variant	

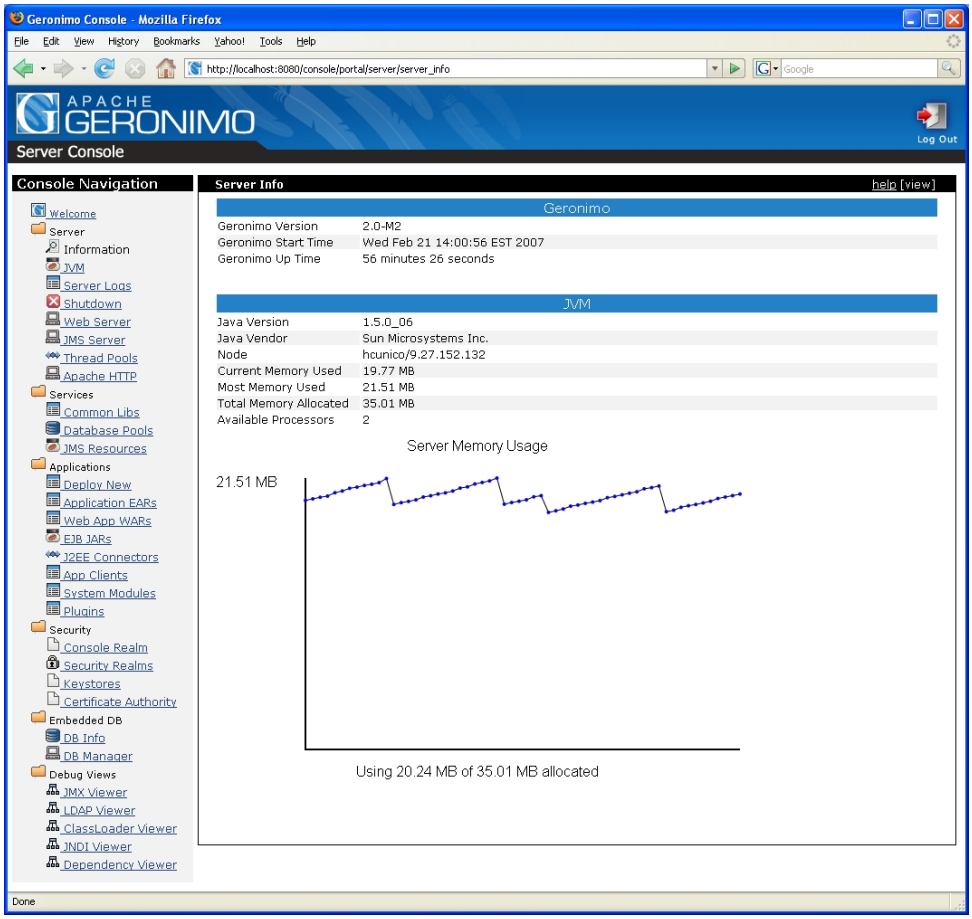
...

Etc	
awt.toolkit	sun.awt.windows.WToolkit
catalina.base	D:\geronimo-tomcat6-jee5-2.0-M2\var\catalina
catalina.home	var/catalina
catalina.useNaming	false
common.loader	${catalina.home}/lib ${catalina.home}/lib/*.jar
derby.storage.fileSyncTransactionLog	true
derby.system.home	D:\geronimo-tomcat6-jee5-2.0-M2\bin
duct tape	
file.encoding	Cp1252
file.encoding.pkg	sun.io
file.separator	\
java.naming.factory.initial	org.apache.xbean.naming.global.GlobalContextManager
java.naming.factory.url.pkgs	org.apache.openejb.core.ivm.naming:org.apache.xbean.naming
java.naming.provider.url	rmi://0.0.0.0:1099
java.rmi.server.RMIClassLoaderSpi	org.apache.geronimo.system.rmi.RMIClassLoaderSpiImpl
javax.security.jacc.PolicyConfigurationFactory.provider	org.apache.geronimo.security.jacc.GeronimoPolicyConfigurationFactory
javax.security.jacc.policy.provider	org.apache.geronimo.security.jacc.GeronimoPolicy
line.separator	
noBanner	true
openejb.naming	xbean
openejb.nobanner	true
org.apache.activeio.journal.active.lockMap	[D:\geronimo-tomcat6-jee5-2.0-M2\var\activemq\journal\control.dat]
org.apache.geronimo.base.dir	D:\geronimo-tomcat6-jee5-2.0-M2
org.apache.geronimo.home.dir	D:\geronimo-tomcat6-jee5-2.0-M2
org.apache.geronimo.log.ConsoleLogLevel	INFO
org.apache.geronimo.server.dir	D:\geronimo-tomcat6-jee5-2.0-M2
org.objectweb.howl.D:\geronimo-tomcat6-jee5-2.0-M2\.\var\txlog\howl_1.log.locked	true
org.objectweb.howl.D:\geronimo-tomcat6-jee5-2.0-M2\.\var\txlog\howl_2.log.locked	true
package.access	sun.,org.apache.catalina.,org.apache.coyote.,org.apache.tomcat.,org.apache
package.definition	sun.,java.,org.apache.catalina.,org.apache.coyote.,org.apache.tomcat.,org.a
path.separator	;
server.loader	
shared.loader	
sun.desktop	windows
sun.jnu.encoding	Cp1252
sun.management.compiler	HotSpot Client Compiler
tomcat.util.buf.StringCache.byte.enabled	true

1.1.2.7. サーバー状況のモニター

This page last changed on 4 21, 2008 by JAGUG.

Geronimo管理コンソールの最初のオプションとして Information ポートレットが利用可能です。サーバーの連続稼動時間やJVMが使っているリソースに関する情報を表示します。このポートレットはメモリーの利用状況をリアルタイムでグラフ表示することもできます。

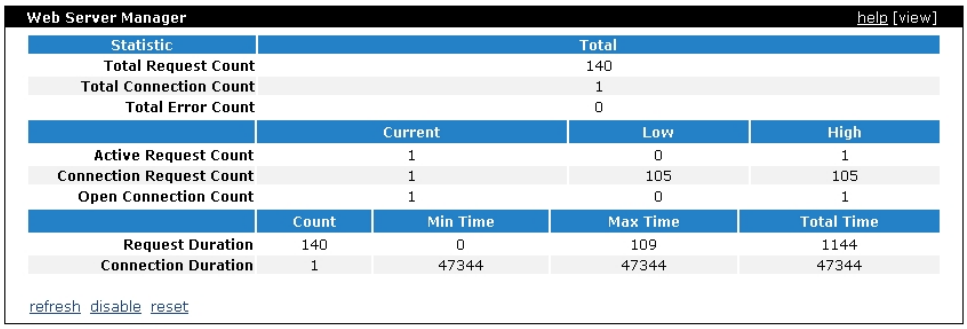


1.1.2.8. パフォーマンスのモニター

This page last changed on 4 21, 2008 by JAGUG.

Webサーバーのパフォーマンスをモニターするには、左側の Console Navigation メニューの Web Server を選べば Web Server Manager ポートレットが利用できます。現在、この機能が使えるのはApache GeronimoのJettyディストリビューションだけです。このポートレットはデフォルトではenabled状態ではありませんので、統計情報の収集を開始する場合は enable ボタンをクリックしてください。

enableにすることで、リクエストの数、現在のコネクションとリクエストの遅延などについての情報の収集が始まります。以下の図は収集されるすべての値を示しています。



Statistic	Total
Total Request Count	140
Total Connection Count	1
Total Error Count	0

	Current	Low	High
Active Request Count	1	0	1
Connection Request Count	1	105	105
Open Connection Count	1	0	1

	Count	Min Time	Max Time	Total Time
Request Duration	140	0	109	1144
Connection Duration	1	47344	47344	47344

下の3つのリンク、refresh は現在の統計をリフレッシュし、disable はポートレットを無効化して新たなデータ収集を停止し、reset は収集されたデータをリセットします。

> ⚠ Geronimo管理コンソールからのすべてのリクエスト(例えば統計のリフレッシュ)もこのデータ収集の値に反映される点にご注意ください。

1.1.2.9. サーバーの開始と停止

This page last changed on 4 21, 2008 by JAGUG.

コマンド行からサーバーを開始するにはいくつかの方法があります。コマンド行ウインドウまたはターミナル上で、ディレクトリーを <geronimo_home>/bin に変更します。その場所から以下のいずれかをタイプします。

`java -Djava.endorsed.dirs=lib/endorsed -javaagent:bin/jpa.jar -jar bin/server.jar` or simply **`startup`**

または

`geronimo run` or **`geronimo start`**

1.9.7. startup と 1.9.4. geronimo コマンドのすべてのオプションは 1.9. ツールとコマンド セクションを参照してください。

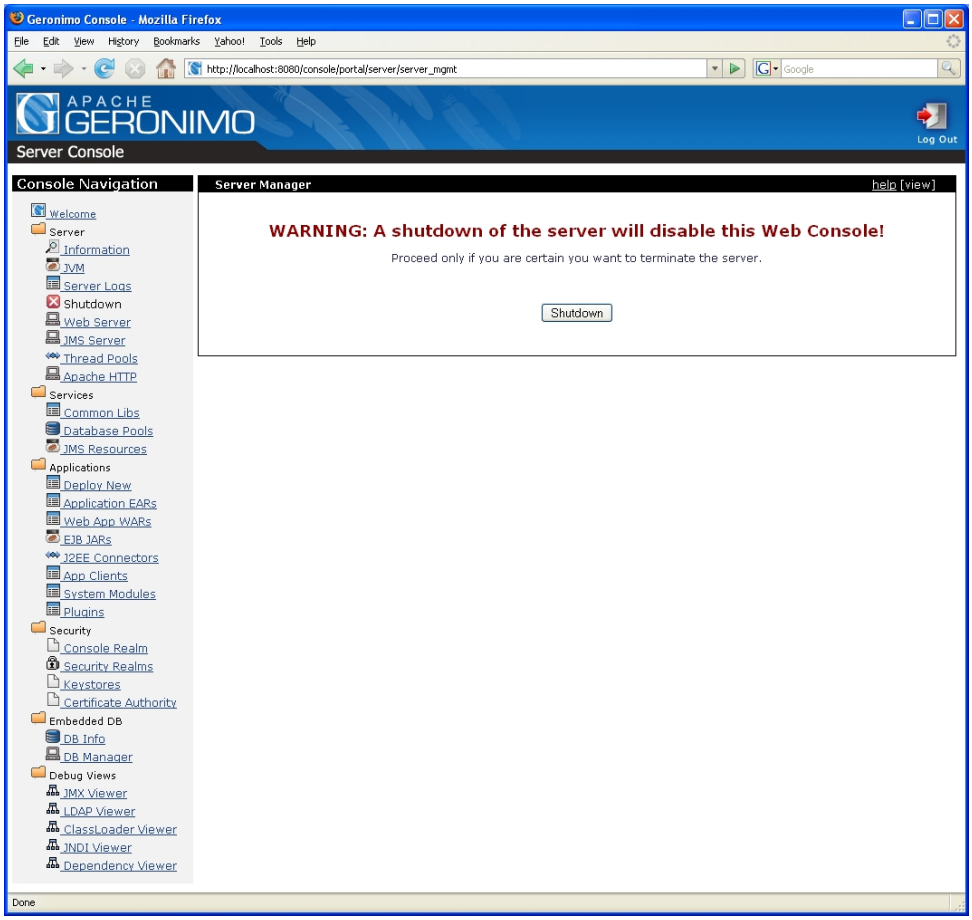
サーバーを停止するには新たなコマンド行またはターミナル上でディレクトリーを　<geronimo_home>/bin に変更して、**`shutdown`** コマンドを実行します。その場合、プロンプトからusernameとpasswordの入力を促される場合もあります。別の方法として、プロセスをkillするにはGeronimoが稼動しているターミナルで Ctrl+C を押します。1.9.6. shutdown と 1.9.4. geronimo コマンドのすべてのオプションは 1.9. ツールとコマンド セクションを参照してください。

サーバーが稼動中であれば、リモートからサーバーを停止するために 1.6. Geronimo 管理コンソール を利用する方法もあります。稼動しているサーバーに対しWebブラウザーを開き、コンソールにアクセスするために以下のURLを指し示します。

http://localhost:8080/console

Geronimo管理コンソールにログインし、左のConsole Navigation menuから

Shutdown をクリックします。



Server Managerポートレットで Shutdown をクリックすると、サーバーの停止の確認のプロンプトが表示されます。サーバーを停止してもよい場合は OK をクリックします。停止の結果管理コンソールとの接続が失われるので、再始動は再度ターミナルかコマンド行ウインドウから行う必要がある点は申すまでもないでしょう。

1.1.3. セキュリティの構成

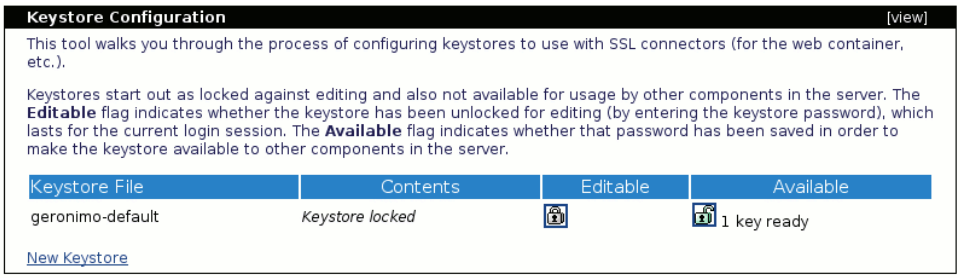
This page last changed on 4 21, 2008 by JAGUG.

このセクションでは一般的なセキュリティ関連のタスク、例えばユーザーやグループの追加と削除、電子証明書の取り扱い、様々なレルムや認証方法を使用してセキュリティ・レベルを高める方法などを説明します。

1.1.3.1. 証明書の管理

This page last changed on 4 21, 2008 by JAGUG.

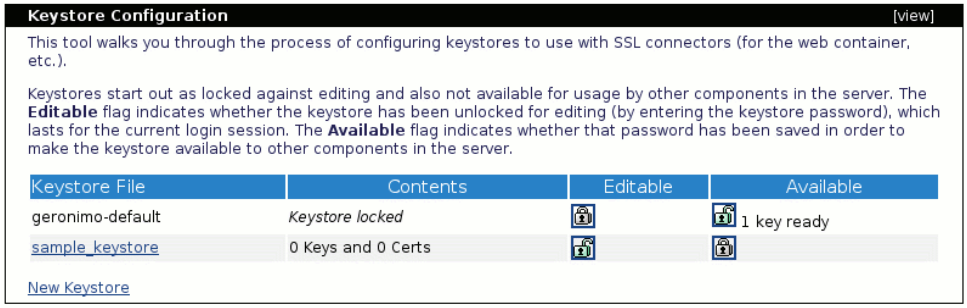
左側の Console Navigation メニューで Keystore を選んで表示される Keystore Configuration ポートレットを使用するとSSL証明書を管理できます。このポートレットから既存の証明書をインポートできますし、新規の証明書リクエストを作成することもできます。



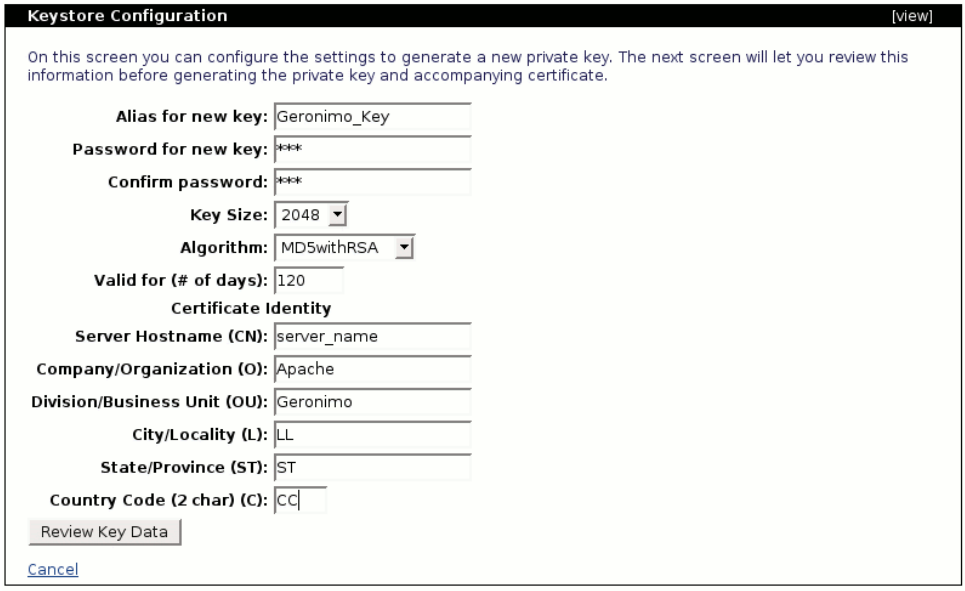
Geronimoの証明書は <geronimo_home>¥var¥security¥keystores¥geronimo-default 配下のキー・ストアに格納されています。

もしもデフォルトで提供されているもの以外の別の証明書を使いたい場合には、New Keystore をクリックすれば作成できます。キー・ストアの名前とパスワードの入力を求めるプロンプトが表示されますので、値を入力のうえ、Create Keystore をクリックします。例えば、各々sample_keystore、passwordと入力したものとしましょう。

下図のように、今作成したばかりのキー・ストアには証明書も鍵も含まれていません。またデフォルトではキー・ストアはロックされており、Available カラムが施錠された状態になっている点にもご注目ください。いったん証明書を作成したら、その証明書を利用可能にするために鍵の絵をクリックしてください。するとキー・ストアと証明書用のパスワードの入力を求めるプロンプトが表示されます。



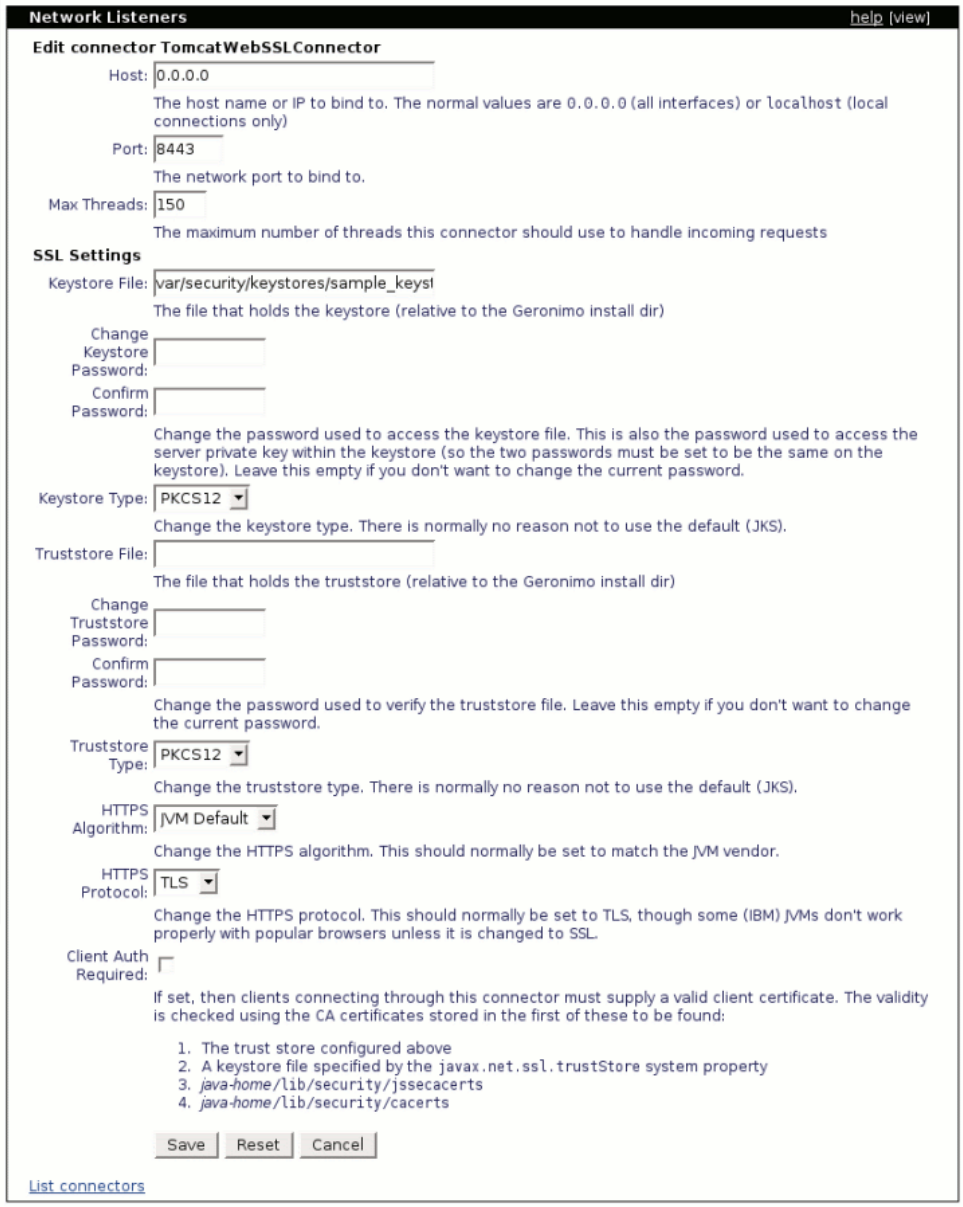
プライベート・キーを作成するには、いましがた作成したキー・ストアのキーをクリックし、さらに Create Private Key をクリックします。次に、各々のフィールドに適切な値を入力してください。



Review Key Data をクリックし、次に Generate Key をクリックします。すると、Keystore Configurationポートレット上に今生成したばかりのキーが表示されます。

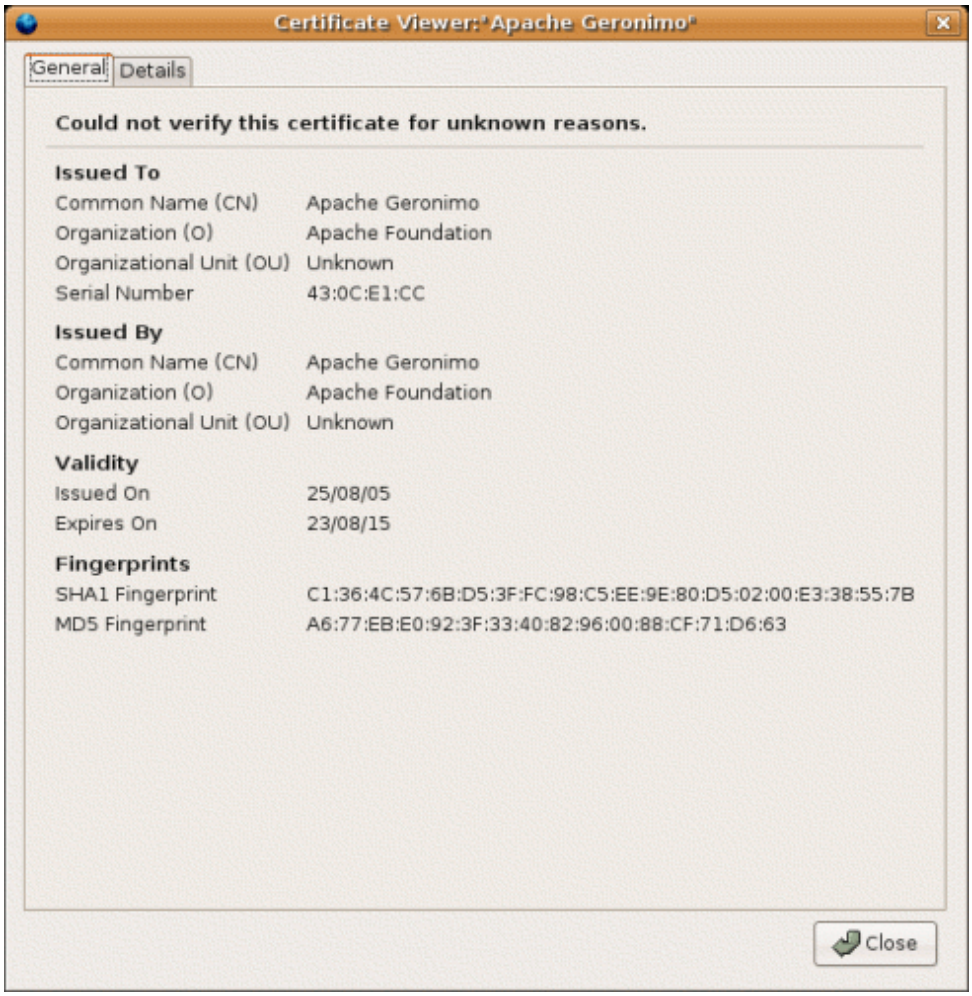
ここで作成した証明書は、「新規HTTPSリスナーの追加」のセクションで説明されているHTTPSコネクターの構成の際に使用できます。鍵の絵をクリックして証明書とキー・ストアを利用可能にすることをお忘れなく。この例では、既存のTomcatWebSSLConnectorを変更し、新しいキー・ストアを生成して、構成を保管しました。

この構成を有効にするためには、コネクターを再始動する必要があります。今しがた更新したネットワーク・リスナー、今回はTomcatWebSSLConnectorですが、に関連付けられている stop のリンクをクリックし、次に start をクリックします。これで該当のコネクターは新しいキー・ストアと証明書を使うようになります。



ここで、ブラウザー上で特定のポートを指定すればサーバーが前述の証明書を使用していることを確認できます。この例では既存のSSLコネクターを使用しているので、ブラウザー上で下記のように入力します。

https://localhost:8443/console


Certificate Viewer:"Apache Geronimo"
General
Details
Could not verify this certificate for unknown reasons.
Issued To
Common Name (CN) Apache Geronimo
Organization (O) Apache Foundation
Organizational Unit (OU) Unknown
Serial Number 43:0C:E1:CC
Issued By
Common Name (CN) Apache Geronimo
Organization (O) Apache Foundation
Organizational Unit (OU) Unknown
Validity
Issued On 25/08/05
Expires On 23/08/15
Fingerprints
SHA1 Fingerprint C1:36:4C:57:6B:D5:3F:FC:98:C5:EE:9E:80:D5:02:00:E3:38:55:7B
MD5 Fingerprint A6:77:EB:E0:92:3F:33:40:82:96:00:88:CF:71:D6:63
Close

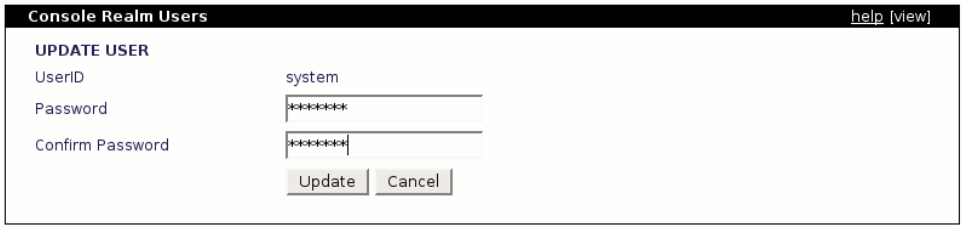
1.1.3.2. ユーザーとグループの管理

This page last changed on 4 21, 2008 by JAGUG.

Geronimo管理コンソールを使うか、または直接構成ファイルを変更すれば、ユーザーやグループを追加できます。まずはGeronimoがデフォルトで提供しているレルムを使って簡単にはじめましょう。次に別の構成やレルムを調べてから、必要ないくつかのトピックを再び取り上げます。

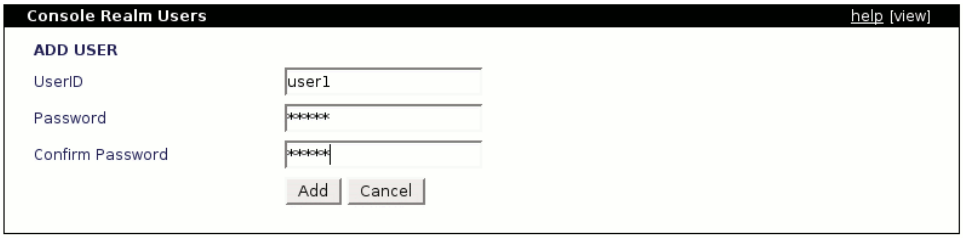
Geronimo管理コンソールの左側の Console Navigation メニューから Console Realm ポートレットを使えば、ユーザーとグループを管理できます。ここでは2つのポートレットが見えるはずです。下記の図にあるように、ひとつはユーザーを管理するためのもので、もうひとつはグループを管理するものです。

ユーザーのパスワードを変更するには、Console Realm User ポートレット上で変更したいユーザーの横の Details をクリックします、するとユーザーIDとパスワードが出てきますのでプロファイルを更新できます。



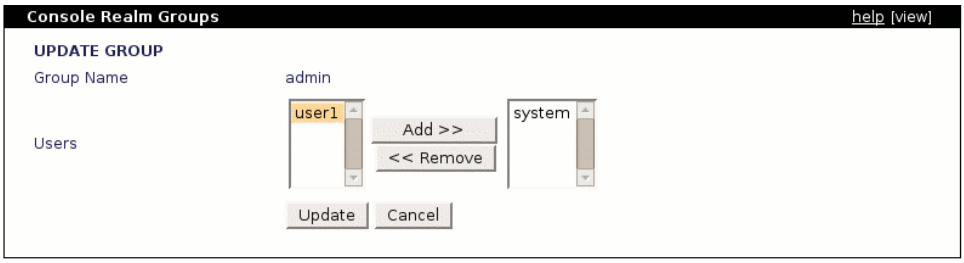
ユーザーを削除するには同様に Delete をクリックします。削除の確認を求めるプロンプトが表示されるので OK をクリックします。

ユーザーを新規に追加するには Create New User をクリックします。ユーザーIDをパスワード(2回)を求めるプロンプトが表示されるので必要な値を入力して Add をクリックします。



ユーザーを新規に作成したらグループに追加できます。デフォルトではグループ admin が利用可能で、ユーザー system がそのグループに属しています。Admin グループの横の Details をクリックすると右側のウインドウにはユーザー system がリストされ、左側のウインドウには他の利用可能なユーザーIDがリストされます。

このグループに新たにユーザーを追加する場合は、まずユーザーを選択してから Add >> をクリックし、次に Update をクリックします。



新規にグループを作成するには Create New Group をクリックします。前項でユーザーを扱った場合ととてもよく似た操作です。ユーザーをグループに追加するためのプロンプトが表示される以外に、今回はグループの名前を指定する必要があります。まず新たなグループ名を入力し、ユーザーを追加したら最後に Add をクリックして完了です。

Console Realm Users ポートレットと Console Realm Groups ポートレットで貴方が行った変更は users.properties と groups.properties 2つの別々のファイルに反映されます。これらのファイルは <geronimo_home>¥var¥security ディレクトリーに存在します。

ユーザーやグループの管理はこれらのファイルを直接変更することでも同様に可能です。

- `users.properties`
- `groups.properties`

`users.properties` は <user_name>=<password> の形式です。groups.properties は <group_name>=<user_name> の形式です。詳細は下記のサンプルを参照してください。

users.properties

```
system=manager
user2=password
user1=password
```

今はデフォルトの基本的な構成を使っているので、ユーザーIDやパスワードが単純なテキストとして格納されているのがお分かりでしょう。このファイルを使用してパスワードの追加、除去、変更を行えます。

groups.properties

```
admin=system,user1
users=user2
```

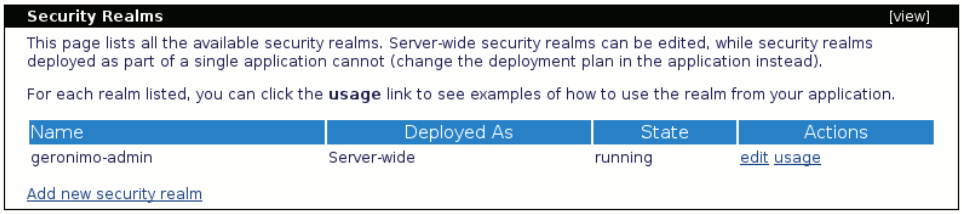
ユーザーの場合と同様、グループの追加・除去、及びれらのグループへのユーザーの追加・除去は **`groups.properties`** で行います。

ユーザーの名前とパスワードに加え、このセクションで言及されたファイル群および全てのセキュリティの構成は geronimo-properties-realm レルムで定義しますが、それらは 1.1.3.3. セキュリティ・レルムの管理 のセクションで取り扱います。

1.1.3.3. セキュリティ・レルムの管理

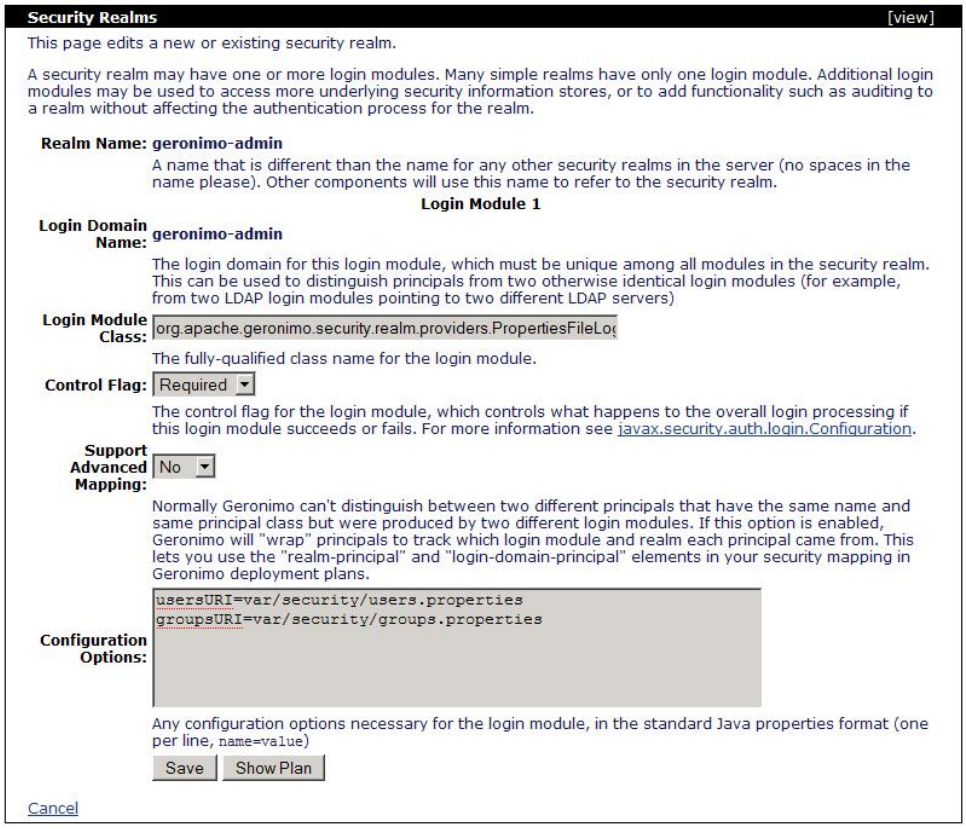
This page last changed on 4 21, 2008 by JAGUG.

Geronimo管理コンソールの左側の Console Navigation メニューの Security Realms ポートレットを使用して、セキュリティ・レルムを管理できます。このポートレットで新たなセキュリティ・レルムを追加したり既存のレルムを編集できます。レルムを除去するには通常はDeployerツールのコマンド行オプションを使用して行います。



このポートレットにはすべての利用可能なセキュリティ・レルムがリストされます。デフォルトでは、プロパティ・ファイルを使ってユーザーを認証するために使用されるGeronimoのレルムは geronimo-admin です。

既存のレルム(この場合にはgeronimo-adminです)を編集する際には以下のような画面が表示されます。realm nameやlogin domain nameは変更できない点にご注意ください。



以下の例はこのレルムによって生成されたデプロイメント・プランを示しています。

geronimo-properties-realm

```
<module xmlns="http://geronimo.apache.org/xml/ns/deployment-1.2">
<environment>
<moduleId>
<groupId>console.realm</groupId>
<artifactId>geronimo-admin</artifactId>
<version>1.0</version>
<type>car</type>
</moduleId>
<dependencies>
<dependency>
```

```
<groupId>org.apache.geronimo.configs</groupId>
<artifactId>j2ee-security</artifactId>
<type>car</type>
</dependency>
</dependencies>
</environment>
<gbean name="geronimo-admin" class="org.apache.geronimo.security.realm.GenericSecurityRealm"
xsi:type="dep:gbeanType"
xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
<attribute name="realmName">geronimo-admin</attribute>
<reference name="ServerInfo">
<name>ServerInfo</name>
</reference>
<xml-reference name="LoginModuleConfiguration">
<log:login-config xmlns:log="http://geronimo.apache.org/xml/ns/loginconfig-2.0">
<log:login-module control-flag="REQUIRED" wrap-principals="false">
<log:login-domain-name>geronimo-admin</log:login-domain-name>
<log:login-module-class>
org.apache.geronimo.security.realm.providers.PropertiesFileLoginModule
</log:login-module-class>
<log:option name="usersURI">var/security/users.properties</log:option>
<log:option name="groupsURI">var/security/groups.properties</log:option>
</log:login-module>
</log:login-config>
</xml-reference>
</gbean>
</module>
```

前述のとおり、これはデフォルトのプランで、プロパティ・ファイルを使用したセキュリティ・レルムです。新たにレルムを作成する際は以下の利用可能なレルムの種類から選択する必要があります。:

- Certificate Properties File Realm
- Database (SQL) Realm
- LDAP Realm
- Properties File Realm
- Other

最後のオプションは、貴方の環境がいずれにも当てはまらない場合にカスタマイズしたレルムを作成するためのものです。

デフォルトの Properties File Real について説明したので、以降はその他の選択肢について説明します。

- 1.1.3.3.1. 証明書プロパティ・ファイル・レルム
- 1.1.3.3.2. データベース(SQL)レルム
- 1.1.3.3.3. LDAPレルム

1.1.3.3.1. 証明書プロパティ・ファイル・レルム

This page last changed on 4 21, 2008 by JAGUG.

このレルムの種類はユーザーのWebアプリケーションに対する認証を構成するために使用します。そのために、まずGeronimoがカスタムSSLリスナーを使用するように構成する必要があります。また、SSLキーとキー・ストアを構成する必要もあります。以下のセクションではこれらのモジュールのそれぞれを構成する方法を、順を追って説明します。

- キー・ストアと証明書の作成
- 証明要求(CSR)の作成とCA応答のインポート
- 信頼された証明書のインポート
- クライアント認証を使用するHTTPリスナーへの追加
- Install certificate on client

キー・ストアと証明書の作成

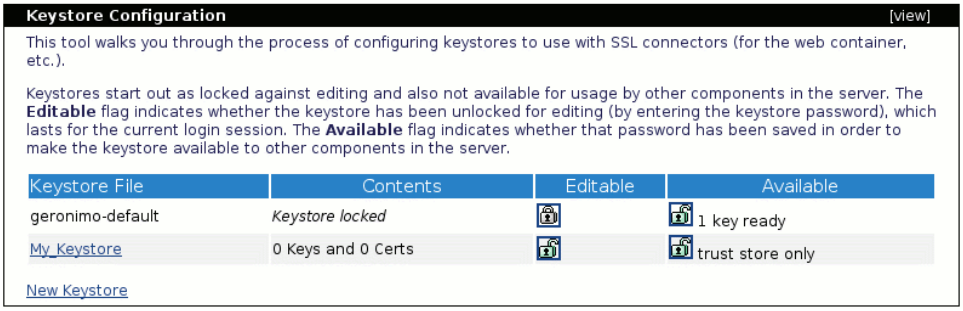
当構成のためには、まず新規にキー・ストアとプライベート・キーと証明要求(CSR)を作成し、次にCAからの応答をインポートします。

すでに「1.1.3.1. 証明書の管理」のセクションでキー・ストアとプライベート・キーの作成方法は述べましたので、このセクションではCSRを生成しCAから応答をインポートする方法を述べ、全体を明らかにします。

Geronimoのキー・ストアは <geronimo_home>¥var¥security¥keystores ディレクトリーに格納されており、デフォルトのキー・ストアであるgeronimo-defaultが導入時に提供されています。今回は新規にキー・ストアを作成してみます。

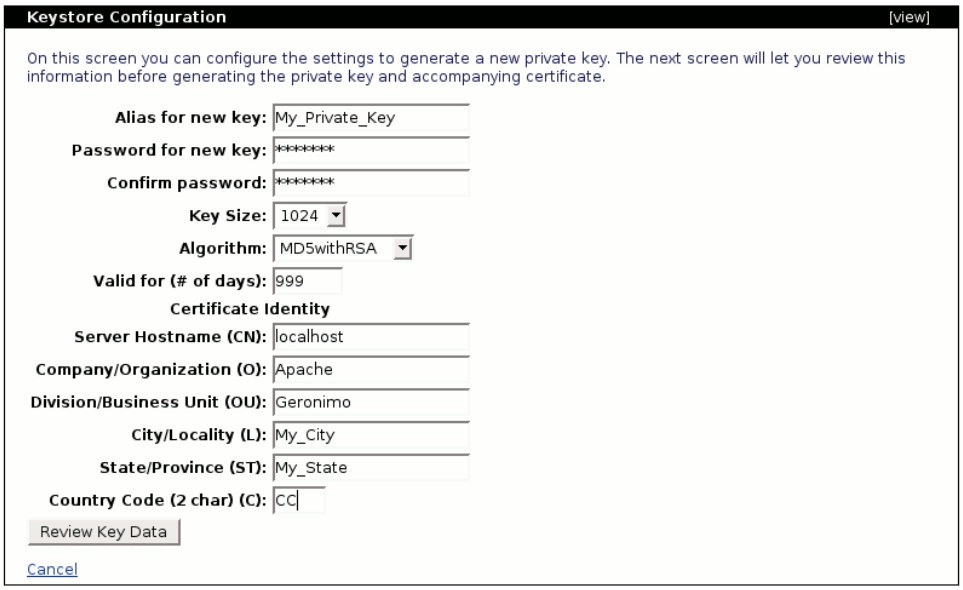
Geronimoの管理コンソールから Keystores をクリックして Keystore Configuration ポートレットにアクセスします。

New Keystore をクリックし、新しいキー・ストアの名前とパスワードを指定後、 Create Keystore をクリックしてください。この例では各々My_Keystoreとpasswordを使います。

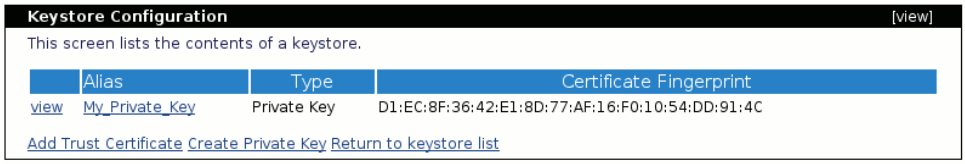


今しがた作成したキー・ストア・ファイルをクリックし、適切なリンクをクリックしてプライベート・キーを作成します。

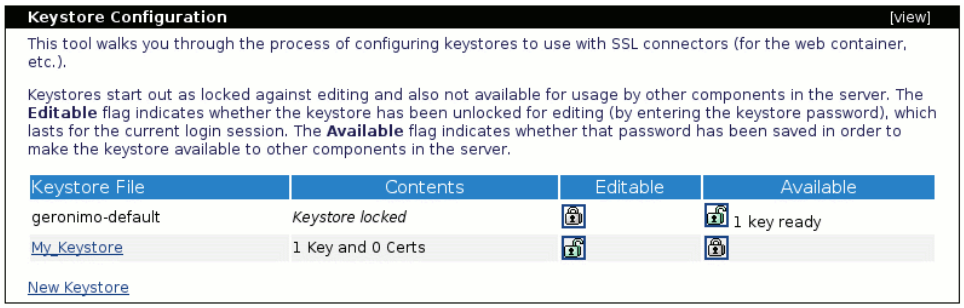
適切なデータを入力のうえ、Review Key Data をクリックしてください。



値が正しいことを確認したら Generate Key をクリックします。

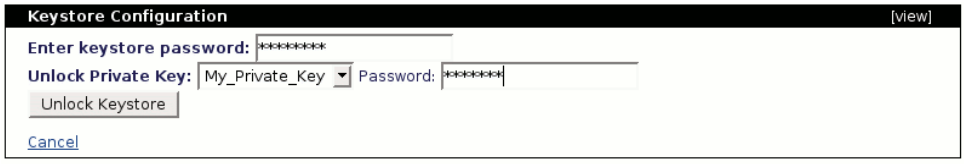


新規にプライベート・キーを作成した直後は、このキーは自動的にロックされています。つまり、参照か削除しかできません。Certificate Signing Request (CSR)を作成するにはキーのロックを解除する必要があります。そのためには、Return to keystore list をクリックしてください。



プライベート・キーをアンロックするために

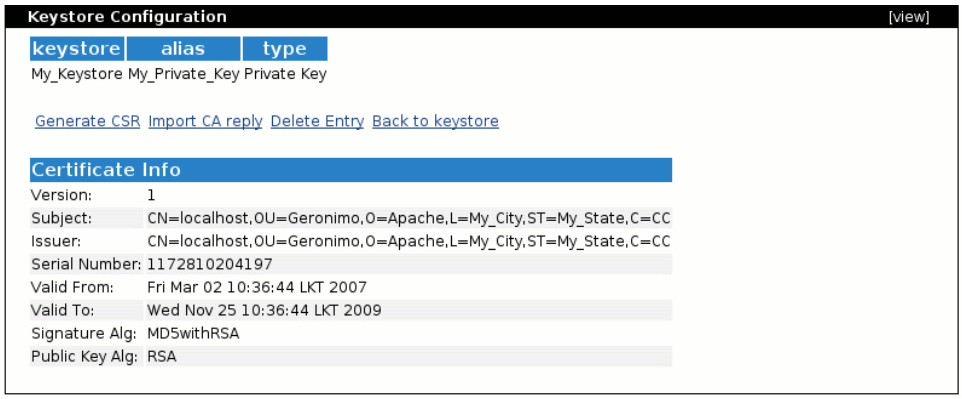
をクリックします。当該キー・ストアとプライベート・キー用のパスワードを求めるプロンプトが表示されます。



Unlock Keystore をクリックします。

証明要求(CSR)の作成とCA応答のインポート

プライベート・キーをアンロックしたので続いてCSRを作成します。Keystore Configuration ポートレットで作成したキー・ストアをクリックし、現在の中身を表示します。この例ではプライベート・キーはひとつだけです。view または現在のプライベート・キーに該当するリンクをクリックし、詳細と追加のアクションを表示します。



Generate CSR をクリックすると、下記に示されているように証明要求が表示されるはずです。

これは PKCS10 の証明要求です。このテキストをコピーして、CAに送付するためにフラット・テキストに貼り付けます。

csr.txt

```
-----BEGIN CERTIFICATE REQUEST-----
MIIBqDCCARECAQAwajESMBAGA1UEAxMJbG9jYWxob3N0MREwDwYDVQQLEwhHZXJvbmltbz
EPMA0GA1UEChMGQXBhY2hlMRAwDgYDVQQHDAdNeV9DaXR5MREwDwYDVQQIDAhNeV9TdGF0
ZTELMAkGA1UEBhMCQ0MwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBANTOM2j05ACU4N
49B4l51oxFSQX1SaX2+MBCWEpMILWriYxpBYRukMjyOOLBqreyUj6nv64j0qm1HgnOeYER
2fRtk6ERBGGRG//HprVBZzXFV5T/kwB4Ocg8NKQFWibLtT9MSjQyYBy0NGRgGL8krn+LDL
/YucueG+NbPfDzKD4xAgMBAAEwDQYJKoZIhvcNAQEEBQADgYEAp0g6oJ2WLllBmXpCnbcd
iyHtWAtFCODRKJaTzC09N+/0s+BugiGOxTGLB65C0xbIeumSog8Yxy26LFTtcvIP1lC7wg
V1e1KaJBTuuop7jOYFo4Tpx3oCL7ZJ6BtHrx0vSNlOdnkY6y+ZUPmQcWJq6lLP85NWu9N5
B94KI7U/QpM=
-----END CERTIFICATE REQUEST-----
```

Back をクリックすればprivate key details ポートレットに戻れます。

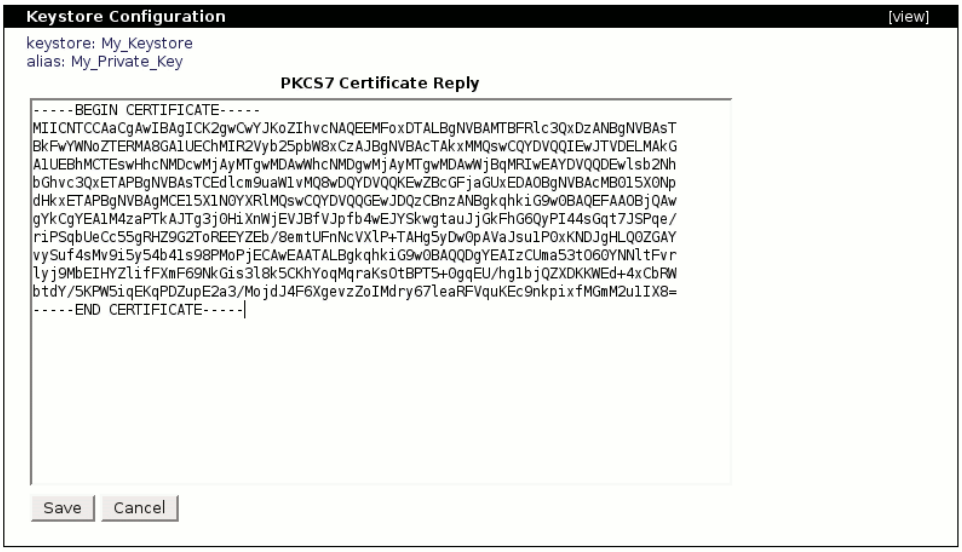
この例では独自に自分で作ったCAを使いますので、このテストでは自分で自分の証明書に署名でき、標準的な手順を変更する必要はありません。もし貴方がCSRをCAに送付した場合は、CAは、CAによって署名された内容を含むファイルを別途返送してくるはずです。

csr_ca_reply.txt

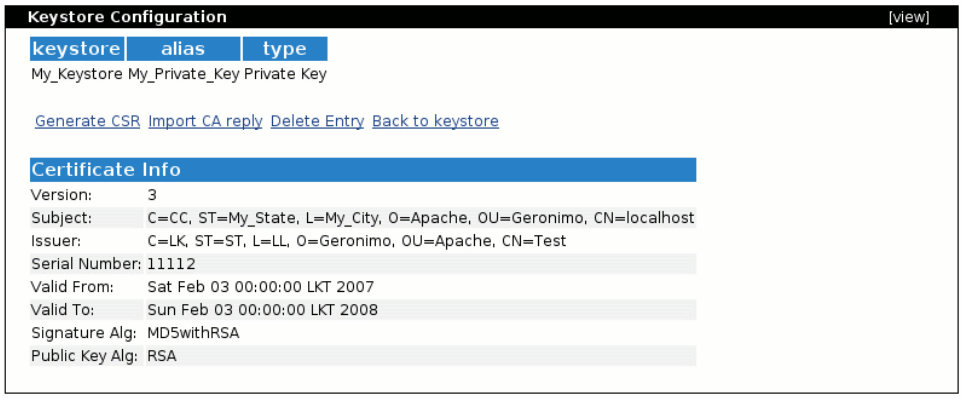
```
-----BEGIN CERTIFICATE-----
MIICNTCCAaCgAwIBAgICK2gwCwYJKoZIhvcNAQEEMFoxDTALBgNVBAMTBFRlc3QxDzANBgNVBAsT
```

```
BkFwYWNoZTERMA8GA1UEChMIR2Vyb25pbW8xCzAJBgNVBAcTAkxMMQswCQYDVQQIEwJTVDELMAkG
A1UEBhMCTEswHhcNMDcwMjAyMTgwMDAwWhcNMDgwMjAyMTgwMDAwWjBqMRIwEAYDVQQDEwlsb2Nh
bGhvc3QxETAPBgNVBAsTCEdlcm9uaW1vMQ8wDQYDVQQKEwZBcGFjaGUxEDAOBgNVBAcMB015X0Np
dHkxETAPBgNVBAgMCE15X1N0YXRlMQswCQYDVQQGEwJDQzCBnzANBgkqhkiG9w0BAQEFAAOBjQAw
gYkCgYEA1M4zaPTkAJTg3j0HiXnWjEVJBfVJpfb4wEJYSkwgtauJjGkFhG6QyPI44sGqt7JSPqe/
riPSqbUeCc55gRHZ9G2ToREEYZEb/8emtUFnNcVXlP+TAHg5yDw0pAVaJsu1P0xKNDJgHLQ0ZGAY
vySuf4sMv9i5y54b41s98PMoPjECAwEAATALBgkqhkiG9w0BAQQDgYEAIzCUma53tO60YNNltFvr
lyj9MbEIHYZlifFXmF69NkGis3l8k5CKhYoqMqraKsOtBPT5+0gqEU/hg1bjQZXDKKWEd+4xCbRW
btdY/5KPW5iqEKqPDZupE2a3/MojdJ4F6XgevzZoIMdry67leaRFVquKEc9nkpixfMGmM2u1IX8=
-----END CERTIFICATE-----
```

private key detailsポートレットから Import CA reply をクリックします。証明応答ウインドウで事前に埋まっているテキストはすべて削除して、CAからの応答ファイルのテキストを貼り付けてから Save をクリックします。



CAからの応答を保管すると、証明書が別の Issuer になっていることに気が付かれることでしょう。Back to keystore をクリックして、次に Return to keystore list をクリックしてください。



信頼された証明書のインポート

クライアント認証を有効にするには貴方のCSRが信頼された証明書であると署名済のCAをインポートする必要があり、この処理は一度だけ行います。そのCAは、署名済のCSRをCA自身とは別の証明書として提供しなくてはなりません。この例では自己証明書を使いますので、以下のようにCA証明書を生成しました。

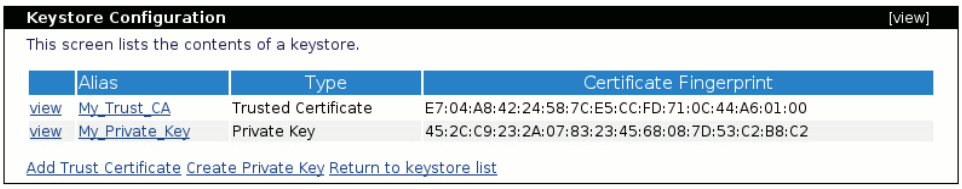
My_Own_CA_Certificate.txt

```
-----BEGIN CERTIFICATE-----
MIICJTCCAZCgAwIBAgICK2cwCwYJKoZIhvcNAQEEMFoxDTALBgNVBAMTBFRlc3QxDzANBgNVBAsT
BkFwYWNoZTERMA8GA1UEChMIR2Vyb25pbW8xCzAJBgNVBAcTAkxMMQswCQYDVQQIEwJTVDELMAkG
A1UEBhMCTEswHhcNMDcwMjAyMTgwMDAwWhcNMDgwMjAyMTgwMDAwWjBaMQ0wCwYDVQQDEwRUZXN0
```

```
MQ8wDQYDVQQLEwZBcGFjaGUxETAPBgNVBAoTCEdlcm9uaW1vMQswCQYDVQQHEwJMTDELMAkGA1UE
CBMCU1QxCzAJBgNVBAYTAkxLMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCUZl1e1eTKLoh0
15vfYqqvhk6Iviva7BWQxZ6mOV9Ye2mii37Btmxajnngz0jKfiwHKqWRQBp6CUzbd9gfZrz2go9g
TwsUBWQwSf6iVypKX1q0Y4WhtTwLcEx78Lx5XN1YCqk34pn4by26SJiHdugs7/ClOiIlcpCt9QVa
Q9BH7wIDAQABMAsGCSqGSIb3DQEBBAOBgQAnmoT/dLvJa7jGstvZJLrsWtMwWQNVJ1ZQmbrDGq9u
oFnkAH1mGHIDbaz2avy/wotHJUIysGBlDP0btk5GVskl45EG/feWHLgCVmqwf3NkdRdLl+CznBBJ
KCC5tINbcI6GqXsbO8hhjIrOGweNyV1653WEvZiQVuMYaHTnGNx+RA==
-----END CERTIFICATE-----
```

Keystore Configurationポートレットで貴方の作成したキー・ストア・ファイルをクリックし、次に Add Trust Certificate をクリックします。Trusted Certificate ウインドウに事前に含まれていた中身はすべて削除して、CA証明書からの内容を貼り付けた後、その証明書に別名を追加します。

Review Certificate をクリックして、それから Import Certificate をクリックします。今インポートしたばかりの信頼された証明書が見えるはずです。

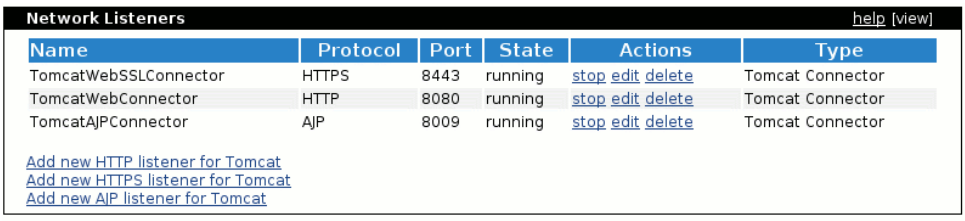


クライアント認証を使用するHTTPリスナーへの追加

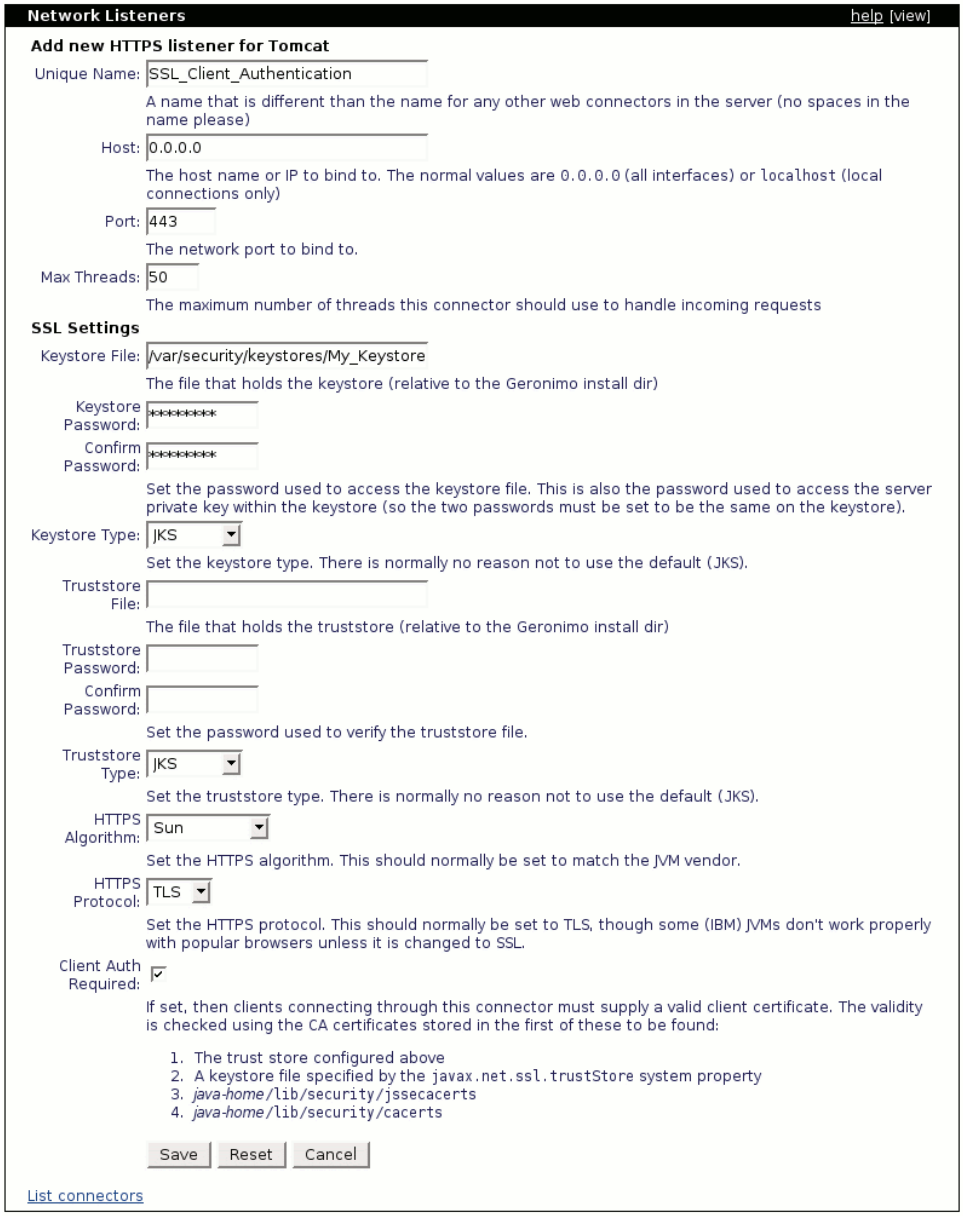
Apache Geronimoには事前にポート8443でHTTPSリスナーが定義済ですが、クライアント認証要用には構成されていません。この例では新規にHTTPSリスナーを追加し、前のステップで作成してインポートした証明書を使ってクライアント認証を要求するように構成を行います。

この例ではGeronimoのTomcatディストリビューションを使います。Jettyディストリビューションを使う場合でも手順は同じですが、名前やリンクが若干異なっているかもしれませんのでご注意ください。

Geronimo管理コンソールから Web Server をクリックしNetwork Listenerポートレットにアクセスします。



Network Listenerポートレットから Add new HTTPS listener for Tomcat をクリックします。



フィールドへ適切なデータを入力し、Save をクリックします。この例ではキーストアのみを指定し、truststoreは使用しません。(訳注：原文にtrustoreはtruststoreの意と思われます)　キーストアのファイル・パスを指定する際は var/security/keystores/<your_keystore> のような形式で追加しますが、このパスはGeronimoの導入されたホーム・ディレクトリーへの相対パスです。

Client Auth Required のチェックボックスを選択することで、有効なクライアント証明書を提示したクライアントのみが暗号化コネクションを確立するようHTTPSリスナーに対し指示します。クライアント証明書は、以下のいずれかのロケーションに格納されたCA証明書と（順番に）検証されます。

1. 上記で構成したトラスト・ストア
2. javax.net.ssl.trustStore system propertyに記載されているキーストア・ファイル
3. java-home/lib/security/jssecacerts
4. java-home/lib/security/cacerts

このHTTPSネットワーク・リスナー構成を保存するとこの構成が自動的に開始したことがステータスから見て取れます。もし貴方のブラウザーでこのポートにアクセスを試みても失敗しますが、それはまだ貴方のクライアントが有効な証明書を使うように構成されていないためです。

1.1.3.3.2. データベース(SQL)レルム

This page last changed on 4 21, 2008 by JAGUG.

このセクションではデータベースを使ってユーザー名とパスワードを検証・検索する方法を説明します。

この例では、組み込まれているDerbyデータベースを使って SecurityDatabase という名前の新しいデータベースを作ります。以下のステップではデータベースとテーブルを作り、サンプル・データをロードし、コネクション・プールを作る手順を要約します。データベースへコネクション・プールを定義するための詳細な方法はConfiguring database poolsセクションに記載されています。

データベースの作成とサンプル・データのロード

- 左側の Console Navigation メニューから Database Manager をクリックします。
- Create DB: フィールドに SecurityDatabase と入力し、Create をクリックします。
- Use DB: プルダウン・メニューから SecurityDatabase を選択し、以下のコマンドを入力して Run SQL をクリックします。

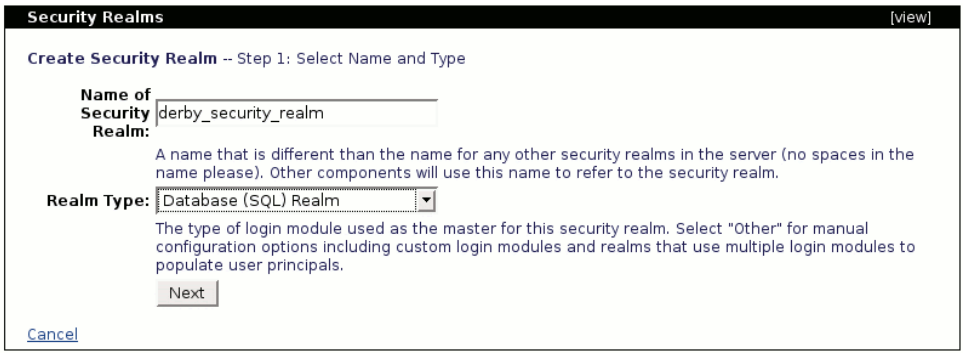
```
create table users
(username varchar(15),
password varchar(15));
create table groups
(username varchar(15),
groupname varchar(15));
insert into users values('userone','p1');
insert into users values('usertwo','p2');
insert into users values('userthree','p3');
insert into groups values('userone','admin');
insert into groups values('usertwo','admin');
insert into groups values('userthree','user');
```

コネクション・プールの作成

- 左側の Console Navigation メニューから Database Pools をクリックします。
- Using the Geronimo database pool wizard をクリックします。
- データベース・プール名に SecurityDatabasePool と入力します。
- database pool typeプルダウン・メニューから Derby embedded XA を選択し、Next をクリックします。
- Driver JAR スクロール・ボックスから org.apache.geronimo.configs/system-database/2.0.1/car を選択します。
- DB user name と passwordは 空白 にします。
- Databaseに SecurityDatabase と入力します。
- Deploy をクリックします。

新規セキュリティ・レルムの追加

新規にセキュリティ・レルムを作成するには Security Realms ポートレットで Add new security realm をクリックします



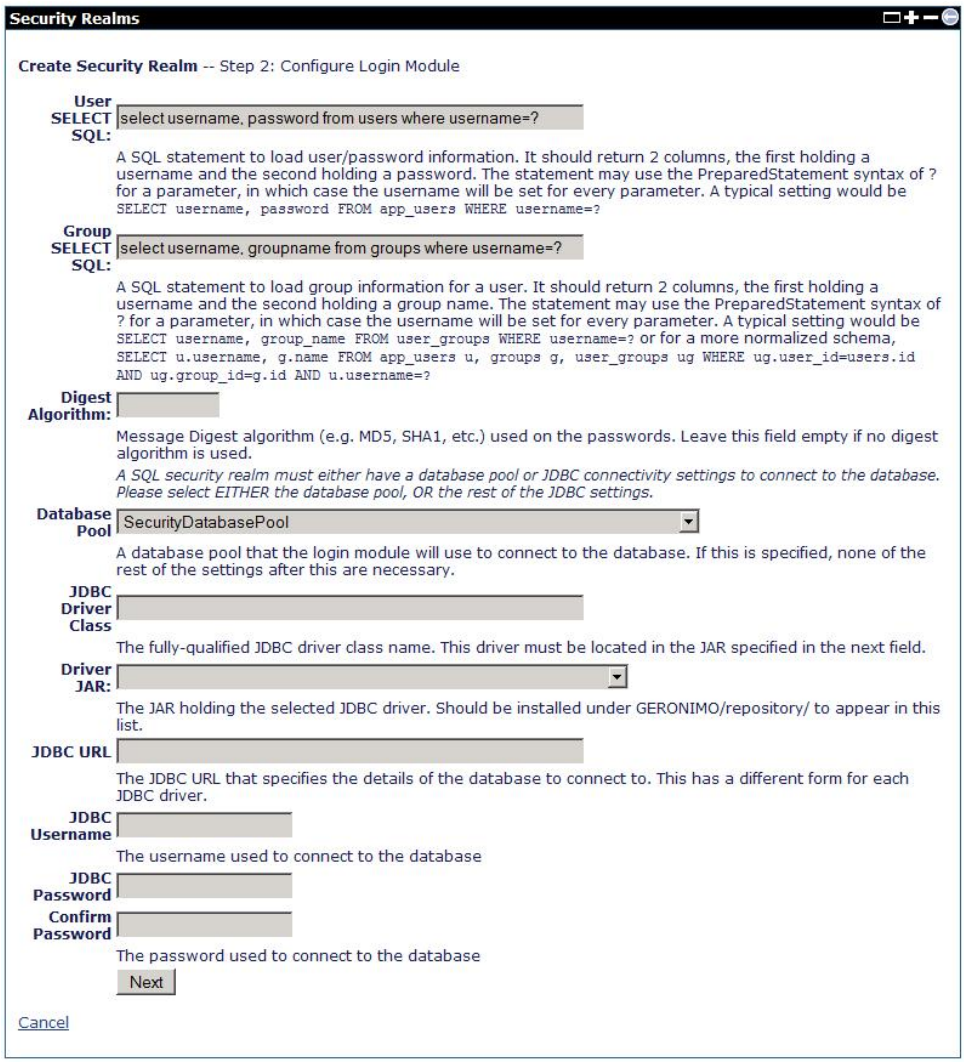
Name of Security Realm: フィールドに derby_security_realm と入力し、Realm type: プルダウン・メニューから Database (SQL) Realm を選択して Next をクリックします。

以下の画面でログイン・モジュールを構成します。冒頭の2つのフィールドは利用しているデータベースにより異なる値を入力する必要があるでしょう。今回はembedded Derby database　を使用しますので、ユーザーとグループを入手するためのSQLは下記のようになります。

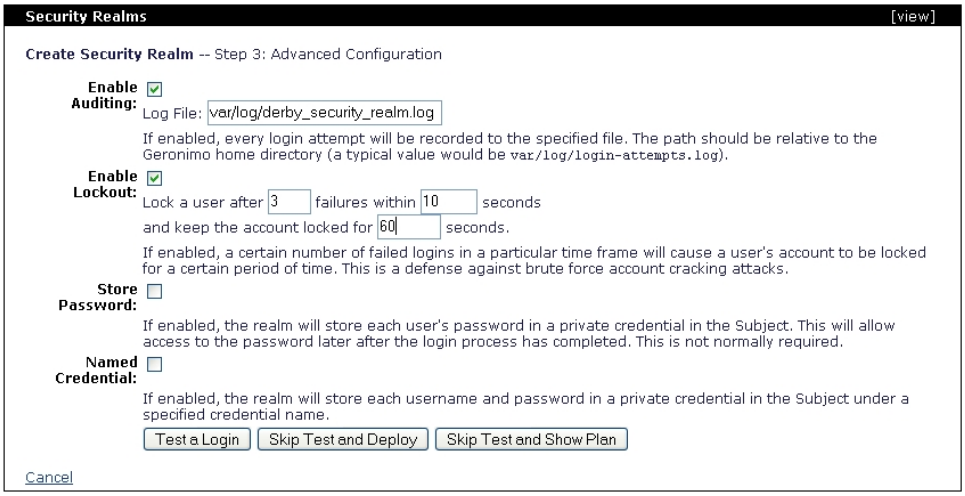
User SELECT SQL: `select username, password from users where username=?`
Group SELECT SQL: `select username, groupname from groups where username=?`

ユーザーとグループを検索するSQLステートメントを入力したら、前のステップで作成したデータベース・コネクション・プールを Database Pool プルダウン・メニューから選択してください。下記を参考に必要なフィールドに値を追加して Next をクリックします。

Database Pool: **`SecurityDatabasePool`**

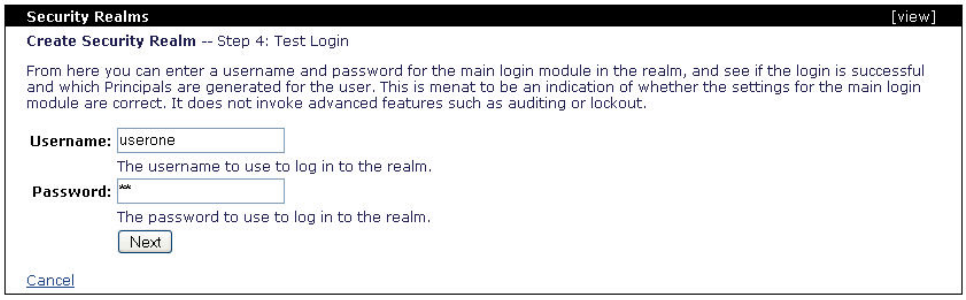


以下の手順を踏めば、このレルムに対するログインの試みを監査しモニターできます。また設定した期間内に失敗したログインの回数に基づき、アカウントをロックアウトするように構成することもできます。Store Password を有効にすれば、ユーザーのパスワードを"Subject"内のプライベート・クリデンシャルに格納できます。また Naming Credential も有効にすれば、ユーザーのパスワードのみならずユーザーの名前もプライベート・クリデンシャルに格納できます。

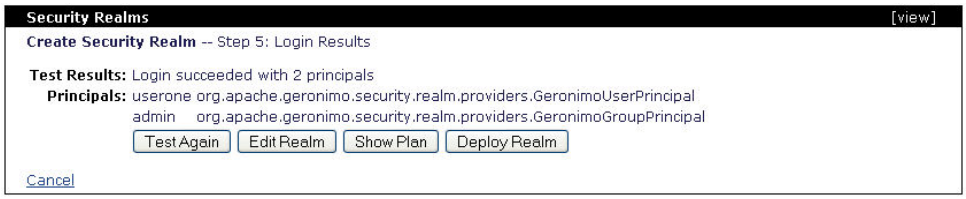


ここまでで新しいセキュリティ・レルムを構成してきました。次は構成をテストしてデプロイしましょう。Test a Login をクリックします。

データベースから読み取れる有効なユーザー名とパスワードを入力し Next をクリックします。



ログイン成功を示す確認メッセージが表示されたら Deploy Realm をクリックしてこの構成をサーバーにロードします。



これでDerbyデータベース内のユーザー名とパスワードを使った、新規に完全に構成されたセキュリティ・レルムが完成しました。

> ⚠ このレルムの初回の検証時にエラーが発生した場合は、画面やログに **`SQL Exception: Failed to start database ...`** が出ているかもしれません。これはDerbyの既知の問題で、Geronimoをリスタートすれば新規データベースとの通信が正しく行えるようになります。

以下はこのセキュリティ・レルムのデプロイメント・プランの例です。Geronimo管理コンソールを使う方法以外に、このサンプルをファイル(derby_security_realm.xml)に保存して 1.9.3. Deployer tool - デプロイヤー・ツール を使って以下のコマンドを実行する方法もあります。

```
<geronimo_home>\bin\deploy --user system --password manager deploy <realm_path>
\derby_security_realm.xml
```

derby_security_realm

```
<module xmlns="http://geronimo.apache.org/xml/ns/deployment-1.2">
<environment>
<moduleId>
<groupId>console.realm</groupId>
<artifactId>derby_security_realm</artifactId>
<version>1.0</version>
<type>car</type>
```

```
</moduleId>
<dependencies>
<dependency>
<groupId>org.apache.geronimo.configs</groupId>
<artifactId>j2ee-security</artifactId>
<type>car</type>
</dependency>
<dependency>
<groupId>console.dbpool</groupId>
<artifactId>SecurityDatabasePool</artifactId>
<version>1.0</version>
<type>rar</type>
</dependency>
</dependencies>
</environment>
<gbean name="derby_security_realm" class="org.apache.geronimo.security.realm.GenericSecurityRealm"
xsi:type="dep:gbeanType"
xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
<attribute name="realmName">derby_security_realm</attribute>
<reference name="ServerInfo">
<name>ServerInfo</name>
</reference>
<xml-reference name="LoginModuleConfiguration">
<log:login-config xmlns:log="http://geronimo.apache.org/xml/ns/loginconfig-2.0">
<log:login-module control-flag="REQUIRED" wrap-principals="false">
<log:login-domain-name>derby_security_realm</log:login-domain-name>
<log:login-module-class>org.apache.geronimo.security.realm.providers.SQLLoginModule</log:login-
module-class>
<log:option name="dataSourceName">SecurityDatabasePool</log:option>
<log:option name="dataSourceApplication">null</log:option>
<log:option name="groupSelect">select username, groupname from groups where username=?</log:option>
<log:option name="userSelect">select username, password from users where username=?</log:option>
</log:login-module>
<log:login-module control-flag="OPTIONAL" wrap-principals="false">
<log:login-domain-name>derby_security_realm-Audit</log:login-domain-name>
<log:login-module-class>org.apache.geronimo.security.realm.providers.FileAuditLoginModule</
log:login-module-class>
<log:option name="file">var/log/derby_security_realm.log</log:option>
</log:login-module>
<log:login-module control-flag="REQUISITE" wrap-principals="false">
<log:login-domain-name>derby_security_realm-Lockout</log:login-domain-name>
<log:login-module-
class>org.apache.geronimo.security.realm.providers.RepeatedFailureLockoutLoginModule</log:login-
module-class>
<log:option name="failureCount">3</log:option>
<log:option name="failurePeriodSecs">10</log:option>
<log:option name="lockoutDurationSecs">60</log:option>
</log:login-module>
</log:login-config>
</xml-reference>
</gbean>
</module>
```

一旦セキュリティ・レルムを作ってしまえば、usageリンクから、そのレルムをアプリケーションで使用するためのサンプルを参照できます。

1.1.3.3.3. LDAPレルム

This page last changed on 4 21, 2008 by JAGUG.

GeronimoにLDAPレルムをデプロイするためのガイドです。サンプルはデフォルトであるApacheDSセットアップ向けの設定を反映したものです。

ApacheDS

Geronimoはディレクトリー・サービスとして Apache Directory Server を使うように構成できます。Geronimo 1.1の時点ではApacheDSはデフォルトではインストールされていません。もし貴方の環境にApacheDSがインストールされていないなら下記の簡単な一連の手順を踏んでApacheDSをGeronimoのプラグインとしてインストールできます。

GeronimoへのApacheDSのインストール

ApacheDS をGeronimoのプラグインとしてインストールする手順は以下のとおりです。

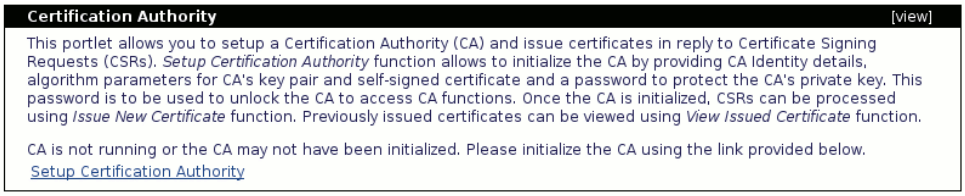
1. Geronimo管理コンソールにログインします
2. 左側のメニューの Plugins から Create/Install をクリックします
3. リスト中にリポジトリーが存在しない場合は **`Update Repository List`** リンクをクリックします
4. 検索のためのRepositoryを選択します
5. Search for Plugins をクリックします

1.1.3.4. 認証局(CA)

このリリースのApache Geronimoでは、自分用の認証局(CA)を定義して署名要求(CSR)への応答としての証明書を発行することができます。Apache Geronimo の管理コンソールの左側のCertificate Authority をクリックすると、Certification Authority ポートレットを利用できます。

認証局の構成

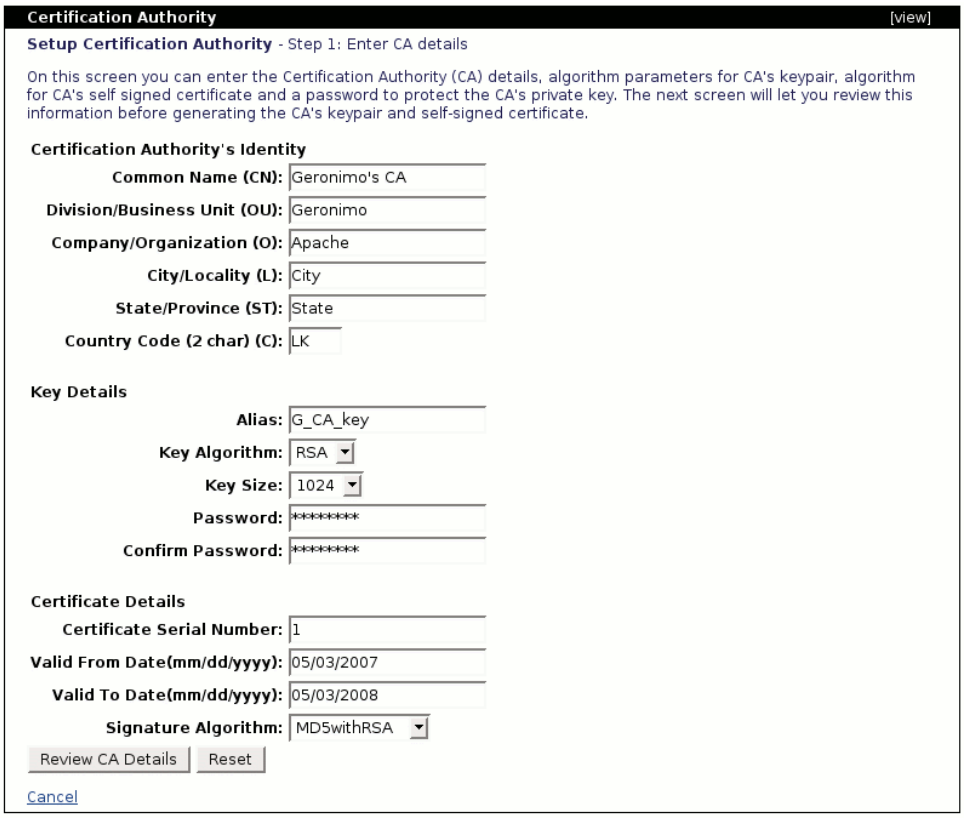
初めてこのポートレットを呼び出した際にはCAはまた構成されていませんので、このような画面が見えるはずです。



Geronimo をCAとして構成するには Setup Certification Authority をクリックします。

同種の情報を提供できるように事前に準備しておく必要があるという意味では、この手順は 1.1.3.1. 証明書の管理で説明したキー・ストア及び証明書を定義する手順に少し似ています。

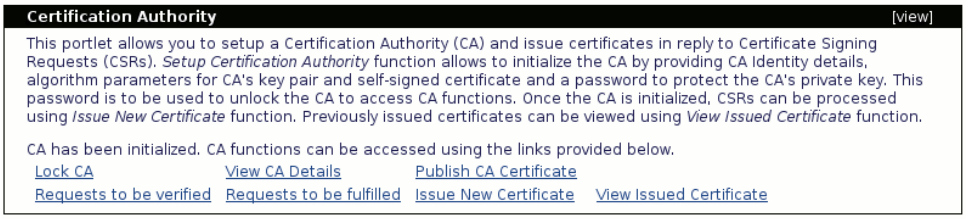
始めのステップは下記のイメージにある通り、Certification Authorityの詳細を定義することです。このフォームで入力された情報がCAと自己署名された各々のキー・ペアを作成するために使われます。



これは「情報収集」のステップであり、この時点では証明書はまだ作成していません。Review CA Details をクリックしてから、次に Setup Certification Authority をクリックしてください。

うまく作成できたら、作成した証明書の詳細と共に CA Setup is successful! というメッセージが表示されるはずです。

次回 Certification Authority へアクセスしたときには、今しがた作成したCAが見えるはずです。このポートレットから、CSRを管理し、証明書をレビュー・発行することができます。



署名要求への署名

1.1.3.3.1. 証明書プロパティ・ファイル・レルム のセクションで新しいキー・ストアと証明書の作成方法、および署名要求(CSR)の作成方法、およびCAの応答をインポートする方法を詳しく説明しました。このセクションではこのCAを使ってクライアントからの署名要求(CSR)をどのように管理し、署名すればよいかを説明します。

まず署名要求(CSR)を生成したところから始めましょう。1.1.3.3.1. 証明書プロパティ・ファイル・レルム セクションで使った例が使えます。

1.1.4. サービスの構成

This page last changed on 4 20, 2008 by JAGUG.

このセクションでカバーされている主なタスクのリストです。以下を含みます。

1.1.4.1. アーカイブのGeronimoリポジトリーへの追加

This page last changed on 4 21, 2008 by JAGUG.

左側の Console Navigation メニューから Common Libs を選択して表示される Repository Viewer ポートレットを使用すれば、アーティファクトをリポジトリーに追加できます。リポジトリーのレイアウトはApache Mavenで使われているものと同じですので、ファイルを上書きコピーすることも簡単です。

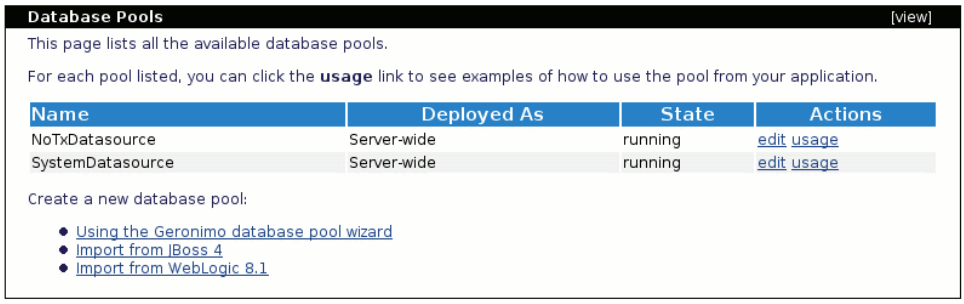
リストの先頭のものを例として取り上げますと、アプリケーションでアーティファクトを使用するにはアプリケーションのデプロイメント・プランの　<environment> 要素中の <dependencies> の下に <dependency> 要素を追加します。一部の抜粋ですが、デプロイメント・プランは下記のような感じになっているはずです。

```
<environment>
...
<dependencies>
...
<dependency>
<groupId>activeio</groupId>
<artifactId>activeio</artifactId>
<version>2.0-r118</version>
<type>jar</type>
</dependency>
</dependencies>
</environment>
```

1.1.4.2. データベース・プールの構成

This page last changed on 4 25, 2008 by JAGUG.

データベース・プールを構成するには、左端の Console Navigation メニューから Database Pools を選択して表示される Database Pools ポートレットを使います。下図のような Database Pools ポートレットにすべての利用可能なデータベース・プールが表示され、さらに新しいプールのインポートや作成ウィザードへのリンクがあります。



1.1.4.2.1. DB2データソースの構成

This page last changed on 4 21, 2008 by JAGUG.

本稿はApache Geronimo v2.0でDB2データソースを構成する方法を示します。Geronimo v1.1のリリースから、Geronimo管理コンソールの接続プール作成ウィザードで複数のJDBCドライバーを選択できます。

このシナリオでは、DB2データベースの接続プールを作成しています。DB2データベースに接続するためには少なくとも2つのドライバーjarファイルを定義する必要があるでしょう。　それは、JDBCドライバー自体と対応するライセンス・ファイルです。選択したJDBCドライバー毎に異なるライセンス・ファイルが必要となります。

ここで示す手順は、サーバにドライバーが未インストール、複数のドライバー・ファイルを必要とするかもしれない、DB2以外のデータソースをデプロイするときにも当てはまります。

本稿では接続プールをデプロイする2通りの方法を提供します。最初に取り組むアプローチはGeronimo管理コンソールを使用するもので、次にコマンド・ライン・オプションの使い方を説明します。

DB2ドライバーをリポジトリーに追加する

DB2ドライバーとライセンスをGeronimoリポジトリーに追加するため、これらのファイルを特定のディレクトリ構造の中に配置する必要があります。通常は2個もしくは3個のファイルをリポジトリーに追加する必要があるでしょう。これらのファイルは、

- db2jcc.jar - これが実際のDB2ユニバーサルJDBCドライバーjarファイルです。
- db2jcc_license_cu.jar - これが標準 DB2 ユニバーサルJDBCドライバーのライセンス・ファイルです。これによりLinux、UNIX、そしてWindows用 DB2 ユニバーサル・データベースへのアクセスが許可されます。
- db2jcc_license_cisuz.jar - これがz/OSとiSeries用の DB2(DB2 ESEとDB2 Connect)JDBC ドライバー・ライセンスです。これは標準ライセンスに加えてこれらのサーバを利用する際に使われるべきです。

これらのファイルは<sqllib_home>¥javaディレクトリから取得できます。DB2 JDBCドライバーとライセンスについてのより多くの情報は下記URLからDB2インフォメーションセンターを参照してください。

http://publib.boulder.ibm.com/infocenter/db2luw/v8/topic/com.ibm.db2.udb.doc/ad/t0010264.htm

Geronimoでこれらのファイルを利用するため、以下に示すように名前を変更(コピーして名前変更)する必要があるでしょう。

変更前	変更後
db2jcc.jar	db2jcc-8.1.8.jar
db2jcc_license_cu.jar	db2jcc_license_cu-8.1.8.jar
db2jcc_license_cisuz.jar	db2jcc_license_cisuz-8.1.8.jar

これは、この特定のケースのためにDB2 v8.1.8.806が使われたことを意味します。

この時点で、これらのファイルをGeronimoリポジトリーに追加する準備が整いました。追加する方法は2通りあります。様々なグラフィカルもしくはコマンド・ライン・ツールを使ってこれのファイルをコピーし、必要なディレクトリを作成することもできますし、Geronimo管理コンソールを使いドライバーとライセンスを共有ライブラリに追加することもできます。続いて、2通りの方法を説明していきます。

コマンドラインを使う場合

<geronimo_home>¥repositoryディレクトリの中に次に示すディレクトリ構造を作成し、適切なファイルをそれぞれのディレクトリにコピーする必要があります。

- **`com/ibm/db2/db2jcc/8.1.8`**
  このディレクトリに db2jcc-8.1.8.jar をコピーします
- **`com/ibm/db2/db2jcc_license_cisuz/8.1.8`**
  このディレクトリに db2jcc_license_cisuz-8.1.8.jar をコピーします
- **`com/ibm/db2/db2jcc_license_cu/8.1.8`**
  このディレクトリに db2jcc_license_cu-8.1.8.jar をコピーします

Geronimo管理コンソールを使う場合

管理コンソールを使うため、Apache Geronimoが起動中でなければなりません。ブラウザで次のURLを参照し、Geronimo管理コンソールにアクセスしてください。

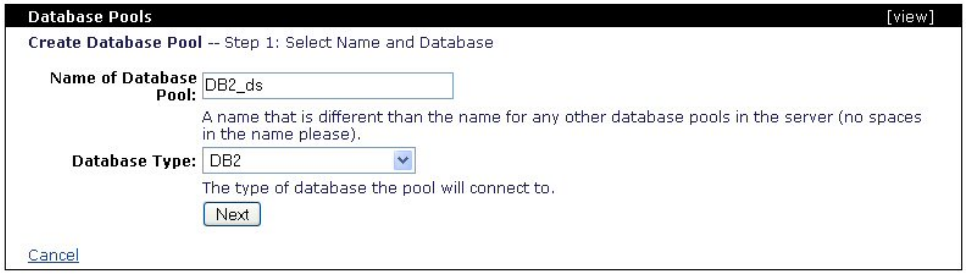
http://localhost:8080/console
ユーザに system 、パスワードに manager を入力し、Login をクリックしてください。
Common Libs をクリックして、Repository viewer にアクセスしてください。
参照をクリックして、インストールする最初のファイルを選択してください。今回の場合、最初にdb2jcc-8.1.8.jarをインストールしましょう。
幾つかの値が初期値として設定されるでしょう。Group: に com.ibm.db2 を設定し、それ以外は初期値のままにして、Install をクリックします。
db2jcc_license_cisuz-8.1.8.jar と db2jcc_license_cisuz-8.1.8.jar についても上の2つの手順を繰返します。

ドライバーとライセンス・ファイルがインストールされたら、新しいデータベース接続プールを作成できます。

Geronimo管理コンソールからウィザードを使ってデータベース接続プールを作成する

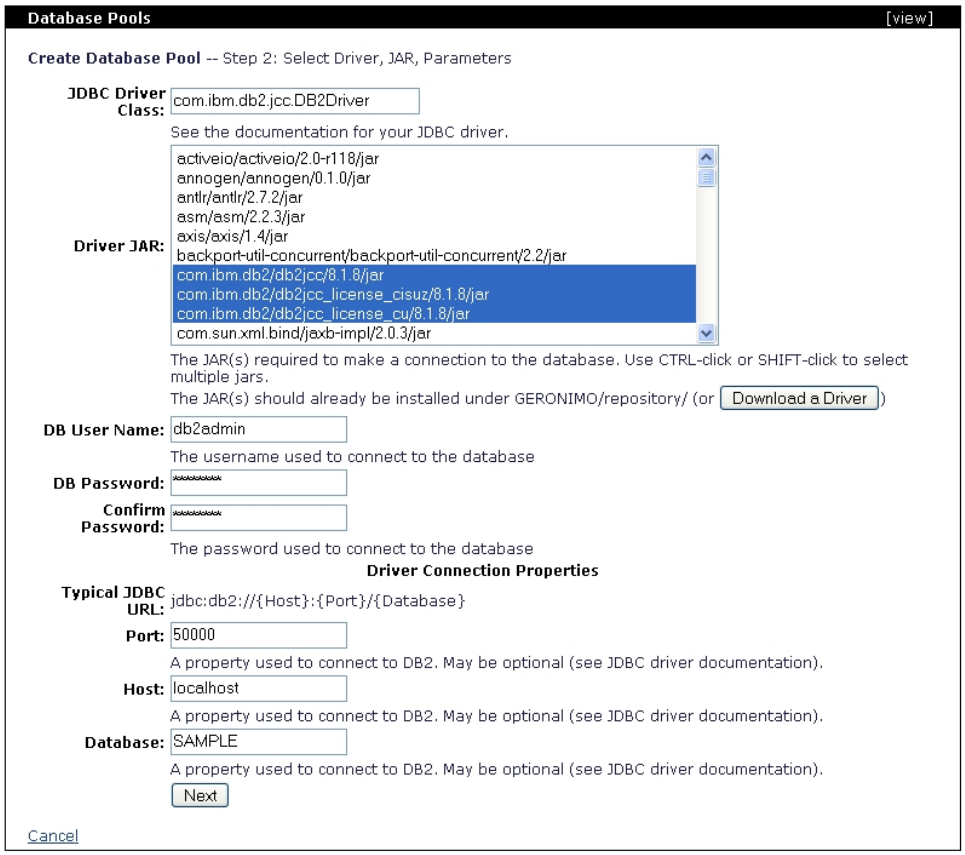
Geronimo管理コンソールから Database Pools を選択し、Using the Geronimo database pool wizard をクリックして新しいプールを作成します。

データベース・プール作成ウィザードのStep 1では、下図に示すようにプール名とデータベース・タイプを入力し、Next をクリックします。



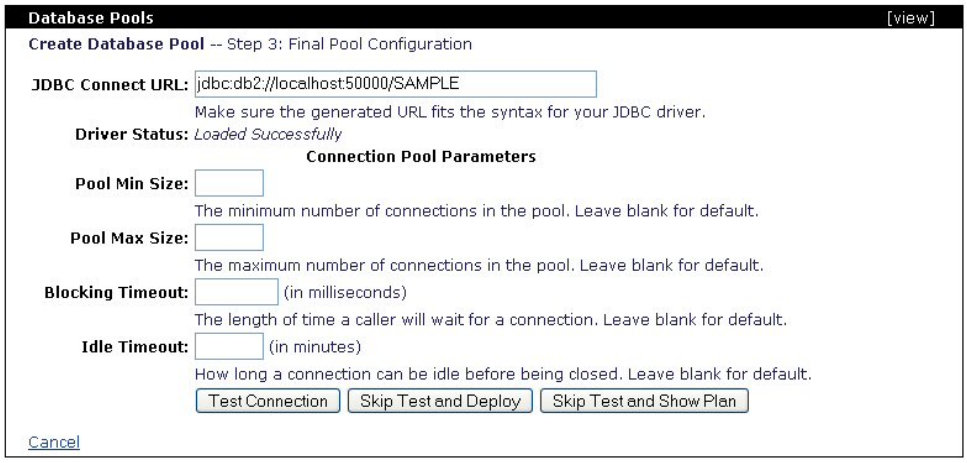
Step 2では、JDBC Driver Class: 項目は初期値(com.ibm.db2.jcc.DB2Driver)のままにしてください。もし Driver JAR: CTRLとクリックもしくはSHIFTとクリックにより、下図に示すように複数ファイルを選択します。

下図に示したように残りの接続情報を入力します。この例のために、デフォルトの db2admin ユーザとパスワードが使用され、SAMPLE データベースがDB2コントロール・センター経由で作成されています。

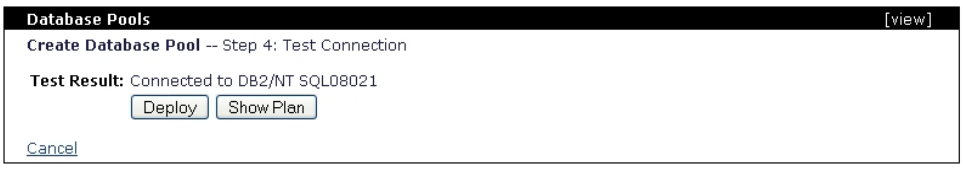


Nextをクリックします。

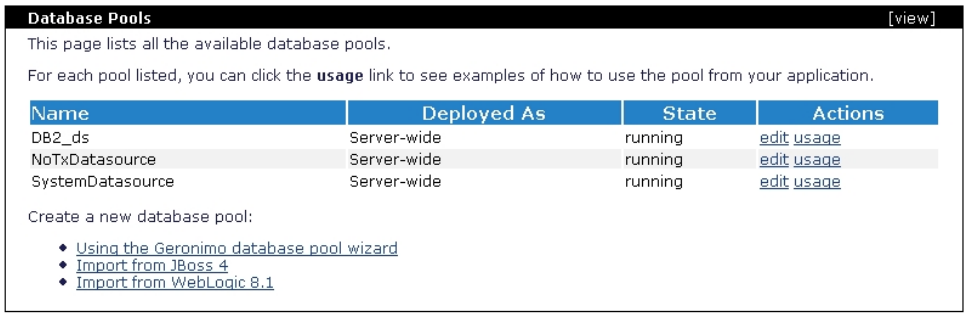
Step 3のオプションは初期値のままにして、Driver Status: Loaded Successfully メッセージを確認します。



Test Connectionをクリックします。DB2に接続した確認メッセージが出力されます。



Deploy をクリックします。Database Pools 内に DB2_ds がリストされます。



コマンドラインからデータベース接続プールをデプロイする

ウィザードの代わりに、手動でデプロイメント・プランを作成することができます。そしてコマンド・ライン版デプロイ・ツールを使ってそれをデプロイします。このオプションを使うため、db2-plan.xmlファイルを作成し、次に示す例をコピーします。

db2-plan.xml deployment plan

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.2">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2">
<dep:moduleId>
<dep:groupId>console.dbpool</dep:groupId>
<dep:artifactId>DB2_ds</dep:artifactId>
<dep:version>1.0</dep:version>
<dep:type>rar</dep:type>
</dep:moduleId>
<dep:dependencies>
<dep:dependency>
<dep:groupId>com.ibm.db2</dep:groupId>
<dep:artifactId>db2jcc</dep:artifactId>
<dep:version>8.1.8</dep:version>
<dep:type>jar</dep:type>
</dep:dependency>
<dep:dependency>
<dep:groupId>com.ibm.db2</dep:groupId>
<dep:artifactId>db2jcc_license_cisuz</dep:artifactId>
<dep:version>8.1.8</dep:version>
<dep:type>jar</dep:type>
</dep:dependency>
<dep:dependency>
<dep:groupId>com.ibm.db2</dep:groupId>
<dep:artifactId>db2jcc_license_cu</dep:artifactId>
<dep:version>8.1.8</dep:version>
<dep:type>jar</dep:type>
</dep:dependency>
</dep:dependencies>
</dep:environment>
<resourceadapter>
<outbound-resourceadapter>
<connection-definition>
<connectionfactory-interface>javax.sql.DataSource</connectionfactory-interface>
<connectiondefinition-instance>
<name>DB2_ds</name>
<config-property-setting name="Password">db2admin</config-property-setting>
<config-property-setting name="Driver">com.ibm.db2.jcc.DB2Driver</config-property-setting>
<config-property-setting name="UserName">db2admin</config-property-setting>
<config-property-setting name="ConnectionURL">jdbc:db2://localhost:50000/SAMPLE</config-property-
setting>
<connectionmanager>
<local-transaction/>
<single-pool>
<max-size>10</max-size>
```

```
<min-size>0</min-size>
<match-one/>
</single-pool>
</connectionmanager>
</connectiondefinition-instance>
</connection-definition>
</outbound-resourceadapter>
</resourceadapter>
</connector>
```

すぐにこのプランを分析しましょう。<dep:environment> セクションを見てください。そこにはデプロイされたリソースやコンポーネントを識別する moduleId が見つかります。管理コンソールの中で、Database Pools の Name カラムに moduleId は表示されます。

moduleIdの直後に依存定義が続きます。このケースでは DB2 JDBC ドライバと2つのライセンスjarに関係する3つの <dep:dependency> ブロックを見つけられます。このプラン内の最後の"大きな"ブロックは <resourceadapter> です。そこにはドライバー、ユーザとパスワード、接続URLなどの接続パラメータが定義されています。

データソースのデプロイ

これまで作成してきた DB2 データソースをデプロイするため、<geronimo_home>¥binディレクトリから次のコマンドを実行します。

deploy --user system --password manager deploy <dep_plan_home>\db2-plan.xml ..\repository\org\tranql\tranql-connector-ra\1.3\tranql-connector-ra-1.3.rar

次のメッセージが出力されます。

```
D:\geronimo-tomcat6-jee5-2.0\bin>deploy --user system --password manager deploy \tmp\db2-
plan.xml ..\repository\org\tranql\tranql-connector-ra\1.3\tranql-connector-ra-1.3.rar
Using GERONIMO_BASE:   D:\geronimo-tomcat6-jee5-2.0
Using GERONIMO_HOME:   D:\geronimo-tomcat6-jee5-2.0
Using GERONIMO_TMPDIR: D:\geronimo-tomcat6-jee5-2.0\var\temp
Using JRE_HOME:        C:\Java\jdk1.5.0_06\\jre
Deployed console.dbpool/DB2_ds/1.0/rar
```

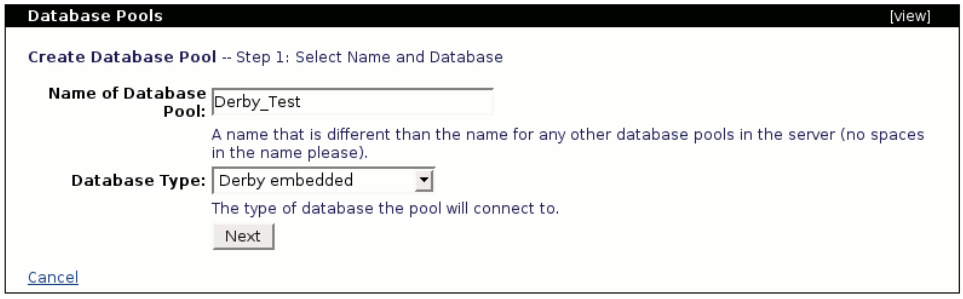
Back to Top

1.1.4.2.2. データベース・プールの新規作成

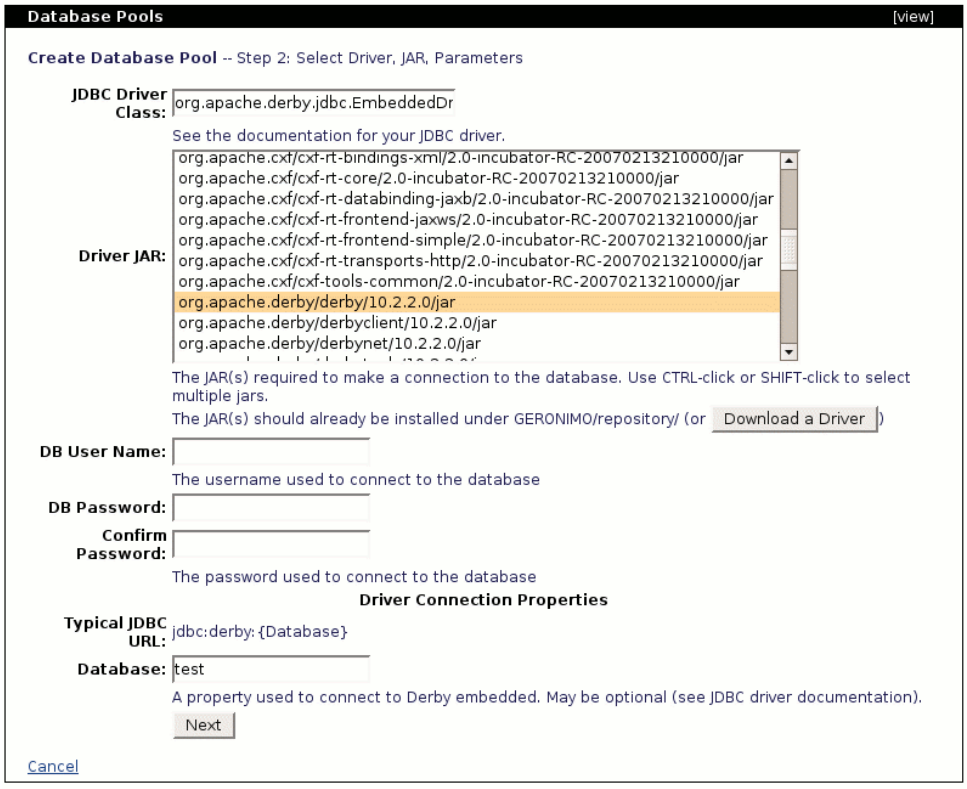
This page last changed on 4 21, 2008 by JAGUG.

データベース・プール・ポートレットの Using the Geronimo database pool wizard リンクをクリックすると新しいプールを作成できます。このウィザードは4ステップの簡単な手続きでガイドします。

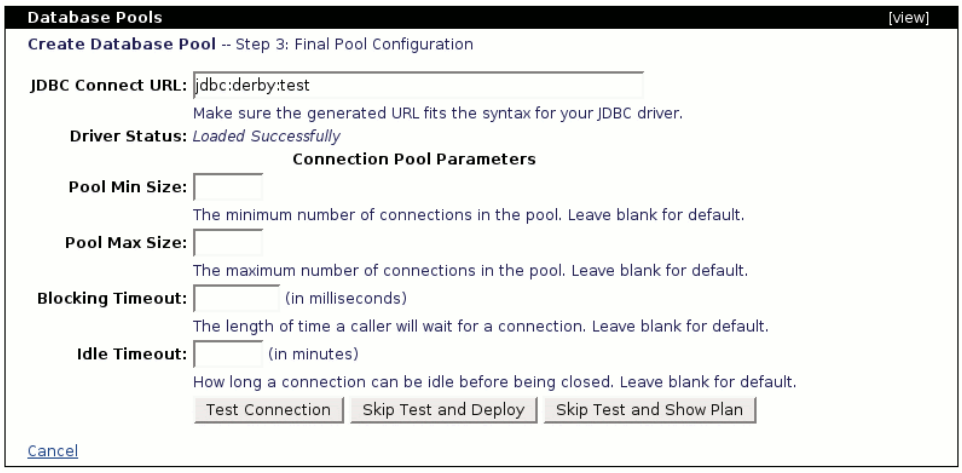
最初に、データベース・プールの名前とデータベース・タイプを指定します。



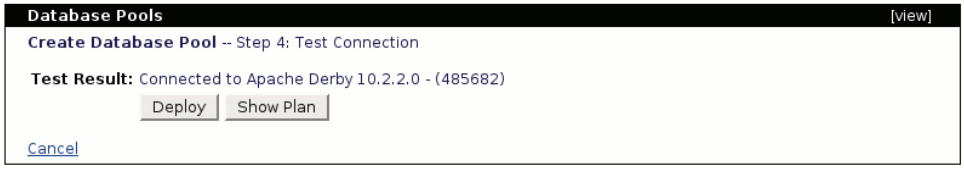
それから、JDBCドライバーを選択します。JDBCドライバーは前画面で選択したデータベース・タイプに従って事前設定されます。次にドライバーJARを選択し、そしてデータベース名を指定します。この例では test データベースが事前に作成されています。データベース作成自体は1ステップの手順で 1.5. データベースの作成で説明されています。



次のステップでは、プールサイズ（最小と最大）やタイムアウト等の接続パラメータを構成します。パラメータを設定したら、Test Connetion をクリックします。



接続テストが成功したら、Deploy をクリックします。そうではなく Show Plan をクリックすることもできます。こうすると、データベース・プールのデプロイメント・プランが表示され編集できます。手動でどのようにDBプランをデプロイするかについては 1.1.4.2.1. DB2データソースの構成セクションを参照してください。



1.1.4.2.3. JBoss 4データベース・プールのインポート

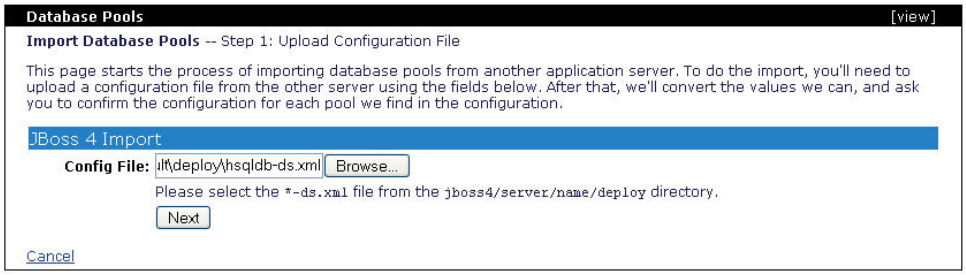
This page last changed on 4 21, 2008 by JAGUG.

このウィザードは JBoss 4 データベース・プールのインポートを助けてくれます。この例ではJBossサーバーにデフォルトで提供されている hsqldb-ds.xml を使います。

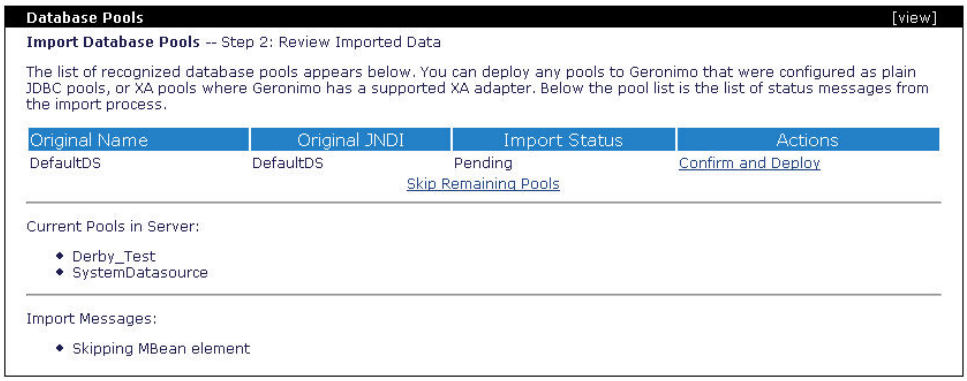
まず、GeronimoにHypersonic HSQL用のドライバーjarを与えます。これを行うため、次に示す手順でGeronimoのリポジトリにドライバーをインストールします。

1. HSQLドライバ **`hsqldb.jar`** を見つけてください。このファイルは <jboss_home>¥server¥default¥lib ディレクトリに存在します。このファイルをコピーし、**`hsqldb-1.8.0.jar`** にリネームします。
2. Repository Viewer ポートレットを使って、HSQL ドライバーjarをインストールします。管理コンソールの Common Libs をクリックし、ポートレットにアクセスします。参照 をクリックしてデータベース・ドライバjarを選択します。Group: を Hipersonic に変更し、他の項目は初期値のままで Install をクリックします。repository entries リストの先頭付近に **`Hipersonic/hsqldb/1.8.0/jar`** が表示されます。

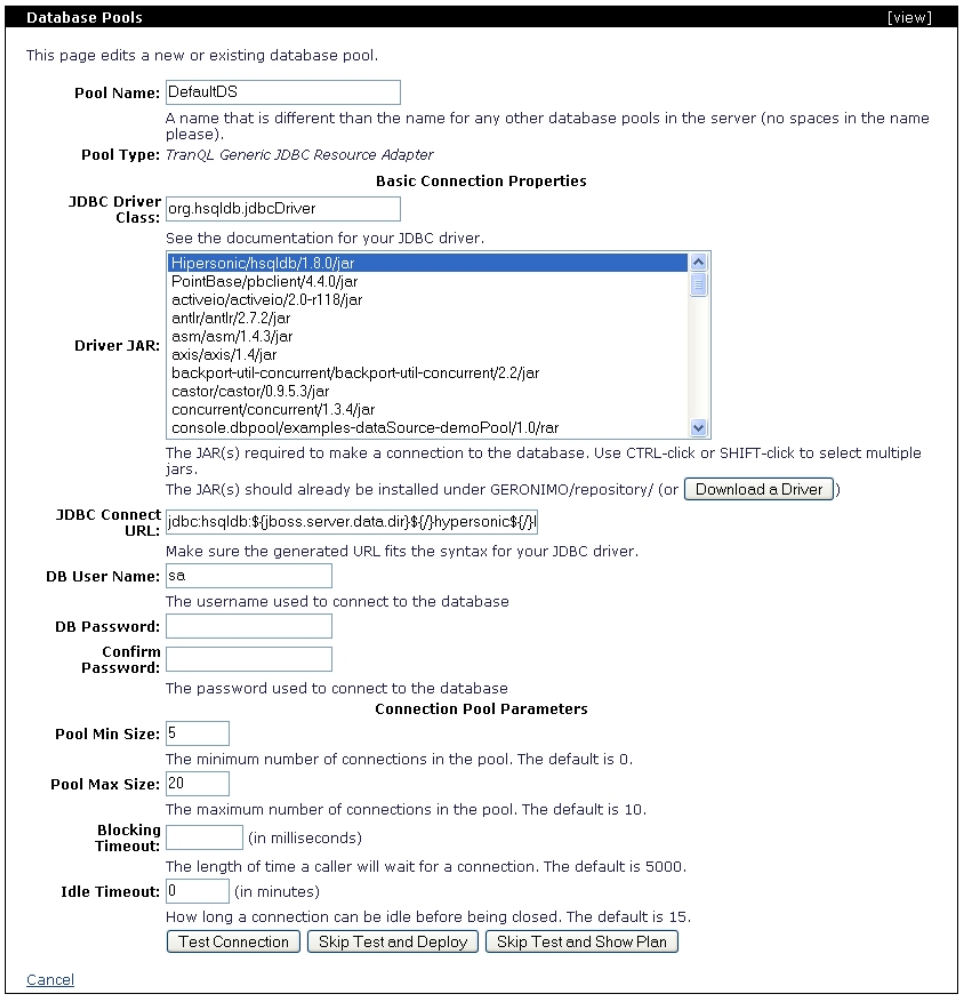
Geronimo管理コンソールの Database Pools リンクをクリックし、Database Pools ポートレットの Import from JBoss 4 をクリックします。データベース・プールのインポート Step 1 で、*-ds.xml ファイルの場所を指定し、Next をクリックします。



JBoss データソースが読み込まれると、Step 2 で hsqldb-ds.xml ファイルからウィザードが認識できたデータベース・プールがリストされます。Confirm and Deploy をクリックします。

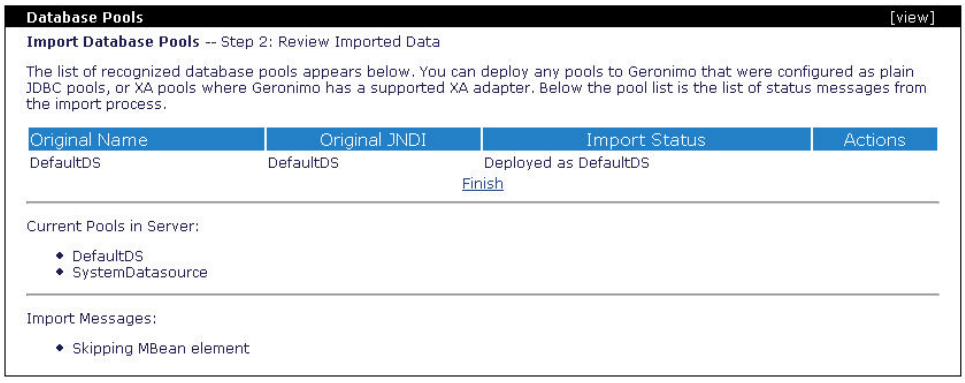


次のステップでは認識できたデータベース・プールの編集ができます。このステップでGeronimoリポジトリーに定義したドライバーjarを指定します。*Driver JAR:*がどのようにリストされるかを下図に示します。

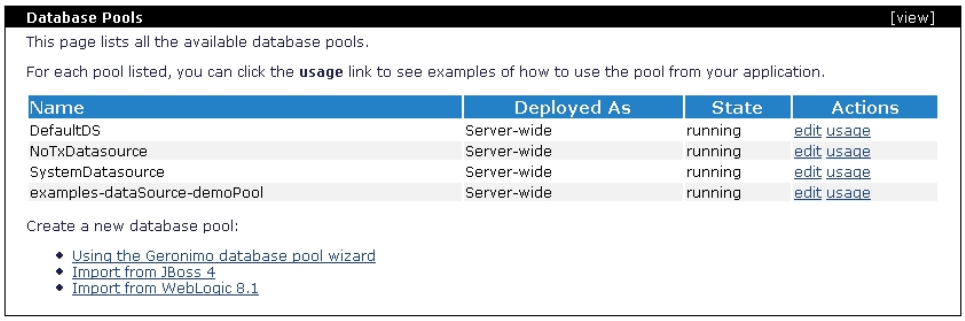


Test Connection をクリックして、必要なデータが全て入力され、Geronimoが接続を確立できるか確認します。

Deploy をクリックします。これにより Step 2 と同じ画面が表示されます。もしファイルに定義されたプールを更に取り込むなら、このステップを繰り返し、選択しながらデータベース・プールをインポートできます。この例ではデータソースが1つなので、Finish をクリックしインポート・ウィザードを終了します。



インポートされた DefaultDS データベース・プールが表示されます。


Database Pools
[view]
This page lists all the available database pools.
For each pool listed, you can click the usage link to see examples of how to use the pool from your application.
Name
Deployed As
State
Actions
DefaultDS
Server-wide
running
edit usage
NoTxDatasource
Server-wide
running
edit usage
SystemDatasource
Server-wide
running
edit usage
examples-dataSource-demoPool
Server-wide
running
edit usage
Create a new database pool:
Using the Geronimo database pool wizard
Import from JBoss 4
Import from WebLogic 8.1

1.1.4.2.4. WebLogic 8.1データベース・プールのインポート

This page last changed on 4 21, 2008 by JAGUG.

このウィザードはBEA WebLoginデータベース・プールをインポートする2通りの方法を提供します。最初の方法は構成ファイル(config.xml)を使用します。もしこの方法を選択すると、ウィザードはできるだけ多くの項目を変換し、変換できない項目を手動で入力するように聞いてきます。入力しなくてはいけない値の1つはデータベースのパスワードです。

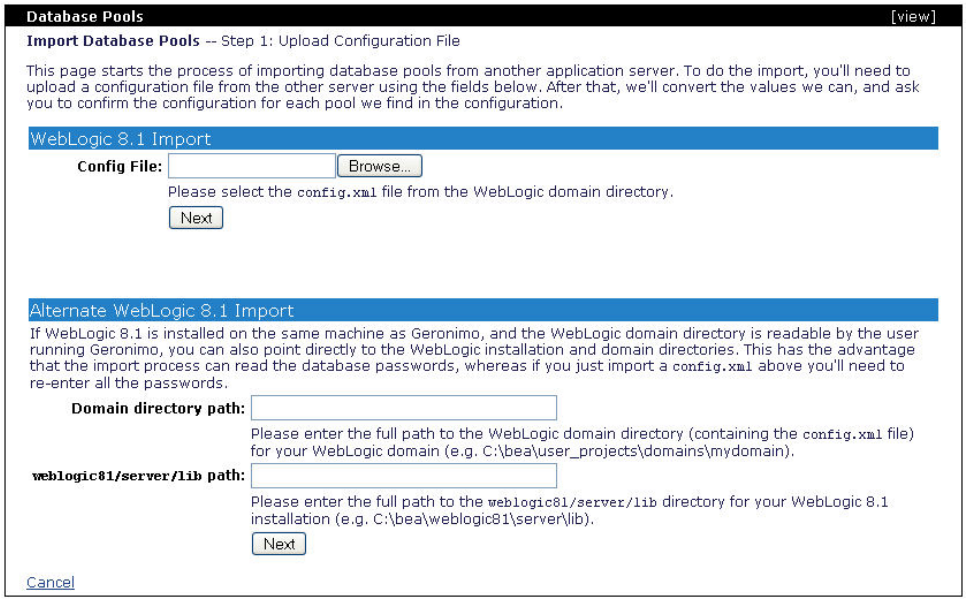
2つ目の方法は、Apache GeronimoとBEA WebLogicサーバーが同一のマシンにインストールされているときに可能です。この方法ではWebLogicのインストール先とドメイン・ディレクトリを直接指定します。データベースのパスワードを直接読み取れる利点があります。

どちらの方法をとっても、データベース・ドライバーjarのありかをGeronimoに教える必要があります。この例では、Repository Viewer を使い PointBase ドライバーjarをインストールします。

以下が PointBase データベース・ドライバーjarをGeronimoにインストールする手順の要約です。

1. PointBase クライアント・ドライバーjar **`pbclient44.jar`** を見つけてください。このファイルは <bea_home>weblogic81¥common¥eval¥pointbase¥lib ディレクトリに存在します。このファイルをコピーし、**`pbclient-4.4.0.jar`** にリネームします。
2. Repository Viewer を使って PointBase ドライバーjarをインストールします。管理コンソールの Common Libs をクリックし、ポートレットにアクセスします。参照 をクリックしてデータベース・ドライバjarを選択します。Group: を PointBase に変更し、他の項目は初期値のままで Install をクリックします。repository entries リストの先頭付近に **`PointBase/pbclient/4.4.0/jar`** が表示されます。

Geronimo管理コンソールの Database Pools リンクをクリックし、Database Pools ポートレットの Import from WebLogic 8.1 をクリックします。下図にインポート・ウィザードを示します。



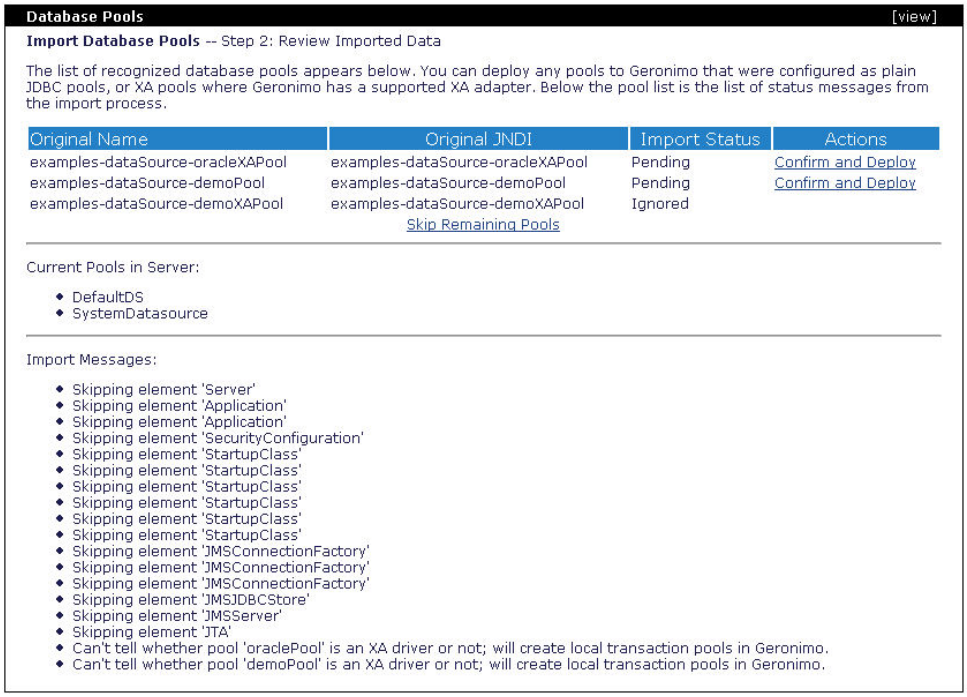
この例では、2番目の方法に焦点を当てます。WebLogicサーバには全てのサンプル・アプリケーションを含むデフォルトの example ドメインが作成されています。このドメインは <bea_home>¥user_projects¥domains¥examples ディレクトリに存在します。

(上に示した)インポート・ウィザードの最初の画面で、Domain directory path: と weblogic81/server/lib path: に入力し、Next をクリックします。

Domain directory path: `<bea_home>\user_projects\domains\examples`

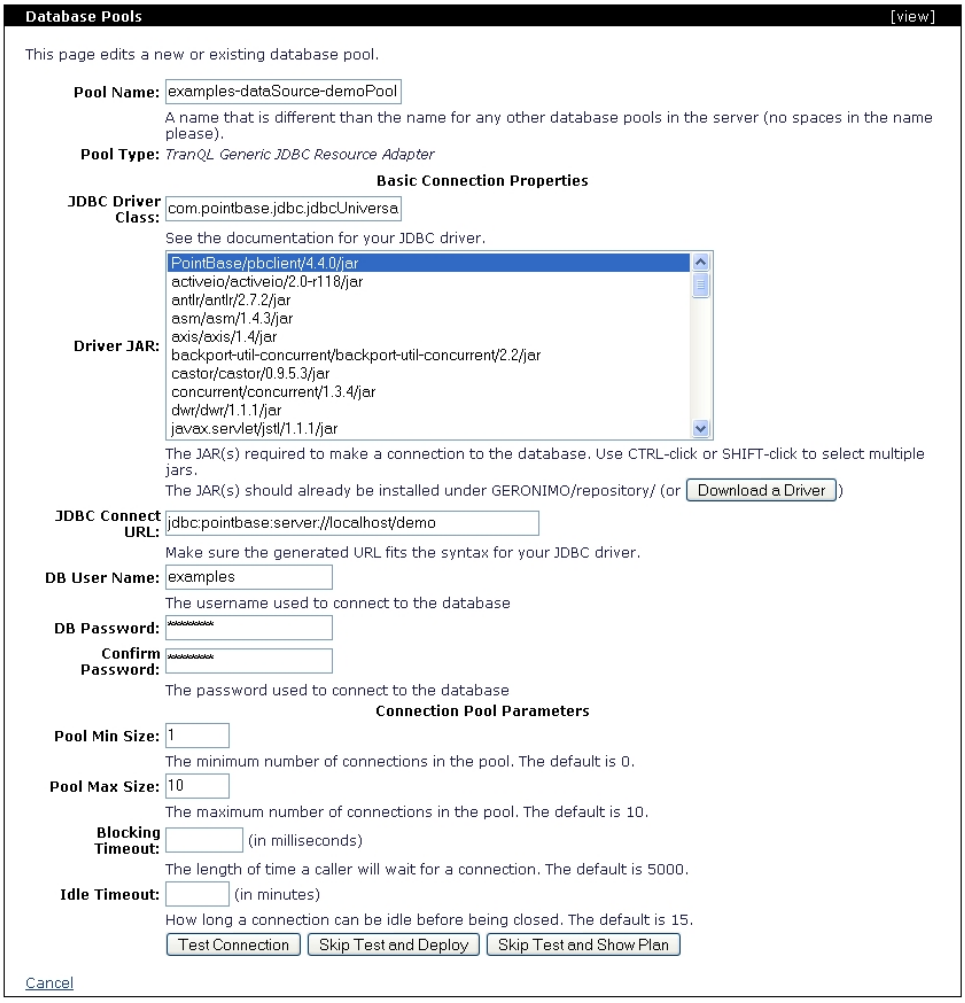
weblogic81/server/lib path: `<bea_home>\weblogic81\server\lib`

下図の Step 2 では指定したWebLogic ドメインから Apache Geronimoにインポートされ、認識されたデータベース・プールのリストが示されています。



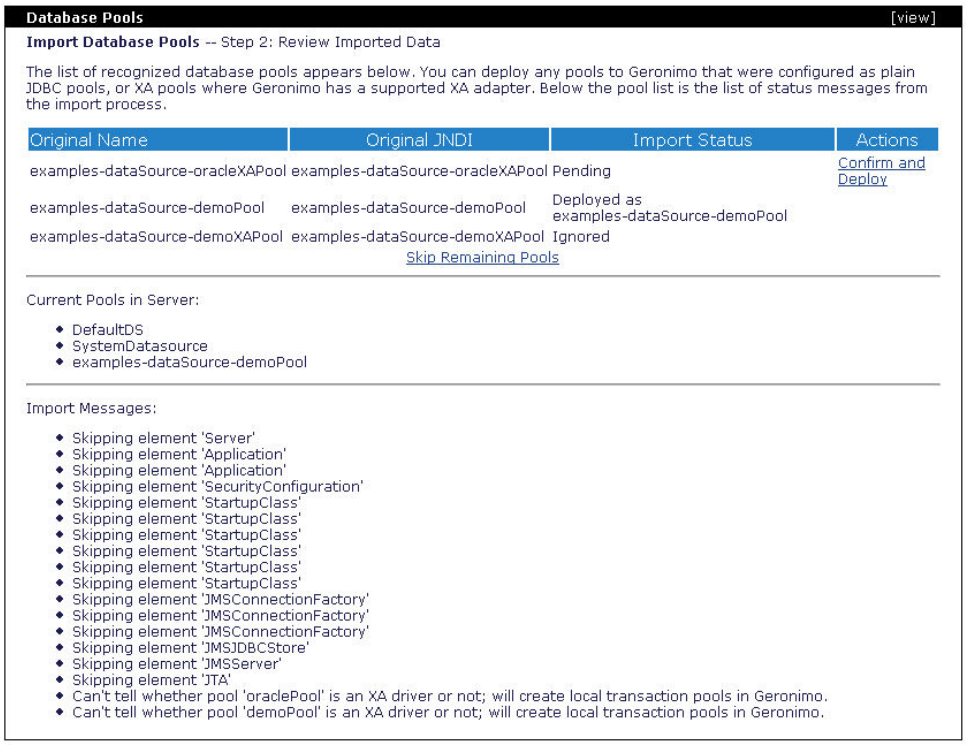
注意:もし接続テストを成功させたいなら、データベース・プールをインポートしようとするWebLogic ドメインは実行中でなければいけません。この例では examples-dataSource-demoPool に対応する、リストの上から2番目の Confirm and Deploy をクリックします。

次のステップでは、先ほどGeronimoリポジトリに作成した Driver JAR: を選択します。

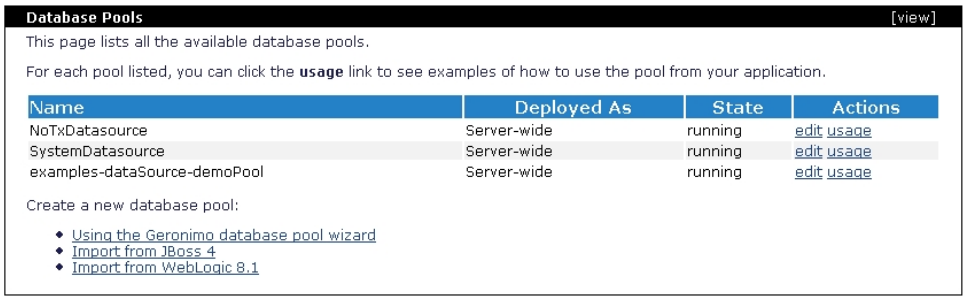


注意:データベースのパスワードは設定されています。Test Connection をクリックし。下図のようになることを確認します。Deploy をクリックします。

次のページは Step 2 に戻ります。残りのインポート可能なデータベース・プールが表示され、インポート済みのデータベース・プールのインポート・ステータスも表示されています。ここで Skip Remaining Pools をクリックしてインポート・ウィザードを終了できます。



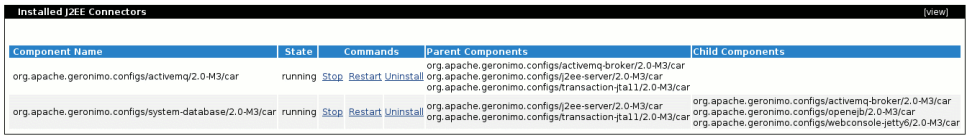
データベース・プール・ポートレットにはインポートしたデータベース・プールが表示されます。



1.1.4.2.5. データベース・プールの削除

データベース・プールを削除するには2通りの方法があります。コマンドライン 1.9.3. Deployer tool - デプロイヤー・ツール と 1.6. Geronimo 管理コンソール のInstalled J2EE Connectors ポートレットです。

もしグラフィカルに行いたいなら管理コンソールの J2EE Connectors をクリックし、Installed J2EE Connectors ポートレットにアクセスします。



このポートレットでは全ての利用可能なコネクターがリストされ、停止と開始そしてこれらのJ2EEコネクターをアンインストールすることができます。削除したいコネクターの Uninstall をクリックするだけです。

また、1.9.3. Deployer tool - デプロイヤー・ツール を使ってデータベース・プールを削除することもできます。このツールは利用可能なモジュールをリストし、コネクション・プールは `moduleID` でリストされます。`moduleID` は Component Name カラムでリストされたものです。

全ての利用可能なモジュールをリストするには、<geronimo_home>¥bin ディレクトリで以下のコマンドを発行します。

```
deploy --user system --password manager list-modules
```

以下に示す例ようなリストが出力されます。

```
root@karthi-laptop:/home/karthi/Projects/Geronimo/geronimo-jetty-j2ee-1.2-beta/bin>java -jar
deployer.jar --user
system --password manager list-modules
Found 59 modules
+ org.apache.geronimo.configs/activemq/2.0-M3/car
+ org.apache.geronimo.configs/activemq-broker/2.0-M3/car
+ org.apache.geronimo.configs/axis-deployer/2.0-M3/car
+ org.apache.geronimo.configs/client-deployer/2.0-M3/car
+ org.apache.geronimo.configs/clustering/2.0-M3/car
+ org.apache.geronimo.configs/connector-deployer/2.0-M3/car
+ org.apache.geronimo.configs/cxf-deployer/2.0-M3/car
+ org.apache.geronimo.configs/dojo-jetty6/2.0-M3/car @ http://karthi-laptop:8080/dojo
+ org.apache.geronimo.configs/geronimo-gbean-deployer/2.0-M3/car
+ org.apache.geronimo.configs/hot-deployer/2.0-M3/car
+ org.apache.geronimo.configs/j2ee-deployer/2.0-M3/car
+ org.apache.geronimo.configs/j2ee-security/2.0-M3/car
+ org.apache.geronimo.configs/j2ee-server/2.0-M3/car
+ org.apache.geronimo.configs/j2ee-system/2.0-M3/car
+ org.apache.geronimo.configs/jasper/2.0-M3/car
+ org.apache.geronimo.configs/javamail/2.0-M3/car
+ org.apache.geronimo.configs/jaxws-deployer/2.0-M3/car
+ org.apache.geronimo.configs/jee-specs/2.0-M3/car
+ org.apache.geronimo.configs/jetty6/2.0-M3/car
+ org.apache.geronimo.configs/jetty6-deployer/2.0-M3/car
+ org.apache.geronimo.configs/openejb/2.0-M3/car
+ org.apache.geronimo.configs/openejb-deployer/2.0-M3/car
+ org.apache.geronimo.configs/persistence-jpa10-deployer/2.0-M3/car
+ org.apache.geronimo.configs/remote-deploy-jetty/2.0-M3/car @ http://karthi-laptop:8080/remote-
deploy
+ org.apache.geronimo.configs/rmi-naming/2.0-M3/car
+ org.apache.geronimo.configs/sharedlib/2.0-M3/car
+ org.apache.geronimo.configs/system-database/2.0-M3/car
+ org.apache.geronimo.configs/transaction-jta11/2.0-M3/car
+ org.apache.geronimo.configs/webconsole-jetty6/2.0-M3/car
`-> framework.war @ http://karthi-laptop:8080/console
`-> standard.war @ http://karthi-laptop:8080/console-standard
+ org.apache.geronimo.configs/webservices-common/2.0-M3/car
+ org.apache.geronimo.configs/welcome-jetty/2.0-M3/car @ http://karthi-laptop:8080/
```

```
org.apache.geronimo.configs/axis/2.0-M3/car
org.apache.geronimo.configs/axis2/2.0-M3/car
org.apache.geronimo.configs/axis2-deployer/2.0-M3/car
org.apache.geronimo.configs/client/2.0-M3/car
org.apache.geronimo.configs/client-corba-yoko/2.0-M3/car
org.apache.geronimo.configs/client-security/2.0-M3/car
org.apache.geronimo.configs/client-system/2.0-M3/car
org.apache.geronimo.configs/client-transaction/2.0-M3/car
org.apache.geronimo.configs/cxf/2.0-M3/car
org.apache.geronimo.configs/directory/2.0-M3/car
org.apache.geronimo.configs/j2ee-corba-yoko/2.0-M3/car
org.apache.geronimo.configs/jetty6-clustering-builder-wadi/2.0-M3/car
org.apache.geronimo.configs/jetty6-clustering-wadi/2.0-M3/car
org.apache.geronimo.configs/jsr88-cli/2.0-M3/car
org.apache.geronimo.configs/jsr88-deploymentfactory/2.0-M3/car
org.apache.geronimo.configs/jsr88-ear-configurer/2.0-M3/car
org.apache.geronimo.configs/jsr88-jar-configurer/2.0-M3/car
org.apache.geronimo.configs/jsr88-rar-configurer/2.0-M3/car
org.apache.geronimo.configs/jsr88-war-configurer/2.0-M3/car
org.apache.geronimo.configs/ldap-realm/2.0-M3/car
org.apache.geronimo.configs/offline-deployer/2.0-M3/car
org.apache.geronimo.configs/online-deployer/2.0-M3/car
org.apache.geronimo.configs/openejb-corba-deployer/2.0-M3/car
org.apache.geronimo.configs/openjpa/2.0-M3/car
org.apache.geronimo.configs/shutdown/2.0-M3/car
org.apache.geronimo.configs/transformer-agent/2.0-M3/car
org.apache.geronimo.configs/uddi-jetty6/2.0-M3/car
org.apache.geronimo.configs/wadi-clustering/2.0-M3/car
```

1.1.4.3. JMSの構成

This page last changed on 4 21, 2008 by JAGUG.

JMSを構成する際はGeronimo管理コンソールで JMS Resourcesポートレットが利用できます。このポートレットを使えば、利用可能なリソースをリストするだけでなく、Geronimo上で既に利用可能となっているActiveMQ用や、別のJMSプロバイダー用のリソースを新規に作成できます。

1.2. WindowsサービスとしてのGeronimoの構成

This page last changed on 4 25, 2008 by JAGUG.

この記事では Apache Geronimo v2.0 を Microsoft Windows のサービスとして構成する方法をご紹介します。この構成をするためには別途 Java Service Wrapper が必要になります。Java Service Wrapper を使えば、サービスに対し ping を発行する、サービスがダウンした際のアクションを取る、というような追加の"コンロトール"を行うことができます。

前提となるソフトウエア

Apache Geronimo v2.0 を Microsoft Windows のサービスとしてセットアップするには、Java Service Wrapper が必須です。この例では下記の URL からダウンロード可能な Java Service Wrapper 3.2.3 を使用します。

http://sourceforge.net/projects/wrapper/

この例は Apache Geronimo v2.0 を Windows XP 上で動かしています。Apache Geronimo のバイナリー・イメージは下記の URL からダウンロードできます。

http://geronimo.apache.org/downloads.html

Apache Geronimo のインストール

Geronimo をバイナリーからインストールするのはとても簡単です。お好みのディレクトリーに zip ファイルを展開するだけです。以降、この記事では展開先のディレクトリーを <geronimo_home> と表記します。

Java Service Wrapper のインストール

Java Service Wrapper のインストールも Apache Geronimo 同様、簡単です。貴方の環境に合ったバージョンをダウンロードしたら、お好きなディレクトリーに解凍します。以降、この記事では展開先のディレクトリーを <jsw_home> と表記します。

今回の例では wrapper 関係のファイルは Geronimo のディレクトリー構造に配置しますので、いくつかのファイルが上書きされます。

wrapper をインストールしたら、下記のテーブルに従ってファイルをコピーしてください。

Source	Destination
<jsw_home>/bin/wrapper.exe	<geronimo_home>/bin/wrapper.exe
<jsw_home>/lib/wrapper.jar	<geronimo_home>/lib/wrapper.jar
<jsw_home>/lib/wrapper.dll	<geronimo_home>/lib/wrapper.dll

更にこの構成の過程では、以下のファイルも作成されます。

- <geronimo_home>/bin/g_service.bat
- <geronimo_home>/bin/Install_Geronimo_NT.bat
- <geronimo_home>/bin/Uninstall_Geronimo_NT.bat
- <geronimo_home>/var/config/wrapper.conf

ラッパーの構成

Java Service Wrapper はテスト・スクリプトを実行するために <jsw_home>¥conf ディレクトリー上に定義済の wrapper.conf という構成ファイルのサンプルを提供しています。このセクションでは利便性を考えて Geronimo 専用のwrapper.conf ファイルを使うことにします。

今回はすべての wrapper ファイルを Geronimo のディレクトリー構造の中に収めようとしていますので、下記の例のように構成ファイルを作成し、<geronimo_home>/var/conf/wrapper.conf ディレクトリーに置いて下さい。

wrapper.conf

```
# Location of java.exe (in Windows)

wrapper.java.command=<java_home>/bin/java

# These additional parameters are required to start the server since we are not setting any
environment variables prior to running the wrapper.

wrapper.java.additional.1=-javaagent:"<geronimo_home>/bin/jpa.jar"
wrapper.java.additional.2=-Djava.ext.dirs="<geronimo_home>/lib/ext;<java_home>/jre/lib/ext"
wrapper.java.additional.3=-Djava.endorsed.dirs="<geronimo_home>/lib/endorsed;<java_home>/jre/lib/
endorsed"
wrapper.java.additional.4=-Dorg.apache.geronimo.base.dir="<geronimo_home>"
wrapper.java.additional.5=-Djava.io.tmpdir="<geronimo_home>/var/temp"

# Good old classpath, make sure to include /bin/server.jar and /bin/shutdown.jar

wrapper.java.classpath.1=../lib/wrapper.jar
wrapper.java.classpath.2=../bin/server.jar
wrapper.java.classpath.3=../bin/shutdown.jar
wrapper.java.classpath.4=../lib/geronimo-cli-2.0-SNAPSHOT.jar
wrapper.java.classpath.5=../lib/geronimo-kernel-2.0-SNAPSHOT.jar
wrapper.java.classpath.6=../lib/geronimo-transformer-2.0-SNAPSHOT.jar
wrapper.java.classpath.7=../lib/commons-cli-1.0.jar
wrapper.java.classpath.8=../lib/commons-logging-1.0.4.jar
wrapper.java.classpath.9=../lib/cglib-nodep-2.1_3.jar
wrapper.java.classpath.10=../lib/log4j-1.2.14.jar
wrapper.java.classpath.11=../lib/xpp3-1.1.3.3.jar
wrapper.java.classpath.12=../lib/xstream-1.1.3.jar

# Location of the wrapper.dll (in Windows)

wrapper.java.library.path.1=../lib

# Main class the wrapper will use.

wrapper.java.mainclass=org.tanukisoftware.wrapper.WrapperSimpleApp

# Geronimo start class and parameters.

wrapper.app.parameter.1=org.apache.geronimo.cli.daemon.DaemonCLI

# Log settings.

wrapper.console.format=PM
wrapper.console.loglevel=INFO
wrapper.logfile=../var/log/wrapper.log
wrapper.logfile.format=LPTM
wrapper.logfile.loglevel=INFO
wrapper.logfile.maxsize=0
wrapper.logfile.maxfiles=0
wrapper.syslog.loglevel=INFO

# Windows service name and description.

wrapper.console.title=Apache Geronimo v2.0 Server
wrapper.ntservice.name=Geronimo
wrapper.ntservice.displayname=Apache Geronimo v2.0 Server
wrapper.ntservice.description=Apache Geronimo v2.0 Server
wrapper.ntservice.dependency.1=
wrapper.ntservice.starttype=AUTO_START
wrapper.ntservice.interactive=false
```

このサービス・ラッパーは、同じクラスを使うか、サーバーを開始・終了するかどうか、により3種類の異なった統合の方法を提供します。

Geronimo 向けに、System.exit() が呼ばれた時にきれいにシャットダウンさせるために最もシンプルな方法を使います。この統合ではラッパーのメイン・クラスは **`WrapperSimpleApp`** を設定し、サーバーを始動するためのメイン・クラスには **`org.apache.geronimo.cli.daemon.DaemonCLI`** クラスを指定します。今の時点では、サーバーを始動する時にパラメーターは渡しませんが **`wrapper.app.parameter`** で追加のパラメーターを指定することもできます。ラッパーは System.exit() を呼び出すことで停止します。

前述の通り、この例ではすべてのラッパー関係のファイルは Geronimo のディレクトリー構造の中に配置するようにしています。上の wrapper.conf を見ると判るように、殆どのディレクトリーの参照は Geronimo のインストレーション・ディレクトリーへの相対参照としています。貴方は <geronimo_home> と <java_home> を自分の環境に合った適切な値に設定するだけです。

次に、サービスをコマンド行から実行するためのバッチ・ファイルを作成しますが、実際にサービスをインストールする必要はありません。この点は、テストやデバッグをする際にはとても重宝します。<geronimo_home>/bin ディレクトリーに **`g_service.bat`** というバッチ・ファイルを作成し、下記の例の内容をコピーしてください。

g_service.bat

```
@echo off
setlocal

rem Copyright (c) 1999, 2006 Tanuki Software Inc.
rem
rem Java Service Wrapper general startup script
rem

rem
rem Resolve the real path of the wrapper.exe
rem  For non NT systems, the _REALPATH and _WRAPPER_CONF values
rem  can be hard-coded below and the following test removed.
rem
if "%OS%"=="Windows_NT" goto nt
echo This script only works with NT-based versions of Windows.
goto :eof

:nt
rem
rem Find the application home.
rem
rem %~dp0 is location of current script under NT
set _REALPATH=%~dp0

rem Decide on the wrapper binary.
set _WRAPPER_BASE=wrapper
set _WRAPPER_EXE=%_REALPATH%%_WRAPPER_BASE%-windows-x86-32.exe
if exist "%_WRAPPER_EXE%" goto conf
set _WRAPPER_EXE=%_REALPATH%%_WRAPPER_BASE%-windows-x86-64.exe
if exist "%_WRAPPER_EXE%" goto conf
set _WRAPPER_EXE=%_REALPATH%%_WRAPPER_BASE%.exe
if exist "%_WRAPPER_EXE%" goto conf
echo Unable to locate a Wrapper executable using any of the following names:
echo %_REALPATH%%_WRAPPER_BASE%-windows-x86-32.exe
echo %_REALPATH%%_WRAPPER_BASE%-windows-x86-64.exe
echo %_REALPATH%%_WRAPPER_BASE%.exe
pause
goto :eof

rem
rem Find the wrapper.conf
rem
:conf
set _WRAPPER_CONF="%~f1"
if not %_WRAPPER_CONF%=="" goto startup
set _WRAPPER_CONF="%_REALPATH%..\var\config\wrapper.conf"

rem
rem Start the Wrapper
rem
```

```
:startup
"%_WRAPPER_EXE%" -c %_WRAPPER_CONF%
if not errorlevel 1 goto :eof
pause
```

このファイルはラッパーによりサンプルとして提供されているものですが、**wrapper.conf** の場所を Geronimo の var/config ディレクトリーを指すように編集した変更されたバージョンです。

これで Geronimo をコマンド・ラインからサービスとして起動できるようになりました。下記のコマンドを実行してみてください。

<geronimo_home>/bin/g_service.bat

下記の例のような画面が表示されるはずです。表示の一部は切り捨てられていることにご注意ください。

```
{D:\geronimo-tomcat6-jee5-2.0\bin>g_service.bat
wrapper  | --> Wrapper Started as Console
wrapper  | Launching a JVM...
jvm 1    | Wrapper (Version 3.2.3) http://wrapper.tanukisoftware.org
jvm 1    |   Copyright 1999-2006 Tanuki Software, Inc.  All Rights Reserved.
jvm 1    |
jvm 1    | Booting Geronimo Kernel (in Java 1.5.0_06)...
jvm 1    | Starting Geronimo Application Server v2.0
jvm 1    |
jvm 1    | [*                                ]  0%   0s Loading
jvm 1    | [*-                               ]  0%   0s  Loading org.apache.ge...
jvm 1    | [*>                               ]  6%   0s  Loading org.apache.ge...

...

jvm 1    | [*********************************] 100%  19s Startup complete
jvm 1    |   Listening on Ports:
jvm 1    |     1050 127.0.0.1 CORBA Naming Service
jvm 1    |     1099 0.0.0.0   RMI Naming
jvm 1    |     1527 0.0.0.0   Derby Connector
jvm 1    |     2001 127.0.0.1 OpenEJB ORB Adapter
jvm 1    |     4201 0.0.0.0   org.apache.geronimo.openejb.EjbDaemonGBean
jvm 1    |     4242 0.0.0.0   Remote Login Listener
jvm 1    |     6882 127.0.0.1 OpenEJB ORB Adapter
jvm 1    |     8009 0.0.0.0   Tomcat Connector AJP
jvm 1    |     8080 0.0.0.0   Tomcat Connector HTTP
jvm 1    |     8443 0.0.0.0   Tomcat Connector HTTPS
jvm 1    |     9999 0.0.0.0   JMX Remoting Connector
jvm 1    |    61613 0.0.0.0   ActiveMQ Transport Connector
jvm 1    |    61616 0.0.0.0   ActiveMQ Transport Connector
jvm 1    |
jvm 1    |   Started Application Modules:
jvm 1    |     EAR: org.apache.geronimo.configs/webconsole-tomcat/2.0/car
jvm 1    |     RAR: org.apache.geronimo.configs/activemq-ra/2.0/car
jvm 1    |     RAR: org.apache.geronimo.configs/system-database/2.0/car
jvm 1    |     WAR: org.apache.geronimo.configs/dojo-tomcat/2.0/car
jvm 1    |     WAR: org.apache.geronimo.configs/remote-deploy-tomcat/2.0/car
jvm 1    |     WAR: org.apache.geronimo.configs/welcome-tomcat/2.0/car
jvm 1    |
jvm 1    |   Web Applications:
jvm 1    |     http://localhost:8080/
jvm 1    |     http://localhost:8080/console
jvm 1    |     http://localhost:8080/console-standard
jvm 1    |     http://localhost:8080/dojo
jvm 1    |     http://localhost:8080/remote-deploy
jvm 1    |
jvm 1    | Geronimo Application Server started
```

おめでとうございます!!! これで Geronimo サーバーを Windows サービスとして動かすことができました。

次のステップでは、他の既存サービスと同様に Windows の始動時に管理されるように、実際にサービスをインストールします。

サービスのインストールと除去

以下の2つのバッチ・ファイルは、Geronimo を Windows サービスとしてインストールおよび除去するためのものです。これらのファイルを **<geronimo_home>/bin** ディレクトリーに作成し、下記の例の中身をコピーしてください。

Install_Geronimo_NT.bat

```
@echo off
setlocal

rem Copyright (c) 1999, 2006 Tanuki Software Inc.
rem
rem Java Service Wrapper general NT service install script
rem

if "%OS%"=="Windows_NT" goto nt
echo This script only works with NT-based versions of Windows.
goto :eof

:nt
rem
rem Find the application home.
rem
rem %~dp0 is location of current script under NT
set _REALPATH=%~dp0

rem Decide on the wrapper binary.
set _WRAPPER_BASE=wrapper
set _WRAPPER_EXE=%_REALPATH%%_WRAPPER_BASE%-windows-x86-32.exe
if exist "%_WRAPPER_EXE%" goto conf
set _WRAPPER_EXE=%_REALPATH%%_WRAPPER_BASE%-windows-x86-64.exe
if exist "%_WRAPPER_EXE%" goto conf
set _WRAPPER_EXE=%_REALPATH%%_WRAPPER_BASE%.exe
if exist "%_WRAPPER_EXE%" goto conf
echo Unable to locate a Wrapper executable using any of the following names:
echo %_REALPATH%%_WRAPPER_BASE%-windows-x86-32.exe
echo %_REALPATH%%_WRAPPER_BASE%-windows-x86-64.exe
echo %_REALPATH%%_WRAPPER_BASE%.exe
pause
goto :eof

rem
rem Find the wrapper.conf
rem
:conf
set _WRAPPER_CONF="%~f1"
if not %_WRAPPER_CONF%=="" goto startup
set _WRAPPER_CONF="%_REALPATH%..\var\config\wrapper.conf"

rem
rem Install the Wrapper as an NT service.
rem
:startup
"%_WRAPPER_EXE%" -i %_WRAPPER_CONF%
if not errorlevel 1 goto :eof
pause
```

Apache Geronimo を MS Windows サービスとしてインストールするには下記のコマンドを実行してください。

<geronimo_home>/bin/Install_Geronimo_NT.bat

この例のような確認画面が出力されます。

```
D:\geronimo-tomcat6-jee5-2.0\bin>Install_Geronimo_NT.bat
wrapper  | Apache Geronimo v2.0 Server installed.
```

Uninstall_Geronimo_NT.bat

```
@echo off
setlocal

rem Copyright (c) 1999, 2006 Tanuki Software Inc.
rem
rem Java Service Wrapper general NT service uninstall script
rem

if "%OS%"=="Windows_NT" goto nt
echo This script only works with NT-based versions of Windows.
goto :eof

:nt
rem
rem Find the application home.
rem
rem %~dp0 is location of current script under NT
set _REALPATH=%~dp0

rem Decide on the wrapper binary.
set _WRAPPER_BASE=wrapper
set _WRAPPER_EXE=%_REALPATH%%_WRAPPER_BASE%-windows-x86-32.exe
if exist "%_WRAPPER_EXE%" goto conf
set _WRAPPER_EXE=%_REALPATH%%_WRAPPER_BASE%-windows-x86-64.exe
if exist "%_WRAPPER_EXE%" goto conf
set _WRAPPER_EXE=%_REALPATH%%_WRAPPER_BASE%.exe
if exist "%_WRAPPER_EXE%" goto conf
echo Unable to locate a Wrapper executable using any of the following names:
echo %_REALPATH%%_WRAPPER_BASE%-windows-x86-32.exe
echo %_REALPATH%%_WRAPPER_BASE%-windows-x86-64.exe
echo %_REALPATH%%_WRAPPER_BASE%.exe
pause
goto :eof

rem
rem Find the wrapper.conf
rem
:conf
set _WRAPPER_CONF="%~f1"
if not %_WRAPPER_CONF%=="" goto startup
set _WRAPPER_CONF="%_REALPATH%..\var\config\wrapper.conf"

rem
rem Uninstall the Wrapper as an NT service.
rem
:startup
"%_WRAPPER_EXE%" -r %_WRAPPER_CONF%
if not errorlevel 1 goto :eof
pause
```

Apache Geronimoサービスを除去するには下記のコマンドを実行してください。

<geronimo_home>/bin/Uninstall_Geronimo_NT.bat

この例のような確認画面が出力されます。

```
D:\geronimo-tomcat6-jee5-2.0\bin>Uninstall_Geronimo_NT.bat
wrapper  | Apache Geronimo v2.0 Server removed.
```

1.3. Geronimo-Jettyでの仮想ホストの構成

This page last changed on 4 21, 2008 by JAGUG.

1.4. Geronimo-Tomcatでの仮想ホストの構成 のセクションではGeronimoの **config.xml** に仮想ホストを定義し、アプリケーションのデプロイメント・プラン側での構成作業を最小にしつつアプリケーションを特定のホスト (Virtual Hosts) にデプロイする方法を説明しました。このように仮想ホストをサーバー・レベル (config.xml) で定義する必要があるのはGeronimoのTomcatディストリビューションだけです。

Apache GeronimoのJettyディストリビューションでは構成作業は劇的に簡単になります。必要なのは、アプリケーションのデプロイメント・プランに **<virtual-host>** タグを定義して、そのホスト名 (virtual host) がクライアントから名前解決できることを確認するだけです。Geronimoサーバーの側ではそれ以外の追加の構成作業は必要ありません。

この例では 5. クイック・スタート - いますぐ始めたい人の Apache Geronimo セクションで使ったHelloWorld Webサンプル・アプリケーションを引き続き使います。

1. ファイルシステムの適当な場所にHelloWorldディレクトリーを作成します。以降、この場所を<HelloWorld_home>と呼ぶことにします。
2. <HelloWorld_home>ディレクトリーに HelloWorld.jsp という名前のファイルを作成し、以下の内容を貼り付けます。

```
code: Border style is not a valid CSS2 border-style value

<html>
<head>
<jsp:useBean id="datetime" class="java.util.Date"/>
<title>Basic HelloWorld JSP</title>
</head>
<body bgcolor="#909DB8">
<h1><font face="tahoma" color="white">Hello world from GERONIMO!</font></h1>
<font face="tahoma" color="white">on $

Unknown macro: {datetime}
</font>
</body>
</html>
```

1. <HelloWorld_home>配下に WEB-INF ディレクトリーを作成します。
2. <HelloWorld_home>¥WEB-INFディレクトリーに **web.xml** ファイルを作成し、以下の内容を貼り付けます。

```
code: Border style is not a valid CSS2 border-style value

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"

xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

<welcome-file-list>
<welcome-file>HelloWorld.jsp</welcome-file>
</welcome-file-list>

</web-app>
```

1. <HelloWorld_home>¥WEB-INF ディレクトリーに **geronimo-web.xml** という名前のファイルを作成し、以下の内容を貼り付けます。

```
code: Border style is not a valid CSS2 border-style value

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.2">
<environment>
<moduleId>
<groupId>sample.applications</groupId>
<artifactId>HelloWorldApp</artifactId>
```

```
<version>2.0</version>
<type>war</type>
</moduleId>
</environment>
<context-root>/hello</context-root>

<!-- Add this line to define a new Virtual Host in Geronimo - Jetty -->
<virtual-host>virtualhost1.com</virtual-host>
</web-app>
```

このデプロイメント・プランを 5. クイック・スタート - いますぐ始めたい人の Apache Geronimo セクションで説明したデプロイメント・プランと比較すると、両者の違いは **<virtual-host>virtualhost1.com</virtual-host>** という一行だけであることに気付かれることでしょう。Jetty版のGeronimoで仮想ホストを定義するために必要なことは、これですべてです。

このアプリケーションをデプロイする際には、プロイヤー・ツールで --inPlace オプションを使えます。そうすれば、アプリケーションをパッケージする必要はありませんし、別のどこかにコピーする必要もありません。下記のコマンドを実行するだけです。

<geronimo_home>\bin\deploy --user system --password manager deploy --inPlace <HelloWorld_home>

```
D:\geronimo-jetty6-jee5-2.0-M2\bin>deploy --user system --password manager deploy --inPlace
\HelloWorld_2.0
Using GERONIMO_BASE:   D:\geronimo-jetty6-jee5-2.0-M2
Using GERONIMO_HOME:   D:\geronimo-jetty6-jee5-2.0-M2
Using GERONIMO_TMPDIR: D:\geronimo-jetty6-jee5-2.0-M2\var\temp
Using JRE_HOME:        C:\Java\jdk1.5.0_06\\jre
Deployed sample.applications/HelloWorldApp/2.0/war @
http://hcunico:8080/hello
```

デプロイが成功すると、アプリケーションが http://XYZ:8080/hello というホストにデプロイされたとのメッセージが画面に表示されますが、実際は該当のアプリケーションはgeronimo-web.xmlというデプロイメント・プラン中で<virtual-host>タグにより定義された仮想ホストからのみアクセス可能である点にご注意ください。

1.4. Geronimo-Tomcatでの仮想ホストの構成

This page last changed on 4 21, 2008 by JAGUG.

この記事では、Tomcat版のApache Geronimoで仮想ホストを構成する方法をご紹介します。デフォルトでは、Geronimoのアプリケーションをデプロイしてスタートすると、そのアプリケーションは、利用可能なホスト名すべてに対してリスンしています。仮想ホストを構成すれば、アプリケーションが特定のホスト名やIPアドレスのみをリスンするようにできます。この記事で述べている構成手順はいくつかのホスト名で同一のIPアドレスを共用しているときでも有効です。

Geronimoで仮想ホストを構成する際は、基本的に以下の手順を踏みます。

- ローカルホストまたはDNSの構成
- 仮想ホストの定義
- デプロイメント・プランでの仮想ホストの宣言
- アプリケーションのデプロイ

この記事は参考としてHelloWorldアプリケーションを使いますが、このアプリケーションは 5. クイック・スタート - いますぐ始めたい人の Apache Geronimo セクションでカバーされています。

ローカルホストまたはDNSの構成

この構成を組むためには、貴方がこれから定義しようとしている仮想ホスト名が名前解決されなくてはなりません。貴方のネットワーク環境により、DNSにGeronimoサーバーのIPをエントリーとして追加しても結構です。また別の方法として、ローカルホストのテーブルへエントリーを追加するのでもよいです。ローカルホスト・テーブルへの登録の仕方はOSにより様々です。例えばWindowsでは `%SystemRoot%\system32\drivers\etc\hosts` ですし、UNIXベースのオペレーティング・システムでは通常は `/etc/hosts` になります。

例として、ローカルホスト・テーブルに定義済の下記のホスト名を使っていきましょう。

```
127.0.0.1 localhost virtualhost1.com virtualhost2.com virtualhost3.com virtualhost4.com
```

貴方のシステムでこれらの名前解決ができることを確認してください。

仮想ホストの定義

次に、仮想ホストをGeronimoに認識させるためにGeronimoの **`config.xml`** に仮想ホストを定義します。このセクションでは2つの異なる仮想ホストの定義を使います。つまりGeronimoの構成では2つの新しい HostGBean (TomcatVirtualHost1 と TomcatVirtualHost2)を作成します。そのうちの1つは複数のホスト別名を持つようにします。この例では、あるアプリケーションがひとつの仮想ホスト(virtualhost1.comとします)をリスンしており、もうひとつのアプリケーションは追加の2つのホスト別名(virtualhost3.comとvirtualhost4.comとします)を持っている別の仮想ホスト(virtualhost2.comとします) をリスンしているような構成を作ってみます

`<geronimo_home>/var` ディレクトリー上の config.xml ファイルを開き、**`<module name="org.apache.geronimo.configs/tomcat6/2.0/car">`** という行を探してください。これがTomcat構成モジュールの始まりの行であり、この行のすぐ下に仮想ホストの定義を追加します。

最初のHostGBeanである **`TomcatVirtualHost_1`** を定義するには、**`<module name="org.apache.geronimo.configs/tomcat6/2.0/car">`** という行のすぐ次に以下の行を追加してください。

confog.xmlからの抜粋

```
...
<gbean gbeanInfo="org.apache.geronimo.tomcat.HostGBean" name="org.apache.geronimo.configs/
tomcat6/2.0/car?ServiceModule=org.apache.geronimo.configs/tomcat6/2.0/
car,j2eeType=Host,name=TomcatVirtualHost_1">
<attribute name="className">org.apache.catalina.core.StandardHost</attribute>
<attribute name="initParams">name=virtualhost1.com
appBase=
workDir=work</attribute>
</gbean>
...
```

2番目のHostGBeanである **TomcatVirtualHost_2** の定義は、最初のHostGBeanのすぐ下に追加してください。この2つのHostGBeanは別々に分離されているので見分けるのは容易でしょう。この2つは、別名を定義するための **<attribute name="aliases">..,..</attribute>** の部分が異なっています。

confog.xmlからの抜粋

```
...
<gbean gbeanInfo="org.apache.geronimo.tomcat.HostGBean" name="org.apache.geronimo.configs/
tomcat6/2.0/car?ServiceModule=org.apache.geronimo.configs/tomcat6/2.0/
car,j2eeType=Host,name=TomcatVirtualHost_2">
<attribute name="className">org.apache.catalina.core.StandardHost</attribute>
<attribute name="initParams">name=virtualhost2.com
appBase=
workDir=work</attribute>
<attribute name="aliases">virtualhost3.com,virtualhost4.com</attribute>
</gbean>
...
```

これでGeronimo.に2つの異なる仮想ホストを構成することができました。**config.xml** に対して行った変更を保管してGeronimoを start してください。

Back to Top

ご参考までに、下記に2つのHostGBeanが定義済の **config.xml** の全体を掲載します。

config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Licensed to the Apache Software Foundation (ASF) under one or more
contributor license agreements.  See the NOTICE file distributed with
this work for additional information regarding copyright ownership.
The ASF licenses this file to You under the Apache License, Version 2.0
(the "License"); you may not use this file except in compliance with
the License.  You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
-->

<!-- $Rev: 562176 $ $Date: 2007-08-02 12:31:43 -0400 (Thu, 02 Aug 2007) $ -->

<!-- ============================================================ -->
<!-- Warning - This XML file is re-generated by Geronimo when -->
<!-- changes are made to Geronimo's configuration, therefore  -->
<!-- any comments added to this file will be lost.            -->
<!-- ============================================================ -->

<attributes xmlns="http://geronimo.apache.org/xml/ns/attributes-1.1">

<module name="org.apache.geronimo.configs/rmi-naming/2.0/car">
<gbean name="RMIRegistry">
<attribute name="port">${NamingPort + PortOffset}</attribute>
</gbean>
<gbean name="NamingProperties">
<!-- Check whether this really works if host name is 0.0.0.0 -->
<attribute name="namingProviderUrl">rmi://${ServerHostname}:${NamingPort + PortOffset}</attribute>
</gbean>
<gbean name="DownloadedPluginRepos">
<attribute name="repositoryList">http://geronimo.apache.org/plugins/plugin-repository-
list-2.0.txt</attribute>
<attribute name="userRepositories">[]</attribute>
```

```
</gbean>
</module>

<module name="org.apache.geronimo.configs/j2ee-server/2.0/car"/>

<module name="org.apache.geronimo.configs/transaction/2.0/car"/>

<module name="org.apache.geronimo.configs/j2ee-security/2.0/car">
<gbean name="JMXService">
<attribute name="protocol">rmi</attribute>
<attribute name="host">${ServerHostname}</attribute>
<attribute name="port">${JMXPort + PortOffset}</attribute>
<attribute name="urlPath">/jndi/rmi://${ServerHostname}:${NamingPort + PortOffset}/JMXConnector</
attribute>
</gbean>
</module>

<module name="org.apache.geronimo.configs/axis/2.0/car" load="false"/>
<module name="org.apache.geronimo.configs/axis2/2.0/car" load="false"/>
<module name="org.apache.geronimo.configs/cxf/2.0/car" load="false"/>

<module name="org.apache.geronimo.configs/openejb/2.0/car">
<gbean name="EJBNetworkService">
<attribute name="port">${OpenEJBPort + PortOffset}</attribute>
<attribute name="host">${ServerHostname}</attribute>
</gbean>
</module>

<module name="org.apache.geronimo.configs/j2ee-corba-yoko/2.0/car" load="false">
<gbean name="NameServer">
<attribute name="port">${COSNamingPort + PortOffset}</attribute>
<attribute name="host">${COSNamingHost}</attribute>
</gbean>
<gbean name="Server">
<attribute name="port">${ORBSSLPort + PortOffset}</attribute>
<attribute name="host">${ORBSSLHost}</attribute>
</gbean>
<gbean name="UnprotectedServer">
<attribute name="port">${ORBPort + PortOffset}</attribute>
<attribute name="host">${ORBHost}</attribute>
</gbean>
</module>

<module name="org.apache.geronimo.configs/system-database/2.0/car">
<gbean name="DerbyNetwork">
<attribute name="host">${ServerHostname}</attribute>
<attribute name="port">${DerbyPort + PortOffset}</attribute>
</gbean>
</module>

<module name="org.apache.geronimo.configs/activemq-broker/2.0/car">
<gbean name="ActiveMQ.tcp.default">
<attribute name="host">${ServerHostname}</attribute>
<attribute name="port">${ActiveMQPort + PortOffset}</attribute>
</gbean>
<gbean name="ActiveMQ.stomp.default">
<attribute name="host">${ServerHostname}</attribute>
<attribute name="port">${ActiveMQStompPort + PortOffset}</attribute>
</gbean>
</module>

<module name="org.apache.geronimo.configs/activemq-ra/2.0/car">
<gbean name="ActiveMQ RA">
<attribute name="ServerUrl">tcp://${ServerHostname}:${ActiveMQPort + PortOffset}</attribute>
</gbean>
</module>
```

```
<module name="org.apache.geronimo.configs/jasper/2.0/car" load="false"/>

<module name="org.apache.geronimo.configs/myfaces/2.0/car"/>

<module name="org.apache.geronimo.configs/tomcat6/2.0/car">
<gbean gbeanInfo="org.apache.geronimo.tomcat.HostGBean"
name="org.apache.geronimo.configs/tomcat6/2.0/car?ServiceModule=org.apache.geronimo.configs/
tomcat6/2.0/car,
j2eeType=Host,name=TomcatVirtualHost_1">
<attribute name="className">org.apache.catalina.core.StandardHost</attribute>
<attribute name="initParams">name=virtualhost1.com
appBase=
workDir=work
</attribute>
</gbean>
<gbean gbeanInfo="org.apache.geronimo.tomcat.HostGBean"
name="org.apache.geronimo.configs/tomcat6/2.0/car?ServiceModule=org.apache.geronimo.configs/
tomcat6/2.0/car,
j2eeType=Host,name=TomcatVirtualHost_2">
<attribute name="className">org.apache.catalina.core.StandardHost</attribute>
<attribute name="initParams">name=virtualhost2.com
appBase=
workDir=work
</attribute>
<attribute name="aliases">virtualhost3.com,virtualhost4.com</attribute>
</gbean>
<!-- To disable accesslogging uncomment the following lines
<gbean name="TomcatEngine">
<reference name="TomcatValveChain" />
</gbean>
<gbean name="FirstValve" load="false"></gbean>
-->
<gbean name="TomcatResources"/>
<gbean name="TomcatWebConnector">
<attribute name="host">${ServerHostname}</attribute>
<attribute name="port">${HTTPPortPrimary + PortOffset}</attribute>
<attribute name="redirectPort">${HTTPSPortPrimary + PortOffset}</attribute>
</gbean>
<gbean name="TomcatAJPConnector">
<attribute name="host">${ServerHostname}</attribute>
<attribute name="port">${AJPPortPrimary + PortOffset}</attribute>
<attribute name="redirectPort">${HTTPSPortPrimary + PortOffset}</attribute>
</gbean>
<gbean name="TomcatWebSSLConnector">
<attribute name="host">${ServerHostname}</attribute>
<attribute name="port">${HTTPSPortPrimary + PortOffset}</attribute>
</gbean>
</module>

<!--
NOTE: n.b. be sure the gbean deployer is explicitly loaded before j2ee-deployer
so that defaultEnvironment overrides work properly
-->
<module name="org.apache.geronimo.configs/geronimo-gbean-deployer/2.0/car"/>

<module name="org.apache.geronimo.configs/j2ee-deployer/2.0/car">
<gbean name="WebBuilder">
<attribute name="defaultNamespace">http://geronimo.apache.org/xml/ns/j2ee/web/tomcat-2.0</
attribute>
</gbean>
<gbean name="EnvironmentEntryBuilder">
<attribute name="eeNamespaces">http://java.sun.com/xml/ns/j2ee,http://java.sun.com/xml/ns/javaee</
attribute>
</gbean>
</module>
```

```
<module name="org.apache.geronimo.configs/connector-deployer/2.0/car">
<gbean name="ResourceRefBuilder">
<attribute name="eeNamespaces">http://java.sun.com/xml/ns/j2ee,http://java.sun.com/xml/ns/javaee</
attribute>
<attribute name="defaultEnvironment">
<environment xmlns="http://geronimo.apache.org/xml/ns/deployment-1.2">
<dependencies>
<dependency>
<groupId>org.apache.geronimo.configs</groupId>
<artifactId>j2ee-corba-yoko</artifactId>
<type>car</type>
</dependency>
</dependencies>
</environment>
</attribute>
</gbean>

<gbean name="AdminObjectRefBuilder">
<attribute name="eeNamespaces">http://java.sun.com/xml/ns/j2ee,http://java.sun.com/xml/ns/javaee</
attribute>
</gbean>

<gbean name="ClientResourceRefBuilder">
<attribute name="eeNamespaces">http://java.sun.com/xml/ns/j2ee,http://java.sun.com/xml/ns/javaee</
attribute>
<attribute name="defaultEnvironment">
<environment xmlns="http://geronimo.apache.org/xml/ns/deployment-1.2">
<dependencies>
<dependency>
<groupId>org.apache.geronimo.configs</groupId>
<artifactId>client-corba-yoko</artifactId>
<type>car</type>
</dependency>
</dependencies>
</environment>
</attribute>
</gbean>
</module>

<module name="org.apache.geronimo.configs/persistence-jpa10-deployer/2.0/car">
<gbean name="PersistenceUnitBuilder">
<attribute
name="defaultPersistenceProviderClassName">org.apache.openjpa.persistence.PersistenceProviderImpl</
attribute>
<attribute name="defaultPersistenceUnitProperties">
openjpa.Log=commons
#   openjpa.jdbc.DBDictionary=org.apache.openjpa.jdbc.sql.DerbyDictionary
openjpa.jdbc.SynchronizeMappings=buildSchema(ForeignKeys=true)
openjpa.jdbc.UpdateManager=operation-order
openjpa.Sequence=table(Table=OPENJPASEQ, Increment=100)
</attribute>
<attribute name="defaultEnvironment">
<environment xmlns="http://geronimo.apache.org/xml/ns/deployment-1.2">
<dependencies>
<dependency>
<groupId>org.apache.geronimo.configs</groupId>
<artifactId>openjpa</artifactId>
<type>car</type>
</dependency>
</dependencies>
</environment>
</attribute>
</gbean>
</module>
```

```
<module name="org.apache.geronimo.configs/openejb-deployer/2.0/car">
<gbean name="EjbRefBuilder">
<attribute name="eeNamespaces">http://java.sun.com/xml/ns/j2ee,http://java.sun.com/xml/ns/javaee</
attribute>
</gbean>

<gbean name="ClientEjbRefBuilder">
<attribute name="eeNamespaces">http://java.sun.com/xml/ns/j2ee,http://java.sun.com/xml/ns/javaee</
attribute>
</gbean>
</module>

<module name="org.apache.geronimo.configs/client-deployer/2.0/car">
</module>

<module name="org.apache.geronimo.configs/axis2-deployer/2.0/car"
condition="props.getProperty('org.apache.geronimo.jaxws.provider', 'axis2') == 'axis2'">

<gbean name="Axis2ModuleBuilderExtension">
<attribute name="listener">?name=TomcatWebContainer</attribute>
<attribute name="defaultEnvironment">
<environment xmlns="http://geronimo.apache.org/xml/ns/deployment-1.2">
<dependencies>
<dependency>
<groupId>org.apache.geronimo.configs</groupId>
<artifactId>tomcat6</artifactId>
<type>car</type>
</dependency>
</dependencies>
</environment>
</attribute>
</gbean>
</module>

<module name="org.apache.geronimo.configs/cxf-deployer/2.0/car"
condition="props['org.apache.geronimo.jaxws.provider'] == 'cxf'">

<gbean name="CXFModuleBuilderExtension">
<attribute name="listener">?name=TomcatWebContainer</attribute>
<attribute name="defaultEnvironment">
<environment xmlns="http://geronimo.apache.org/xml/ns/deployment-1.2">
<dependencies>
<dependency>
<groupId>org.apache.geronimo.configs</groupId>
<artifactId>tomcat6</artifactId>
<type>car</type>
</dependency>
</dependencies>
</environment>
</attribute>
</gbean>
</module>

<module name="org.apache.geronimo.configs/axis-deployer/2.0/car">
<gbean name="AxisServiceRefBuilder">
<attribute name="eeNamespaces">http://java.sun.com/xml/ns/j2ee,http://java.sun.com/xml/ns/javaee</
attribute>
</gbean>
<gbean name="AxisModuleBuilderExtension">
<attribute name="listener">?name=TomcatWebContainer</attribute>
<attribute name="defaultEnvironment">
<environment xmlns="http://geronimo.apache.org/xml/ns/deployment-1.2">
<dependencies>
<dependency>
<groupId>org.apache.geronimo.configs</groupId>
<artifactId>tomcat6</artifactId>
```

```
<type>car</type>
</dependency>
</dependencies>
</environment>
</attribute>
</gbean>
</module>

<module name="org.apache.geronimo.configs/javamail/2.0/car">
<gbean name="SMTPTransport">
<attribute name="host">localhost</attribute>
<attribute name="port">25</attribute>
</gbean>
</module>

<module name="org.apache.geronimo.configs/sharedlib/2.0/car">
<gbean name="SharedLib">
<attribute name="classesDirs">var/shared/classes</attribute>
<attribute name="libDirs">var/shared/lib</attribute>
</gbean>
</module>

<module name="org.apache.geronimo.configs/tomcat6-deployer/2.0/car"/>

<module name="org.apache.geronimo.configs/jasper-deployer/2.0/car"/>

<module name="org.apache.geronimo.configs/myfaces-deployer/2.0/car"/>

<module name="org.apache.geronimo.configs/welcome-tomcat/2.0/car"/>

<module name="org.apache.geronimo.configs/dojo-tomcat/2.0/car"/>

<module name="org.apache.geronimo.configs/webconsole-tomcat/2.0/car"/>

<module name="org.apache.geronimo.configs/uddi-tomcat/2.0/car" load="false"/>

<module name="org.apache.geronimo.configs/remote-deploy-tomcat/2.0/car" />

<module name="org.apache.geronimo.configs/hot-deployer/2.0/car"/>

<module name="org.apache.geronimo.configs/jsr88-rar-configurer/2.0/car"/>

<module name="org.apache.geronimo.configs/ca-helper-tomcat/2.0/car" load="false"/>

</attributes>
```

Back to Top

デプロイメント・プランでの仮想ホストの宣言

以前述べた通り、この例では 5. クイック・スタート - いますぐ始めたい人の Apache Geronimo でご説明したHelloWorldサンプル・アプリケーションを使います。これはWebアプリケーションなので、変更するべきデプロイメント・プランは **`geronimo-web.xml`** です。もし別の種類のアプリケーションを利用している場合は例えば `geronimo-application.xml` のような別の種類のデプロイメント・プランを使う必要がある場合もあるでしょう。

前のセクションでは2つの仮想ホストを定義しました。ではこれから2つのアプリケーションが各々の仮想ホストを別々に利用するような形でデプロイしてみます。

双方のケースで同一のアプリケーションを使いますが、お互いを区別するために異なった **`artifactId`** を与えてデプロイします。そうすれば、コードを変更せずに各々のデプロイメントを区別することができます。

貴方が 5. クイック・スタート - いますぐ始めたい人の Apache Geronimo のセクションで説明した手順を実施済とすると、貴方の環境は下記のような構造になっているはずです。

```
<app_home>\
```

```
+ HelloWorld.jsp
+ WEB-INF\
+ geronimo-web.xml
+ web.xml
```

geronimo-web.xml ファイルを開き、デプロイメントがユニークになるように **artifactId** と **context-root** を編集してください。更に **web-app** セクションに **host** 属性を追加して、このアプリケーションにリスンさせたい仮想ホスト、このケースでは virtualhost1.com を記述します。

geronimo-web.xml for HelloWorld_1

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.2">
<environment>
<moduleId>
<groupId>sample.applications</groupId>
<artifactId>HelloWorldApp_1</artifactId>
<version>2.0</version>
<type>war</type>
</moduleId>
</environment>
<context-root>/hello_1</context-root>
<host>virtualhost1.com</host>
</web-app>
```

geronimo-web.xml ファイルへの変更を保管し<app_home>ディレクトリーで下記のコマンドを入力してWARファイルを生成します。:

```
 jar -cvf HelloWorld_1.war *
```

一旦デプロイされれば、このアプリケーションはホスト名 virtualhost1.com だけをリスンしているはずです。

この手順を繰り返して2番目のWARを作ります、再度 **geronimo-web.xml** ファイルを編集し下記の例から中身をコピーしてください。変更するのは **artifactId** , **context-root** と **host** だけですのでご注意ください。

geronimo-web.xml for HelloWorld_2

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.2">
<environment>
<moduleId>
<groupId>sample.applications</groupId>
<artifactId>HelloWorldApp_2</artifactId>
<version>2.0</version>
<type>war</type>
</moduleId>
</environment>
<context-root>/hello_2</context-root>
<host>virtualhost2.com</host>
</web-app>
```

geronimo-web.xml ファイルへの変更を保管し<app_home> ディレクトリーで下記のコマンドを入力してWARファイルを生成します。

```
 jar -cvf HelloWorld_2.war *
```

これで2つのアプリケーションを2つの別々の仮想ホストへデプロイすることができました。

Back to Top

アプリケーションのデプロイ

ここまでで、Geronimoを2つの仮想ホストを使うように構成できました。そのうちの片方は追加の別名もリスンするよう構成されています。あとはアプリケーションをデプロイしてテストをするだけです。アプリケーションをデプロイするには以下のコマンドを<geronimo_home>¥binディレクトリーで入力してください。

deploy --user system --password manager deploy <app_home>\HelloWorld_1.war

下のような成功の確認メッセージが表示されるはずです。

```
D:\geronimo-tomcat6-jee5-2.0\bin>deploy --user system --password manager deploy
\HelloWorld_2.0\HelloWorld_1.war
Using GERONIMO_BASE:   D:\geronimo-tomcat6-jee5-2.0
Using GERONIMO_HOME:   D:\geronimo-tomcat6-jee5-2.0
Using GERONIMO_TMPDIR: var\temp
Using JRE_HOME:        C:\Java\jdk1.5.0_06\jre
Deployed sample.applications/HelloWorldApp_1/2.0/war @ /hello_1
```

2番目のアプリケーションも同様にデプロイします。

java -jar deployer.jar --user system --password manager deploy <app_home>\HelloWorld_2.war

下のような成功の確認メッセージが表示されるはずです。

```
D:\geronimo-tomcat6-jee5-2.0\bin>deploy --user system --password manager deploy
\HelloWorld_2.0\HelloWorld_2.war
Using GERONIMO_BASE:   D:\geronimo-tomcat6-jee5-2.0
Using GERONIMO_HOME:   D:\geronimo-tomcat6-jee5-2.0
Using GERONIMO_TMPDIR: var\temp
Using JRE_HOME:        C:\Java\jdk1.5.0_06\jre
Deployed sample.applications/HelloWorldApp_2/2.0/war @ /hello_2
```

あとはデプロイされたアプリケーションをテストするだけです、まず先に hello_1 に対してGeronimoサーバーのマシンに定義されているホスト名を使ってアクセスしてみます。

Host name / Virtual Host	Access
http://localhost:8080/hello_1	Fail
http://virtualhost1.com:8080/hello_1	SUCCESS!!!
http://virtualhost2.com:8080/hello_1	Fail
http://virtualhost3.com:8080/hello_1	Fail
http://virtualhost4.com:8080/hello_1	Fail

次は hello_2 で繰り返します。

Host name / Virtual Host	Access
http://localhost:8080/hello_2	Fail
http://virtualhost1.com:8080/hello_2	Fail
http://virtualhost2.com:8080/hello_2	SUCCESS!!!
http://virtualhost3.com:8080/hello_2	SUCCESS!!!
http://virtualhost4.com:8080/hello_2	SUCCESS!!!

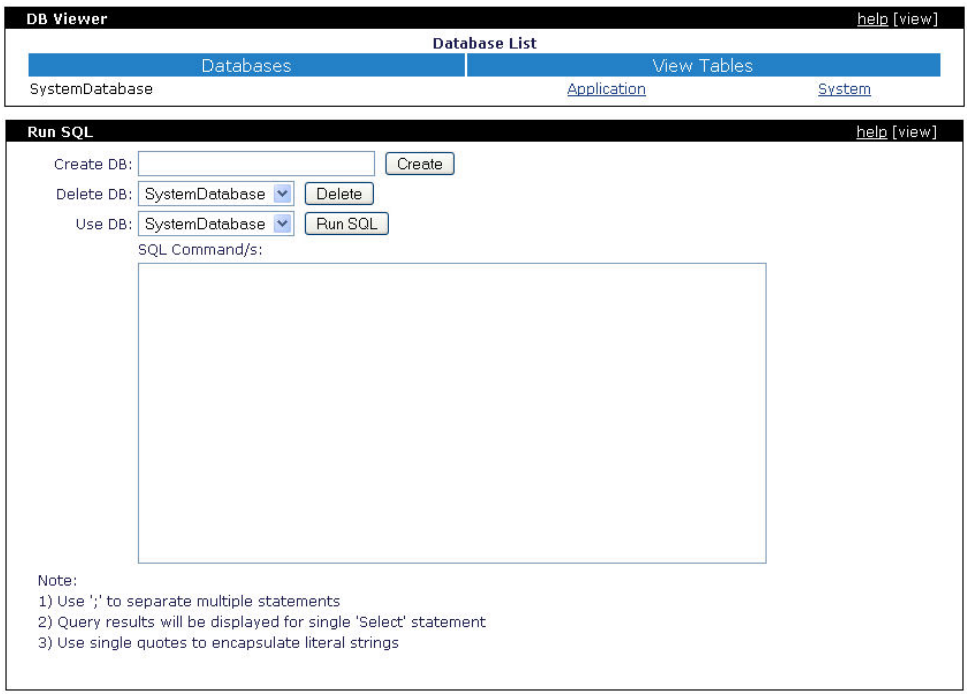
おめでとうございます!!! 2つのアプリケーションを2つの異なる仮想ホストと別名に割当てる構成とデプロイが成功しました。

Back to Top

1.5. データベースの作成

This page last changed on 4 21, 2008 by JAGUG.

Geronimo に新規で埋め込み型データベースを作成する場合には、DB Manager ポートレットを使用できます。Console Navigation メニューの一番下部に近い部分で、Embedded DB -> DB Manager を選択してください。このポートレットは下図に示す DB Viewer および Run SQL ポートレットを起動します。



DB Viewer ポートレットは、使用可能なすべてのデータベース、テーブル（アプリケーションおよびシステム）、およびそれらテーブルのコンテンツを表示します。

RunSQL ポートレットでは、SQL コマンドを実行し、データベースの作成／削除、テーブルのコンテンツの変更ができます。このポートレットでは、プルダウン・メニューにより、コマンドの実行対象としたいデータベースを選択できます。

テスト用にデータベースを作成してみます。Create DB: フィールドに test と指定し、Create　をクリックします。数秒後には確認用のメッセージが出力され、Run SQL ポートレットの底部近くに Database created: test と表示されます。 DB Viewer ポートレットに test データベースのエントリーが追加されます。

このデータベースに対して SQL コマンドを発行したい場合、Use DB: プルダウン・メニューより test を選択し、SQL コマンドを入力し、Run SQL ボタンを押下します。ポートレットの底部近くにコマンドの実行結果が表示されます。

1.6. Geronimo 管理コンソール

This page last changed on 4 24, 2008 by JAGUG.

Geronimo 管理コンソールは Web ベースのインターフェースで、Geronimo サーバーを多数の局面にわたって管理するための、利便性の高い、ユーザー・フレンドリーな手法を提供してくれます。Apache Geronimo サーバーが開始されたら (参照: 1.9. ツールとコマンド) 、以下の URL から管理コンソールに接続します。
http://localhost:8080/console

管理コンソールのログイン・ページは、下図で示されます。

ユーザー名 system、パスワード manager でログインすると、welcome ページにリダイレクトされます。コンソールは2つの主な領域で構成されます。左側のナビゲーション・パネルと右側のメイン・コンテンツ・エリアです。

左側のナビゲーション・パネルで、個々のタスクへ遷移ができます。常に表示されていてタスクからタスクへの遷移が容易に行えます。右側のメイン・コンテンツ・エリアでは実際の作業を行いますが、左側のナビゲーション・パネルで選択したタスクに連動します。下図の例を参照ください。ビュー内には1つ以上のポートレットが表示され、通常はヘッダーにヘルプへのリンクが掲載されます。

Back to Top

コンソール・ナビゲーション（左側のパネル）

前章で簡単に述べましたが、ナビゲーション・パネルによって Geronimo リソース管理のさまざまなタスクへ接続できます。このナビゲーション・パネルは下記のグループで構成されます。

- Welcome
- Server
- Services
- Applications
- Security
- Embedded DB
- Debug Views

Welcome

ログイン後の welcome ページです。Geronimo 管理コンソールの概要が記載され、さらに以下のよく使われるタスクへのショートカットが提供されます。

Common Console Actions

- Database Pools
- Security Realms
- JMS Resources
- Deploy Applications

- 
- 

Web Server Ports
Memory & Uptime

このページでは下記リンクも提供されます。

Geronimo Online

- The Geronimo Home Page
- Problem Tracking Database
- Users Mailing List
- Developers Mailing List
- Geronimo IRC chat

Geronimo Documentation

- FAQ
- Wiki
- Geronimo Documentation
- Additional Documentation

Back to Top

Server

Server グループ下には下記の通り、サーバーごとの構成・管理上の統計情報やリンクが提供されます。








Information
JVM
Server Logs
Shutdown
Web Server
JMS Server
Thread Pools
Apache HTTP

Back to Top

Information

このポートレットではサーバーの起動時間や JVM によって使用されるリソース状況が表示されます。

JVM

このポートレットではサーバー・プロセスが使用可能なシステム設定値が表示されます。

Server Logs

ここでは4つのポートレットが表示されます。Log Manager, Server Log Viewer, Derby Log Viewer そして Web Access Log Viewer　です。

- Log Manager
  このポートレットではログ構成ファイルの選択、ログ・レベルやリフレッシュ間隔の変更が可能です。デフォルトの "Config file" が標準の log4j ファイルで、Geronimo サーバーの出力結果をどこに保存するかを定義しています。構成ファイルはまた、ログ・レベル、最大ログ・ファイル・サイズ、そしてログに関するその他属性を定義します。

- Server Log Viewer

このポートレットでは Geronimo サーバー・ログを表示し、Geronimo コンソールやデプロイされたアプリケーションについての問題をデバッグする上での手助けとなります。

- Derby Log Viewer
  このポートレットでは Geronimo の内蔵データベースである、Derby のログ・ファイルを表示します。

- Web Access Log Viewer
  このポートレットは Web コンテナ・ログ・ファイルを表示・抽出します。HTTP 接続状況や JSP／サーブレット・アプリケーションからのメッセージを表示します。デフォルトでは、現在の日付のログ・ファイルのコンテンツを表示します。必要なデータのみに絞るためのフィルタリングの基準を適用することもできます。

Back to Top

Shutdown

このポートレットではサーバーのウォーム・リブートおよびシャットダウンを実行します。ウォーム・リブートではすべてのアプリケーションとサービスをシャットダウンし同一プロセス内で Geronimo カーネルを再起動します。管理コンソールは非接続状態になりますので、サーバーが再起動したら再接続してください。

shutdown ボタンを押下することで、サーバーをシャットダウンし、JVM を終了します。Geronimo コンソールを使用し続ける場合は、Geronimo を再起動します。

Back to Top

Web Server

ここでは2つのポートレットが提供されます。Web Server Manager と Network Listeners です。

- Web Server Manager
  このポートレットが enabled 状態である場合、サーバーのパフォーマンスに関する統計情報を収集します。測定の対象となる値は以下の通りです。
  - Requests
  - Connections
  - Errors
  - Active Requests
  - Request Duration
  - Connections Open
  - Connection Requests
  - Connection Duration

    最後の5つの値については、Count, Average および Maximum も同時に表示されます。

- Network Listeners
  このポートレットではすべてのコネクター・リスナ (HTTP, HTTPS, AJP) および Jetty / Tomcat 双方の Web コンテナのステータスが表示されます。このポートレットから新規のリスナの追加 や既存コネクターのステータス (stop, start, delete) の変更が可能になります。

Back to Top

JMS Server

ここでは2つのポートレットが提供されます。JMS Server Manager と JMS Network Listeners です。

- JMS Server Manager
  このポートレットでは使用可能な　JMS ブローカーおよびそのステータスを表示します。また、JMS ブローカーに対して start, stop, edit, add および delete といった操作ができます。

- JMS Network Listeners
  このポートレットでは現在使用可能な JMS ブローカーに対して構成されたすべてのネットワーク・コネクターを表示します。また、JMS ネットワーク・コネクターに対して start, stop, edit, add および delete といった操作ができます。

Back to Top

Thread Pools

このポートレットではサーバー上で定義されたすべてのスレッド・プールをリストします。プール・サイズを表示し、リストされたスレッド・プールを個々にモニターすることができます。これらスレッドのモニタリングによって、最大プールや記録された最低値・最高値、使用中のスレッドといった統計情報を収集できます。

Back to Top

Apache HTTP

このポートレットは、リモートの Apache 2 HTTP サーバーを使用するための Apache Geronimo の構成プロセスをウィザード形式でガイドしてくれます。リモート HTTP サーバーを構成するにはリモート・サーバーに mod_jk Apache モジュールを導入する必要があります。一連の質問に従っていけば、ウィザードが構成プロセスを案内します。

Back to Top

Services

このグループ分類では以下のサービス構成についてのリンクを紹介します。



Common Libs
Database Pools
JMS Resources

Common Libs

このポートレットではサーバーのリポジトリーに導入された artifact を表示します。リポジトリーのレイアウトは Apache Maven で使用されているものと同じで、ファイルのコピーが簡単にできるようになっています。このポートレットから新規の artifact も導入できます。

Back to Top

Database Pools

このポートレットでは使用可能なデータベース・プールとそのステータスをすべて表示します。このポートレットでは、サーバー・ワイドなデータベース・プールのみ編集可能で、シングル・アプリケーションの一部としてデプロイされたデータベース・プールについては編集できませんので、その場合は、アプリケーション内のデプロイメント・プランを更新する必要があります。

このポートレットからは使用可能な、システム・ワイドなデータベース・プールを編集可能です。ご自身のアプリケーションからプールをどう使用するかという例に関する usage リンクもあります。
このポートレットには新規プールのデータベース・プール作成ウィザードが含まれていて、また、JBoss 4 and WebLogic 8.1 からデータベース・プールのインポートができます。

Back to Top

JMS Resources

このポートレットでは Geronimo サーバーで使用可能な JMS リソース (queues および factories) をすべてリストします。このポートレットでは新規の JMS リソース・グループを作成でき、2つのウィザードにより、ActiveMQ および JMS provider がそれぞれ作成できます。後者については、固有のリソース・アダプター RAR が特定のプロバイダーに接続するために必要です。

Back to Top

<!-- CONTINUE HERE -->

Applications

このグループ分類では、アプリケーションの導入・管理についてのリンクを紹介します。

Deploy New

このポートレットでは Apache Geronimo サーバーへ新規のアプリケーションをデプロイすることができます。Web アプリケーションにはデプロイメント・プランが必要ですが、war ファイルとしてパッケージされているか、通常 geronimo-web.xml と呼ばれる別ファイルとして保存されています。

Back to Top

Application EARs

このポートレットでは導入済のアプリケーション EAR およびそのステータスを表示します。使用可能なアプリケーション EAR の停止、始動、アンインストールといったことが可能です。

Back to Top

Web App WARs

このポートレットでは導入済の Web アプリケーションおよびそのステータスを表示します。使用可能な Web アプリケーションの停止、始動、アンインストールといったことが可能です。

Back to Top

EJB JARs

このポートレットでは導入済の EJB JAR およびそのステータスを表示します。使用可能な EJB JAR の停止、始動、アンインストールといったことが可能です。

Back to Top

J2EE Connectors

このポートレットでは導入済の J2EE コネクタ およびそのステータスを表示します。使用可能な J2EE コネクタの停止、始動、アンインストールといったことが可能です。

Back to Top

App Clients

このポートレットでは導入済のアプリケーション・クライアントおよびそのステータスを表示します。使用可能なアプリケーション・クライアントの停止、始動、アンインストールといったことが可能です。

Back to Top

System Modules

このポートレットでは導入済のシステム・モジュールおよびそのステータスを表示します。使用可能なシステム・モジュールの停止、始動、アンインストールといったことが可能です。

Back to Top

Plugins

このポートレットでは Geronimo プラグインの導入および作成が可能です。リモート・プラグイン・リポジトリーの選択や使用可能なプラグインの検索、導入、もしくは Geronimo にすでに導入しているモジュールに関するプラグインのエクスポートといったことが可能です。

Back to Top

Security

Console Realm
Security Realm
Keystores
Certificate Authority

Console Realm

このオプションでは2つのポートレットが提供されます。Console Realm Users および Console Realm Groups です。

- Console Realm Users
  このポートレットでは Console Realm Users 情報を表示し、ユーザー情報の追加・削除、およびパスワード変更が可能です。
- Console Realm Groups
  このポートレットでは Console Realm Groups 情報を表示し、グループ情報の追加・変更・削除が可能です。

Back to Top

Security Realm

このポートレットでは使用可能なすべてのセキュリティ・レルムおよびそのステータスを表示します。サーバー・ワイドなセキュリティ・レルムだけがこのポートレットで表示可能で、シングル・アプリケーションの一部としてデプロイされたセキュリティ・レルムについては（そのままでは）表示できませんので、その場合はアプリケーションのデプロイメント・プランを変更してください。

このポートレットでは新規レルム作成用のセキュリティ・レルム作成ウィザードも含まれています。

Back to Top

Keystores

このポートレットでは SSL コネクターとあわせて使用するための keystores 構成プロセスをガイドします。keystores の追加・編集、および Trust Certificates の追加、プライベート・キーの作成が可能です。

Back to Top

Certificate Authority

このポートレットでは Geronimo の Certification Authority (CA) を作成し、Certificate Signing Requests (CSRs) への応答として証明書を発行します。

Back to Top

Embedded DB

このグループには以下のリンクがあります。

- 
- 

DB Info
DB Manager

DB Info

このポートレットでは下記のような内蔵データベースに関する情報が表示されます。

- データベース製品名およびバージョン
- ドライバーおよびバージョン
- コネクション URL
- サポート機能およびサポート SQL コマンド

Back to Top

DB Manager

このポートレットでは2つのポートレットが提供されます。 DB VIewer および Run SQL です。

- DB Viewer
  このポートレットでは使用可能なデータベースおよびテーブル、およびテーブルのコンテンツが表示されます。特定のデータベーステーブルを参照する場合は、データベース・リンクをクリックします。そうするとデータベース内の全テーブル情報が参照できます。テーブルのコンテンツを参照するには、View Contents リンクをクリックします。DB Viewer ポートレットの底部に View Databases もしくは View Tables リンクがあり直前の画面に戻れます。

- Run SQL
  このポートレットでは SQL コマンドを実行しテーブル内のデータを収集し、また新規データベースを作成・削除することができます。SQL Command(s) テキスト・ボックスに SQL コマンドを入力し、対象となるデータベースを選択し、Run SQL をクリックしコマンドを実行します。

Back to Top

Debug Views

このグループには以下のリンクがあります。





JMX Viewer
LDAP Viewer
ClassLoader Viewer
JNDI Viewer
Dependency Viewer

JMX Viewer

このポートレットでは tree 構造に似た形式でいろいろな種類の MBeans をブラウズすることができます。各々の MBeans はツリー・ノードとして、オブジェクト名とあわせて表示されます。すべての MBeans はドメインによってグループ化された MBeans のリストとして表示されます。

Back to Top

LDAP Viewer

このポートレットでは LDAP サーバーへの接続を可能にし、そのコンテンツをブラウズできますが、編集まではできません。ただし、Geronimo に埋め込まれた LDAP へ接続するには事前定義が必要で、サーバーへ接続する前にサービスが実行されていることを確認してください。

Back to Top

ClassLoader Viewer

このビューはサーバーに関連するクラスローダーおよびロードされるクラスのために使用されます。

Back to Top

JNDI Viewer

このビューはさまざまなモジュールの JNDI コンテキストを参照するために使用されます。

Back to Top

Dependency Viewer

このビューは全モジュールおよびその依存関係のために使用されます。

Back to Top

1.7. Geronimo の実行

This page last changed on 4 21, 2008 by JAGUG.

Geronimo の実行方法は何種類かあります。この章では各々のケースでの機能性やケイパビリティーについて記載します。利用可能なオプションは以下の通りです。

1.7.1. 複数リポジトリー

本記事は、Geronimo 2.0 以降を対象としています。

この記事では Geronimo の一つの導入インスタンスから、複数サーバーのインスタンスを実行する方法を述べます。サーバーのインスタンスができるだけ、ベースとなる Geronimo アーティファクトおよびファイル・システムのスペースを共有します。下記はデフォルトでの Geronimo ディレクトリーおよび <geronimo_home> 下のサブ・ディレクトリーです。

- bin
- lib
- schema
- var
- repository
- deploy (hot deployment)

<geronimo_home> は Geronimo を導入したディレクトリーです。bin, lib そして schema ディレクトリーは読み取り専用ですので、インスタンス間で共有できます。導入したばかりの Geronimo インスタンスが通常占有するディスク・スペースは var と repository で均等にそれぞれ約 50 MB に分割されています。その他は無視できるくらいの容量です。var のほとんどを占める 40 MB は ActiveMQ ジャーナル・ファイルです。これによって特に、各サーバー・インスタンスに対して一つずつ、第二リポジトリーとして使用できる、という考えができます。

単純な read/write パーティションのケイパビリティーに加えて、ディレクトリーへのアクセス許可というセキュリティの観点もあります。各サーバー・インスタンスは、自分自身の、たとえば var ディレクトリーだけに read/write 権限があります。

構成サブスティテューション・プロパティ

リポジトリーを共有しているかどうかにかかわらず、複数サーバーを実行させるためのフィーチャーは、構成サブスティテューション・プロパティ・ファシリティである config.xml を以下の形式で記述します。

```
${prop1 + prop2 + prop3}
```

その他、この記述方法で、下記により設定される値を使用し、jexl に関しての評価を行います。

- プロパティ・ファイル。デフォルトで var/config/config-substitutions.properties このファイル名はコマンドライン・プロパティによって変更できます。
  -Dorg.apache.geronimo.config.substitutions.file=var/config/some-other-file.properties
- システム環境プロパティ
- コマンドライン・プロパティ
  上記の順序で、お互いにオーバーライドします。コマンドライン・プロパティは他のすべてをオーバーライドします。システム環境プロパティおよびコマンドライン・プロパティは接頭辞がデフォルトで、"org.apache.geronimo.config.substitution." (末尾の "." にご注意ください。) とつきます。コマンドライン・プロパティで接頭辞を変更することもできます。
  -Dorg.apache.geronimo.config.substitution.prefix=myPrefix.

複数のサーバー・インスタンス

サーバー・インスタンスを Geronimo に作成するのは容易です。

1. `#########org.apache.geronimo.server.name` システム・プロパティをインスタンス名に変更します。
   - シンタックス `-Dorg.apache.geronimo.server.name=foo` を使用し、あなたのインスタンス foo を命名します。
   - 以下の2つの方法があります。
     a. `GERONIMO_OPTS` 環境変数に追加する、もしくは
     b. サーバーの java コマンドライン・インボケーションに渡す
   - このサーバーの var およびデプロイ・ディレクトリーは、それによって、<geronimo_home>/foo の下に位置します。
   - `org.apache.geronimo.server.name` 名は <geronimo_home> からの（降順で）相対するパス名になります。たとえば、`servers/bar ####### var ######### <geronimo_home>/servers/bar #########`
   - `org.apache.geronimo.server.dir ###################`
     `org.apache.geronimo.server.name` をオーバーライドします。
   - `org.apache.geronimo.server.dir` として絶対パスを指定してください。<geronimo_home> との相対的な関係は必要ありません。たとえば、`/ag20/servers/bar` サーバーの var ディレクトリーを `/ag20/servers/bar` 下に配置します。そのようにしない場合は、2つのシステム・プロパティは同一に振舞います。

2. **mkdir foo**
3. var/* を foo/var/ にコピーします。
4. foo/var/config/config-substitutions.properties を編集し、PortOffset のコメントを外し、1,2,10,11,12,20,21,22,...##(それに代わる方法として、コマンドラインからプロパティ -Dorg.apache.geronimo.config.substitution.PortOffset=3 を指定しサーバーを開始します。)
5. サーバーを始動します。

新規サーバー・インスタンスへアプリケーションをデプロイするには、NamingPort+PortOffset を指定し、PortOffset=1 のように使用します。

- deploy -port 1100 list-modules

複数のリポジトリー

まず、シングル・サーバー・インスタンスの場合で、セカンド・レジストリーの追加を考えてみましょう。Geronimo はリポジトリーに残したまま、アプリケーションのデプロイ用にセカンド・リポジトリーを追加します。セカンド・リポジトリーの追加はとても簡単です。

1. リポジトリー・モジュール用にプラン(repo2.xml)を作成します。
   repo2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="http://geronimo.apache.org/xml/ns/deployment-1.2">
<environment>
<moduleId>
<groupId>org.example.configs</groupId>
<artifactId>myrepo</artifactId>
<version>2.0.1</version>
<type>car</type>
</moduleId>
<dependencies>
<dependency>
<groupId>org.apache.geronimo.configs</groupId>
<artifactId>j2ee-system</artifactId>
<version>2.0.1</version>
<type>car</type>
</dependency>
</dependencies>
<hidden-classes/>
<non-overridable-classes/>
</environment>
<!--Repository-->
<gbean name="Repo2" class="org.apache.geronimo.system.repository.Maven2Repository">
<attribute name="root">repo2s/</attribute>
<attribute name="resolveToServer">true</attribute>
<reference name="ServerInfo">
<name>ServerInfo</name>
</reference>
</gbean>
<!--Configuration Store service-->
<gbean name="Local2"
class="org.apache.geronimo.system.configuration.RepositoryConfigurationStore">
<reference name="Repository">
<name>Repo2</name>
</reference>
</gbean>
</module>
```

2. リポジトリーのルート・ディレクトリーを **mkdir <geronimo_home>/repo2s**#* コマンドにより作成します。上記の例では、ディレクトリーは、Maven2Repository GBean, repo2s/ の root 属性で指定されます。ベース・ディレクトリー <geronimo_home> と相対関係にあります。
   - resolveToServer 属性はリポジトリーの場所を特定します。
     - true の場合はこのパスが baseServer と相対関係にあり、複数サーバー・インスタンスを扱うには便利です。
     - false の場合はこのパスがベース・ディレクトリー <geronimo_home> と相対関係にあります。
3. リポジトリー・モジュールをデプロイします。**deploy deploy repo2.xml** ####repo2.xml をデプロイすることにより実施します。

deploy コマンドはスクリプト <geronimo_home>/bin/deploy.{bat,sh}であり、オペレーティング・システムあわせてどちらか(bat,sh)を使用します。<geronimo_home>/bin ディレクトリーからか、もしくはパスにそのディレクトリーを通しておいて、deploy とタイプするだけです。

新規リポジトリーの使用

新規のリポジトリーを使用するのは少し癖があります。現在のところコマンドラインからのみのサポートとなります。重要な点は **deploy** コマンドの `--targets` オプションを使用することであり、新規リポジトリーにご自身のモジュールをデプロイするためにターゲットします。まず、デプロイヤー list-targets コマンドを使用し、リポジトリーを参照します。ターゲット名は長くて扱いにくくなっています。

```
deploy list-targets
```

利用可能なターゲット:
`org.example.configs/myrepo/2.0-SNAPSHOT/car?ServiceModule=org.example.configs/myrepo/2.0-SNAPSHOT/car,j2eeType=ConfigurationStore,name=Local2`
`org.apache.geronimo.configs/j2ee-system/2.0-SNAPSHOT/car?ServiceModule=org.apache.geronimo.configs/j2ee-system/2.0-SNAPSHOT/car,j2eeType=ConfigurationStore,name=Local`

環境変数の使用にはコマンドラインを推奨します。たとえば、set SEPO2=org.example.configs/myrepo/2.0-SNAPSHOT/car?ServiceModule=org.example.configs/myrepo/2.0-SNAPSHOT/car,j2eeType=ConfigurationStore,name=Local2.

新規のリポジトリーをデプロイするには **`deploy deploy --targets %REPO2% sample.war`** コマンドを使用します。

`deploy list-modules` はまた、各 repo に対して長いターゲット名を付与します。それに続くのが、repo にデプロイされた各モジュールの、通常の短い名前です。
`deploy list-modules %REPO2%` によってターゲットの repo に対してのいつもの短いアウトプットが与えられます。

repo からのアンデプロイのシンタックスは **`deploy undeploy "%REPO2%|geronimo/jsp-examples/1.1.1/war"`** となります。

文字 | がリポジトリー名とモジュール名を区分します。引用句 " はパラメーター全体にわたって使用され、コマンド・シェルが解釈できるように特別な文字を制御(エスケープ)します。

もし `--targets` がデプロイ・コマンドで指定されなかった場合、モジュールは全リポジトリーに対してデプロイされます。これは明らかに望ましくありません。モジュールを特定のあるリポジトリーに配置したいのであれば、必ず `--targets` オプションを使用してください!

それぞれが自身のリポジトリーを所有する複数サーバー・インスタンス

複数サーバー・インスタンスの場合には、各サーバーに対して、別のパスが、Maven2Repository root 属性として付与されます。`config.xml` とともに、各サーバー・インスタンスに対してオーバーライドされます。もっとよいやり方は `GBean ## resolveToServer` 属性の追加です。それによって、`root` 属性が `repository ##########################################` `<geronimo_home>/<instance_name>/repository` に配置されるリポジトリーを持ちます。

複数リポジトリーへのコンソール・サポート、およびホット・デプロイおよびプラグインのサポートはありません。

ここでちょっとご紹介しておきたいのは、どのリポジトリーに、新規のリポジトリーがデプロイされるか、ということです。明らかに、セカンド・リポジトリー(たとえば上記 `repo2.xml`) はファースト・リポジトリーにデプロイされます。このように、ファースト・リポジトリーを読み取り専用とすることは、動的に追加されるリポジトリーとしては動作しません。おそらく、セカンド・リポジトリーが追加され、これらサーバー・ユニークなリポジトリーを含むことになります。セカンド・リポジトリーは共有され読み取り専用でなくてはなりませんので、ユースケースにはぴったりとフィットしません。

1.7.2. 非 root ユーザーでの Geronimo 実行

This page last changed on 4 22, 2008 by JAGUG.

スーパー・ユーザー root のもとで Geronimo サーバー・プロセスを稼動させることには躊躇する場合が多いと思います。以下は簡単なステップによって、オーナーシップとパーミッションを必要なファイルに対して変更し、Geronimo を異なるユーザーで実行できるようにします。

以下のタスクを root もしくは同等の権限を所有するユーザーで実行してください。

1. 新規ユーザー（ここでは gmo）を作成し特定のユーザー・グループに追加します。例えば、あらかじめ定義された www ユーザー・グループを使用します。
2. 作成したユーザーの .bashrc を編集し、PATH および JAVA_HOME 変数を追加します。
   ```
   PATH=$PATH:<java_home>/bin
   export PATH
   JAVA_HOME=<java_home>
   export JAVA_HOME
   ```
3. <geronimo_home>下の全サブディレクトリーへのユーザー・グループ権限を変更します。
   ```
   chown -R root:www <geronimo_home>
   ```
4. <geronimo_home> ディレクトリー構造内に deploy ディレクトリーを作成します。
5. 以下のディレクトリーへのアクセス・パーミッションを変更します。
   ```
   chmod -R 774 <geronimo_home>/bin
   chmod -R 774 <geronimo_home>/deploy
   chmod -R 774 <geronimo_home>/repository
   chmod -R 774 <geronimo_home>/var
   ```
6. 新規ユーザー（ここでは gmo）でログインします。
7. <geronimo_home>/bin にディレクトリーを変更します。
   ```
   cd <geronimo_home>/bin
   ```
8. ./geronimo.sh を実行します。
   ```
   ./geronimo.sh run
   ```

root 以外のユーザーで実行し、アプリケーションを deploy できるようになりました。

1.7.3. Geronimo の複数インスタンスを実行

This page last changed on 4 22, 2008 by JAGUG.

本稿は以下の構成となっています。

同一マシン上で Geronimo の複数インスタンスを実行させることができます。現仕様では、Geronimo の複数インスタンスは Geronimo 導入先 <geronimo_home> の以下のディレクトリーを共有します。

- bin
- lib
- schema
- repository

各インスタンスは `<geronimo_home>/<instance_name>` にある以下のコピーを持ちます。

- var
- deploy (hot deployment)

bin, lib そして schema は読み取り専用ですので、インスタンス間で共有可能です。リポジトリーは共有されます。つまり、アプリケーションがあるインスタンスにデプロイされるとデプロイ済のモジュール・リストに掲載されますが、その他のインスタンス上では稼動していません。

新規サーバー・インスタンスの作成

> **Be Careful**
>
> Geronimo のフレッシュ・イメージで開始します。デフォルト・インスタンスを実行させるために使用したイメージを使わないでください。

`foo` という名前のインスタンスを作成するには以下の通り実施します。全インスタンス・データを `<geronimo_home>/foo` ディレクトリーに配置します。下記に名前のあげられたディレクトリーはすべて `<geronimo_home>#########`

1. **`mkdir foo`**
2. var を foo にコピーします。
3. `foo/var/config/config.substitutions.properties` ファイルを編集し、portOffset の値を変更します。さまざまなインスタンスに対して、10, 20, 30.. と試してみてください。

例

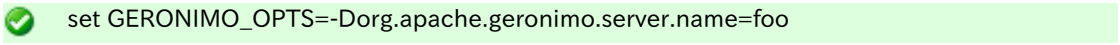
var をテンプレートとして使用します。

- ディレクトリー名 `servers` の下にあるインスタンスを作成し、`var` を `servers/geronimoi` にコピーし、各インスタンスの portOffset を変更します。
- ith インスタンスを実行するために、org.apache.geronimo.server.name property を servers/geronimoi に設定します。

インスタンスの実行

1. システム・プロパティ `org.apache.geronimo.server.name` をインスタンス名に設定してから、サーバーを始動します。`<geronimo_home>/foo` に位置する `foo` という名称のインスタンスとして、シンタックス `-Dorg.apache.geronimo.server.name=foo` を使用し、`GERONIMO_OPTS` 環境変数に追加します。

> Windows Hint



2. Start the server using 1.9.7. startup
   1.9.7. startup を参照し、サーバーを始動します。

   ```
   <geronimo_home>/bin/startup
   ```
3. このインスタンスを 1.9.6. shutdown するために RMI Naming port (デフォルト 1099) のポート番号を使用します。

   ```
   <geronimo_home>/bin/shutdown --port=<port_num>
   ```
4. このインスタンスにアプリケーションをデプロイするためにRMI Naming port のポート番号を使用します。

   ```
   <geronimo_home>/bin/deploy -port <port_num> deploy ......
   ```

1.6. Geronimo 管理コンソール も参照してこれら操作を実施ください。正しい HTTP ポート (デフォルト 8080) を使用し、インスタンスへ接続してください。

デフォルト・インスタンス

デフォルトのインスタンスには名前がありません。つまり、<geronimo_home>/varから実行されます。削除もできます!

Coming Soon!
'geronimo' デフォルト・インスタンスが作成され、以下により実行されます。インスタンス・データには <geronimo_home>/geronimo を使用します。

```
<geronimo_home>/bin/startup
```

その他拡張した点

複数のリポジトリーを使用するには:
1.7.1. 複数リポジトリー

1.8. システム・モジュール

This page last changed on 4 25, 2008 by JAGUG.

Apache Geronimo はすでに導入済のコンポーネントのシリーズによって構成されています。これらコンポーネントは 特に、関心のない機能や、導入したいと考えている追加フィーチャーがある場合、Geronimo 導入に際してカスタマイズできます。

deploy list-modules・コマンドを使用し、Geronimo システムにおけるモジュールのリストを入手します。出力フォーマットは一般的には org.apache.geronimo.configs/module-name/2.0.1/car となります。下記テーブルでは module-name のみ記載されます。

モジュールのディペンデンシーおよびヒエラルキーを参照するには、管理コンソールのデバッグ・ビューを参照します。特に Dependency Viewer および ClassLoader Viewer を使用します。後者のビューでは、ディペンデンシー・ツリーを反転できます。

オリジナル・マニュアル・テーブル

configs/*/pom.xml でのディスクリプション

テーブルのマージ

Geronimo のパッケージは以下のシステム・モジュールで構成されています。

テーブルのマージ

Module	pom Name	Started	Description	pom Description
activemq-broker	ActiveMQ Broker	+	ActiveMQ メッセージ・キューイング・サービス	Apache ActiveMQ の Geronimo への統合. 本モジュールは Geronimo に埋め込まれた activemq ブローカーを開始
activemq-ra	ActiveMQ Resource Adapter	+	ActiveMQ メッセージング・サービスへのリソース・アダプター	Geronimo ActiveMQ 統合: ActiveMQ リソース・アダプターによる (埋め込み) activemq ブローカーへの接続
axis	Axis		サーバー内での Axis web サービス提供者	Apache Axis 1 統合
axis2	Axis2		Axis2 web サービス・エンジン	Geronimo Web サービス Apache Axis2 統合
axis2-deployer	Axis2 Deployer	+	Axis2 Web サービス・デプロイヤー	Axis 2 への Geronimo web サービス・デプロイヤー
axis-deployer	Axis Deployer	+	Axis Web サービス・デプロイヤー	Geronimo Axis 1 統合への Web サービス・デプロイヤー
ca-helper-tomcat	CA Helper app - Tomcat			認証局(CA)用のヘルパー・アプリケーション。CSR(Certificate Signing Request:証明書署名要求)の受信およびブラウザーへの証明書のアップロードを可能にします

client	J2EE Client	Never	サーバーによって使用され application client をサポートします。このモジュールを開始しないでください!	
client-corba-yoko	Corba J2EE Client	Never	サーバーによって使用され CORBA 接続サポートを application client に提供します。このモジュールを開始しないでください!	
client-deployer	Application Client Deployments	+	アプリケーション・クライアント・デプロイヤー	
client-security	J2EE Client Security	Never	サーバーによって使用され application client へセキュリティ・サポートを提供します。このモジュールを開始しないでください!	
client-system	Client System	Never	サーバーによって使用され application client へ基本的なサポートを提供します。このモジュールを開始しないでください	
client-transaction	J2EE Client transaction	Never	サーバーによって使用され application client へトランザクション・サポートを提供します。このモジュールを開始しないでください	
connector-deployer	Connector Deployer	+	リソース・コネクション・デプロイヤー	J2CA コネクター用デプロイヤー
cxf	CXF		CXF web サービス・コンテナ。典型的には、Axis もしくは CXF がサーバーで使用され、web サービス・サポートを提供。よって、どちらか一方を起動します。	Geronimo Web サービス Apache CXF 統合
cxf-deployer	CXF Deployer		サーバーでの web サービス・アセットの CXF デプロイヤー	Apache CXF への Geronimo JAX-WS デプロイヤー
dojo-tomcat	Dojo app - Tomcat	+	Dojo の DHTML サポート	Geronimo 用 Dojo AJAX ライブラリー。/dojo にバインドし、dojo サポートが必要なアプリケーションが /dojo/dojo.js 参照をインクルードするだけで、Dojo サポートを可能にします

geronimo-gbean-deployer	GBean Deployer	+	GBean デプロイヤー	
hot-deployer	Hot Deployer	+	ホット・デプロイ	ホット・デプロイヤーは、Maven でのデプロイ・コマンド、コマンドライン、コンソールを使用するための代替としては良くありません
j2ee-corba-yoko	J2EE Corba Yoko ORB	+	サーバーへの CORBA 接続サポートを提供する Yoko ORB	Yoko Orb サーバーを、ネーミング・サービス、プロテクト/アンプロテクト orb とともにセットアップします
j2ee-deployer	J2EE Deployer	+	web アセットを除く J2EE アセットへのデプロイヤー	ear デプロイヤーを含む、基本的な JavaEE デプロイヤー機能
j2ee-security	J2EE Security	+	サーバー・セキュリティー・サポート。管理コンソールのセキュリティ・レルムを含みます	基本的な Geronimo サーバー・セキュリティ・インフラストラクチャー
j2ee-server	J2EE Server	+	基本的な J2EE サーバー・サポート。Web コンテナー、EJB コンテナー、J2EE コネクター・サービスを開始する GBeans を含みます	基本的な Geronimo JavaEE サポート・コンポーネント。トランザクション・マネージャーとコネクター・フレームワークを含みます
j2ee-system	J2EE System	+	ロギングや構成ストアのような基本的なサービスを含む。 サーバー始動時に開始される最初のモジュールで、他の全モジュールの親モジュール（直接もしくは非直接）としての役割をなします	ベースの Geronimo サーバーで、カーネルをセットアップします
jasper	Jasper	+	Jasper Report サービスを提供します	Geronimo Jasper jsp 統合で、注入サポートを含みます
jasper-deployer	Jasper Deployer	+	サーバーへの Jasper Report アセットをデプロイします	Jasper jsp のデプロイヤー。注入サポート・コンポーネントを導入します
javamail	JavaMail	+	SMTP を含むサーバーでの JavaMail サポートを提供します	Geronimo JavaMail サポート
jaxws-deployer	JAXWS Deployer	+	サーバー内の JAX-WS web サービス・アセットをデプロイします	JAX-WS Web サービスのベースの構成

jee-specs	JavaEE Specs	+		シングル・クラスローダーの JavaEE スペックの jar
jsr88-cli	JSR88 CLI		コマンド・ラインからの JSR-88 デプロイメント	
jsr88-deploymentfactory	JSR88 DeploymentFactory		JSR-88 デプロイメントを可能にします	
jsr88-ear-configurer	JSR88 EAR Configurer		JSR-88 経由での EAR デプロイメントを許可します	
jsr88-jar-configurer	JSR88 JAR Configurer		JSR-88 経由での JAR デプロイメントを許可します	
jsr88-rar-configurer	JSR88 RAR Configurer	+		
jsr88-war-configurer	JSR88 WAR Configurer		JSR-88 経由での WAR デプロイメントを許可します	
myfaces	MyFaces	+	サーバーへの Java Server Faces (JSF) サポートを提供します	Geronimo MyFaces jsf 統合
myfaces-deployer	MyFaces Deployer	+	サーバーへの Java Server Faces (JSF) アセットをデプロイします	
offline-deployer	Offline Deployer	Never	サーバーが実行中でないときサーバーへのアセットのデプロイに使用します	オフライン・デプロイヤー
online-deployer	Online Deployer			Geronimo オンライン・デプロイヤー
openejb	OpenEJB	+	サーバーへの EJB サポートを提供する OpenEJB Enterprise Java Bean (EJB) コンテナー	OpenEJB ejb コンテナーの Geronimo 統合
openejb-corba-deployer	Geronimo CORBA Deployer			openejb の corba セキュリティ構成用の Geronimo デプロイヤー
openejb-deployer	OpenEJB Deployer	+	サーバーへの EJB アセットのデプロイ	OpenEJB コンテナーへの Geronimo デプロイヤー
openjpa	OpenJPA with dependencies	+	OpenJPA は EJB 3.0 のパーシステンス部分を提供。Java Persistence API (JPA) としても既知	このモジュールは openejb とすべての依存関係とをあわせて提供

persistence-jpa10-deployer	Persistence Deployer	+	サーバー・アプリケーションのパーシステンス・アセットをデプロイ	Geronimo パーシステンス・ユニット・デプロイヤー
remote-deploy-tomcat	Remote Deploy Tomcat	+	リモート・ホストからの Tomcat web コンテナーに web アセットをデプロイ	Geronimo リモート・デプロイ・アップロード・サーブレット (Tomcat)
rmi-naming	RMI Naming	+	サーバーへの基本的な RMI およびネーミング・サービスを提供	プラグイン・インストーラーを含む Geronimo サービス基盤
server-security-config	Server SecurityConfiguration	+		デモ用 Geronimo サーバーのサンプル・セキュリティ構成。開発用途には不適
sharedlib	Shared Library	+	サーバーへの共有ライブラリー・サポートを提供	
shutdown	Shutdown	Never	シャットダウン・コマンドを実行しサーバーをシャットダウンする用途。このモジュールを開始しないでください!	
system-database	System Database	+	システムにより使用される Apache Derby データベース	
tomcat6	Tomcat	+	サーバーへの Tomcat web コンテナーを提供	Geronimo Tomcat web サーバー統合
tomcat6-deployer	Tomcat Deployer	+	Tomcat web コンテナーへの web アセットのデプロイ	Tomcat web コンテナーへの Geronimo デプロイヤー
transaction	Transaction Manager (JTA11)	+	サーバーへのトランザクション・サポートを提供	Geronimo トランザクション・マネージャー・モジュール
transformer-agent	Transformer Agent			Geronimo クラス・トランスフォーマーレジストリー・エージェント
uddi-tomcat	UDDI Tomcat			
webconsole-tomcat	WebConsole Tomcat	+	サーバー管理用 Geronimo 管理コンソール の提供	tomcat への Geronimo 管理コンソール
webservices-common	WebServices Common	+	サーバーへの Web サービスのベース・サポート	すべての web サービス・プロバイダー統合への共通クラス
welcome-tomcat	Welcome app Tomcat	+	Geronimo Welcome アプリケーションの提供。使用可能なデフォルト・アプリケー	

			ションは以下http://localhost:8080/.Geronimoの基本的な情報が記載。各種ドキュメントや管理コンソールの説明など。	
xmlbeans	XMLBeans	+	サーバーでのXMLBeansサービスを提供。他のモジュールにより使用され、XML を Java タイプにバインドすることでXML 接続を実施	シングル・クラスローダーでの xmlbeans フレームワークを提供する Geronimo モジュール

configs//pom.xml でのディスクリプション

これらのディスクリプションは、サーバー構成ディレクトリーでの pom.xml ディスクリプション要素です。

Module	Name	Description
activemq-broker	ActiveMQ Broker	Apache ActiveMQ の geronimo への統合。このモジュールは geronimo に埋め込まれた activemq ブローカーを開始
activemq-ra	ActiveMQ Resource Adapter	Geronimo ActiveMQ 統合: (埋め込み型) activemq ブローカーへ接続する ActiveMQ リソース・アダプター
axis-deployer	Axis Deployer	Geronimo Axis 1 統合用の Web サービス・デプロイヤー
axis	Axis	Apache Axis 1 統合
axis2-deployer	Axis2 Deployer	Axis 2 用 Geronimo web サービス・デプロイヤー
axis2-ejb-deployer	Axis2 EJB Deployer	Apache Axis2 用 Geronimo JAX-WS EJB デプロイヤー
axis2-ejb	Axis2 EJB	Geronimo Web サービス Apache Axis2 EJB 統合
axis2	Axis2	Geronimo Web サービス Apache Axis2 統合
ca-helper-jetty	CA Helper app - Jetty	認証局（CA）用のヘルパー・アプリケーション。CSR（Certificate Signing Request：証明書署名要求）の受信およびブラウザーへの証明書のアップロードを可能にする
ca-helper-tomcat	CA Helper app - Tomcat	認証局（CA）用のヘルパー・アプリケーション。CSR（Certificate Signing Request：証明書署名要求）の受信およびブラウザーへの証明書のアップロードを可能にする
client-corba-yoko	Corba J2EE Client	
client-deployer	Application Client Deployments	

client-security	J2EE Client Security	
client-system	Client System	
client-transaction	J2EE Client transaction	
client	J2EE Client	
clustering	Clustering	WADI で使用する基本的なクラスタリング・サポート
connector-deployer	Connector Deployer	J2CA コネクターのデプロイヤー
cxf-deployer	CXF Deployer	Apache CXF　用の Geronimo JAX-WS デプロイヤー
cxf-ejb-deployer	CXF EJB Deployer	Apache CXF 用の Geronimo JAX-WS EJB デプロイヤー
cxf-ejb	CXF EJB	Geronimo Web サービス Apache CXF EJB 統合
cxf	CXF	Geronimo Web サービス Apache CXF 統合
dojo-jetty6	Dojo app - Jetty6	Geronimo 用 Dojo AJAX ライブラリー。/dojo にバインドし dojo サポートを必要とするアプリケーションが、/dojo/dojo.js 参照するだけで　Dojo サポートを可能とする
dojo-tomcat	Dojo app - Tomcat	Geronimo 用 Dojo AJAX ライブラリー。/dojo にバインドし dojo サポートを必要とするアプリケーションが、/dojo/dojo.js 参照するだけで　Dojo サポートを可能とする
geronimo-gbean-deployer-bootstrap	GBean DeployerBoostrap version	
geronimo-gbean-deployer	GBean Deployer	
hot-deployer	Hot Deployer	ホット・デプロイヤーは、Maven でのデプロイ・コマンド、コマンドライン、コンソールを使用するための代替としては良くありません
j2ee-corba-yoko	J2EE Corba Yoko ORB	Yoko Orb サーバーを、ネーミング・サービス、プロテクト/アンプロテクト orb とともにセットアップ
j2ee-deployer	J2EE Deployer	ear デプロイヤーを含む、基本的な JavaEE デプロイヤー機能
j2ee-security	J2EE Security	基本的な Geronimo サーバー・セキュリティ・インフラストラクチャー
j2ee-server	J2EE Server	基本的な Geronimo JavaEE サポート・コンポーネント。トランザクション・マネージャーとコネクター・フレームワークを含みます

j2ee-system	J2EE System	ベースの Geronimo サーバーで、カーネルをセットアップ
jasper-deployer	Jasper Deployer	Jasper jsp のデプロイヤー。注入サポート・コンポーネントを導入
jasper	Jasper	Geronimo Jasper jsp 統合で、注入サポートを含みます
javamail	JavaMail	Geronimo JavaMail サポート
jaxws-deployer	JAXWS Deployer	JAX-WS Web サービスのベースの構成
jaxws-ejb-deployer	JAXWS EJB Deployer	JAX-WS EJB Web サービスのベース構成
jee-specs	JavaEE Specs	シングル・クラスローダーの JavaEE スペックの jar
jetty6-clustering-builder-wadi	Jetty 6 :: ClusteringBuilder for WADI	Jetty6 上での WADI クラスタリングのデプロイ
jetty6-clustering-wadi	Jetty 6 Clustering over WADI	Jetty 用 WADI クラスタリング
jetty6-deployer	Jetty Deployer	Jetty 6 Web サーバー統合用 Geronimo デプロイヤー
jetty6	Jetty 6	Geronimo Jetty Web サーバー統合
jsp-examples-jetty	JSP Examples Jetty	JSP サンプルで、元々は Tomcat 用に開発されたもの。導入後、HTTP 経由で /jsp-examples/ で参照可能。JSP 開発の初歩的な紹介をサンプル・コードとともに紹介
jsp-examples-tomcat	JSP Examples Tomcat	JSP サンプルで、元々は Tomcat 用に開発されたもの。導入後、HTTP 経由で /jsp-examples/ で参照可能。JSP 開発の初歩的な紹介をサンプル・コードとともに紹介
jsr88-cli	JSR88 CLI	
jsr88-deploymentfactory	JSR88 DeploymentFactory	
jsr88-ear-configurer	JSR88 EAR Configurer	
jsr88-jar-configurer	JSR88 JAR Configurer	
jsr88-rar-configurer	JSR88 RAR Configurer	
jsr88-war-configurer	JSR88 WAR Configurer	
ldap-demo-jetty	LDAP Demo for Jetty	
ldap-demo-tomcat	LDAP Demo for Tomcat	
ldap-realm	LDAP Security Realm	
mejb	Management EJB (MEJB)	Geronimo 用 Management EJB
myfaces-deployer	MyFaces Deployer	

myfaces	MyFaces	Geronimo MyFaces jsf 統合
offline-deployer	Offline Deployer	オフライン・デプロイヤー
online-deployer	Online Deployer	Geronimo オンライン・デプロイヤー
openejb-corba-deployer	Geronimo CORBA Deployer	openejb の Corba セキュリティ構成用 Geronimo デプロイヤー
openejb-deployer	OpenEJB Deployer	OpenEJB コンテナー用 Geronimo デプロイヤー
openejb	OpenEJB	OpenEJB ejb コンテナー用 Geronimo 統合
openjpa	OpenJPA with dependencies	このモジュールは openejb とすべての依存関係とをあわせて提供
persistence-jpa10-deployer	Persistence Deployer	Geronimo パーシステンス・ユニット・デプロイヤー
remote-deploy-jetty	Remote Deploy Jetty	Geronimo リモート・デプロイ・アップロード・サーブレット (Jetty)
remote-deploy-tomcat	Remote Deploy Tomcat	Geronimo リモート・デプロイ・アップロード・サーブレット (Tomcat)
rmi-naming	RMI Naming	プラグイン・インストーラーを含む Geronimo サービス基盤
server-security-config	Server SecurityConfiguration	デモ用 Geronimo サーバーのサンプル・セキュリティ構成。開発用途には不適
servlet-examples-jetty	Servlet Examples for Jetty	
servlet-examples-tomcat	Servlet Examples for Tomcat	
sharedlib	Shared Library	
shutdown	Shutdown	
system-database	System Database	
tomcat6-deployer	Tomcat Deployer	Tomcat web コンテナーへの Geronimo デプロイヤー
tomcat6	Tomcat	Geronimo Tomcat web サーバー統合
transaction	Transaction Manager (JTA11)	Geronimo トランザクション・マネージャー・モジュール
transformer-agent	Transformer Agent	Geronimo クラス・トランスフォーマーレジストリー・エージェント
uddi-jetty6	UDDI Jetty6	
uddi-tomcat	UDDI Tomcat	
unavailable-client-deployer	Unavailable ClientDeployer	
unavailable-ejb-deployer	Unavailable EJB Deployer	

unavailable-webservices-deployer	Unavailable WebServices Deployer	
upgrade-cli	CLI Upgrade	
upgrade	Plan Upgrade	
wadi-clustering	Clustering over WADI	Geronimo WADI 統合
webconsole-jetty6	Geronimo Admin Console (Jetty)	web　ベースの Geronimo 管理コンソールで、HTTP 経由で /console で参照可能。Geronimo 導入や JDBC コネクション・プールや JMS リソース、セキュリティ・レルム、keystores などを構成するのに使用。実行されている Geronimo サーバーの単純な統計を提供し、同時にアプリケーションおよびプラグインをレビューしデプロイする
webconsole-tomcat	WebConsole Tomcat	tomcat 用 Geronimo 管理コンソール
webservices-common	WebServices Common	全 Web サービス・プロバイダー統合用共通クラス
welcome-jetty	Welcome app Jetty	
welcome-tomcat	Welcome app Tomcat	
xmlbeans	XMLBeans	シングル・クラスローダーでの xmlbeans フレームワークの提供する Geronimo モジュール

オリジナル・マニュアル・テーブル

Module	Started	Description
activemq-broker	+	ActiveMQ メッセージ・キューイング・サービス
activemq-ra	+	ActiveMQ メッセージング・サービス用リソース・アダプター
axis-deployer	+	Axis Web サービス・デプロイヤー
axis2-deployer	+	Axis2 Web サービス・デプロイヤー
client-deployer	+	アプリケーション・クライアント・デプロイヤー
connector-deployer	+	リソース・コネクション・デプロイヤー
dojo-tomcat	+	Dojo DHTML サポート
geronimo-gbean-deployer	+	Gbean デプロイヤー
hot-deployer	+	ホット・デプロイ
j2ee-corba-yoko	+	Yoko ORB でサーバーへの CORBA 接続サポートを提供
j2ee-deployer	+	web アセットを除く J2EE アセットへのデプロイヤー

j2ee-security	+	サーバー・セキュリティー・サポート。管理コンソールのセキュリティ・レルムを含みます
j2ee-server	+	基本的な J2EE サーバー・サポート。Web コンテナー、EJB コンテナー、J2EE コネクター・サービスを開始する GBeans を含みます
j2ee-system	+	ロギングや構成ストアのような基本的なサービスを含みます。サーバー始動時に開始される最初のモジュールで、他の全モジュールの親モジュール（直接もしくは非直接）としての役割をなします
jasper	+	Jasper Report サービスを提供
jasper-deployer	+	サーバーへ Jasper Report アセットをデプロイ
javamail	+	SMTP を含むサーバーでの JavaMail サポートを提供します
jaxws-deployer	+	サーバー内の JAX-WS web サービス・アセットをデプロイします
jee-specs	+	
jsr88-rar-configurer	+	
myfaces	+	サーバーへの Java Server Faces (JSF) サポートを提供します
myfaces-deployer	+	サーバーへの Java Server Faces (JSF) アセットをデプロイします
openejb	+	サーバーへの EJB サポートを提供する OpenEJB Enterprise Java Bean (EJB) コンテナー
openejb-deployer	+	サーバーへの EJB アセットのデプロイ
openjpa	+	OpenJPA は EJB 3.0 のパーシステンス部分を提供。Java Persistence API (JPA) としても既知
persistence-jpa10-deployer	+	サーバー・アプリケーションのパーシステンス・アセットをデプロイ
remote-deploy-tomcat	+	リモート・ホストからの Tomcat web コンテナーに web アセットをデプロイ
rmi-naming	+	サーバーへの基本的な RMI およびネーミング・サービスを提供
server-security-config	+	
sharedlib	+	サーバーへの共有ライブラリー・サポートを提供
system-database	+	システムにより使用される Apache Derby データベース

tomcat6	+	サーバーへの Tomcat web コンテナーを提供
tomcat6-deployer	+	Tomcat web コンテナーへの web アセットのデプロイ
transaction	+	サーバーへのトランザクション・サポートを提供
webconsole-tomcat	+	サーバー管理用 Geronimo Administration Console の提供
webservices-common	+	サーバーへの Web サービスのベース・サポート
welcome-tomcat	+	Geronimo Welcome アプリケーションの提供。使用可能なデフォルト・アプリケーションは以下http://localhost:8080/. Geronimo の基本的な情報が記載。各種ドキュメントや管理コンソールの説明など。
xmlbeans	+	サーバーでの XMLBeans サービスを提供。他のモジュールにより使用され、XML を Java タイプにバインドすることで XML 接続を実施
axis		サーバーでの Axis web サービス・プロバイダー
axis2		Axis2 web　サービス・エンジン
ca-helper-tomcat		
client	Never	サーバーによって使用され application client をサポートします。このモジュールを開始しないでください！
client-corba-yoko	Never	サーバーによって使用され CORBA 接続サポートを application client に提供します。このモジュールを開始しないでください！
client-security	Never	サーバーによって使用され application client へセキュリティ・サポートを提供します。このモジュールを開始しないでください！
client-system	Never	サーバーによって使用され application client へ基本的なサポートを提供します。このモジュールを開始しないでください
client-transaction	Never	サーバーによって使用され application client へトランザクション・サポートを提供します。このモジュールを開始しないでください
cxf		CXF web サービス・コンテナ。典型的には、Axis もしくは CXF がサーバーで使用され、web サービス・サポートを提供。よって、どちらか一方を起動します。

cxf-deployer		サーバーでの web サービス・アセットの CXF デプロイヤー
jsr88-cli		コマンド・ラインからの JSR-88 デプロイメント
jsr88-deploymentfactory		JSR-88 デプロイメントを可能にします
jsr88-ear-configurer		JSR-88 経由での EAR デプロイメントを許可します
jsr88-jar-configurer		JSR-88 経由での JAR デプロイメントを許可します
jsr88-war-configurer		JSR-88 経由での WAR デプロイメントを許可します
offline-deployer	Never	サーバーが実行中でないときサーバーへのアセットのデプロイに使用します
online-deployer		
openejb-corba-deployer		
shutdown	Never	シャットダウン・コマンドを実行しサーバーをシャットダウンする用途。このモジュールを開始しないでください！
transformer-agent		
uddi-tomcat		

1.9. ツールとコマンド

This page last changed on 4 21, 2008 by JAGUG.

Apache Geronimo はサーバー管理用にいくつかのツールを提供しています。これらのツールはコマンド・ラインから使用可能で、Web ベースのコンソール経由のものもあります。Web ベースの Geronimo コンソールは次セクション 1.6. Geronimo 管理コンソール で詳述します。現在利用可能なコマンドライン・ツールは <geronimo_home>/bin ディレクトリーに配置され以下のリストに挙げられたものになります。

ツール名はそれ自身の動作を示していますが、そのパラメーターについては必ずしもそうではありません。次セクションでは本ツールとコマンドの使用につき詳細に説明します。

1.9.1. client.jar

This page last changed on 4 21, 2008 by JAGUG.

client.jar はクライアント・アプリケーション・コンテナーを起動します。以下のシンタックスです。

```
java -jar client.jar config-name [app arg] [app arg] ...
```

最初の引数は Geronimo 構成を明示し、実行させたいアプリケーション・クライアントを意味します。残りの引数は、クライアント・アプリケーションが開始されたときに渡されます。

1.9.2. deploy - デプロイ

This page last changed on 4 21, 2008 by JAGUG.

デプロイヤー・ツールは導入、アンインストール、再インストール、アプリケーションやモジュールの開始・停止、構成（たとえばある構成特有のデプロイメント・プランやセキュリティ・レルム、データベース・コネクション・プールなど）の導入やアンインストールに使用されます。ここでは多数のオプションが存在するため、このツールについては次セクション 1.9.3. Deployer tool - デプロイヤー・ツール で詳細にカバーします。

多数のパラメーターとオプションをもつ強力なツールですが、使用するのはまったく難しくありません。

startup コマンドに関しては、デプロイメント・ツールが2種類の方法で起動できます。

```
java -jar deployer.jar [general_options] <command> [command_options]
```

もしくは簡単に

```
deploy [general_options] <command> [command_options]
```

このツールのオプションや使用方法など詳細については 1.9.3. Deployer tool - デプロイヤー・ツール を参照してください。

1.9.3. Deployer tool - デプロイヤー・ツール

This page last changed on 4 21, 2008 by JAGUG.

デプロイヤー・アプリケーションは Java アプリケーションであり、Geronimo サーバーにおいて J2EE アーティファクトと GBean コンポーネントを管理します。Geronimo の実行中はサーバーに接続しサーバーのデプロイメント・サービスを通してアクションを実行します。実行中のサーバーが見つからない場合は、エラーを throw し、「サーバーに接続できなかった」もしくは「サーバーがダウンしている」と出力します。

デプロイメント・ツールは java -jar を使用することで始動し、<geronimo_home>/bin/deployer.jar におけるメイン・クラスを起動します。

典型的には、デプロイメント・ツールは deploy スクリプトを使用することによって始動しますが、アプリケーションの実行には Java 仮想マシンを以下のシンタックスによって開始します。

```
java -jar deployer.jar <general_options> <command> <command_options>
```

<general_options> は全コマンドに適用できる共通オプションを指定し、アプリケーションがどのように振舞うかをコントロールします。<command> はコマンド名で実行するアクションを指定し、<command_options> は指定されたコマンドで決まるオプションです。

一般的なオプション

このセクションでは Geronimo デプロイヤー・ツールの使用可能な一般的な全オプションをリストします。

- --uri <identifier>
  <identifier> は Universal Resource Identifier (URI) を指し、デプロイヤーがサーバーとどのようにコンタクトするかを指定します。このフラグが指定されない場合、デプロイヤーは localhost の標準ポートを使用してサーバーへのコンタクトをしようとします。 identifier は以下の形式で記述されます。
  deployer:geronimo:jmx:rmi:///jndi/rmi:[//host[:port]]/JMXConnector
  <host> はホスト名もしくはサーバーの起動しているシステムの TCP/IP アドレスで置き換えられ、<port> はサーバーが listen しているポート番号です。指定しない場合は localhost とデフォルト・ポートが使用されます。

- --host <host>
  <host> はアプリケーションやリソースをデプロイしようとしているサーバーのホスト名です。このオプションによりリモート・サーバーへのリソースとアプリケーションのデプロイが可能です。このパラメーターはオプションで、localhost がデフォルトです。

- --port <port>
  <port> はアプリケーションやリソースをデプロイしようとしているリモート・サーバーのポートです。このパラメーターはオプションで、ポート 1099 がデフォルトです。.
- --driver <driver_path>
  <driver_path> はドライバー JAR へのパスで、Geronimo 以外のサーバーとあわせて使用します。現在のところは、JAR の中の manifest Class-Path エントリーは無視されます。

- --user <username>
  <username> はサーバー管理者権限を付与されたユーザー名です。コマンドが認証を必要とする場合、このオプションを使用します。

- --password <password>
  <password> はユーザー名を認証するのに必要なパスワードです。指定しない場合は、デプロイヤーはパスワードなしでコマンドを実行しようとしますが、失敗すると、パスワードを入力するようプロンプトします。

- --syserr <select>
  <select> は true もしくは false のどちらかです。指定しない場合は false が使用されます。 syserr デバイスにエラー情報をログしたい場合は true を指定してください。

- --verbose <select>
  <select> は true もしくは false のどちらかです。指定しない場合は false が使用されます。エラー原因を特定するためのメッセージを出力したい場合は true を指定してください。

Back to top

コマンド

Geronimo デプロイヤー・ツールとして使用可能なコマンドは以下の通りです。

- 共通コマンド
  - deploy
  - login
  - redeploy
  - start
  - stop
  - undeploy
- その他コマンド
  - distribute
  - list-modules
  - list-targets
- Geronimo プラグイン
  - install-plugin
  - search-plugins

加えて、あるコマンドの詳細をさらに知りたい場合は、help と入力します。シンタックスは以下の通りです。

```
java -jar deployer.jar help <commands>
```

Back to top

Deploy

deploy コマンドを使用し新規のモジュールを起動してみます。以下のシンタックスとなります。

```
java -jar deployer.jar <general_options> deploy <module> <deployment_plan>
```

<module> はアプリケーション・ファイル名とロケーションを指定します。<deployment_plan> は XML によるデプロイメント・プランのファイル名とロケーションを指定します。アプリケーション・モジュールがパッケージ内にデプロイメント・プランをすでに含んでいることがあり、また、アプリケーションが非常に単純でデプロイメント・プランをまったく必要としない場合があります。こういった場合には、このパラメーターは省略できます。

モジュール・ファイルは以下のどれかになります。

- J2EE Enterprise Application Archive (EAR) file
- J2EE Web Application Archive (WAR) file
- J2EE Enterprise JavaBean Archive (JAR) file
- J2EE Java Resource Archive (RAR) file

アプリケーションのデプロイ時にサーバーが実行されていなかった場合は、サーバーの次回起動時にモジュールが有効になります。

もっとも一般的な <general_options> は、--user および --password となります。--inPlace オプションでは、アプリケーションのパッケージングの必要さえなく、Geronimo の外にあるディレクトリーから直接アプリケーションを指定しデプロイできるようにしてくれます。言い換えると、Geronimo でアプリケーションを起動させていても、そのアプリケーションは任意のファイル・システム上でよい、ということを意味します。

このオプションは以下のように使用します。

```
java -jar deployer.jar <general_options> deploy --inPlace <app_home>
```

<app_home> はアプリケーションのホーム・ディレクトリーを指定します。

Geronimo が起動していない状態でアプリケーションをデプロイしたい場合は、--offline オプションを使用します。シンタックスは以下の通りです。

```
java -jar deployer.jar <general_options> --offline deploy <module>
```

もちろん、--offline および --inPlace を組み合わせることもできます。

```
java -jar deployer.jar <general_options> --offline deploy --inPlace <app_home>
```

Back to top

Login

login コマンドを使用し、カレント・ユーザーのホーム・ディレクトリーで **.geronimo-deployer** ファイルに、現在の接続情報としてユーザー名とパスワードを保存します。次から同じサーバーへの接続は、この保存された認証情報を使用し、可能な場合はプロンプトしません。

この情報はコネクション URL とは別に保存されます。よって、コマンドライン上で --url もしくは --host もしくは --port を指定し、異なるサーバーへのログインを保存できます。

login コマンドは以下のシンタックスとなります。

```
java -jar deployer.jar --user <user_name> --password <password> login
```

したがって、次回異なるコマンドを発行する際にも、本来であればユーザー名とパスワードを必要とするような場合でも、直接以下のようにコマンドを発行できます。

```
deploy list-modules
```

ログイン情報がクリア・テキストに保存されている訳ではないからといって、必ずしも安心とはいえません。もし安全に認証情報を保存しておきたい場合は、他者から読み書きされないように、自身のホーム・ディレクトリー内に **.geronimo-deployer** ファイルを変更します。

Back to top

Redeploy

redeploy コマンドを使用し、停止、置換、およびデプロイ済のモジュールの再起動を行います。リデプロイ・コマンドは以下のシンタックスを持ちます。

```
java -jar deployer.jar <general_options> redeploy <module> <deployment_plan>
```

deploy command と同様に、リデプロイ・コマンドも次のモジュールのファイル・タイプに対応します。

- J2EE Enterprise Application Archive (EAR) file
- J2EE Web Application Archive (WAR) file
- J2EE Enterprise JavaBean Archive (JAR) file
- J2EE Java Resource Archive (RAR) file

典型的には、モジュールとプランの両方がオプションとして指定されます。モジュールがプランを含んでいるか、デフォルト・プランが使用されれば、プラン用のオプションは省略できます。しかし、この場合にプランが指定されれば、他のプランをオーバーライドします。プランが、すでにサーバー環境にデプロイされているサーバー・コンポーネントを参照している場合、モジュールのオプションは省略できます。

Back to top

Start

start コマンドを使用し以前デプロイしたモジュールを開始します。以下のシンタックスとなります。

```
java -jar deployer.jar <general_options> start <moduleIDs>
```

<moduleIDs> はブランク・スペースによって区切られた1つ以上のモジュール (configID) のリストです。モジュールのID (つまりConfigID) はデプロイ時に個々のデプロイメント・プランにおいて先にデプロイされた各モジュールに対して定義されます。

Back to top

Stop

stop コマンドを使用し、起動中のモジュールを停止します。以下のシンタックスとなります。

```
java -jar deployer.jar <general_options> stop <moduleIDs>
```

<moduleIDs> はブランク・スペースによって区切られた1つ以上のモジュール (configID) のリストです。モジュールのID (つまりConfigID) はデプロイ時に個々のデプロイメント・プランにおいて先にデプロイされた各モジュールに対して定義されます。

Back to top

Undeploy

undeploy コマンドを使用し、停止および（起動中かどうかにかかわらず）モジュールおよびサーバーからのデプロイ情報の除去を行います。以下のシンタックスとなります。

```
java -jar deployer.jar <general_options> undeploy <moduleIDs>
```

<moduleIDs> はブランク・スペースによって区切られた1つ以上のモジュール (configID) のリストです。モジュールのID (つまりConfigID) はデプロイ時に個々のデプロイメント・プランにおいて先にデプロイされた各モジュールに対して定義されます。

このコマンドは deploy の場合と同様、サーバーが起動していない状態でアプリケーションをアンインストールします。以下のシンタックスとなります。

```
java -jar deployer.jar <general_options> --offline undeploy <moduleID>
```

Back to top

Distribute

distribute コマンドを使用し、サーバーへ新規モジュールの追加を行います。このコマンドではモジュールの開始を行ったり、開始の予定として登録したりしません。以下のシンタックスとなります。

```
java -jar deployer.jar <general_options> distribute <module> <deployment_plan>
```

デプロイ・コマンドと同様、<module> はアプリケーションのファイル名とロケーションを指定します。モジュールがプランを含んでいるか、デフォルト・プランが使用されれば、プラン用のオプションは省略できます。しかし、この場合にプランが指定されれば、他のプランをオーバーライドします。プランが、すでにサーバー環境にデプロイされているサーバー・コンポーネントを参照している場合、モジュールのオプションは省略できます。

モジュール・ファイルは以下のうちの一つ となります。

- J2EE Enterprise Application Archive (EAR) file
- J2EE Web Application Archive (WAR) file
- J2EE Enterprise JavaBean Archive (JAR) file
- J2EE Java Resource Archive (RAR) file

Back to top

List-modules

list-modules コマンドを使用し、サーバー上の使用可能なモジュールをすべてリストします。このコマンドを実行するにはサーバーを起動していなければならないことに注意してください。以下のシンタックスとなります。

```
java -jar deployer.jar <general_options> list-modules [-{}{}all|started|-stopped ]
```

- --all : デフォルトで他のオプションが指定されていない場合に使用されます。すべての使用可能なモジュールをリストします。
- --started : このオプションでは起動しているモジュールのみリストされます。
- --stopped : このオプションでは起動していないモジュールのみリストされます。

Back to top

List-targets

list-targets コマンドを使用し、接続しているサーバーへの既知のターゲットをリストします。以下のシンタックスとなります。

```
java -jar deployer.jar <general_options> list-targets
```

Geronimo の場合は、各構成ストアは別々のターゲットとなっています。ですので、Geronimo はターゲットとしてのクラスターをサポートしません。

Back to top

Install-plugin

install-plugin コマンドを使用し、Geronimo サーバーから先にエクスポートした、または、リポジトリーからダウンロードした Geronimo プラグインを導入します。Geronimo プラグインはアプリケーションであり、そういったデータソースやドライバーの構成であり、またはそれらの混合されたものとなります。以下のシンタックスとなります。

```
java -jar deployer.jar install-plugin <plugin_file>
```

Back to top

Search-plugins

search-plugins コマンドを使用し、Maven リポジトリーで使用可能な Geronimo プラグインを全てリストします。以下のシンタックスとなります。

```
java -jar deployer.jar search-plugins <maven_repository_URL>
```

Back to top

1.9.4. geronimo

This page last changed on 4 21, 2008 by JAGUG.

geronimo コマンドは先述した2つのアクションを実行します。指定されたパラメーターに基づき異なるモードで、サーバーの「開始」と「停止」を行います。以下のシンタックスとなります。

<geronimo_home>/bin/geronimo [*options*]

使用可能なオプションは以下の通りです。

debug
このオプションは JDB デバッガーとしてサーバーを開始します。

jpda run
このオプションは JPDA デバッガーのもとでサーバーをフォアグラウンドで開始します。

jpda start
このオプションは JPDA デバッガーのもとでサーバーをバックグラウンドで開始します。

run
このオプションはカレント・ウィンドウでサーバーを開始します。

start
このオプションは 1.9.7. startup コマンド同様、別のウィンドウでサーバーを開始します。

stop
このオプションはサーバーを停止します。

start および stop の本コマンドへの両オプションとも、1.9.7. startup および 1.9.6. shutdown コマンドと同じパラメーターを持ちます。

1.9.5. jpa.jar

This page last changed on 4 21, 2008 by JAGUG.

jpa.jar は OpenJPA パーシスタンス・サポートを提供します。

実行環境の拡張性を高めるためには、jpa エージェントとともにサーバーを開始する必要があります。コマンドは以下の通りです。

```
<geronimo_home>java -javaagent:bin/jpa.jar -jar bin/server.jar
```

1.9.6. shutdown

This page last changed on 4 21, 2008 by JAGUG.

`startup` コマンドが Apache Geronimo サーバーを開始する一方、**`shutdown`** は停止するコマンドです。以下のシンタックスとなります。

`<geronimo_home>/bin/shutdown [`*`options`*`]`

使用可能なオプションは以下の通りです。

--user [user_name]
ユーザー名を指定し、サーバーの停止をする権限を所有します。デフォルトでは通常 system をユーザー名として使用します。

--password [password]
ユーザー名へのパスワードを指定します。デフォルトでは通常 manager をパスワードとして使用します。

--port [port_number]
RMI ネーミング・ポートを指定し、サーバーへ接続します。(たとえば、JMX コネクション・ポート)　デフォルトでは通常ポート 1099 を使用します。

何もパラメーターを指定しない場合、このコマンドはユーザー名とパスワードをプロンプトし、ポートについては 1099 をデフォルトで使うものとし、ポートへのプロンプトはありません。

1.9.7. startup

このコマンドは Apache Geronimo サーバーを開始するのに使用されます。以下に示す2つの方法で Apache Geronimo サーバーを開始することができます。

<geronimo_home>java -Djava.endorsed.dirs=lib/endorsed -javaagent:bin/jpa.jar -jar bin/server.jar

またはシンプルに

<geronimo_home>/bin/startup

startup コマンドは server.jar を起動します。

両方の場合ともこのコマンドは同じパラメーターが使用できます。使用可能なパラメーターは以下の通りです。

--quiet
通常の起動時に表示される進捗状況を示すプログレス・バーを表示しないようにします。これを使用するのはコンソールへの出力をファイルにリダイレクトさせる場合、または、IDE や他のツールからサーバーを起動させる場合です。

--long
起動時の進捗状況をコンソールに表示します。コンソールへの出力をファイルにリダイレクトさせるのに、もしくは IDE や他のツールからサーバーを起動させるのに、適したフォーマットとなります。(--quietまたは --long オプションを指定しなければデフォルトで使用される進捗情報を更新するためのラインフィードは使用されません。)

-v --verbose
コンソール・ログ・レベルを INFO に設定します。通常表示されるよりも多くのコンソール出力がされます。

-vv --veryverbose
コンソール・ログ・レベルを DEBUG に設定します。さらに多くのコンソール出力がされます。

-override [configId]
コマンド・ラインにリストされた構成のみが開始されるように、<geronimo_home>/var/config.list における構成をオーバーライドします。多くの J2EE フィーチャーは始動済みの構成によって決定されます。よって、省略する場合は細心の注意を払ってください。-override の後の引数はすべて構成の名称として処理されます。

Microsoft Windows コマンド・ラインからこのコマンドを実行すると、そのウィンドウは起動時の結果画面によってロックされてしまいます。Linux ベースのオペレーティング・システムでは、このコマンドをバックグラウンドで実行し、このウィンドウへの制御を取り戻すことができます。

startup コマンドをまったくパラメーターなしで実行すると、以下と似た画面が表示されます。

```
Booting Geronimo Kernel (in Java 1.5.0_06)...
16:59:13,734 INFO  [root] -------------------------------------------
16:59:13,734 INFO  [root] Started Logging Service
16:59:13,734 INFO  [root] Runtime Information:
16:59:13,734 INFO  [root]   Install Directory = D:\geronimo-jetty6-jee5-2.0-M2
16:59:13,734 INFO  [root]   JVM in use = Sun Microsystems Inc. Java 1.5.0_06
16:59:13,734 INFO  [root] Java Information:
16:59:13,734 INFO  [root]   System property [java.runtime.name]  = Java(TM) 2 Runtime Environment,
Standard Edition
16:59:13,734 INFO  [root]   System property [java.runtime.version]  = 1.5.0_06-b05
16:59:13,734 INFO  [root]   System property [os.name]             = Windows XP
16:59:13,734 INFO  [root]   System property [os.version]          = 5.1
16:59:13,734 INFO  [root]   System property [sun.os.patch.level]  = Service Pack 2
16:59:13,750 INFO  [root]   System property [os.arch]             = x86
16:59:13,750 INFO  [roor]   System property [java.class.version]  = 49.0
16:59:13,750 INFO  [root]   System property [locale]              = en_US
16:59:13,750 INFO  [root]   System property [unicode.encoding]    = UnicodeLittle
16:59:13,750 INFO  [root]   System property [file.encoding]       = Cp1252
16:59:13,750 INFO  [root]   System prt]   System poperty [java.vm.name]        = Java HotSpot(TM)
Client VM
16:59:13,750 INFO  [root]   System property [java.vm.vendor]      = Sun Microsystems Inc.
16:59:13,750 INFO  [root]   System property [java.vm.version]     = 1.5.0_06-b05
16:59:13,750 INFO  [root]   System property [java.vm.info]        = mixed mode
16:59:13,750 INFO  [root]   System property [java.home]           = C:\Java\jdk1.5.0_06\jre
```

```
16:59:13,750 INFO  [root]   System property [java.classpath]      = null
16:59:13,750 INFO  [root]   System property [java.library.path]   = C:\Java\jdk1.5.0_06\jre\bin;.C:
\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\Program Files\Intel\DMIX;C:\Program ...
16:59:13,750 INFO  [root]   System property [java.endorsed.dirs]  = C:\Java\jdk1.5.0_06\\jre\lib
\endorsed;D:\geronimo-jetty6-jee5-2.0-M2\lib\endorsed
16:59:13,750 INFO  [root]   System property [java.ext.dirs]       = C:\Java\jdk1.5.0_06\\jre/lib/
ext;D:\geronimo-jetty6-jee5-2.0-M2/lib/ext
16:59:13,750 INFO  [root]   System property [sun.boot.class.path] = D:\geronimo-jetty6-jee5-2.0-
M2\lib\endorsed\xercesImpl-2.8.1.jar;D:\geronimo-jetty6-jee5-2.0-M2\lib\endorsed\yoko-rmi-spec-1...
16:59:13,750 INFO  [root] -------------------------------------------
Module  1/27 org.apache.geronimo.configs/rmi-naming/2.0-M2/car                started in   .344s
Module  2/27 org.apache.geronimo.configs/j2ee-server/2.0-M2/car               started in   .235s
Module  3/27 org.apache.geronimo.configs/transaction-jta11/2.0-M2/car         started in   .359s
Module  4/27 org.apache.geronimo.configs/j2ee-security/2.0-M2/car             started in   .485s
Module  5/27 org.apache.geronimo.configs/openejb/2.0-M2/car                  16:59:20,765 INFO
[OpenEJB] Using directory D:\geronimo-jetty6-jee5-2.0-M2\var\temp for stateful session passivation
started in  4.922s
Module  6/27 org.apache.geronimo.configs/system-database/2.0-M2/car           started in   .000s
Module  7/27 org.apache.geronimo.configs/activemq-broker/2.0-M2/car           started in  2.703s
Module  8/27 org.apache.geronimo.configs/activemq/2.0-M2/car                 started in   .281s
Module  9/27 org.apache.geronimo.configs/jetty6/2.0-M2/car                  2007-02-05
16:59:24.468::INFO:  Logging to STDERR via org.mortbay.log.StdErrLog
2007-02-05 16:59:24.468::INFO:  jetty-6.1.x
2007-02-05 16:59:24.953::INFO:  Started GeronimoSSLListener @ 0.0.0.0:8443
2007-02-05 16:59:24.968::INFO:  Started SocketConnector @ 0.0.0.0:8080
started in   .859s
Module 10/27 org.apache.geronimo.configs/geronimo-gbean-deployer/2.0-M2/car   started in   .422s
Module 11/27 org.apache.geronimo.configs/j2ee-deployer/2.0-M2/car             started in   .281s
Module 12/27 org.apache.geronimo.configs/connector-deployer/2.0-M2/car        started in   .188s
Module 13/27 org.apache.geronimo.configs/persistence-jpa10-deployer/2.0-M2/car started in   .109s
Module 14/27 org.apache.geronimo.configs/openejb-deployer/2.0-M2/car          started in   .172s
Module 15/27 org.apache.geronimo.configs/client-deployer/2.0-M2/car           started in   .109s
Module 16/27 org.apache.geronimo.configs/cxf-deployer/2.0-M2/car              started in   .469s
Module 17/27 org.apache.geronimo.configs/axis2-deployer/2.0-M2/car            started in   .140s
Module 18/27 org.apache.geronimo.configs/axis-deployer/2.0-M2/car             started in   .985s
Module 19/27 org.apache.geronimo.configs/javamail/2.0-M2/car                  started in   .062s
Module 20/27 org.apache.geronimo.configs/sharedlib/2.0-M2/car                 started in   .000s
Module 21/27 org.apache.geronimo.configs/jetty6-deployer/2.0-M2/car           started in   .219s
Module 22/27 org.apache.geronimo.configs/welcome-jetty/2.0-M2/car             started in   .578s
Module 23/27 org.apache.geronimo.configs/dojo-jetty6/2.0-M2/car               started in   .062s
Module 24/27 org.apache.geronimo.configs/webconsole-jetty6/2.0-M2/car         started in  3.375s
Module 25/27 org.apache.geronimo.configs/remote-deploy-jetty/2.0-M2/car       started in   .125s
Module 26/27 org.apache.geronimo.configs/hot-deployer/2.0-M2/car              started in   .437s
Module 27/27 console.dbpool/LocalDB/1.0/rar                                   started in   .844s
Startup completed in 21 seconds
Listening on Ports:
1099 0.0.0.0 RMI Naming
1527 0.0.0.0 Derby Connector
4201 0.0.0.0 org.apache.geronimo.openejb.EjbDaemonGBean
4242 0.0.0.0 Remote Login Listener
8080 0.0.0.0 Jetty Connector HTTP
8443 0.0.0.0 Jetty Connector HTTPS
9999 0.0.0.0 JMX Remoting Connector
61613 0.0.0.0 ActiveMQ Transport Connector
61616 0.0.0.0 ActiveMQ Transport Connector

Started Application Modules:
EAR: org.apache.geronimo.configs/webconsole-jetty6/2.0-M2/car
RAR: console.dbpool/LocalDB/1.0/rar
RAR: org.apache.geronimo.configs/activemq/2.0-M2/car
RAR: org.apache.geronimo.configs/system-database/2.0-M2/car
WAR: org.apache.geronimo.configs/dojo-jetty6/2.0-M2/car
WAR: org.apache.geronimo.configs/remote-deploy-jetty/2.0-M2/car
WAR: org.apache.geronimo.configs/welcome-jetty/2.0-M2/car

Web Applications:
```

```
http://hcunico:8080/
http://hcunico:8080/console
http://hcunico:8080/console-standard
http://hcunico:8080/dojo
http://hcunico:8080/remote-deploy

Geronimo Application Server started
```

2. デプロイメント・プラン

This page last changed on 4 27, 2008 by JAGUG.

3. 導入

This page last changed on 4 21, 2008 by JAGUG.

ここでは Apache Geronimo の導入に関して詳細を記述します。前提となるソフトウェアが何か、Geronimo をどこからダウンロードすればいいのか、デフォルトではないポートを使用したカスタマイズされた導入について、などを採り上げます。

リモートの Web サーバーによる2層トポロジーでの導入や構成に関しても、詳細を後述しています。

本章は下記セクションで構成されています。

前提となるソフトウェア

Apache Geronimo v2.0 のビルド環境として J2SE 1.5 上で Maven 2 を使用します。下記サイトより適切なバージョンの JVM を入手します。

- J2SE SDK (http://java.sun.com/)
- Apache Maven (http://maven.apache.org/)

Geronimo のダウンロード

Geronimo v2.0 のソース・コードおよびバイナリーは下記 URL より入手します。

http://geronimo.apache.org/downloads.html

ソースからのビルド

コマンド・ラインのコンソールから、ソース・コード (.zip または tar.gz) を解凍し、ディレクトリーを <geronimo_home> に変更します。
以下のコマンドをコマンド・ラインより入力し、Apache Geronimo をビルドします。

```
mvn install
```

ソースからの Geronimo のビルドに関しての詳細なステップと要件については、"Building Apache Geronimo with Maven 2" のセクションで解説しています。

バイナリーからの Geronimo 導入

Apache Geronimo を導入・実行したいプラットフォームに合わせて、適切な導入イメージをダウンロード してください。オペレーティング・システムに合った圧縮フォーマット(.zip, .tar.gz) を選択します。リンクをクリック、ダウンロードし、ハード・ディスクの新しいディレクトリーにバイナリーを展開します。

Apache Geronimo の導入は zip/tar ファイルを解凍するだけのシンプルさです！コマンド・ラインからディレクトリーを <geronimo_home>/bin に変更し、下記コマンドを使用しサーバーを始動してください。

```
geronimo run
```

Apache Geronimo はサーバー/アプリケーション管理用に便利な一連のスクリプトを提供しています。詳細は 1.9. ツールとコマンド セクションをご覧ください。

デフォルトのポート変更

HTTP ポートは Web コンテナが使用する標準ネットワークです。このポート番号は、Geronimo 上で稼動する Web アプリケーションを呼び出す、どの HTTP URL でも使用されます。Geronimo のサーバーおよび始動構成の各モジュールは **config.xml** によって制御され、デフォルトの HTTP リスナーはポート番号が 8080 で構成されます。

ネットワークのポートを変更するのにはいくつか理由があり、それによってGeronimo のインスタンスを複数実行させることができます。デフォルトのポートを構成するには　<geronimo_home>/var/config/config.xml ファイルを編集します。Apache Geronimo v2.0 (Tomcat ディストリビューション版) での config.xml を例として下記に掲載します。

config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ======================================================== -->
<!-- Warning - This XML file is re-generated by Geronimo when -->
<!-- changes are made to Geronimo's configuration, therefore  -->
<!-- any comments added to this file will be lost.            -->
<!-- Do not edit this file while Geronimo is running.         -->
<!-- ======================================================== -->
<attributes xmlns="http://geronimo.apache.org/xml/ns/attributes-1.1">
<module name="org.apache.geronimo.configs/rmi-naming/2.0-M2/car">
<gbean name="RMIRegistry">
<attribute name="port">1099</attribute>
</gbean>
<gbean name="NamingProperties">
<attribute name="namingProviderUrl">rmi://0.0.0.0:1099</attribute>
</gbean>
<gbean name="DownloadedPluginRepos">
<attribute name="repositoryList">http://geronimo.apache.org/plugins/plugin-repository-
list-2.0.txt</attribute>
<attribute name="userRepositories">[]</attribute>
</gbean>
</module>
<module name="org.apache.geronimo.configs/j2ee-server/2.0-M2/car"/>
<module name="org.apache.geronimo.configs/transaction-jta11/2.0-M2/car"/>
<module name="org.apache.geronimo.configs/j2ee-security/2.0-M2/car">
<gbean name="JaasLoginServiceRemotingServer">
<attribute name="host">0.0.0.0</attribute>
<attribute name="port">4242</attribute>
</gbean>
<gbean name="JMXService">
<attribute name="protocol">rmi</attribute>
<attribute name="host">0.0.0.0</attribute>
<attribute name="port">9999</attribute>
<attribute name="urlPath">/jndi/rmi://0.0.0.0:1099/JMXConnector</attribute>
</gbean>
</module>
<module name="org.apache.geronimo.configs/axis/2.0-M2/car"/>
<module load="false" name="org.apache.geronimo.configs/axis2/2.0-M2/car"/>
<module load="false" name="org.apache.geronimo.configs/cxf/2.0-M2/car"/>
<module name="org.apache.geronimo.configs/openejb/2.0-M2/car"/>
<module load="false" name="org.apache.geronimo.configs/j2ee-corba-yoko/2.0-M2/car">
<gbean name="NameServer">
<attribute name="port">1050</attribute>
<attribute name="host">localhost</attribute>
</gbean>
</module>
<module load="false" name="org.apache.geronimo.configs/j2ee-corba-sun/2.0-M2/car">
<gbean name="NameServer">
<attribute name="port">1050</attribute>
<attribute name="host">localhost</attribute>
</gbean>
```

```
</module>
<module name="org.apache.geronimo.configs/system-database/2.0-M2/car">
<gbean name="DerbyNetwork">
<attribute name="host">0.0.0.0</attribute>
<attribute name="port">1527</attribute>
</gbean>
</module>
<module name="org.apache.geronimo.configs/activemq-broker/2.0-M2/car">
<gbean name="ActiveMQ.tcp.default">
<attribute name="host">0.0.0.0</attribute>
<attribute name="port">61616</attribute>
</gbean>
</module>
<module name="org.apache.geronimo.configs/activemq/2.0-M2/car"/>
<module name="org.apache.geronimo.configs/tomcat6/2.0-M2/car">
<gbean name="TomcatResources"/>
<gbean name="TomcatWebConnector">
<attribute name="host">0.0.0.0</attribute>
<attribute name="port">8080</attribute>
<attribute name="redirectPort">8443</attribute>
</gbean>
<gbean name="TomcatAJPConnector">
<attribute name="host">0.0.0.0</attribute>
<attribute name="port">8009</attribute>
<attribute name="redirectPort">8443</attribute>
</gbean>
<gbean name="TomcatWebSSLConnector">
<attribute name="host">0.0.0.0</attribute>
<attribute name="port">8443</attribute>
</gbean>
<gbean name="org.apache.geronimo.configs/tomcat6/2.0-M2/car?
ServiceModule=org.apache.geronimo.configs/tomcat6/2.0-M2/
car,j2eeType=GBean,name=TomcatWebContainer">
<attribute name="catalinaHome">var/catalina</attribute>
</gbean>
</module>
<module name="org.apache.geronimo.configs/geronimo-gbean-deployer/2.0-M2/car"/>
<module name="org.apache.geronimo.configs/j2ee-deployer/2.0-M2/car">
<gbean name="WebBuilder">
<attribute name="defaultNamespace">http://geronimo.apache.org/xml/ns/j2ee/web/tomcat-1.2</
attribute>
</gbean>
<gbean name="EnvironmentEntryBuilder">
<attribute name="eeNamespaces">http://java.sun.com/xml/ns/j2ee,http://java.sun.com/xml/ns/javaee</
attribute>
</gbean>
</module>
<module name="org.apache.geronimo.configs/connector-deployer/2.0-M2/car">
<gbean name="ResourceRefBuilder">
<attribute name="eeNamespaces">http://java.sun.com/xml/ns/j2ee,http://java.sun.com/xml/ns/javaee</
attribute>
</gbean>
<gbean name="AdminObjectRefBuilder">
<attribute name="eeNamespaces">http://java.sun.com/xml/ns/j2ee,http://java.sun.com/xml/ns/javaee</
attribute>
</gbean>
</module>
<module name="org.apache.geronimo.configs/persistence-jpa10-deployer/2.0-M2/car"/>
<module name="org.apache.geronimo.configs/openejb-deployer/2.0-M2/car">
<gbean name="EJBBuilder">
<reference name="ServiceBuilders">
<pattern>
<name>GBeanBuilder</name>
</pattern>
<pattern>
<name>PersistenceUnitBuilder</name>
```

```
</pattern>
</reference>
<reference name="WebServiceBuilder">
<pattern>
<name>CXFBuilder</name>
</pattern>
<pattern>
<name>Axis2Builder</name>
</pattern>
<pattern>
<name>WebServiceBuilder</name>
</pattern>
</reference>
</gbean>
<gbean name="EjbRefBuilder">
<attribute name="eeNamespaces">http://java.sun.com/xml/ns/j2ee,http://java.sun.com/xml/ns/javaee</
attribute>
</gbean>
<gbean name="ClientEjbRefBuilder">
<attribute name="eeNamespaces">http://java.sun.com/xml/ns/j2ee,http://java.sun.com/xml/ns/javaee</
attribute>
</gbean>
</module>
<module name="org.apache.geronimo.configs/client-deployer/2.0-M2/car"/>
<module name="org.apache.geronimo.configs/cxf-deployer/2.0-M2/car"/>
<module name="org.apache.geronimo.configs/axis2-deployer/2.0-M2/car"/>
<module name="org.apache.geronimo.configs/axis-deployer/2.0-M2/car">
<gbean name="AxisServiceRefBuilder">
<attribute name="eeNamespaces">http://java.sun.com/xml/ns/j2ee,http://java.sun.com/xml/ns/javaee</
attribute>
</gbean>
</module>
<module name="org.apache.geronimo.configs/javamail/2.0-M2/car">
<gbean name="SMTPTransport">
<attribute name="host">localhost</attribute>
<attribute name="port">25</attribute>
</gbean>
</module>
<module name="org.apache.geronimo.configs/sharedlib/2.0-M2/car">
<gbean name="SharedLib">
<attribute name="classesDirs">var/shared/classes</attribute>
<attribute name="libDirs">var/shared/lib</attribute>
</gbean>
</module>
<module name="org.apache.geronimo.configs/tomcat6-deployer/2.0-M2/car"/>
<module name="org.apache.geronimo.configs/welcome-tomcat/2.0-M2/car"/>
<module name="org.apache.geronimo.configs/webconsole-tomcat/2.0-M2/car"/>
<module load="false" name="org.apache.geronimo.configs/uddi-tomcat/2.0-M2/car"/>
<module name="org.apache.geronimo.configs/remote-deploy-tomcat/2.0-M2/car"/>
<module name="org.apache.geronimo.configs/hot-deployer/2.0-M2/car"/>
<module load="false" name="org.apache.geronimo.configs/ca-helper-tomcat/2.0-M2/car"/>
<module name="console.dbpool/LocalDB/1.0/rar"/>
<module name="console.dbpool/jdbc%2FTradeDataSource/1.0/rar"/>
</attributes>
```

ここで、config.xml ファイルからの抜粋による例を2つご紹介します。一つは Tomcat 版で、もう一つは Jetty 版です。これら抜粋ではリスナーが定義されている全セクションを掲載しています。ここで、標準の 8080 ポートを、任意のポート(ここでは 9000) で置き換えてみましょう。

Excerpt from config.xml - Tomcat

```
<module name="org.apache.geronimo.configs/tomcat6/2.0-M2/car">
<gbean name="TomcatResources"/>
<gbean name="TomcatWebConnector">
<attribute name="host">0.0.0.0</attribute>
<attribute name="port">9000</attribute>
```

```
<attribute name="redirectPort">8443</attribute>
</gbean>
<gbean name="TomcatAJPConnector">
<attribute name="host">0.0.0.0</attribute>
<attribute name="port">8009</attribute>
<attribute name="redirectPort">8443</attribute>
</gbean>
<gbean name="TomcatWebSSLConnector">
<attribute name="host">0.0.0.0</attribute>
<attribute name="port">8443</attribute>
</gbean>
<gbean name="org.apache.geronimo.configs/tomcat6/2.0-M2/car?
ServiceModule=org.apache.geronimo.configs/tomcat6/2.0-M2/
car,j2eeType=GBean,name=TomcatWebContainer">
<attribute name="catalinaHome">var/catalina</attribute>
</gbean>
</module>
```

Excerpt from config.xml - Jetty

```
<module name="org.apache.geronimo.configs/jetty6/2.0-M2/car">
<gbean name="JettyWebConnector">
<attribute name="host">0.0.0.0</attribute>
<attribute name="port">9000</attribute>
<attribute name="redirectPort">8443</attribute>
</gbean>
<gbean name="JettySSLConnector">
<attribute name="host">0.0.0.0</attribute>
<attribute name="port">8443</attribute>
</gbean>
</module>
<module name="org.apache.geronimo.configs/geronimo-gbean-deployer/2.0-M2/car"/>
<module name="org.apache.geronimo.configs/j2ee-deployer/2.0-M2/car">
<gbean name="WebBuilder">
<attribute name="defaultNamespace">http://geronimo.apache.org/xml/ns/j2ee/web/jetty-1.2</attribute>
</gbean>
<gbean name="EnvironmentEntryBuilder">
<attribute name="eeNamespaces">http://java.sun.com/xml/ns/j2ee,http://java.sun.com/xml/ns/javaee</
attribute>
</gbean>
</module>
```

ファイルを保存し、サーバーを再起動します。新しい構成情報をロードし、新たに割り当てたポートをブラウザーに指定します。

http://localhost:9000/console

ユーザー名とパスワードの変更

Apache Geronimo 導入時はデフォルトのユーザー名 - system およびパスワード - manager が使用されています。これらの変更には3つのオプションがあります。

1. *.properties ファイルを手動で編集する
2. 管理コンソールからセキュリティ構成を更新する
3. 新規のセキュリティ・レルムを作成する

この最後のオプションについては、1.1.3. セキュリティの構成 のセクションで詳細に述べられています。

*.properties ファイルのマニュアルでの変更

<geronimo_home>/var/security/groups.properties ファイルを開き、編集します。必要なユーザー名を追加し、ファイルを保存します。

groups.properties

```
admin=user1,user2
```

次に **<geronimo_home>/var/security/users.properties** ファイルを開きます。既存ユーザーのパスワード変更や、新規ユーザー追加ができます。新規ユーザーを追加する場合は、groups.properties ファイルに追加したものと同じにしてください。

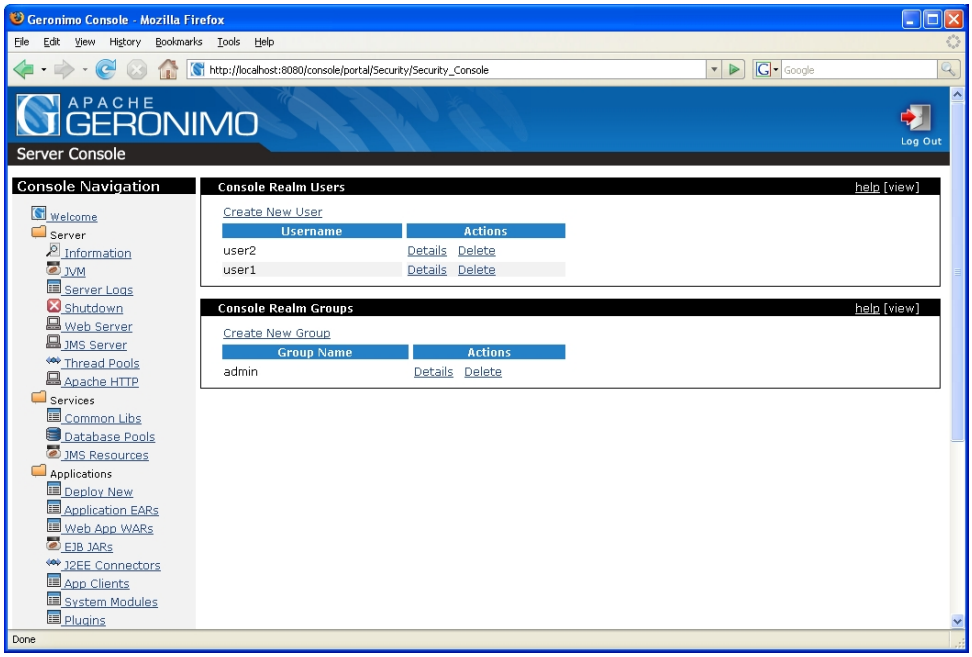
users.properties

```
user1=password1
user2=password2
```

この例では、2ユーザー (user1 / user2) を追加し、デフォルトのシステム・アカウントは削除しています。user1 / user2 の両方ともコンソールやコマンド・ライン・デプロイメント・ツールにアクセスできます。

管理コンソールからセキュリティ構成の変更

コンソールにログインし、コンソール・ナビゲーション・パネル上で Console Realm をクリックします。これによって、Console Realm Users と Console Realm Groups ポートレットが表示されます。



- "Create New User" リンクをクリックし、新規ユーザーの追加、もしくは Details にて既存ユーザーを編集します。
- "Create New Group" リンクをクリックし、グループに新規ユーザーを追加します。
- 新規のユーザー名とパスワードが追加されたら、コンソールからログアウトし、新しいユーザー名とパスワードを試してみてください。

Apache Geronimo v2.0 でのセキュリティ構成に関しては、1.1.3. セキュリティの構成 のセクションで詳細に述べられています。

トポロジー

今日のグローバリズムが進んだ世界では、組織が日常的に直面するオポチュニティーやチャレンジは多数存在し、組織を支える IT インフラへの責任の重大さからも柔軟・迅速な構成が組めるかどうかが決め手になります。Apache Geronimo アプリケーション・サーバーは中堅企業向けアプリケーションをサポートし最新の J2EE 仕様に対しても堅牢でセキュアなサポートを提供します。このセクションでは、皆さんご自身の本番環境で使用可能な、いくつかの構成オプションをご紹介します。

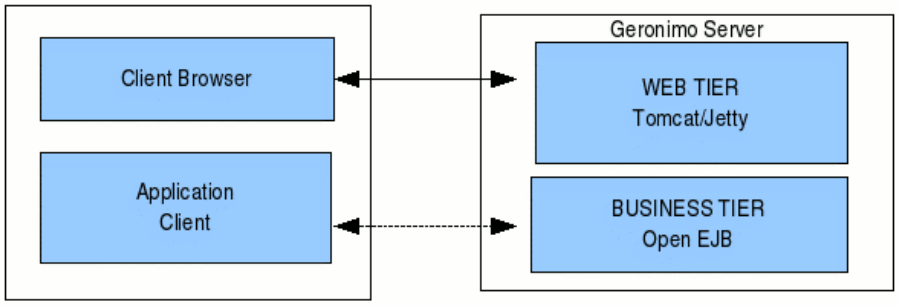
2層システム

下図では、Geronimo サーバーにホストされているアプリケーションに接続するクライアントを示しています。図には1台のクライアント・マシンしか記載されていませんが、複数マシンの場合でもサーバーへ接続可能です。それらマシン上のユーザーは、標準的な Web ブラウザーを使用して、Geronimo サーバーにホストされた Web アプリケーションに接続できます。

クライアント側のアプリケーションは範囲が広く、単純なコマンド・ライン・インターフェースから、GUI で作成されたインターフェースまであります。これらアプリケーションは HTTP 接続を使用しサーバーに接続することで Web 層に接続します。ビジネス/EIS 層へは、Geronimo クライアント・アプリケーション・コンテナーを通して接続します。
(参考URL: http://www.ibm.com/developerworks/jp/opensource/library/os-ag-client/index.html)

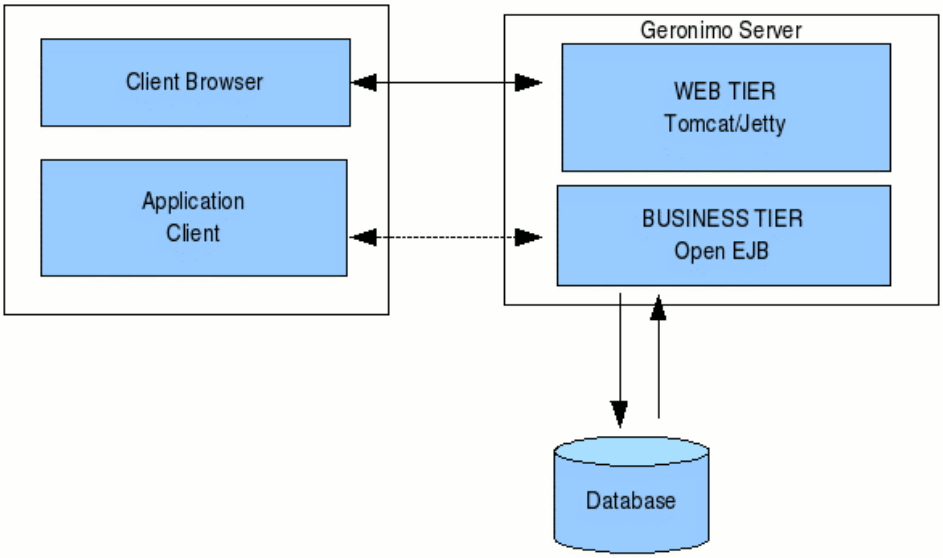
Geronimo アプリケーション・クライアントの特徴:

- Geronimo サーバーとの分離
- Geronimo サーバーとのコミュニケーションはネットワークを介して実施
- クライアント・アプリケーションとのマッピング上、疎結合の簡易な仕組みを提供



3層システム

3層アーキテクチャーによるシステムは2層よりもスケーラブルで、多人数の人と組織に対応可能で、柔軟性と自由度が増します。

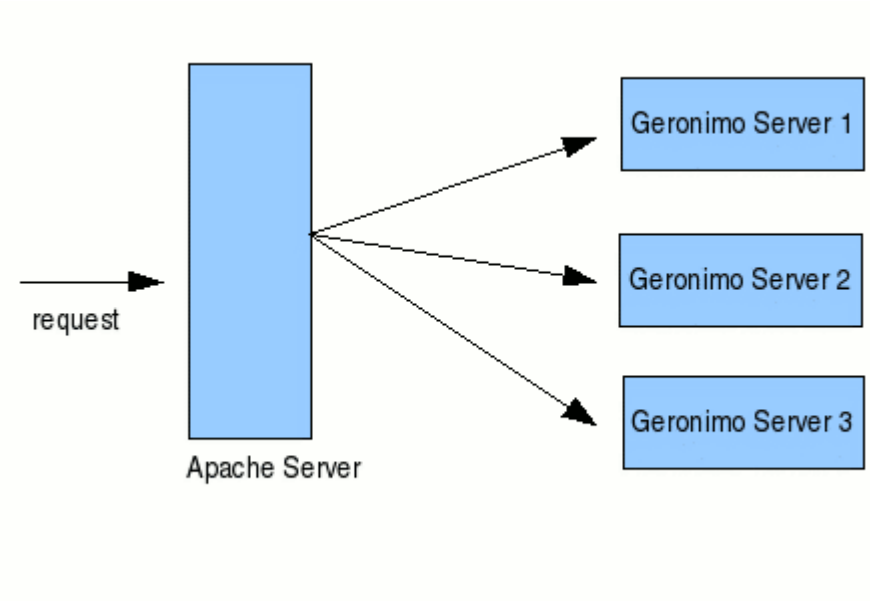


Apache HTTPd での構成

Apache web サーバーはもっともポピュラーな HTTP サーバーとして今日のインターネット環境で利用されています。本番環境で Geronimo を Apache web サーバーと合わせて使用することで、下記にあげた利点を享受できます。

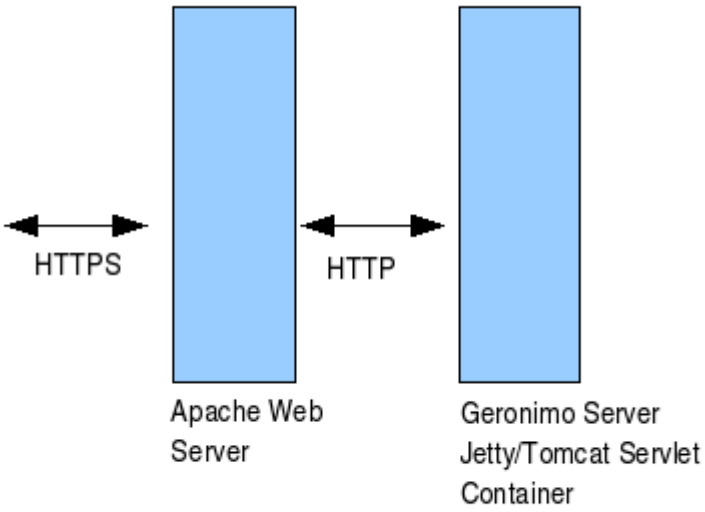
1. クラスタリング&ロード・バランシング

クラスタリングによってアプリケーション・サーバーは複数ノードのフェイルオーバーやセッション・データ共有、およびロード・バランシングを、ネットワーク上の多数のノード間にわたってサポートします。アプリケーションのロード・バランシングは、Apache サーバーに組み込まれた機能によってサポートされます。



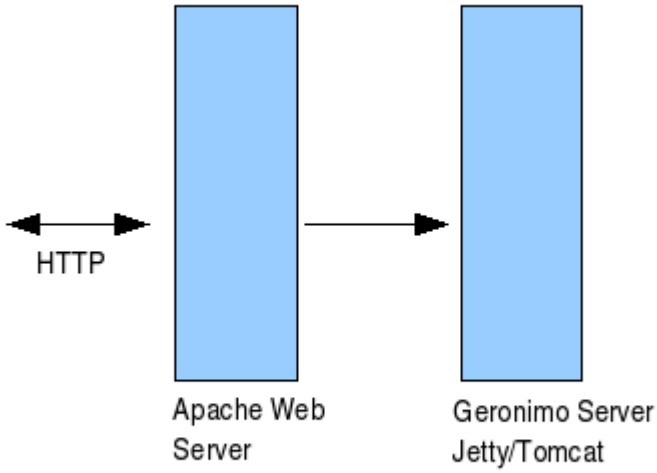
2. SSL エンコーダー

Web サーバー・プロキシの利点は、追加のハードウェアなしに、SSL で暗号化されたメッセージをサーバーへ安全に提供することです。しかし広範囲にわたる SSL プロセシングはプロキシに対して負荷をかけることになります。それを処理するのが Apache サーバーになります。一方で、Geronimo はアプリケーションのビジネス・ロジックをつかさどります。



3. アプリケーション・ロード・シェアリング

サーバー環境の多くでは、web サーバーとアプリケーション・サーバーが一緒に稼動し、すべての HTTP リクエストを取り扱います。Apache が扱うのは静的ページ（HTML, JPEG, GIF など）で、Geronimo はTomcat / Jetty のサポートにより動的ページ（JSP, サーブレット）を扱います。Geronimo サーバーも静的なページを扱いますが、両者混合のシステムでは、通常動的なリクエストを扱うよう構成されます。



サマリー

Geronimo アプリケーション・サーバーを使用することによって、セキュアなエンタープライズ・インフラストラクチャーを構築し、最新の技術標準や要件に適合します。ここでは標準的な Geronimo の導入に対してのカスタム構成を、いくつかの導入例を通して洞察しました。

4. Apache Geronimo への移行

This page last changed on 4 25, 2008 by JAGUG.

移行ツール

JBoss to Geronimo (J2G) は、JBoss アプリケーション・サーバー、もしくは、Java 2 Enterprise Edition (J2EE) 向けに書かれているアプリケーションのソースコードを Apache Geronimo プラットホームへ移行するのを支援するために設計された Eclipse SDK プラグインとして構築されているツール・セットです。

J2G のダウンロードに関する情報については、Geronimo Development Tools ページを開いてください。

サンプル・アプリケーションの移行

次に取り上げる内容は、JBoss v4.x から Apache Geronimo V2.0 へ、アプリケーションの移行を支援する解説集です。

J2EE アプリケーション間で差異のある特徴/機能に焦点を合わせた解説もいくつかあります。
これらの全解説は、自己完結であり、互いにまったく依存していません。
そして、これらの解説には、JBoss から Apache Geronimo への実装上における相違点に関するフィーチャー・トゥー・フィーチャーを比較した分析を提供しており、これは移行計画を立てる時に大きな手助けとなってくれることでしょう。

さらに、各解説には、一方のプラットホームから他方のプラットホームへ移行する体験に慣れたり、習得するためのサンプル・アプリケーションを提供しています。
これらの全解説は、同じ内部体系で構成されているので、複数の解説を越えた異なるトピックの周辺にある類似情報を探すのが、とても容易です。

完成済みの解説:

- 4.6. JBoss to Geronimo - Hibernate の移行
- 4.9. JBoss to Geronimo - サーブレットとJSPの移行
- 4.5. JBoss to Geronimo - EJB-セッションビーンの移行
- 4.2. JBoss to Geronimo - EJB-BMP 移行
- 4.7. JBoss to Geronimo - JDBC の移行

工事中の解説:

- 4.8. JBoss to Geronimo - Security Migration
- 4.a. JBoss to Geronimo - Web Services Migration
- 4.3. JBoss to Geronimo - EJB-CMP Migration
- 4.4. JBoss to Geronimo - EJB-MDB Migration

4.1. J2G 移行ツール

This page last changed on 4 21, 2008 by JAGUG.

概要

JBoss to Geronimo (J2G) は、JBoss アプリケーション・サーバー、もしくは、Java 2 Enterprise Edition (J2EE) 向けに書かれているアプリケーションのソースコードを Apache Geronimo プラットホームへ移行するのを支援するために設計された Eclipse SDK プラグインとして構築されているツール・セットです。

ダウンロード

J2G のダウンロードに関する情報については、Geronimo Development Tools ページを開いてください。

使用

4.1.1. ソースから J2G をビルド

This page last changed on 4 21, 2008 by JAGUG.

ソースからビルド

アプリケーションの移行する目的で J2G の使用に興味を持っていましたら、この解説はあなたのためでないかもしれません。それは、ソースから J2G をコンパイルする手順として提供するつもりです。J2G のバイナリー・コピーのダウンロードに関する情報については、Geronimo Development Tools ページを開いてください。

最新ソースから J2G をダウンロード、および、ビルドすることにより、ツール・セットに対して変更や追加が行えます。なお、必要条件や手順は、次の通りです。

必要条件

- Apache Maven (2.0.6+)
- Apache Maven (2.0.6 以上)
- Subversion (1.2+)
- Subversion (1.2 以上)
- JDK 5.0+ (J2SE 1.5.0+)
- JDK 5.0 以上 (J2SE 1.5.0 以上)

ソースコードの入手

必要条件の通りにインストールすれば、リポジトリーからソースコードをチェックアウトしてもよいタイミングです。これを実行するには、コマンドライン上から "svn co" コマンドを使用してください。現在、J2G のリポジトリーは https://svn.apache.org/repos/asf/geronimo/devtools/j2g です。

```
svn co http://svn.apache.org/repos/asf/geronimo/devtools/j2g/trunk j2g
```

ビルド

J2G のルート・ディレクトリーより、下記のコマンドを実行してください:

```
mvn install assembly:assembly
```

テストの無効化

ビルド中のユニット・テストを無効化するには、下記のコマンドを実行してください:

```
mvn install -Dtest=false assembly:assembly
```

デプロイ

ツールをビルド完了後、次のステップを踏むことによってデプロイ可能となります:

1. <j2g-home>/target 上で、作成された配布物を解凍します。
2. コマンドライン上で、配布物の bin ディレクトリーへ移動します。
3. OS に適した j2g-configure スクリプトを実行します。

4.1.2. J2G の使用

This page last changed on 4 25, 2008 by JAGUG.

コンポーネント

J2G は、基本となる Eclipse IDE UI エクステンションの3種類の実行可能なユーティリティーで構成されています。"コンフィグレーター" コンポーネント、3種類の主要コンポーネント、各々の関連機能を、次のように順序よく実行しなければいけません。

必要条件

J2G ツール・セットは、少なくとも、以下のアプリケーションのバージョンを必要とします。
(これらは、J2G の動作確認が取れているバージョンです)

- Sun JDK 5.0 以上 (J2SE 1.5)
- Eclipse SDK 3.3M7 (JDT Core Plug-in が付属していること)
- Apache Geronimo 1.1.1 以上
- JBoss 3.2 以上、もしくは、J2EE 1.2 以上で書き下ろされたアプリケーション

環境変数

J2G の各ツールでは、次の環境変数が正しい場所に設定されていなければいけません:

- JAVA_HOME (JDK の場所)
- ECLIPSE_HOME (Eclipse SDK の場所)
- WORKSPACE (一時領域となる eclipse ワークスペースの場所)

コンフィグレーター

このコンポーネントを実行するには、bin へ移動し、j2g-configure を実行してください。

```
user@lappy-486:~/j2g$ cd bin
user@lappy-486:~/j2g/bin$ ./j2g-configure.sh

Copying J2G plugins to /home/user/.m2/repository/eclipse/eclipse/plugins/ ...
```

```
`../plugins/org.apache.geronimo.j2g.common_1.0.0-SNAPSHOT.jar' ->
`/home/user/.m2/repository/eclipse/eclipse/plugins/org.apache.geronimo.j2g.common_1.0.0-
SNAPSHOT.jar'
`../plugins/org.apache.geronimo.j2g.descriptors_1.0.0-SNAPSHOT.jar' ->
`/home/user/.m2/repository/eclipse/eclipse/plugins/org.apache.geronimo.j2g.descriptors_1.0.0-
SNAPSHOT.jar'
`../plugins/org.apache.geronimo.j2g.jasper_1.0.0-SNAPSHOT.jar' ->
`/home/user/.m2/repository/eclipse/eclipse/plugins/org.apache.geronimo.j2g.jasper_1.0.0-
SNAPSHOT.jar'
`../plugins/org.apache.geronimo.j2g.resources_1.0.0-SNAPSHOT.jar' ->
`/home/user/.m2/repository/eclipse/eclipse/plugins/org.apache.geronimo.j2g.resources_1.0.0-
SNAPSHOT.jar'
`../plugins/org.apache.geronimo.j2g.sources_1.0.0-SNAPSHOT.jar' ->
/home/user/.m2/repository/eclipse/eclipse/plugins/org.apache.geronimo.j2g.sources_1.0.0-
SNAPSHOT.jar'
`../plugins/org.apache.geronimo.j2g.util_1.0.0-SNAPSHOT.jar' ->
`/home/user/.m2/repository/eclipse/eclipse/plugins/org.apache.geronimo.j2g.util_1.0.0-SNAPSHOT.jar'
`../properties/org.apache.geronimo.j2g.descriptors.ejb/persistence_differences.properties' ->
`/home/user/.m2/repository/eclipse/eclipse/plugins/org.apache.geronimo.j2g.descriptors.ejb/
persistence_differences.properties'
`../properties/org.apache.geronimo.j2g.descriptors.ejb.annotation/
annotation_differences.properties' ->
/home/user/.m2/repository/eclipse/eclipse/plugins/
org.apache.geronimo.j2g.descriptors.ejb.annotation/annotation_differences.properties'
`../properties/org.apache.geronimo.j2g.sources.dependence/default_jars.properties' ->
`/home/user/.m2/repository/eclipse/eclipse/plugins/org.apache.geronimo.j2g.sources.dependence/
default_jars.properties'
`../properties/org.apache.geronimo.j2g.sources.dependence/class_analogies.properties' ->
`/home/user/.m2/repository/eclipse/eclipse/plugins/org.apache.geronimo.j2g.sources.dependence/
class_analogies.properties'
`../properties/org.apache.geronimo.j2g.sources.dependence/compatible_sources.properties' ->
`/home/user/.m2/repository/eclipse/eclipse/plugins/org.apache.geronimo.j2g.sources.dependence/
compatible_sources.properties'
`../properties/org.apache.geronimo.j2g.sources.environment/beans-types.properties' ->
`/home/user/.m2/repository/eclipse/eclipse/plugins/org.apache.geronimo.j2g.sources.environment/
beans-types.properties'
`../properties/org.apache.geronimo.j2g.sources.environment/beans-interfaces.properties' ->
`/home/user/.m2/repository/eclipse/eclipse/plugins/org.apache.geronimo.j2g.sources.environment/
beans-interfaces.properties'
`../properties/org.apache.geronimo.j2g.sources.environment/beans-references.properties' ->
`/home/user/.m2/repository/eclipse/eclipse/plugins/org.apache.geronimo.j2g.sources.environment/
beans-references.properties'
`../compatibility/org.apache.geronimo.j2g.sources.dependence.compatibility/DigestCallback.java' ->
`/home/user/.m2/repository/eclipse/eclipse/plugins/
org.apache.geronimo.j2g.sources.dependence.compatibility/DigestCallback.java'
`../compatibility/org.apache.geronimo.j2g.sources.dependence.compatibility/
GeronimoLoginModule.java' ->
`/home/user/.m2/repository/eclipse/eclipse/plugins/
org.apache.geronimo.j2g.sources.dependence.compatibility/GeronimoLoginModule.java'
`../compatibility/org.apache.geronimo.j2g.sources.dependence.compatibility/PasswordHasher.java' ->
`/home/user/.m2/repository/eclipse/eclipse/plugins/
org.apache.geronimo.j2g.sources.dependence.compatibility/PasswordHasher.java'
`../compatibility/org.apache.geronimo.j2g.sources.dependence.compatibility/Nobody.java' ->
`/home/user/.m2/repository/eclipse/eclipse/plugins/
org.apache.geronimo.j2g.sources.dependence.compatibility/Nobody.java'
`../compatibility/org.apache.geronimo.j2g.sources.dependence.compatibility/
GeronimoSimpleGroup.java' ->
`/home/user/.m2/repository/eclipse/eclipse/plugins/
org.apache.geronimo.j2g.sources.dependence.compatibility/GeronimoSimpleGroup.java'
`../compatibility/org.apache.geronimo.j2g.sources.dependence.compatibility/Anybody.java' ->
`/home/user/.m2/repository/eclipse/eclipse/plugins/
org.apache.geronimo.j2g.sources.dependence.compatibility/Anybody.java'

A backup of your configuration file has been written to /home/user/.m2/repository/eclipse/eclipse/
configuration/config.bak
```

```
Configuration complete.
user@lappy-486:~/j2g/bin$
```

これは、./plugins、./properties 、および、./compatibility の本体を ECLIPSE_HOME/plugins へ自動コピーするだけでなく、ECLIPSE_HOME/configuration/config.ini も変更します(もしくは、必要に応じて、作成します)。

ソース解析ツールの概略

このツールは、コマンドライン引数として渡されたディレクトリーにおいて、ソースコード (.java および .jsp ファイル) を再帰的に解析します。これを実行している間、Geronimo と同等な要素に対応している、あらゆる JBoss クラスや依存関係の要素を置換します。同じものがなかった場合、置換するクラスを手作業で作成するか、JBoss クラスに依存しているコードの部分を書き換えることによって、解決しなければならないこの課題をユーザーに警告します。

ディスクリプター変換ツールの概略

このツールは、JBoss 固有のデプロイメント・ディスクリプターや J2EE 標準のデプロイメント・ディスクリプターを、(必要に応じて) Geronimo の同等な要素へ変換します。

リソース変換ツールの概略

このツールは、Java メッセージング・サービス (JMS) キュー、JMS トピック、Java データベース接続 (JDBC) データソース、といった JBoss 固有のリソースを Geronimo の同等な要素へ変換します。

ソース解析ツール

⚠ J2G を実行する前に、アプリケーションのソースのバックアップを取っておくことを 強くお勧め します。

説明

ソース解析ツールから、最初にスタートしてください。移行プロセス中に変更の必要なソースや固有ファイルをスキャンします。これらのファイルには、Java クラス (.java ファイル)、JSP (.jsp ファイル) だけでなく、引数 -jspext を指定した JSP ファイルの拡張をオーバーライドしたファイル (例: .html ファイル用の -jspext html) も含まれます。

これらにファイルに対して、当該ツールは以下の内容をチェックします:

- Geronimo の非互換 API の使用 (例. JBoss 内部 API)。
- J2EE ネーミング規則の違反。

ツールは Geronimo 非互換 API 使用のチェック時に、JBoss クラスは Geronimo 互換クラスへ自動的に置換されます。

JBoss クラス	Geronimo 互換クラス
org.jboss.security.SimplePrincipal	org.apache.geronimo.security.realm.providers.GeronimoUserPrincipa
org.jboss.security.auth.spi.DigestCallback	org.apache.geronimo.j2g.sources.dependence.compatibility.DigestC
org.jboss.security.Nobody	org.apache.geronimo.j2g.sources.dependence.compatibility.Nobody
org.jboss.security.Anybody	org.apache.geronimo.j2g.sources.dependence.compatibility.Anybod
org.jboss.security.auth.spi.UsernamePasswordLoginModule	org.apache.geronimo.j2g.sources.dependence.compatibility.Geronim
org.jboss.security.SimpleGroup	org.apache.geronimo.j2g.sources.dependence.compatibility.Geronim

今後は JBoss に依存できなくなるので、上記の一覧にない参照があれば手作業で修正しなければいけません。処理する JSP ファイルには問題の識別に限界があるのと、自動修正が現在サポートされていないことに、注意しておくことが重要です。

ツールが J2EE ネーミングの検証時に、コンポーネント (クラス、または、JSP) が変換されるアプリケーションの部分にない名前のオブジェクトを参照しているケースを検知します (例: データソース、JMS、Eメール・プロバイダーのリファレンス)。単純なケースの場合、コンポーネントがダイレクト、フル・パスつきの外部リソースを要求し、かつ、環境リファレンスがないと、ツールは、コンポーネントのデプロイメント・ディスクリプターに対してリソース・リファレンスを追加することによって自動的にこの問題を修正しようと試み、追加されたリファレンスの名前のダイレクト・パスに置換します。

ソース解析ツールでソースファイルが自動変換されると、変換元のバージョンは .j2g エクステンションに保存され、情報レベルのメッセージが出力されます。その他のケースでは、問題を手作業で修正する必要があるときに、ツールは警告メッセージを簡単に出力します。

追加クラスの指定

JBoss 固有クラスに対する追加のドロップ・イン置換は、正式なリレーションシップを含む <ECLIPSE_HOME>/plugins/org.apache.geronimo.j2g.sources.dependence/class_analogies.properties だけでなく、クラス名を含む <ECLIPSE_HOME>/plugins/org.apache.geronimo.j2g.sources.dependence/compatible_sources.properties を変更することによって、ユーザー定義可能です。これらの置換クラスに対応するソースコードは、正式な含有に対応する <ECLIPSE_HOME>/plugins/org.apache.geronimo.j2g.sources.dependence.compatibility ディレクトリーに配置しなければなりません。

Geronimo の JAR ファイルにすでに存在するクラスは、class-analogies.properties ファイルに追加することが可能です。これらのクラスは、互換ディレクトリーに配置する必要はありませんが、依存関係については、変換元のアプリケーションにある関連ファイルに追加しなければいけません。

以下に示すのは、class_analogies.properties ファイルの初期設定例です。

class_analogies.properties

```
#
#  Licensed to the Apache Software Foundation (ASF) under one or more
#  contributor license agreements.  See the NOTICE file distributed with
#  this work for additional information regarding copyright ownership.
#  The ASF licenses this file to You under the Apache License, Version 2.0
#  (the "License"); you may not use this file except in compliance with
#  the License.  You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
#  Unless required by applicable law or agreed to in writing, software
#  distributed under the License is distributed on an "AS IS" BASIS,
#  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#  See the License for the specific language governing permissions and
#  limitations under the License.
#
##########################################################################################################################
# Jboss classes and it's analogies.
##########################################################################################################################
org.jboss.security.SimplePrincipal =
org.apache.geronimo.security.realm.providers.GeronimoUserPrincipal
org.jboss.security.auth.spi.DigestCallback =
org.apache.geronimo.j2g.sources.dependence.compatibility.DigestCallback
org.jboss.security.Nobody = org.apache.geronimo.j2g.sources.dependence.compatibility.Nobody
org.jboss.security.Anybody = org.apache.geronimo.j2g.sources.dependence.compatibility.Anybody
org.jboss.security.auth.spi.UsernamePasswordLoginModule =
org.apache.geronimo.j2g.sources.dependence.compatibility.GeronimoLoginModule
org.jboss.security.SimpleGroup =
org.apache.geronimo.j2g.sources.dependence.compatibility.GeronimoSimpleGroup
```

以下に示すのは、compatible_sources.properties ファイルの初期設定例です。

compatible_sources.properties

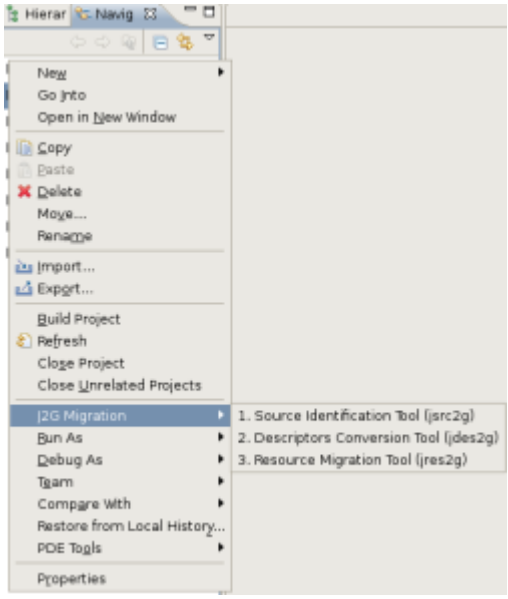
```
#
#  Licensed to the Apache Software Foundation (ASF) under one or more
#  contributor license agreements.  See the NOTICE file distributed with
#  this work for additional information regarding copyright ownership.
#  The ASF licenses this file to You under the Apache License, Version 2.0
```

```
#  (the "License"); you may not use this file except in compliance with
#  the License.  You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
#  Unless required by applicable law or agreed to in writing, software
#  distributed under the License is distributed on an "AS IS" BASIS,
#  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#  See the License for the specific language governing permissions and
#  limitations under the License.
#
########################################################################
# List of compatibility sources
########################################################################
org.apache.geronimo.j2g.sources.dependence.compatibility.DigestCallback
org.apache.geronimo.j2g.sources.dependence.compatibility.Nobody
org.apache.geronimo.j2g.sources.dependence.compatibility.Anybody
org.apache.geronimo.j2g.sources.dependence.compatibility.GeronimoLoginModule
org.apache.geronimo.j2g.sources.dependence.compatibility.GeronimoSimpleGroup
org.apache.geronimo.j2g.sources.dependence.compatibility.PasswordHasher
```

ソース解析ツールの実行 - Eclipse GUI

Eclipse IDE GUI にてこのコンポーネントを実行するには、左パネルがまずナビゲーター・ビューになければいけません。

ナビゲーター・ビューに入ると、右クリックして、コンテキスト・メニューから J2G Migration を選択することが可能です。



J2G Migration サブ・メニューに入り、"1. Source Identification Tool (jsrc2g)" を選択し、後は表示されているプロンプトに従ってください。

出力内容は Eclipse IDE の下部にある Console ウィンドウの中に表示されます。

ソース解析ツールの実行 - コマンドライン

このコンポーネントを実行するには、j2g の bin ディレクトリーから次のコマンドを実行してください。<APPLICATION> は変換するアプリケーションのパスを表し、<GERONIMO_HOME> は Geronimo 2.0のパスを表しています。

*nix

```
./jsrc2g.sh -src <APPLICATION>/src -web <APPLICATION>/web -geronimo <GERONIMO_HOME>
```

Windows

```
jsrc2g.bat -src <APPLICATION>\src -web <APPLICATION>\web -geronimo <GERONIMO_HOME>
```

注意: 変換対象に複数のソースを指定したい時は、次のように、単に , (カンマ)で区切ってください。

```
jsrc2g.cmd -src <APPLICATION>/src,<APPLICATION>/src2,...<APPLICATION>/srcn -web <APPLICATION>/web -geronimo <GERONIMO_HOME>
```

ツールの実行後(システムの速度やアプリケーションの複雑度次第では、数秒程度かかるかもしれません)、コンソール出力に、警告やエラーがすべて表示されます。これらのメッセージは、ユーザーに合うように解釈され、目に見える形で修正されなければいけません。変更を完了すると、ソース解析ツールは、修正の確認を再度実行することが可能です。

ディスクリプター変換ツール

説明

ディスクリプター変換ツールは、次のデプロイメント・ディスクリプターを検索して、自動変換を試みます。その結果、それらは Geronimo で使用されます:

ファイル名	目的
application.xml	J2EE アプリケーション・デプロイメント・ディスクリプター
jboss-app.xml	セキュリティー・ドメインやクラスローダーの構成等といった要素を目的とした JBoss 固有のディスクリプター
web.xml	Web アプリケーション・ディスクリプター
jboss-web.xml	セキュリティー・ロール、セッション設定、Web サービスの構成といった要素を目的とした JBoss 固有のディスクリプター。それは、Web アプリケーションを JBoss のデプロイメントに統合するために使用されます
ejb-jar.xml	EJB デプロイメント・ディスクリプター
jboss.xml	Bean や JNDI 名のカスタム・コンテナー構成のようなオプション情報を提供する JBoss ディスクリプター。それは、JBoss で使用されますが、各々の ejb-jar.xml には記述されません
jbosscmp-jdbc.xml	CMP-データベース・マッピングや、キー生成情報を記述している JBossCMP-JDBC 構成ファイル
standardjbosscmp-jdbc.xml	JBossCMP-JDBC の初期構成ファイル
persistence.xml	データソース・コネクションの構成ファイル

注意: Geronimo は OpenEJB を使用し、専用のディスクリプター (openejb-jar.xml) を必要とします。jboss.xml が与えられない一部のインスタンスが若干あるかもしれません。これらのケースでは、openejb-jar.xml は、J2G に指定されたソース・パスの一番上のレベルのディレクトリー上に作成されます。

アノテーション (EJB 3.0 用)

標準の EJB アノテーションは、Geronimo と JBoss の両方でサポートされています。しかしながら、アノテーションの一部に提供されている属性の中には、差異があります。
以下に示すのは、指定アノテーションの中に発生するかもしれない、提供されているすべての標準属性の違いの一覧です。

Annotation アノテーション	JBoss Attribute JBoss 属性	Geronimo Attribute Geronimo 属性
Resource	mappedName	name
EJB	mappedName	name
MessageDriven	mappedName	name
Stateful	mappedName	name
Stateless	mappedName	name

今後は JBoss に依存できなくなるので、上記の一覧にない参照があれば手作業で修正しなければいけません。

> ⚠ JBoss は、独自の EJB アノテーションの拡張機能を持っています。これらは、必要に応じて、手作業で修正する必要があります。

追加アノテーションの指定

JBoss 固有のアノテーション属性に対する追加のドロップ・イン置換は、正式なリレーションシップを含む <ECLIPSE_HOME>/plugins/org.apache.geronimo.j2g.descriptors.ejb.annotation/annotation_differences.properties を変更することによって、ユーザー定義可能です。というのも、これらのマッピングの本来の形は1対1ではなく、<annotation>=<attribute> という従来の書式の中でプロパティーを書式化することができません。その代わりに、正式な書式化は次のようになります:

標準の EJB アノテーションの場合

```
<property> :: <annotation>=<mapping>
```

<annotation> は 標準の EJB アノテーションです。

Geronimo 互換のリレーションシップに対する標準的な JBoss 固有属性の場合

```
<mapping> :: <j_attribute>--><g_attribute>
```

<j_attribute> は JBoss がサポートしている属性、<g_attribute> は Geronimo がサポートしている属性です。

最後に、単一アノテーションに対して複数属性のマッピングを必要とする場合

```
<mapping> :: <j_attribute>--><g_attribute>,<mapping>
```

<j_attribute> は JBoss がサポートしている属性、<g_attribute> は Geronimo がサポートしている属性です。

以下に示すのは、annotation_differences.properties ファイルの初期設定例です。

annotation_differences.properties

```
#
#  Licensed to the Apache Software Foundation (ASF) under one or more
#  contributor license agreements.  See the NOTICE file distributed with
#  this work for additional information regarding copyright ownership.
#  The ASF licenses this file to You under the Apache License, Version 2.0
#  (the "License"); you may not use this file except in compliance with
#  the License.  You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
#  Unless required by applicable law or agreed to in writing, software
#  distributed under the License is distributed on an "AS IS" BASIS,
```

```
#  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#  See the License for the specific language governing permissions and
#  limitations under the License.
#
#############################################################################
# Jboss classes and it's analogies.
#############################################################################
Resource = mappedName-->name
EJB = mappedName-->name
MessageDriven = mappedName-->name
Stateful = mappedName-->name
Stateless = mappedName-->name
```

OpenEJB

ツールが Geronimo と非互換の EJB リファレンスの使用をチェックすると、これらの JBoss EJB リファレンスは Geronimo 互換の OpenEJB リファレンスに自動的に置換されます。ここにあるのは、(EJB 3.0 で使用される) persistence.xml の中でで変更するために示されている差異です。したがって、構成は密接に結び合っています。

JBoss EJB	Geronimo OpenEJB
hibernate.connection.url	openjpa.ConnectionURL
hibernate.connection.driver_class	openjpa.ConnectionDriverName
hibernate.connection.password	openjpa.ConnectionPassword
hibernate.connection.username	openjpa.ConnectionUserName

今後は JBoss に依存できなくなるので、上記の一覧にない参照があれば手作業で修正しなければいけません。

persistence.xml に対する追加 OpenEJB アノテーションの指定

JBoss 固有のアノテーション属性に対する追加のドロップ・イン置換は、正式なリレーションシップを含む <ECLIPSE_HOME>/plugins/org.apache.geronimo.j2g.descriptors.ejb/persistence_differences.properties を変更することによって、ユーザー定義可能です。

以下に示すのは、persistence_differences.properties ファイルの初期設定例です。

persistence_differences.properties

```
#
#  Licensed to the Apache Software Foundation (ASF) under one or more
#  contributor license agreements.  See the NOTICE file distributed with
#  this work for additional information regarding copyright ownership.
#  The ASF licenses this file to You under the Apache License, Version 2.0
#  (the "License"); you may not use this file except in compliance with
#  the License.  You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
#  Unless required by applicable law or agreed to in writing, software
#  distributed under the License is distributed on an "AS IS" BASIS,
#  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#  See the License for the specific language governing permissions and
#  limitations under the License.
#
#############################################################################
# Jboss persistence.xml property names and its analogies.
#############################################################################
hibernate.connection.url = openjpa.ConnectionURL
hibernate.connection.driver_class = openjpa.ConnectionDriverName
hibernate.connection.password = openjpa.ConnectionPassword
```

```
hibernate.connection.username = openjpa.ConnectionUserName
hibernate.hbm2ddl.auto = suggestion:openjpa.jdbc.SynchronizeMappings
```

X が Y に対する代わりであるかもしれないことをユーザーにしっかり提示するためには (これは直接解釈されないため、条件つきです)、プロパティーは <Y> = suggestion:<X> として書くことも可能です。

ディスクリプター変換ツールの実行 - Eclipse GUI

Eclipse IDE GUI にてこのコンポーネントを実行するには、左パネルがまずナビゲーター・ビューになければいけません。

ナビゲーター・ビューに入ると、右クリックして、コンテキスト・メニューから J2G Migration を選択することが可能です。

J2G Migration サブ・メニューに入り、"2. Descriptors Conversion Tool (jdes2g)", を選択し、後は表示されているプロンプトに従ってください。
出力内容は Eclipse IDE の下部にある Console ウィンドウの中に表示されます。

ディスクリプター変換ツールの実行 - コマンド・ライン

ディスクリプター変換ツールを実行するには、j2g の bin ディレクトリーから次のコマンドを実行してください。繰り返しになりますが、<APPLICATION> は変換するアプリケーションのパスを表します。

*nix

```
./jdesc2g.sh <APPLICATION>
```

Windows

```
jdesc2g.bat <APPLICATION>
```

このツールの出力に示されている警告は、自動的に変換できなかった未サポートの要素のみをレポートしているため、手作業で解決および変換しなければなりません。この処理は、変換されるアプリケーションの構造に大きく依存するため、本文書の範囲を越えてしまいます。

リソース変換ツール

説明

リソース変換ツールは、JBoss 固有のリソースを Geronimo の同等な要素へ変換します。次にあげる種類のリソースを対象に処理します:

- JDBC 接続プールを定義している JBoss/JCA データソース・ディスクリプター (*-ds.xml ファイル)。データソースはデータベースの種類に依存するため、ツールは次のよく知られているデータベース向けの移行を自動的に実行します
  - DB2
  - Derby
  - Hypersonic
  - Informix
  - Microsoft SQL
  - MySQL
  - Oracle
  - PostgreSQL
  - それ以外のデータソースの定義は手作業で移行する必要があります。定義が未知のパラメータを含んでいる場合、ツールは警告メッセージを添えてこのことを知らせます。
- JBoss MBean ディスクリプター (*-service.xml ファイル)。次のリソースの定義のみが移行されます。
  - javax.mail.Session
  - javax.jms.QueueConnectionFactory
  - javax.jms.TopicConnectionFactory
  - javax.transaction.TransactionManager
- JASS ログイン・モジュール構成を提供している JBoss セキュリティー・ポリシー・ディスクリプター (login-config.xml)。ツールは次の JBoss ログイン・モジュール向けにロールやポリシーだけでなくユーザーやグループを移行します。
  - org.jboss.security.auth.spi.IdentityLoginModule

- org.jboss.security.auth.spi.UsersRolesLoginModule
- org.jboss.security.auth.spi.LdapLoginModule
- org.jboss.security.auth.spi.DatabaseServerLoginModule

リソース変換ツールの実行 - Eclipse GUI

Eclipse IDE GUI にてこのコンポーネントを実行するには、左パネルがまずナビゲーター・ビューになければいけません。

ナビゲーター・ビューに入ると、右クリックして、コンテキスト・メニューから J2G Migration を選択することが可能です。

J2G Migration サブ・メニューに入り、"3. Resource Migration Tool (jres2g)", を選択し、後は表示されているプロンプトに従ってください。
出力内容は Eclipse IDE の下部にある Console ウィンドウの中に表示されます。

リソース変換ツールの実行 - コマンド・ライン

ディスクリプター変換ツールを実行するには、j2g の bin ディレクトリーから次のコマンドを実行してください。繰り返しになりますが、<APPLICATION> は変換するアプリケーションのパスを表します。

*nix

```
./jres2g.sh <APPLICATION>
```

Windows

```
jres2g.bat <APPLICATION>
```

ツールは、移行対象の各リソースの名前だけでなく、ファイル名や行番号を添えた情報、エラー、警告のメッセージを出力します。エラーなしでリソースの移行が完了した場合、ツールは成功のメッセージを出力し、同じフォルダーに移行されたりソースをオリジナル・リソースとして生成します。

他のツールと同様に、エラーや警告のメッセージについては、変換処理を完了させるために、手作業で修正する必要があります。繰り返しになりますが、それは大きなアプリケーション固有の内容であり、本文書の範囲を越えています。

完了

上記のツールをすべて実行し、エラーや警告がなければ、変換されたアプリケーションは Geronimo 用にビルドされるかもしれません。また、このことは極端にアプリケーション固有であったり、特定の状況ごとに十分適応させる必要がある、といった重要なことです。

正常にビルドされた後、アプリケーションは、問題なく Geronimo 2.0 の実行インスタンスの中でデプロイすることが可能ですが、それはデプロイメント時に表示される解決すべきエラーがない場合であって、アプリケーションはリビルドさせる必要があります。

4.2. JBoss to Geronimo - EJB-BMP 移行

This page last changed on 4 25, 2008 by JAGUG.

エンティティー・ビーンは、データベースから読み込むことができ、複数のフィールドをデータで埋めているような、永続化された情報を表現するものとして定義されています。エンティティ・ビーンは、更新されたり、データベースへ書き戻されたりすることも可能です。エンティティー・ビーンには2種類あります。BMP(Bean-Managed Persistence) と CMP(Container-Managed Persistence) です。この文書では、BMP サンプル・アプリケーションの移行について取り扱います。この種のエンティティー・ビーンでは、読込や保存、データの検索といった永続化操作を実行するために、実際にコードを書かなければいけません。例えば、データベースに対して、select、insert、update、delete を実行する JDBC のような永続化 API を、開発者は使用する必要があります。

この文書は、次のセクションで構成されています:

BMP 実装の分析

BMP 実装はベンダーにより様々でしょう。本セクションの目的は、JBoss v4.0.5 と Apache Geronimo との間の BMP 仕様の機能比較を提供し、移行前の相違点や明確にし、それに応じて計画を立てられるようにします。

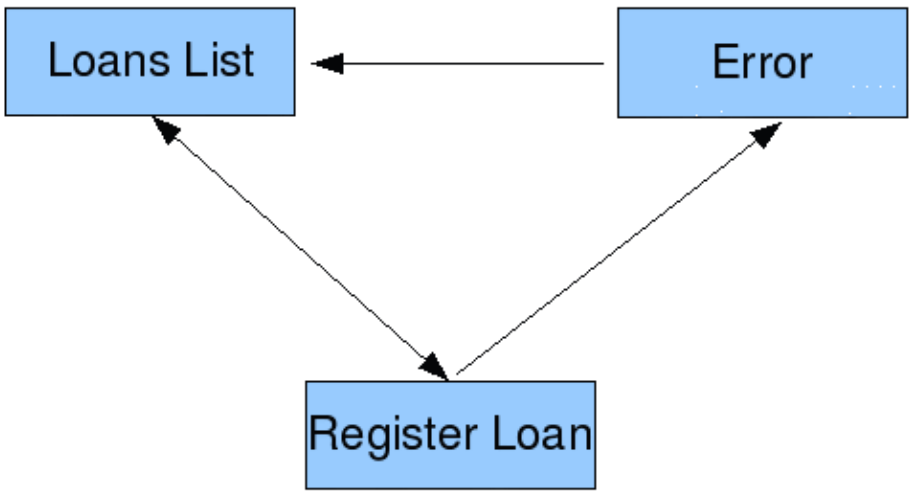
機能	JBoss v4.2.1	Apache Geronimo
EJB コンテナー	JBoss は独自の実装を使用します	Geronimo は OpenEJB を EJB コンテナーとして使用します

トップへ戻る

サンプル・アプリケーション

これは、ユーザーにローンを許可し、管理者がローンを承認/拒否する、シンプルなローン登録のアプリケーションです。顧客は、管理者がローンの承認/拒否するためにシンプルな Java アプリケーションを運用している間、Web サイト上でローンを利用可能です。BMP エンティティー・ビーンは、データベースからローン関連で永続されているデータを管理するために使用されます。それに加え、ステートレス・セッション・ビーンは、ワークフロー関連の状態をハンドルするために使用されます。

次の図は、ローン管理の Web アプリケーション・フローを示します:



ローン登録の初期画面は、Webアプリケーションがローンと最新ステータスの一覧を表示する通知ボードとして、振る舞います。顧客がローンを利用する際、登録フォームを利用可能です。ローンの登録後、保留中の間は保持されます。管理者権限を持つマネージャーはローンのステータスを更新するために、小型のアプリケーションを使用します。当該アプリケーションによって、マネージャーは、ステータスを承認するか拒否するか変更することができます。

アプリケーションのクラスと JSP ページ

- org.apache.geronimo.samples.loan.client
  - LoanStatusChanger - 貸付の承認手続をするスタンドアローン・アプリケーション。
- org.apache.geronimo.samples.loan.dto
  - LoanDTO - Web 層と EJB 層との間のローン関連情報に関するデータ転送オブジェクト。
- org.apache.geronimo.samples.loan.ejb
  - LoanBean - データベースからローン関連の永続データを受け取るための BMP。
  - LoanManagerBean - ハンドルするステートレス・セッション・ビーン。
- org.apache.geronimo.samples.loan.util
  - PropertyLoader - 異なる種類のクライアントに対してアプリケーション・サーバー関連のプロパティーを読み込みます。
- org.apache.geronimo.samples.loan.web
  - LoanManagerDispatchServlet - フロント・エンドから Web 層へ関連アクティビティーを受け取るローンをディスパッチするサーブレット。

また、このローン管理 Web アプリケーションは、次の JSP ページを含んでいます:

- error.jsp - 予期せぬ状況を受け取る共通のエラー画面。
- index.jsp - アプリケーションのローン一覧へ転送します。
- list_loans.jsp - ローン一覧とその情報を表示します。
- register_loan.jsp - 顧客が入力するローンの登録フォーム。

使用したツール

ローン管理アプリケーションの開発および構築に使用したツールは、以下の通りです:

Eclipse

Eclipse IDE を使用してサンプル・アプリケーションは開発されました。非常に強力でポピュラーなオープンソース開発ツールであり、JBoss と Geronimo の両方にとって有効な統合プラグインです。Eclipse は次の URL からダウンロード可能です:
http://www.eclipse.org

Apache Ant

Ant は Pure Java のビルド・ツールです。war ファイルの構築やオンライン仲介業務アプリケーションのデータベースの操作に使用されます。Ant は次の URL からダウンロード可能です:
http://ant.apache.org

トップへ戻る

JBoss 環境

このセクションでは、何処にどのようにして、サンプル JBoss リファレンス環境がインストールされ、シナリオを実装に対応付けるかを示します。この移行例では JBoss v4.1.2 が使用されたことを補足しておきます。

インストール、構成、JBoss の管理の詳細な手順は、製品ドキュメントに記載されています。最も新しく更新されているドキュメントについては、製品の Web サイトを確認してください。

次の一覧は、サンプル・アプリケーションをデプロイする開始地点として、初期環境のインストールと設定を完了しなければいけない一般的なタスクを強調しています。

1. 製品ガイドで説明されているように JBoss v4.2.1 をダウンロードしてインストールします。ここからインストール・ディレクトリーは <jboss_home> として参照されます。
2. 初期状態の JBoss v4.2.1 アプリケーション・サーバーのコピーを作成します。<jboss_home>¥server¥default から <jboss_home>¥server¥<your_server_name> へ再帰的にコピーが行われます。

3. <jboss_home>¥bin ディレクトリーから run.sh -c <your_server_name> コマンドを実行することによって、新規サーバーを始動します。
4. サーバーが始動したら、Webブラウザーを開き URL: http://localhost:8080 を指定して、実行していることを確認することが可能です。JBoss コンソールへアクセス可能な JBoss ウエルカム画面が表示されます。
5. アプリケーション・サーバーが実行中であることを確認したら、次の段階は、サンプル・アプリケーションに必要なソフトウェアのインストールと設定です。この段階は次のセクションに記述されています。

必要なソフトウェアのインストールと設定

この文書に含まれる Loan BMP アプリケーションをビルドして実行させるために、ビルド・ツールおよびアプリケーションで使用されるデータベースのインストールと設定が必要です。

データベース設定の変更

このアプリケーションは JBoss バンドルの一部である HSQL データベースを使用しています。データベースを作成するためのスクリプトを変更する必要があります。<jboss_home>¥server¥<your_server_name>¥data¥hypersonic ディレクトリーにある *localDB.script*ファイルを編集してください:

サンプル HSQL データベースを作成するための次の例の内容を localDB.script ファイルの最初に追加してください。また、JBoss 固有データ配下にある config/db.sql に指定された同ファイルの終端にサンプルのデータを追加してください。

> ⚠ このファイルを編集する際は JBoss が実行していないことを確認してください。

Ant の構成

すでに述べたとおり、Apache Ant は Online Brokerage アプリケーションのバイナリーをビルドするために使用されます。まだ Ant をインストールしていなかったら、そのためのよい機会です。さらに、<ant_home>/bin ディレクトリーがシステムの path 変数に追加されていることも確認してください。

Apache Ant は、次のURLからダウンロード可能です:

http://ant.apache.org

XDoclet の構成

XDoclet は、構成ファイル生成のビルド・ツールとして使用します。オープンソースのコード生成エンジンです。Java の 属性指向プログラミングを可能にします。要約すると、Java ソースに対するメタ・データ(属性)を追加することによって、コードに意味を持たせることが可能となることを意味します。
XDoclet が EJB を作成するツールとして由来したとしても、汎用的なコード生成エンジンへ進化しました。XDoclet は、コアかつ非常に多くのモジュールから構成されます。新種のコンポーネントが必要となる場合、新規モジュールを書くために、かなり一方的です。
http://xdoclet.sourceforge.net/xdoclet/index.html

XDoclet の最新バージョンを解凍し、build.properties ファイルに xdoclet.home パラメータに設定してください。

サンプル・アプリケーションのビルド

この解説に含まれているローン管理アプリケーションでは、当該アプリケーションをビルドするために使用する Ant スクリプトが提供されています。次のリンクからローン・アプリケーションをダウンロードしてください:

Loan

zipファイルの解凍後、loan ディレクトリーが作成されます。そのディレクトリー内にある build.properties ファイルを開き、次の例で示されているように、あなたの環境に適合するプロパティーを編集してください:

build.properties

```
## Set the Geronimo 2.0 home here
geronimo.home=<geronimo_home>

## Set XDoclet 1.2.3 Home
xdoclet.home=<xdoclet_home>
```

ビルド・プロセスを開始する前に、config ディレクトリーの build.properties にある geronimo.home や xdoclet.home エントリーに正しいパスを設定してください。

コマンド・プロンプトまたはシェルから、loan ディレクトリーへ移動し、ant jboss を実行してください。こうすると、ear ファイルがビルドされ、releases/jboss ディレクトリーにそれを直接配置します。

サンプル・アプリケーションのデプロイ

サンプル・アプリケーションをデプロイするには、loan/releases/jboss フォルダーへ作成された Loan.ear を <jboss_home>/server/<your_server_name>/deploy へコピーしてください。

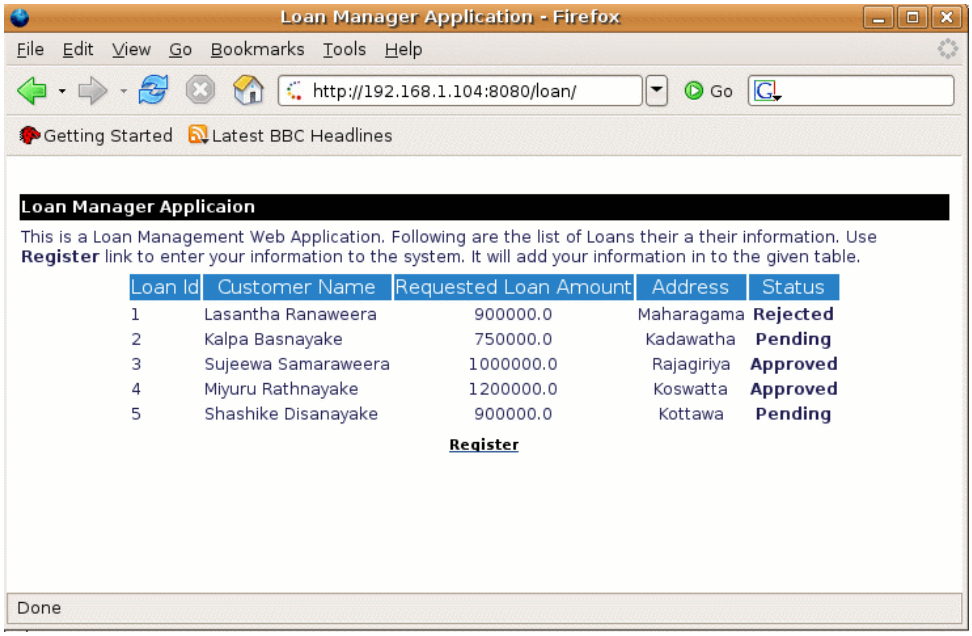
JBoss がすでに始動していれば、アプリケーションを自動的にデプロイし、始動します。そうでなければ、次回の始動時に、当該アプリケーションはデプロイ、始動されます。

サンプル・アプリケーションのテスト

アプリケーションをテスト実行するには、Web ブラウザーを開いて、次の URL へアクセスしてください:

http://localhost:8080/loan

ここへアクセスすると、ローンと最新ステータスの一覧を含む、ローン管理のメイン画面へ遷移します。Register リンクをクリックすると、ローン情報のデータベースに対して、ローンを追加する前にいくつかの項目を入力するローンの登録フォームへ遷移します。



ローンのステータスを変更するには、管理者は、Web アプリケーションからローンIDを検索し、以下のようにして、それをクライアント・アプリケーションに指定してあげる必要があります。
java -jar LoanStatusChanger.jar <ローンID> <ステータス>
ステータスは、次の値にしたがって変更します。以下に示すのは、適切なローンのステータスです。

1. 0 - 保留
2. 1 - 受付済
3. Other - 拒否

> ⚠ 上記のコマンドを実行する前に <jboss-home>/client/jbossall-client.jar ファイルがクラスパスに追加されていることを確認してください。

トップへ戻る

Geronimo 環境

以下の URL より Geronimo をダウンロードし、インストールしてください。

http://geronimo.apache.org/downloads.html

リリース・ノートでは、システム要件、インストール方法、Geronimo の始動方法に関する説明が記述されています。これ以降、Geronimo インストール・ディレクトリーのことを <geronimo_home> と記述します。

> TCP/IP ポートの競合
>
> もし同一マシン上で JBoss と Geronimo を稼動させたい場合には、少なくとも1つの初期サービス・ポートを変更してください。

トップへ戻る

段階的な移行

ローン管理サンプル・アプリケーションをビルドすると、Ant はサンプル・アプリケーションがすでに提供している JBoss の jboss.xml および Geronimo の openejb-jar.xml の両ディスクリプターをパッケージングします。これらのファイルは loan/config ディレクトリーに配置されています。

以下の例は、JBoss デプロイメント・ディスクリプターを示しています。

jboss.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 4.0//EN" "http://www.jboss.org/j2ee/dtd/jboss_4_0.dtd">

<jboss>

<enterprise-beans>

<entity>
<ejb-name>Loan</ejb-name>
<local-jndi-name>Loan</local-jndi-name>
<method-attributes>
</method-attributes>
</entity>

<session>
<ejb-name>LoanManager</ejb-name>
<jndi-name>org.apache.geronimo.samples.loan.ejb.LoanManager</jndi-name>
<local-jndi-name>LoanManager</local-jndi-name>
<method-attributes>
</method-attributes>
</session>

</enterprise-beans>

<resource-managers>
</resource-managers>

</jboss>
```

以下の例で示している Geronimo デプロイメント・プランの内容と比較してみてください。

openejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<openejb-jar xmlns="http://www.openejb.org/xml/ns/openejb-jar-2.1">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2">
<dep:moduleId>
<dep:groupId>org.apache.geronimo.samples</dep:groupId>
<dep:artifactId>LoanManagerEJB</dep:artifactId>
<dep:version>1.0</dep:version>
<dep:type>car</dep:type>
</dep:moduleId>
<dep:dependencies/>
```

```
<dep:hidden-classes/>
<dep:non-overridable-classes/>
</dep:environment>
<enterprise-beans>
<session>
<ejb-name>LoanManager</ejb-name>
<ejb-ref>
<ref-name>ejb/LoanLocal</ref-name>
<ejb-link>Loan</ejb-link>
</ejb-ref>
</session>
<entity>
<ejb-name>Loan</ejb-name>
<resource-ref>
<ref-name>jdbc/LoanDataSource</ref-name>
<resource-link>SystemDatasource</resource-link>
</resource-ref>
</entity>
</enterprise-beans>
</openejb-jar>
```

明確に示されている最初の違いは、Geronimo 固有の構成が JBoss の構成よりも情報が追加されていることです。Geronimo の構成ファイルの一部は Maven 2 のビルド・スクリプトとかなり酷似しています。これらの構成ファイルには EJB の情報を持っています。JBoss では EJB のリンクに対応する ローカル JNDI 名を使用し、Geronimo では EJB 名を直接使用します。上記の違いに加え、openejb-jar.xml ファイルは jboss.xml ファイルよりも EJB リファレンス情報をはっきりと指定しています。

Web アーカイブ関連の構成ファイルにおいても、少々の違があります。

jboss-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>


<jboss-web>

<ejb-local-ref>
<ejb-ref-name>ejb/LoanManagerLocal</ejb-ref-name>
<local-jndi-name>LoanManager</local-jndi-name>
</ejb-local-ref>
</jboss-web>
```

geronimo-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1" xmlns:naming="http://
geronimo.apache.org/xml/ns/naming-1.1">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.1">
<dep:moduleId>
<dep:groupId>org.apache.geronimo.samples</dep:groupId>
<dep:artifactId>LoanManagerWeb</dep:artifactId>
<dep:version>1.0</dep:version>
<dep:type>car</dep:type>
</dep:moduleId>
<dep:dependencies/>
<dep:hidden-classes/>
<dep:non-overridable-classes/>
</dep:environment>

<naming:ejb-local-ref>
<naming:ref-name>ejb/LoanManagerLocal</naming:ref-name>
<naming:ejb-link>LoanManager</naming:ejb-link>
</naming:ejb-local-ref>

</web-app>
```

jboss-web.xml は、上記で指定された JNDI 名を使用している EJB と対応付けられ、geronimo-web.xml は EJB 名を直接使用します。互いにマッピングされているリファレンス名は、サーブレットから EJB を参照するために使用されます。WAR ファイルにあ

る web.xml ファイルには、各 EJB リファレンス名に関するより詳細な情報を含んでいます。それは、このアプリケーションにおける Geronimo および JBoss の特徴に対する共通点です。

サンプル・アプリケーションのビルド

loan ディレクトリーから以下のコマンドを実行し、移行されたサンプル・アプリケーションの Geronimo バージョンをビルドしてください。

ant geronimo

loan/releases/geronimo フォルダーに Loan.ear が作成されます。

移行されたアプリケーションのデプロイ

移行されたローン管理サンプル・アプリケーションをデプロイするには、Geronimo サーバーが始動され実行中であること、さらに、ユーザーがデータベースを操作しなければならないことを確認してください。その後、サンプル・アプリケーションをデプロイしてください。

データベースの操作

このサンプル・アプリケーションでは、アプリケーション専用データを保持するための Geronimo システム・データベースをデフォルトで使用します。
Geronimo コンソールより、以下の手順にしたがってください。

1. 左側のコンソール・ナビゲーションから DB Manager リンクを選択します。
2. Use DB 欄に対して SystemDatabase を選択します。
3. テキスト・エディターから loan/config ディレクトリーにある db.sql を開き、下側にある Geronimo Specific Database SQL 欄に当該ファイルの中身をコピーします。
4. 指定された上記の SQL コマンドの中身をテキストエリアに貼り付けし、Run SQL ボタンを押下します。

サンプル・アプリケーションのデプロイ

ブラウザーより Geronimo コンソールを開き、以下の手順にしたがってください:

1. Console Navigation パネルから Deploy New まで下へスクロールさせます。
2. loan/releases/geronimo フォルダーから loan.ear を Archive 入力欄に 指定します。
3. Install ボタンを押下し、サーバーに対してアプリケーションをデプロイします。

サンプル・アプリケーションのテスト実行

アプリケーションをテスト実行するには、Web ブラウザーを開いて、以下の URL へアクセスしてください。

http://localhost:8080/loan

ローンのステータスを変更するには、ユーザーは Web アプリケーションより ローンID を検索し、それをクライアント・アプリケーションへ下記のように指定する必要があります。

java -jar LoanStatusChanger.jar <ローンId> <ステータス>


上記のコマンドを実行する前に、以下の JAR ファイルがクラスパスに追加されていることを確認してください。

1. geronimo-kernel-2.0.x.jar
2. geronimo-j2ee_connector_1.5_spec-1.x.x.jar
3. geronimo-j2ee_management_1.1_spec-1.x.jar
4. geronimo-security-2.0.x.jar
5. cglib-nodep-2.1_3.jar
6. openejb-core-3.0.jar

トップへ戻る

まとめ

この文書は、BMP エンティティー・ビーンを使用しているサンプル・アプリケーションの JBoss v4.2.1 から Apache Geronimo への移行方法を記述しています。アプリケーションのビルド、配置、稼動の段階的な説明にしたがって Geronimo 環境へ移行してください。

次のリストは、このサンプル・アプリケーションの移行を通した、主な相違点をまとめています。

- JBoss の EJB jar ファイルをデプロイするために、デプロイ先のディレクトリーへ構成ファイルをコピーしておく必要がありますが、Geronimo では、デプロイヤー・ツール、コンソール、ホット・デプロイメント・ディレクトリーのどちらでも利用可能です。
- JBoss および Geronbimo の EJB jar ファイルのデプロイメント・プランの内容は、Maven 2 ビルド・ファイルとほぼ同様な Geronimo の開始部分を除いて、ほとんど酷似しています。

4.3. JBoss to Geronimo - EJB-CMP Migration

This page last changed on 4 20, 2008 by JAGUG.

4.4. JBoss to Geronimo - EJB-MDB Migration

This page last changed on 4 20, 2008 by JAGUG.

4.5. JBoss to Geronimo - EJB-セッションビーンの移行

This page last changed on 4 21, 2008 by JAGUG.

典型的なJ2EEアプリケーションはエンタープライズJavaビーン、つまりEJBを含んでいることがあります。このビーンはアプリケーションのビジネス・ロジックや最新のビジネス・データを含んでいます。ビジネス・ロジックとビジネス・データを保管するには通常のJavaオブジェクトを使うこともできますが、EJBを使用すれば、単純なJavaオブジェクトを使用した場合のスケーラビリティ、ライフサイクルの管理、状態の管理、などの問題を解消することができます。

この文書では、EJBの1種類、セッションEJBについて述べています。セッションEJBはビジネス・プロセス・フロー（またはそれに似たアプリケーションの考え方）をマッピングするのに有用です。セッションEJBにはステートレス、ステートフルの2種類があります。この記事では両方のセッション・ビーンに言及し、と個々のシナリオにおいてそれらを使う方法をご紹介します。

EJBはクライアントと対話をします。対話は基本的にEJBとクライアントの間で行われ、この対話はクライアントからEJBに対するメソッド呼び出しにより行われます。ステートフル・セッション・ビーンはあるクライアントから見た状態を保持しています。反対に、ステートレス・セッション・ビーンはメソッド間での会話の状態を保持しません。別の言葉で言えば、ただ一度のメソッド呼び出しの間だけで会話の状態を保持することを期待されている、とも言えます。

この文章は以下のセクションで構成されています。

EJB実装の分析

EJBの実装は各々のベンダー毎に異なっています。このセクションの目的はJBossとApache Geronimo間でのセッション・ビーンの機能毎の比較を提供し、両者の違いを明確にしたうえで移行の前に計画を立てられるようにすることです。

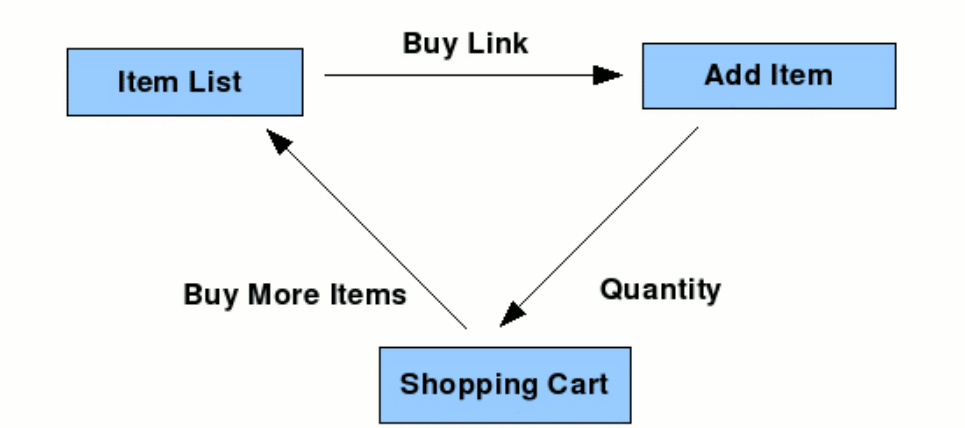
機能	JBoss v4.0.5	Apache Geronimo (OpenEJB)
ステートフルおよびステートレス・セッション・ビーン	サポートあり	サポートあり
BMP (Bean Managed Persistence) エンティティ・ビーン	サポートあり	サポートあり
CMP (Container Managed Persistence) エンティティ・ビーン	サポートあり	サポートあり
メッセージ・ドリブン・ビーン (MDBs)	サポートあり	サポートあり
RMI-IIOP または JAXRPCを使用したインター・オペラビリティ	サポートあり	サポートあり
ステートレス・セッション・ビーンまたはMDBをWebサービスとして公開	サポートあり	サポートあり
Webサービス経由でのメッセージの送受信	サポートあり	サポートあり
EJB 及び JMXベースのWebサービスの容易なプロビジョニングとホット・デプロイメント	サポートあり	サポートあり
外部のCORBAオブジェクトからのEJBへの容易なアクセス	サポートあり	サポートあり

Back to Top

サンプル・アプリケーション

今回のサンプルは単純なeコマースのWebサイトで、コンピューター関係の商品を小売、および卸売りマーケット向けに販売しています。商品を卸売り向けの値段で買うためには、顧客は最低限の決められた数量を購入する必要があります。そうでない場合は、顧客は一般小売向けの値段で商品を購入することになります。このアプリケーションの利用者はショッピング・カートに商品を追加します。商品の割引率を計算するビジネス・ワークフローにはステートレス・セッション・ビーンが使われ、ショッピング・カートの部分にはステートフル・セッション・ビーンが使われています。

下の図はアプリケーションの流れを示しています。



このアプリケーションの顧客は、商品リストのページに直接ログオンします。このページで、在庫のある商品の詳細をすべて見ることができます。顧客は、在庫のある商品を買うため、各々の商品についている「購入」リンクを使います。結果、「商品をカートに入れる」ページに飛びます。購入するなら、買いたい商品の数を入力します。もし購入数が最低購入数を超えているなら、ディスカウント価格が適用され、顧客のショッピング・カートに商品が追加されます。ショッピング・カートWebページはカート登録済みの商品を表示しますが、カートから取り除くこともできます。このアプリケーションでは同じ商品を二度購入することはできません。

アプリケーションのクラスとJSPページ

- org.apache.geronimo.samples.computer.dto
  - ItemDTO - Web層とEJB層の間で商品に関する情報を転送するデータ転送オブジェクト
  - TransactionDTO - Web層とEJB層の間でショッピング・カートに関する情報を転送するデータ転送オブジェクト
- org.apache.geronimo.samples.computer.ejb
  - ItemServiceBean - 商品に関するワークフロー・アクティビティを処理するステートレス・セッション・ビーン
  - ShoppingCartBean - ショッピング・カートに関するワークフロー・アクティビティを処理するステートレス・セッション・ビーン
- org.apache.geronimo.samples.computer.web
  - ItemServiceDispatchServlet - フロント・エンドからWeb層へ商品サービス関連のアクティビティを振り分けるサーブレット
  - ShoppingCartDispatchServlet - フロント・エンドからWeb層へショッピング・カート関連のアクティビティを振り分けるサーブレット

このeコマース・サンプル・アプリケーションは以下のJSPページも含んでいます。

- buy_item.jsp - 顧客の購入した商品の数量を受け入れます
- error.jsp - アプリケーションのエラー状況を表示します
- index.jsp - アプリケーションの商品リストへ転送します
- list_items.jsp - 在庫のある商品を表示します
- shopping_cart.jsp - ショッピング・カートに追加された買い物のリストを表示します

利用したツール

コンピューター・アクセサリーの販売アプリケーションを開発・ビルドするために利用するツールは：

Eclipse

サンプル・アプリケーションの開発にはEclipse統合開発環境を使用しました。これは非常にパワフルで有名なオープンソースの開発ツールです。JBoss と Geronimo用の統合プラグインが利用可能です。Eclipseは下記のURLからダウンロードできます。
http://www.eclipse.org

Apache Ant

AntはピュアJavaのビルド・ツールです。今回はwarファイルをビルドし、オンラインでの仲介アプリケーション用のデータベースを操作する際にAntを用いました。Antは下記のURLからダウンロードできます。
http://ant.apache.org

Back to Top

JBoss環境

このセクションでは、JBoss環境がどのような方法で、どの場所に導入されたかを示しますので、このシナリオを貴方の環境に読み替えてください。この移行シナリオではJBoss v4.0.5を使った点にご留意ください。

JBossのインストール、構成、管理の詳細な方法については製品のドキュメントに記載されています。製品のWebサイトで最新のドキュメントをチェックしてください。

以下のリストがサンプル・アプリケーションをデプロイする際の出発点として、初めの環境のインストール・構成を完了する際に必要となるタスクの概要のハイライトです。

1. 製品ドキュメントのガイドに従ってJBoss v4をダウンロードし、インストールします。以降、導入ディレクトリーは <jboss_home> と記載します。
2. デフォルトのJBoss v4アプリケーション・サーバーのコピーを作成します。<jboss_home>¥server¥default 以下の内容を <jboss_home>¥server¥<your_server_name> の下へサブディレクトリーも含め、すべてコピーします。
3. <jboss_home>¥bin ディレクトリーから run.sh -c <貴方のサーバーの名前> コマンドで新しいサーバーを開始します。
4. サーバーが開始したら、WebブラウザーでURL: http://localhost:8080 を指定してJBossが稼動していることを確認します。
   JBossの「Welcome」画面が表示され、JBossコンソールにアクセスできるはずです。
5. アプリケーション・サーバーが開始して稼動したら、次のステップとして、サンプル・アプリケーションに必要な残りの前提ソフトウエアをインストール・構成します。

前提ソフトウエアのインストールと構成

この記事に含まれているサンプル・アプリケーションをビルド・実行するには、Antビルド・ツールをインストール・構成する必要があります。

Antの構成

前述の通り、オンライン仲介アプリケーションのバイナリーをビルドするためにApache Antを使っています。もし貴方がAntをインストールしていなければ、これが丁度よい機会です。インストール後、システムのpath変数に <ant_home>¥bin ディレクトリーが追加されていることを確認してください。

Apache Antは以下のURLからダウンロードできます。
http://ant.apache.org

XDocletの構成

構成ファイルを生成するためのビルド・ツールにはXDocletが使われています。これはオープンソースのコード生成エンジンです。XDoctletにより、Javaで属性(Attribute)志向のプログラミングが可能になります。要するに、貴方のソースコードにメタ・データ（属性）を記述することにより、貴方のコードに様々な意味を付加できる、ということです。これな特殊なJavaDocタグにより行われます。
XDocletは元々はEJBを作成するためのツールですが、より汎用的なコード生成エンジンに進化しました。XDocletはコア部分と、日々拡張されているいくつかのモジュールから構成されています。もし新しい種類のコンポーネントのための新しいモジュールが必要になったら、それを書くのは比較的簡単なことです。

http://xdoclet.sourceforge.net/xdoclet/index.html

XDocletの最新バージョンを展開して、build.properties ファイル中に xdoclet.home パラメーターをセットしてください。

サンプル・アプリケーションのビルド

この記事のコンピューター・アクセサリー販売アプリケーションは貴方がアプリケーションをビルドする際に利用できるAntスクリプトを提供しています。以下のリンクからコンピューター・アクセサリー販売アプリケーションをダウンロードしてください。

Computer

zipファイルを展開すると computer ディレクトリーが作成されます。そのディレクトリーにあるbuild.properties ファイルを開いて、以下の例のように、貴方の環境に合わせてプロパティを編集してください。

build.properties

```
## Set the Geronimo 1.1 home here
geronimo.home=<geronimo_home>

## Set XDoclet 1.2.3 Home
xdoclet.home=<xdoclet_home>
```

ビルド・プロセスを開始する前に config ディレクトリーの build.properties ファイル中の geronimo.home と xdoclet.home に正しいパスを設定してください。
コマンド・プロンプトかシェルで computer ディレクトリーに移動して ant jboss を実行してください。この操作でearファイルが作成され、release/jboss ディレクトリー上に配置されます。

サンプル・アプリケーションのデプロイ

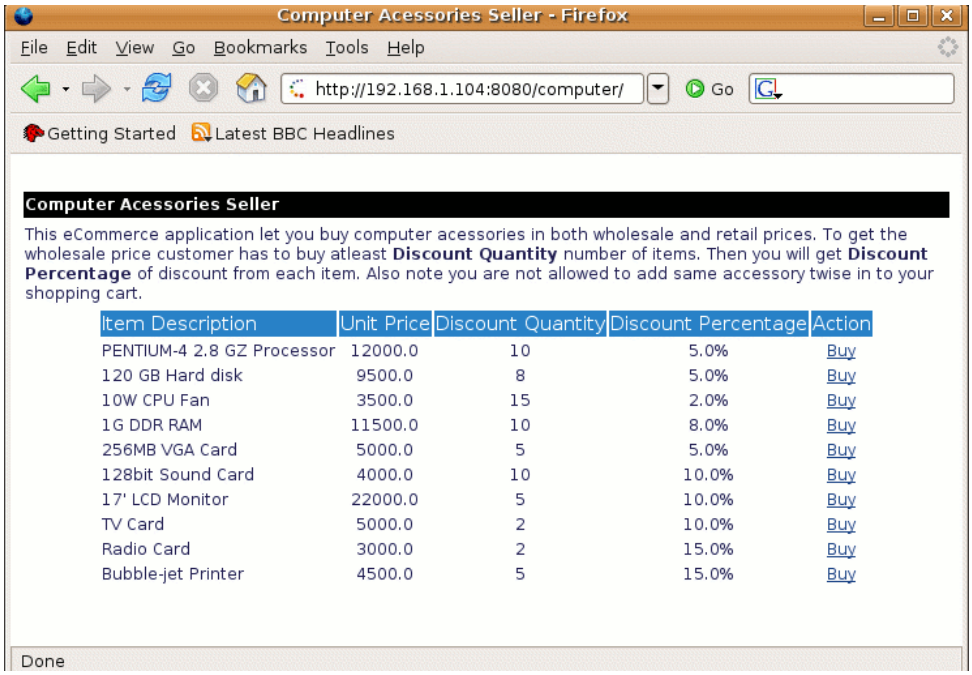
サンプル・アプリケーションをデプロイする際は、computer.ear を computer/releases/jboss フォルダーから <jboss_home>/server/<your_server_name>/deploy フォルダーにコピーするだけです。

既にJBossが開始しているなら、自動的にデプロイされ、アプリケーションが開始します。JBossが開始していなければ、次回の始動時にデプロイされ、開始されます。

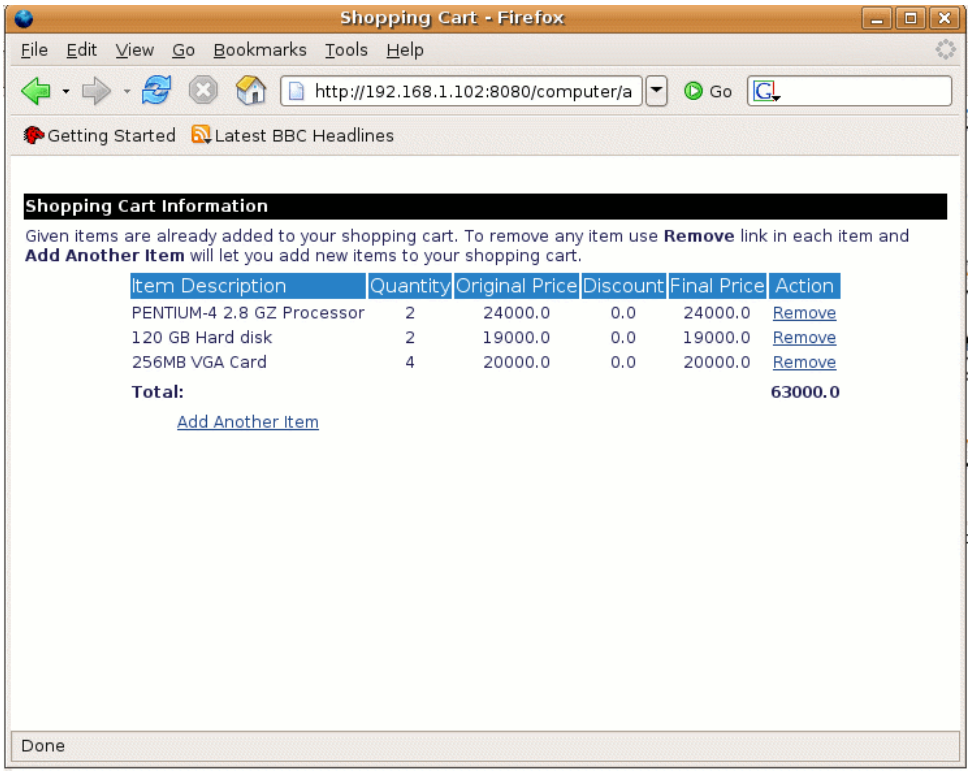
サンプル・アプリケーションのテスト

アプリケーションをテストするには、Webブラウザーを開いて以下のURLへアクセスしてください。
http://localhost:8080/computer

コンピュータ・アクセサリー販売アプリケーションの商品リスト・ページが立ち上がります。このアプリケーションでコンピュータのアクセサリーを購入できます。このアプリケーションは既に構成され稼動しています。



以下がサンプル・アプリケーションのショッピング・カートのデモです。



Back to Top

Geronimo環境

以下のURLからGeronimoをダウンロードしてインストールしてください。

http://geronimo.apache.org/downloads.html

URLにあるリリース・ノートには、システム要件、Geronimoのインストールや構成のための方法が正確に記載されています。以降、当記事では以降、Geronimoの導入ディレクトリーを <geronimo_home> と表記します。

TCP/IPポートの競合

もしJBossとGeronimoを同じマシンで動かそうとしているなら、少なくともどちらかひとつのサーバーのデフォルトのポート番号を変更することを検討する必要があります。

Back to Top

ステップ・バイ・ステップの移行

コンピュータ・アクセサリー販売のサンプル・アプリケーションをビルドすると、Antはサンプル・アプリケーションで提供されたようにJBoss用には　jboss.xml 、Geronimo用には openejb-jar.xml というデプロイメント記述子をパッケージします。これらのファイルは computer/config ディレクトリーに配置されています。

以下のサンプルはJBossのデプロイメント記述子です。

jboss.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<jboss>

<enterprise-beans>

<session>
<ejb-name>ShoppingCart</ejb-name>
<local-jndi-name>ShoppingCart</local-jndi-name>

<method-attributes>
</method-attributes>
</session>
<session>
<ejb-name>ItemService</ejb-name>
<local-jndi-name>ItemService</local-jndi-name>

<method-attributes>
</method-attributes>
</session>

</enterprise-beans>

<resource-managers>
</resource-managers>

</jboss>
```

これらを以下に示されているGeronimo 用のデプロイメント記述子と比べてみてください。

openejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<openejb-jar xmlns="http://www.openejb.org/xml/ns/openejb-jar-2.1">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2">
<dep:moduleId>
<dep:groupId>org.apache.geronimo.samples</dep:groupId>
<dep:artifactId>ComputerEJB</dep:artifactId>
<dep:version>1.0</dep:version>
<dep:type>car</dep:type>
</dep:moduleId>
<dep:dependencies/>
<dep:hidden-classes/>
<dep:non-overridable-classes/>
</dep:environment>
<enterprise-beans>
<session>
```

```
<ejb-name>ShoppingCart</ejb-name>
<ejb-ref>
<ref-name>ejb/ItemServiceLocal</ref-name>
<ejb-link>ItemService</ejb-link>
</ejb-ref>
</session>
<session>
<ejb-name>ItemService</ejb-name>
</session>
</enterprise-beans>
</openejb-jar>
```

初めに気づく違いはGeronimoの固有構成はJBossよりも多くの追加情報を含んでいるということでしょう。Geronimoの構成ファイルのある部分はMaven2のビルド・スクリプトと非常によく似ています。いずれの構成ファイルのEJBに関する情報を含んでいます。JBossはEJBとの関連付けにローカルJNDI名を使いますが、GeronimoではEJBの名前を直接指定します。上記の違い以外にも openejb-jar.xml ファイルは jboss.xml ファイルよりも明示的にEJBリファレンス情報を指定しています。

webアーカイブ・ファイル関連の構成ファイルを見ると、更にいくつかの違いがわかります。

jboss-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web>

<!-- EJB Local References -->
<ejb-local-ref>
<ejb-ref-name>ejb/ItemServiceLocal</ejb-ref-name>
<local-jndi-name>ItemService</local-jndi-name>
</ejb-local-ref>
<ejb-local-ref>
<ejb-ref-name>ejb/ShoppingCartLocal</ejb-ref-name>
<local-jndi-name>ShoppingCart</local-jndi-name>
</ejb-local-ref>
</jboss-web>
```

geronimo-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1" xmlns:naming="http://
geronimo.apache.org/xml/ns/naming-1.1">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.1">
<dep:moduleId>
<dep:groupId>org.apache.geronimo.samples</dep:groupId>
<dep:artifactId>ComputerWeb</dep:artifactId>
<dep:version>1.0</dep:version>
<dep:type>car</dep:type>
</dep:moduleId>
<dep:dependencies/>
<dep:hidden-classes/>
<dep:non-overridable-classes/>
</dep:environment>

<naming:ejb-local-ref>
<naming:ref-name>ejb/ItemServiceLocal</naming:ref-name>
<naming:ejb-link>ItemService</naming:ejb-link>
</naming:ejb-local-ref>

<naming:ejb-local-ref>
<naming:ref-name>ejb/ShoppingCartLocal</naming:ref-name>
<naming:ejb-link>ShoppingCart</naming:ejb-link>
</naming:ejb-local-ref>
</web-app>
```

上記のように jboss-web.xml ファイルはEJBをJNDI名とマップしていますが、geronimo-web.xml ではEJBの名前を直接指定します。それぞれのマッピングでの参照名はサーブレットからEJBを使用する場合に使われるものです。WARファイル中の web.xml は個々のEJB参照名についてより多くの情報を含んでいますが、このアプリケーションではGeronimo版 もJBoss版も共通のものです。

サンプル・アプリケーションのビルド

computer ディレクトリーから以下のコマンドを実行し、移行されたGeronimo版のサンプル・アプリケーションをビルドします。

ant geronimo

結果、computer/releases/geronimo フォルダーに computer.ear が作成されます。

移行されたアプリケーションのデプロイ

移行後のコンピューター・アクセサリー販売アプリケーションをデプロイするために、Geronimoサーバーが起動していることを確認してください。
貴方のブラウザーでGeronimoコンソールを開いて、以下の手順に沿ってください。

1. コンソール・ナビゲーションからDeploy New を選択します。
2. Archive 入力ボックスに computer/releases/geronimo フォルダーの computer.ear を指定します。
3. Install ボタンを押してアプリケーションをサーバーにデプロイします。

アプリケーションがデプロイされたら、Webブラウザーを開いて以下のURLを入力してください。
http://localhost:8080/computer

Back to Top

サマリー

この記事では、セッション・ビーンを使ったサンプル・アプリケーションをJBoss v4.0.5からGeronimoアプリケーション・サーバーに移行する方法を示しました。アプリケーションをビルド、デプロイ、実行してからGeronimo環境へ移行する手順を順番に行いました。

以下のリストはサンプル・アプリケーションの移行の過程で見つかった2つの大きな違いを要約しています。

- JBossへEJB jarファイルをデプロイするにはdeployディレクトリーにコピーするだけですが、Geronimoではデプロイヤー・ツール、コンソール、またはホット・デプロイメント・ディレクトリーを利用できます。
- JBoss と Geronimoのデプロイメント・プランは非常に似通っていますが、Geronimoでは初めの部分がMaven2のビルド・ファイルにとてもよく似ています。

4.6. JBoss to Geronimo - Hibernate の移行

This page last changed on 4 21, 2008 by JAGUG.

この文章は、Hibernate 3.2 を ORM ツールとして利用するアプリケーションを JBoss アプリケーション・サーバー 4.2.1 から Apache Geronimo 2.0 へ移行する手助けとなります。

Hibernate は強力な、高性能なオブジェクト・リレーショナルの永続化とクエリー・サービスです。データベースのフィールドに結びつけられた属性の getter と setter メソッド持つ永続化（POJO）クラスの開発の助けになります。関連・継承・ポリモーフィズム・コンポジション・コレ クションなどのオブジェクト指向の特徴に従うことになります。Hibernate は、ネイティブな SQL、オブジェクト指向検索条件、例示 API などと同じように、移行可能な SQL 表現（HQL）を用いて検索を表現できます。

基本的には、Hibernate は Java クラスとデータベースの表を結びつけます。また、データ検索や検索機構によって、開発時間の短縮につながります。自然と、Java 中間層におけるオブジェクト指向のコード開発ができます。Hibernate のユニークな特徴は、MySQL、Oracle または DB2 など、どのようなデータベースにもアプリケーションが切り替えることができる透過的永続化機構です。Hibernate は Java Swing アプリケーション、Java サーブレット・アプリケーション、または EJB セッション・ビーンを利用する J2EE アプリケーションなどに利用できます。（今回は、Java サーブレット・アプリケーションを利用します）

移行の手順を明らかにするために、最初にサンプル・アプリケーションを JBoss にデプロイし、その後 Geronimo へ移行します。サンプル・アプリケーションは、オンライン取引アプリケーションです。JDBC の移行手順の説明で既に利用した物です。このアプリケーションでは永続化のために Hibernate を利用するように変更します。

この文章には以下のセクションがあります。

Hibernate 実装の解析

Hibernate は以下のサービスを提供します。

- コネクション管理
- トランザクション管理
- オブジェクト・リレーションのマッピング

また、とても柔軟であり、さまざまなサービスと組み合わせて利用できます。JBoss、Geronimo ともトランザクション管理とコネクション管理機能を持っているので、この文章では hibernate を O/R マッピング機能のみ利用できるように構成します。一般的に、Hibernate はアプリケーション・サーバーと一緒に利用するように構成します。Hibernate はコネクション・プールの生成と環境の設定に構成ファイルである `hibernate.cfg.xml` を必要とします。このファイルにはデータベース・ドライバー、接続 url、SQL 方言（これは利用している RDBMS の仕様）、ユーザー名、パスワード、プールの大きさなどのパラメーターを含みます。また、*.hbm.xml というマッピング用ファイルの場所の定義も含みます。マッピング用ファイルはデータベースの表のフィールドと永続化クラスの属性とを結びつけるものです。

これらのプロパティは Apache Geronimo v2.0 を含めたアプリケーション・サーバーで一般的なものです。

しかし、JBoss（さらに言えば、Hibernate MBean）は、2つのさらなるデプロイメント機構を持っています。

一つ目は Hibernate アーカイブ（HAR ファイル）です。これは Hibernate のすべてのクラスとマッピング・ファイルが含まれる特別なアーカイブである HAR ファイルです。JBoss はこのアーカイブを EAR や WAR ファイルと同じ方法でデプロイします。

もうひとつは、他のアプリケーションのクラスといっしょに、Hibernate のすべてのクラスとマッピング・ファイルを、例えば EAR の中に単にいれることです。Hibernate MBean は個別に構成され、マッピング・ファイルがアプリケーションのすべての JAR を探しかたを指定します。両方のデプロイメント機構とも、手作業で構成することなく、通常必要となるような設定コードを記述することなく、Hibernate オブジェクトをアプリケーションに追加することができます。

構造的には、HAR ファイルは JBoss サービス・アーカイブ（SAR ファイル）に似ています。HAR ファイルは Hibernate クラス・ファイルとマッピング・ファイル（*.hbm.xml）を、Hibernate アプリケーションが生成する必要がある Hibernate MBean の構成の定義を持つ標準の `jboss-service.xml` ファイルと一緒に保持します。最新の JBoss 製品では、`hibernate-service.xml` という名前にリネームされていますが、同じ構造と目的を持っています。

Hibernate アーカイブはトップレベルのパッケージとして、または EAR ファイルのコンポーネントとしてデプロイすることができます。Hbernate アーカイブは標準の J2EE デプロイメントの形式ではないので、これらをコンテキスト内で利用できるように EAR ファイルに `jboss-app.xml` ファイルを記述する必要があります。

以下の表はこれらアプリケーション・サーバーの機能ごとの比較です。

Feature	Apache Geronimo v2.0	JBoss v4.2.1
コンテナー管理データソース	サポートされています。Hibernate は与えられた JNDI 名のデータソースを利用することができます。アプリケーションと同じスレッドで実行されるからです。	サポートされています。Hibernate は与えられた JNDI 名のデータソースを利用することができます。
JNDI 自動バインディング	サポートされていません。	サポートされています。プロパティが設定されるとセッション・ファクトリーが JNDI コンテキストに結びつけられます。
JTA セッション・バインディング	この機能は導入直後そのままの状態ではサポートされていません。Geronimo トランザクション・マネージャーが利用できるようにルックアップを記述する必要があります。	導入直後からサポートされています。Hibernate は JBoss トランザクション・マネージャー向けのルックアップ・クラスを持っています。
JMX デプロイメント	導入直後そのままの状態ではサポートされていません。GBean と Hibernate コネクション・プロバイダー・クラスを記述すれば実装できます。	サポートされています。Hibernate は JBoss にデプロイできる `org.hibernate.jmx.HibernateService` を持っています。
Hibernate アーカイブ (HAR)	サポートされていません。Hibernate クラスは J2EE アーカイブの一部としてデプロイされます。	サポートされています。HAR は構成とマッピング・ファイルが含まれ、他のサーバーへのデプロイをサポートできるようになります。
キャッシング	hibernate のキャッシング機構を利用できます。	hibernate のキャッシング機構を利用できます。JBoss キャッシュもサポートされています。
セッション管理	サポートされていません。手動でセッションを開く必要があります。	Hibernate セッション・ライフサイクルは自動的に JTA トランザクションのスコープに結びつけることができます。手動でセッションを開いたり閉じたりする必要がなく、これは JBoss EBJ インターセプターが行います。
Hibernate マッピング・ファイル	Hibernate マッピング・ファイルの場所を指定する必要があります。	HAR デプロイを利用する場合は JBoss は自動的に Hibernate マッピング・ファイルを探しだします。

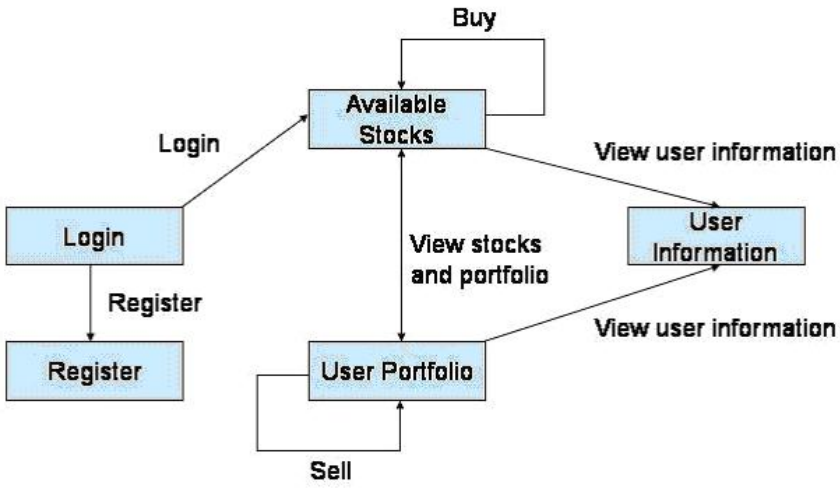
Back to Top

サンプル・アプリケーション

この文章には、JBoss から Geronimo へアプリケーションの移行をデモンストレーションするサンプル・アプリケーションがあります。名前は Online Brokerage です。オンライン取引でユーザーが株の売買を行うシナリオです。このアプリケーションには以下の5つのページがあります。

- Login ページ
- Registration ページ
- User Information ページ
- Available Stocks ページ
- User Portfolio ページ

以下の図は、アプリケーションのフローです。



最初に、ユーザーは Login ページに接続します。ログインページからユーザーはユーザー名とパスワードを入力します。ユーザー名またはパスワードが不正の場合は、アプリケーションは エラーメッセージを表示し、そのログインを拒否します。ユーザー名とパスワードが正確であれば、すべての株についてその時点の株価を見ることができる Available Stocks ページを開きます。
ユーザーは口座にある利用可能な金額に応じて、購入したい株を Buy ボタンをクリックすることで選択します。取引が無事完了すると、Available Stocks ページ が表示され、必要ならばさらに株が購入できます。株の購入に資金が不足していたら、アプリケーションはエラーとなり取引を進めません。エラーメッセージが Available Stocks ページの上部に表示されます。このページには User Info ボタンがあります。これをクリックすると、User Info ページが表示され、ユーザーの詳細情報が表示されます。
Available Stocks ページには、ユーザーが所有するすべての株を確認できる Portfolio へのリンクがあります。このページでは、ユーザーは株と株数を選んで売却することができます。また、User Cash 欄に現在の利用可能な現金の学が表示されています。所有している数より多くの株を売ろうとすると、アプリケーションはエラーになります。エラーメッセージ が同じページ上に表示されます。売却が成功すると、売却金額がユーザーの現金に加算されます。
quantity テキストボックスにはユーザーが持っている特定の会社の株の数量が表示されています。Quantity to Sell 欄では、その会社の株の売りたい数量を入力することができます。売却と購入にはラジオボタンをチェックします。このことは数値を入力した後になされます。 もし quantity to sell テキストボックスが入力されていないか、ラジオボタンがチェックされずに sell ボタンを押すと、必須入力欄が空であるという JavaScript のアラートが表示されます。数量欄に数値以外の文字が入っていると、別のアラートが表示されます。この動き方は、Available Stocks Page と同じようなものです。
新規ユーザーは、login ページで Register ボタンをクリックすることで登録できます。Registration ページでは、ユーザーID、ユーザー名、パスワード、住所と利用可能現金を入力することができます。
Back to Top

アプリケーション・クラスと JSP ページ

オンライン取引サンプル・アプリケーションには、以下のパッケージが含まれます。

- com.dev.trade.bo
  - Stock - 会社の株を表す
  - User - ユーザーを表す
  - UserStock - ユーザーの所有する株を表す
- com.dev.trade.dao
  - TradeDAO - すべてのデータベース接続メソッドを含む
- com.dev.trade.exception
  - DBException - すべてのデータベースの例外を発生させるカスタム例外
- com.dev.trade.servlet
  - TradeDispatcherServlet - すべてのリクエストを、要求されたデータベース機能を実行した後に JSP へ送る処理

オンライン取引には、以下の JSP ページを含みます。

- login.jsp - アプリケーションのログインページ
- error.jsp - アプリケーションのデフォルトのエラーページ
- register.jsp - ユーザー登録のページ
- stocks.jsp - ユーザーが株を購入できる Available Stocks ページ

- userstocks.jsp - ユーザーの保持する株を表示する portfolio ページで、ユーザはここで株を売却する

Back to Top

利用するツール

オンライン取引アプリケーションの開発とビルドのツールは以下の通りです。

Apache Ant

Ant はピュア Java のビルド・ツールです。war ファイルをビルドし、オンライン取引アプリケーション向けデータベースを作成するために利用されています。Ant は以下の URL からダウンロードできます。

http://ant.apache.org

Hibernate

この文章を書いている時点では Hibernate 3.2 が最新の利用可能バージョンです。以下の URL からダウンロードできます。
http://www.hibernate.org

Hibernate に関する他の文章は、以下の URL にあります。
http://hibernate.org/5.html
http://www.hibernate.org/hib_docs/v3/reference/en/html/tutorial.html

Hibernate をダウンロードし、インストールしてください。インストールしたディレクトリーを後ほど <hibernate_home> として参照します。

Back to Top

サンプル・データベース

このアプリケーションのデモンストレーションに利用するデータベースは MySQL です。サンプル・データベース名は adi です。STOCKS、USERS、USERSTOCKS の3つのテーブルも持ちます。各テーブルは以下のフィールドを持ちます。

Table Name	Fields
STOCKS	ID (PRIMARY KEY) NAME PRICE
USERS	USERID (PRIMARY KEY) NAME PASSWORD ADDRESS CASH
USERSTOCKS	ID (PRIMARY KEY) USERID (PRIMARY KEY) NAME PRICE QUANTITY

USERSTOCKS テーブルは、各ユーザが所有する株を保存するために利用されます。USER と STOCKS テーブルはユーザーと株の詳細情報を保存するために利用されます。これは単にサンプル・アプリケーションですので、ユーザーの持つ金額の量は、ユーザー登 録時にユーザー自身に入力してもらいます。

このデータベース生成用の DDL は db.sql です。<brokerage_homge>¥sql ディレクトリーにあります。

Back to Top

JBoss 環境

このセクションでは、サンプルの JBoss 環境がどのように、どこにインストールされるかを示すので、貴方はこのシナリオを貴方の実装に合わせることができます。

JBoss のインストール、構成と管理の詳細な説明については、JBoss のドキュメントにあります。JBoss のウェブ・サイトで最新のドキュメントをチェックしてください。

以下のリストはインストールと初期環境の構成を完了させ、サンプル・アプリケーションのデプロイの準備を完了させるまでに必要な一般的なタスクを示します。

1. 製品の説明書にしたがって、JBoss v4.2.1 をダウンロード、インストールしてください。以下、インストール・ディレクトリーを <jboss_home> として参照します。
2. デフォルトの JBoss v4.2.1 アプリケーション・サーバーのコピーを作ってください。<jboss_home>¥server¥default ディレクトリーを再帰的に <jboss_home>¥server¥<your_server_name> へコピーしてください。
3. <jboss_home>¥bin ディレクトリーで **`run.sh -c <your_server_name>`** コマンドを入力し、新しいサーバーを始動してください。
4. サーバーが始動すると、実行中かどうかの確認はウェブ・ブラウザーで http://localhost:8080を開くことでできます。JBoss Welcome ウィンドウが表示され、JBoss コンソールへ接続することができます。
5. アプリケーション・サーバーが起動し、実行されたら、次はサンプル・アプリケーションに必要なソフトウェアをインストールし、構成するステップです。このステップは以下のセクションで説明します。

事前に必要なソフトウェアのインストールと構成

この文章でのオンライン取引アプリケーションのビルドと実行のために、ビルド・ツールとアプリケーションが利用するデータベースをインストールし、構成する必要があります。

データベースのインストール

前述のとおり、このアプリケーションは MySQL データベースを利用しています。次の URL からダウンロードできます。

http://www.mysql.com

メモ：貴方の持つ MySQL のバージョンに応じて、MySQL Connector/J のふさわしいバージョン (3.1、5.0、5.1) もダウンロードしてください。

この MySQL コネクターは <JBOSS_HOME>/server/default/lib に置いてください。

MySQL のインストールと構成は、かなり直感的です。MySQL リファレンス・マニュアルは次の URL からダウンロードできます。

http://dev.mysql.com/doc/mysql/en

メモ：簡単に構成するために、私はセキュリティ設定を変更し、root ユーザーに "password" というパスワードをつけました。このユーザー ID とパスワードは後ほどサンプル・アプリケーションからデータベースへ接続する際に利用されます。

サンプル・データベースの作成

MySQL のインスタンスが構成されたら、書庫アプリケーション（訳注：オンライン取引アプリケーションの間違い？）が利用するサンプル・データベースを作ります。コマンド・ラインから以下のコマンドを入力し、MySQL モニターを始動してください。

```
mysql -u root -ppassword
```

-p フラグとパスワードとの間にブランク文字がないことに注意してください。

MySQL コマンド・インターフェースが起動すると、以下のような表示になります。

MySQL monitor interface

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 4.1.14-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

```
mysql>
```

MySQL コマンド・インターフェースから、以下のコマンドを入力し、adi サンプル・データベースを作成してください。

```
mysql> create database adi;
```

Ant の構成

前述のとおり、オンライン取引アプリケーションのビルドには Apache Ant を利用しています。もしまだ Ant をインストールしていなかったら、そろそろインストールして、<ant_home>¥bin ディレクトリーをシステムの path 変数に追加してください。

Apache Ant は以下の URL からダウンロードできます。

http://ant.apache.org

Hibernate の構成

JBoss には Hibernate が同梱されていますので、hibernate とは別に jar をダウンロードする必要はありません。

Back to Top

サンプル・アプリケーションのビルド

この文章でのオンライン取引アプリケーションには、アプリケーションのビルドとデータベースの生成に利用できる Ant スクリプトが含まれています。サンプル・アプリケーションを次の URL からダウンロードしてください。Online Brokerage

zip ファイルを解凍すると、brokerage ディレクトリーが作られます。以下、このディレクトリーを <brokerage_home> とします。このディレクトリーで build.properties ファイルを開き、環境に合わせてプロパティを編集してください。以下に例を示します。

build.properties

```
#Replace java.home with your jdk directory
java.home=<java_home>

#Replace jboss.home with the parent directory of lib/j2ee.jar
jboss.home=<jboss_home>/server/<your_server_name>

#Replace geronimo.home with the geronimo home directory.
geronimo.home=<geronimo_home>

#Fully qualified name of the JDBC driver class
db.driver=com.mysql.jdbc.Driver

#database url
db.url=jdbc:mysql://localhost:3306/adi

#database userId
db.userid=root

#database password
db.password=password

#script file for creating the tables
sql.file=sql/db.sql

#location of the jdbc driver jar.
driver.classpath=<mysql-connector_home>/mysql-connector-java-3.1.14-bin.jar

#location of the hibernate jars.
dependency.dir=<hibernate_home>/lib
```

Hibernate の jar ファイルのパスが dependency.dir タグで定義されています。Geronimo と JBoss がそれぞれに Hibernate のコピーを持つ必要があることに注意してください。このディレクトリーに、<hibernage_home> ディレクトリーにある hibernate3.jar ファイルをコピーする必要があります。

⚠ 重要: build.properties ファイルの driver.classpath を設定する際に、スラッシュ "/" を利用することに注意してください。そうしないとコンパイル・エラーになります。

コマンド・プロンプトやシェルで、<brokerage_home> ディレクトリーに移動して、ant コマンドを実行してください。そうすると、<brokerage_home>¥jboss-artefact ディレクトリーに war 、har 、ear ファイルをビルドします。ant によってビルドされた war は、JBoss 固有のデプロイメント・ディスクリプターである jboss-web.xml ファイルを WEB-INF ディレクトリーに含みます。HAR ファイルには JBoss 固有の hibernate-service.xml ファイルを META-INF ディレクトリーに含みます。EAR ファイルには JBoss 固有のデプロイメント・ディスクリプターである jboss-app.xml を含みます。これらのファイルについて、以下に例を示します。

jboss-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web>
<context-root>/brokerage</context-root>
<resource-ref>
<res-ref-name>jdbc/HibernateDB</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<jndi-name>jdbc/HibernateDS</jndi-name>
</resource-ref>
</jboss-web>
```

resource-ref 要素は、web.xml ファイルにある jdbc/HibernateDB という名前で参照されているリソースと、今回の例での MySQL データ・ソース向けである java:jdbc/HibernateDS という JNDI 名のリソースとを結び付けるために利用されます。

hibernate-service.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
<mbean code="org.jboss.hibernate.jmx.Hibernate"
name="jboss.har:service=Hibernate">
<attribute name="DatasourceName">java:jdbc/HibernateDS</attribute>

<attribute name="Dialect">
org.hibernate.dialect.MySQLDialect
</attribute>
<attribute name="SessionFactoryName">
java:/hibernate/BrokerageSessionFactory
</attribute>
<attribute name="CacheProviderClass">
org.hibernate.cache.HashtableCacheProvider
</attribute>
<!-- <attribute name="ScanForMappingsEnabled">true</attribute> -->
<attribute name="ShowSqlEnabled">true</attribute>
</mbean>
</server>
```

このファイルには設定する必要のある hibernate のプロパティを含んでいます。これらの名称と機能は次のとおりです。

- DatasourceName - Hibernate が利用するデータ・ソースの JNDI 名です。
- Dialect - 利用する SQL 方言です。
- SessionFactoryName - セッション・ファクトリーの JNDI 名です。
- CacheProviderClass - キャッシュ・プロバイダーのクラスです。
- ShowSqlEnabled - 実行された SQL 文を出力します。

hibernate-service.xml ファイルは EAR（訳注：原文は EAR ですが、HAR では？）の META-INF ディレクトリーの中にあります。

jboss-app.xml

```
<!DOCTYPE jboss-app PUBLIC "-//JBoss//DTD J2EE Application 1.4//EN" "http://www.jboss.org/j2ee/dtd/
jboss-app_4_0.dtd">
<jboss-app>
<module>
<har>brokerage.har</har>
</module>
</jboss-app>
```

jboss-app.xml ファイルは EAR ファイルの META-INF ディレクトリーの中にあり、har ファイルの名前を記述しています。

以下の例ではこのアプリケーションで利用しているデプロイメント記述である web.xml を示します。web.xml ファイルは brokerage.war ファイルの WEB-INF ディレクトリーにあるデプロイメント記述で、サーブレット名やアプリケーションのデフォルトの JSP などについて記述されています。

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" version="2.4" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
<display-name>brokerage</display-name>
<servlet>
<display-name>Trade-Dispatcher</display-name>
<servlet-name>TradeDispatcher</servlet-name>
<servlet-class>com.dev.trade.servlet.TradeDispatcherServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>TradeDispatcher</servlet-name>
<url-pattern>/login</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>TradeDispatcher</servlet-name>
<url-pattern>/stocks</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>TradeDispatcher</servlet-name>
<url-pattern>/userstocks</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>TradeDispatcher</servlet-name>
<url-pattern>/buy</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>TradeDispatcher</servlet-name>
<url-pattern>/sell</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>TradeDispatcher</servlet-name>
<url-pattern>/register</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>/login.jsp</welcome-file>
</welcome-file-list>
<error-page>
<exception-type>javax.servlet.ServletException</exception-type>
<location>/error.jsp</location>
</error-page>

<resource-ref>
<res-ref-name>jdbc/HibernateDB</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
<res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
</web-app>
```

Back to Top

サンプル・データの生成

前述のとおり、db.sql スクリプトがサンプル・データを生成します。このファイルの場所はすでに build.properties 内で sql.file タグによって定義されています。サンプル・データの生成には、単に <brokerage_home> ディレクトリーで以下のコマンドを実行してください。

```
ant populateDB
```

Back to Top

サンプル・アプリケーションのデプロイ

サンプル・アプリケーションをデプロイする前に、このアプリケーションが必要とするデータ・ソースを構成する必要があります。JBossでデータ・ソースの構成をデプロイするには、<brokerage_home>¥plan ディレクトリーにある mysql-ds.xml ファイルを以下のディレクトリーにコピーしてください。

```
<jboss_home>\server\<your_server_name>\deploy
```

データベースのデプロイと同様に、JBoss でのオンライン取引アプリケーションのデプロイは、Ant でビルドした brokerage.ear ファイルを以下のディレクトリーにコピーしてください。

```
<jboss_home>\server\<your_server_name>\deploy
```

もし JBoss がすでに始動していたら、自動的にアプリケーションがデプロイされ、始動されます。そうでなければ、アプリケーションは次回始動時にデプロイされ、始動されます。

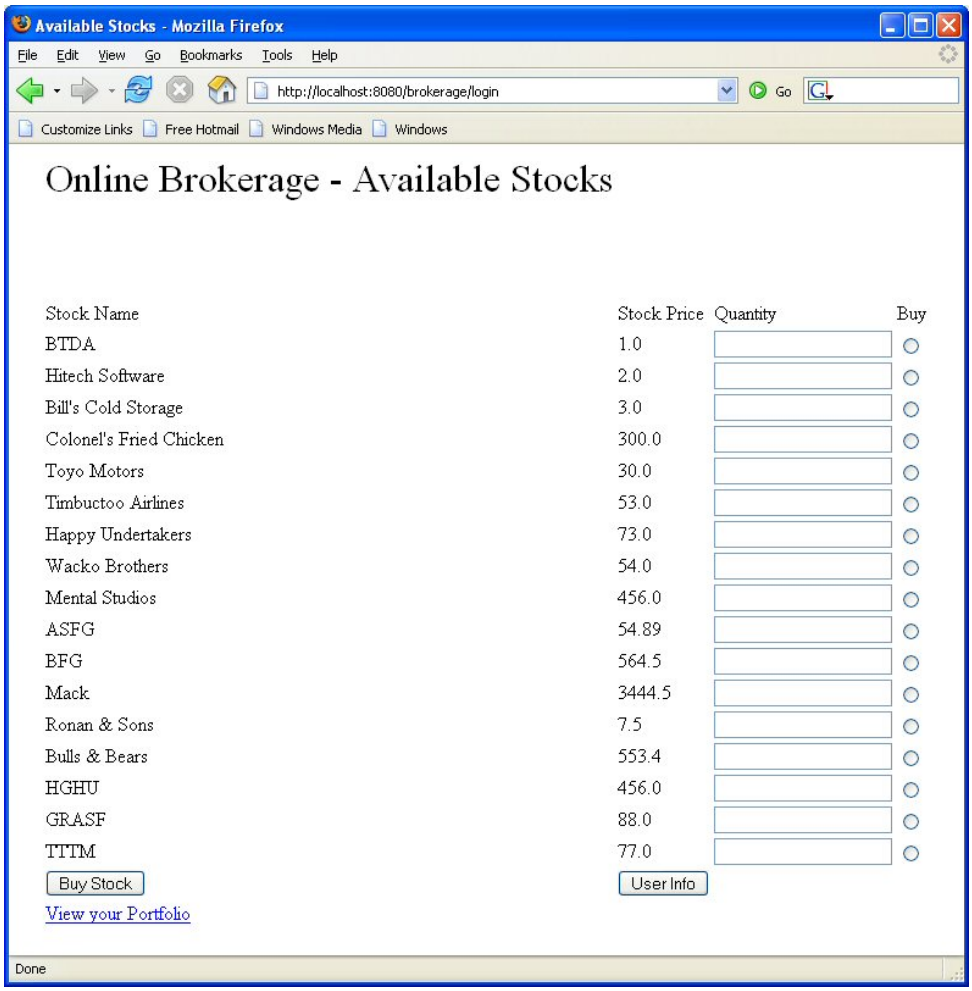
Back to Top

サンプル・アプリケーションのテスト

アプリケーションをテストするには、ウェブ・ブラウザーを開き、以下の URL に接続してください。

http://localhost:8080/brokerage

オンライン取引アプリケーションのログイン画面が表示されます。user name 欄に j2ee 、password 欄に password を入力し、login をクリックしてください。以下の図のような available stocks ページが表示されます。アプリケーションは構成され、実行されています。



Back to Top

Geronimo 環境

以下の URL から Geronimo をダウンロードし、インストールしてください。

http://geronimo.apache.org/downloads.html

そこにあるリリース・ノートには、システム要件とインストール、始動方法がが明示されています。以下、この文章では Geronimo のインストール・ディレクトリーを <geronimo_home> とします。

Back to Top

リソースの構成

Geronimo でオンライン取引アプリケーションを実行するために、JBoss で利用したものと同じ MySQL データベース を利用します。Geronimo 環境に準備のためにするべき作業は、データ・ソースを構成するだけです。

データ・ソースの構成

データ・ソースのデプロイメント・プランで参照可能になるように、MySQL データベースのドライバーを Geronimo リポジトリーにコピーする必要があります。Geronimo リポジトリーは <geronimo_home>/repository ディレクトリにあります。このディレクトリーの中に mysql/jars というディレクトリーを作り、そこへ mysql-connector-java-3.1.14-bin.jar ファイルをコピーしてください。

データ・ソースのデプロイメント・プランを定義する必要があります。簡単になるように、サンプル・アプリケーションにはすでに mysql-geronimo-plan.xml というデプロイメント・プランが <brokerage_home>¥plan ディレクトリーの中にあります。以下の例がデプロイメント・プランの中身です。

mysql-geronimo-plan.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.2">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.1">
<dep:moduleId>
<dep:groupId>user</dep:groupId>
<dep:artifactId>database-pool-HibernateDB</dep:artifactId>
<dep:version>2.0</dep:version>
<dep:type>car</dep:type>
</dep:moduleId>
<dep:dependencies>
<dep:dependency>
<dep:groupId>mysql</dep:groupId>
<dep:artifactId>mysql-connector-java</dep:artifactId>
<dep:version>3.1.14-bin</dep:version>
<dep:type>jar</dep:type>
</dep:dependency>
</dep:dependencies>
</dep:environment>
<resourceadapter>
<outbound-resourceadapter>
<connection-definition>
<connectionfactory-interface>javax.sql.DataSource</connectionfactory-interface>
<connectiondefinition-instance>
<name>HibernateDS</name>
<config-property-setting name="Password">password</config-property-setting>
<config-property-setting name="CommitBeforeAutocommit">false</config-property-setting>
<config-property-setting name="Driver">com.mysql.jdbc.Driver</config-property-setting>
<config-property-setting
name="ExceptionSorterClass">org.tranql.connector.AllExceptionsAreFatalSorter</config-property-
setting>
<config-property-setting name="UserName">root</config-property-setting>
<config-property-setting name="ConnectionURL">jdbc:mysql://localhost:3306/adi</config-property-
setting>
<connectionmanager>
<local-transaction/>
<single-pool>
<max-size>10</max-size>
<min-size>0</min-size>
<blocking-timeout-milliseconds>5000</blocking-timeout-milliseconds>
<idle-timeout-minutes>30</idle-timeout-minutes>
<match-one/>
</single-pool>
</connectionmanager>
</connectiondefinition-instance>
</connection-definition>
</outbound-resourceadapter>
</resourceadapter>
</connector>
```

デプロイメント・プランを作り、ドライバーをコピーしたら、次のステップは実際にデータ・ソースのコネクション・プールをデプロイすることです。もし Geronimo を始動していなければ、以下のコマンドで実行し、始動してください。

<geronimo_home>\bin\geronimo start

データ・ソースのコネクション・プールをデプロイするために、以下のコマンドを実行してください。

<geronimo_home>\bin\deploy --user system --password manager deploy <brokerage_home>\plan\mysql-geronimo-plan.xml ..\repository\org\tranql\tranql-connector-ra\1.3\tranql-connector-ra-1.3.rar

環境によりますが、以下のような確認メッセージが表示されるでしょう。

```
C:\geronimo-2.0\bin>deploy --user system --password manager deploy \brokerage\plan\mysql-geronimo-
plan.xml
..\repository\org\tranql\tranql-connector-ra\1.3\tranql-connector-ra-1.3.rar
```

```
Deployed user/database-pool-HibernateDS/2.0/car
```

Back to Top

ステップ・バイ・ステップの移行

Apache Geronimo はHARアーカイブ形式をサポートしていないので、今回は全てのクラスをWARアーカイブの中に格納します。このアプリケーションをGeronimoで動かすためにクラスを2つ作成する必要があります。ひとつはGeronimo用のTransactionManagerLookupクラスで、もうひとつはセッション・ファクトリーを入手するためのユーティリティ・クラスHibernateUtilです。それ以外にTradeDispatcherServlet とTradeDAOクラスに若干の変更を加える必要があります。

手始めに、アプリケーションに加えるべき変更点を下記のリストで眺めてみましょう。

TradeDAO
HibernateUtil
TradeDispatcherServlet

TradeDAO

JBoss環境に於いては、HibernateMBeanがグローバルJNDIコンテキストへのセッション・ファクトリーを作成・バインドしてくれます。この機構により、単純なルックアップだけでセッション・ファクトリーを入手できるようになるわけです。セッションはセッション・ファクトリー経由で入手することができます。

}}{{{}TradeDAO.java は <brokerage_home>/src/com/dev/trade/dao ディレクトリーに存在します。

Excerpt from TradeDAO.java for JBoss JBoss用のTradeDAO.javaの抜粋

```
package com.dev.trade.dao;

import java.sql.Connection;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;

import com.dev.trade.bo.Stock;
import com.dev.trade.bo.User;
import com.dev.trade.bo.UserStock;
import com.dev.trade.exception.DBException;
import com.dev.trade.util.HibernateUtil;

public class TradeDAO {

SessionFactory factory  = null;
Session session = null;

public TradeDAO() throws Exception {

try {
InitialContext ctx = new InitialContext();
factory  = (SessionFactory)ctx.lookup("java:hibernate/BrokerageSessionFactory");

} catch (NamingException e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
}
```

```
}

public User getUserByUserId(String userId) throws DBException {

session = factory.getCurrentSession();
Query q = session.createQuery("from User u where u.userId=:userId");
q.setString("userId", userId);
return (User) q.uniqueResult();

}
...
```

Geronimo環境ではセッション・ファクトリーを作成するための新しいユーティリティ・クラス、HibernateUtil classを作成することになります。
このクラスには、セッションを入手するための getCurrentSession()メソッドがあります。

TradeDAO.java の冒頭の箇所を以下の抜粋に従って修正します。

Excerpt from TradeDAO.java for Geronimo Geronimo用のTradeDAO.javaの抜

```
package com.dev.trade.dao;

import java.sql.Connection;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

import org.hibernate.Query;
import org.hibernate.Session;

import com.dev.trade.bo.Stock;
import com.dev.trade.bo.User;
import com.dev.trade.bo.UserStock;
import com.dev.trade.exception.DBException;
import com.dev.trade.util.HibernateUtil;

public class TradeDAO {


Session session = null;

public TradeDAO() throws Exception {

}

public User getUserByUserId(String userId) throws DBException {

session = HibernateUtil.getCurrentSession();
Query q = session.createQuery("from User u where u.userId=:userId");
q.setString("userId", userId);
return (User) q.uniqueResult();

}
...
```

以下の文字列を探索・置換します。
`factory.getCurrentSession()`

を

`HibernateUtil.getCurrentSession()`
で置換します。

9箇所が置換されるはずです。

セッションを入手する方法が、Apache Geronimo上でアプリケーションを動かすためのコード上の相違点の代表的なものです。

HibernateUtil

既に述べたとおり、このクラスはセッション・ファクトリーを作成し、アプリケーションにHibernateのセッションを提供するために使われます。このクラスのソース・コードは以下のようになります。

HibernateUtil.java for Geronimo

```
package com.dev.trade.util;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.cfg.Configuration;

/**
* Configures and provides access to Hibernate sessions, tied to the
* current thread of execution.  Follows the Thread Local Session
* pattern, see {@link http://hibernate.org/42.html}.
*/
public class HibernateUtil {

/** location of the Hibernate Configuration File */
private static String CONFIG_FILE_LOCATION = "hibernate.cfg.xml";

/** Holds a single instance of Session */
private static final ThreadLocal threadLocal = new ThreadLocal();

/** The single instance of hibernate configuration */
private static final Configuration cfg = new Configuration();

/** The single instance of hibernate SessionFactory */
private static org.hibernate.SessionFactory sessionFactory;

/**
* Returns the ThreadLocal Session instance.  Lazy initialize
* the <code>SessionFactory</code> if needed.
*
*  @return Session
*  @throws HibernateException
*/
public static Session getCurrentSession() throws HibernateException {
Session session = (Session) threadLocal.get();

if (session == null || ! session.isConnected()) {
if (sessionFactory == null) {
try {
cfg.configure(CONFIG_FILE_LOCATION);
sessionFactory = cfg.buildSessionFactory();
}
catch (Exception e) {
System.err.println("%%%% Error Creating SessionFactory %%%%");
e.printStackTrace();
}
}
session = sessionFactory.openSession();
threadLocal.set(session);
}

return session;
}

/**
*  Close the single hibernate session instance.
*
*  @throws HibernateException
```

```
*/
public static void closeSession() throws HibernateException {
Session session = (Session) threadLocal.get();


if (session != null) {
session.close();
}
}



}
```

HibernateUtil.java は <brokerage_home>/src/com/dev/trade/util ディレクトリーに配置します。ご参考までに、このファイルのコピーは既にサンプル・アプリケーションと一緒に提供されています。

TradeDispatcherServlet

このクラスにもセッション・ファクトリーを入手する際の相違点があります。JBossではセッション・ファクトリーの入手はJNDIコンテキスト経由で行いましたが、Geronimoではユーティリティ・クラス経由で行います。

doGet method for JBoss JBoss用のdoGetメソッド

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {

Session hsession = null;
try {
InitialContext ctx = new InitialContext();
SessionFactory factory  =  (SessionFactory)ctx.lookup("java:hibernate/BrokerageSessionFactory");
hsession = factory.openSession();
} catch (NamingException e1) {
e1.printStackTrace();
}

Transaction tr = hsession.beginTransaction();
```

TradeDispatcherServlet.java は <brokerage_home>/src/com/dev/trade/servlet ディレクトリーに配置されています。doGetメソッドを以下の例に従って置換してください。

doGet method for Geronimo

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {

Transaction tr = HibernateUtil.getCurrentSession().getTransaction();
tr.begin();
```

Hibernateは色々なアプリケーション・サーバーようにトランザクション・マネージャーのルックアップ用のクラスを提供しています。残念ながら、Hibernate 3.2はApache Geronimo用のルックアップ用クラスを提供していませんので、自分たちで作らなくてはなりません。Geronimo用のトランザクション・マネージャーのルックアップ・クラスのコードは以下のようになります。

GeronimoTransactionManagerLookup

```
package org.hibernate.transaction;

import java.util.Iterator;
import java.util.Properties;
import java.util.Set;

import javax.transaction.TransactionManager;
import org.hibernate.HibernateException;
import org.hibernate.transaction.TransactionManagerLookup;

import org.apache.geronimo.gbean.AbstractName;
import org.apache.geronimo.gbean.AbstractNameQuery;
```

```
import org.apache.geronimo.kernel.Kernel;
import org.apache.geronimo.kernel.KernelRegistry;
import org.apache.geronimo.kernel.proxy.ProxyManager;

public class GeronimoTransactionManagerLookup implements
TransactionManagerLookup {

public static final String UserTransactionName = "java:comp/UserTransaction";

public TransactionManager getTransactionManager(Properties props) throws HibernateException {
/*
* try { Kernel kernel = KernelRegistry.getSingleKernel(); ProxyManager
* proxyManager = kernel.getProxyManager(); AbstractNameQuery query =
* new AbstractNameQuery(TransactionManager.class.getName()); Set names =
* kernel.listGBeans(query); AbstractName name = null; for (Iterator it =
* names.iterator(); it.hasNext();) name = (AbstractName) it.next();
* Object transMg = (Object) proxyManager.createProxy(name,
* TransactionManager.class); return (TransactionManager)transMg; }catch
* (Exception e) { e.printStackTrace(); System.out.println(); throw new
* HibernateException("Geronimo Transaction Manager Lookup Failed", e); }
*/
try {
Kernel kernel = KernelRegistry.getSingleKernel();
AbstractNameQuery query = new AbstractNameQuery(TransactionManager.class.getName ());
Set<AbstractName> names = kernel.listGBeans(query);
if (names.size() != 1) {
throw new IllegalStateException("Expected one transaction manager, not " + names.size());
}
AbstractName name = names.iterator().next();
TransactionManager transMg = (TransactionManager)
kernel.getGBean(name);
return (TransactionManager)transMg;
} catch (Exception e) {
e.printStackTrace();
System.out.println();
throw new HibernateException("Geronimo Transaction Manager Lookup Failed", e);
}
}

public String getUserTransactionName() {
return UserTransactionName;
}
}
```

ご参考までに、このクラスは既に <brokerage_home>/TransactionManager ディレクトリーに配置されています。以下のようなディレクトリー構造を作り、そこに GeronimoTransactionManagerLookup.java をコピーしてください。

*brokerage_home>/src/org/hibernate/transaction

さて次にHibernateの構成ファイル hibernate-cfg.xml を作る必要があります。このファイルの中に必要なHibernateの構成上の属性を記述します。JBoss環境での hibernate-service.xml は　hibernate-cfg.xml と同じ働きをしている点にご留意ください。ご参考までに、サンプルアプリケーションと共にこのファイルも既に <brokerage_home>/hibrenate ディレクトリーに提供されています。

hibernate-cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>
<!-- properties -->
<property name="connection.datasource">java:comp/env/jdbc/HibernateDB</property>
<property name="hibernate.transaction.manager_lookup_class">
```

```
org.hibernate.transaction.GeronimoTransactionManagerLookup
</property>

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<!-- Disable the second-level cache  -->
<property name="hibernate.cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
<property name="hibernate.current_session_context_class">org.hibernate.context.JTASessionContext</
property>
<!-- Echo all executed SQL to stdout -->
<property name="hibernate.show_sql">true</property>

<!-- Drop and re-create the database schema on startup -->
<!--<property name="hibernate.hbm2ddl.auto">create</property> -->

<!-- mapping files -->
<mapping resource="Stock.hbm.xml"/>
<mapping resource="UserStock.hbm.xml"/>
<mapping resource="User.hbm.xml"/>

</session-factory>
</hibernate-configuration>
```

各々のプロパティ毎の詳細な機能説明についてはHibernareのマニュアルを参照してください。

ビルド前の最後のステップとして、Geronimo固有のデプロイメント記述子である geronimo-web.xml を作成します。以下の例を参照してください。またご参考までにこのファイルは <brokerage_home>/web/descriptors/geronimo ディレクトリーに既に提供されています。

geronimo-web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1" xmlns:naming="http://
geronimo.apache.org/xml/ns/naming-1.1">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.1">
<dep:moduleId>
<dep:groupId>BrokerageApp</dep:groupId>
<dep:artifactId>MySqlDS</dep:artifactId>
<dep:version>2.0</dep:version>
<dep:type>car</dep:type>
</dep:moduleId>

<dep:dependencies>
<dep:dependency>
<dep:groupId>user</dep:groupId>
<dep:artifactId>database-pool-HibernateDB</dep:artifactId>
<dep:version>2.0</dep:version>
<dep:type>car</dep:type>
</dep:dependency>
</dep:dependencies>

<dep:hidden-classes>
<dep:filter>org.springframework</dep:filter>
<dep:filter>META-INF/spring</dep:filter>
<!--dep:filter>antlr</dep:filter-->
</dep:hidden-classes>
</dep:environment>

<context-root>/brokerage</context-root>

<resource-ref>
<ref-name>jdbc/HibernateDB</ref-name>
<resource-link>HibernateDS</resource-link>
</resource-ref>
</web-app>
```

<hidden-classes> エレメントは、warクラスローダーがクラス探索時にApache Geronimoの提供クラスを参照してしまうことで発生するかもしれないバーション不一致の問題を回避するための指定です。

Back to Top

サンプル・アプリケーションのビルド

サンプル・アプリケーションをビルドするには以下のコマンドを実行します。

- hibernate jar をクラスパスに追加してください。GeronimoTransactionManagerLookup クラスはhibernate と共にビルドする必要があります。貴方がHibernateのソースコードをダウンロードしていないとしても、hibernate jarをクラスパスに追加すればクラスをコンパイルできるようになります。その後、コンパイルしたクラスをhibernate jarファイルに追加します。

```
set CLASSPATH=%CLASSPATH%;<hibernate_home>/lib/hibernate3.jar
```

- クラスパスにgeronimoのカーネルを追加する
  `set CLASSPATH=%CLASSPATH%;<geronimo_home>/lib/geronimo-kernel-2.0.1.jar`
- 今回のサンプルでは必要ありませんが、特定のクラスをご自分で追加するならお好みのツールを使って、GeronimoTransactionManagerLookup.classを <hibernate_home>/lib/hibernate3.jar ファイルの中の org\hibernate\transaction ディレクトリーに追加してください。
- では以下のコマンドを使って移行されたアプリケーションをビルドしてみます。

```
<brokerage_home>/ant war
```

今回のオンライン仲介サンプル・アプリケーションは単なるWebアプリケーションなので war ターゲットを使っています。JBossのセクションではwarとharをパッケージするためにear形式を使いましたが、har形式はGeronimoではサポートされていません。

実行の結果、<brokerage_home>/geronimo-artefact ディレクトリーに brokerage.war ができます。

<brokerage_home>/solutions ディレクトリーにはコンパイルおよびGeronimo環境で稼動させるための移行済みのソースファイル類が含まれています。

Back to Top

移行したサンプル・アプリケーションのデプロイ

移行したオンライン仲介サンプル・アプリケーションをデプロイするには、Geronimoサーバーが起動して正常稼動していることを確認してから以下のコマンドを実行します。

```
deploy --user system --password manager deploy <brokerage_home>/geronimo-artefact/
brokerage.war
```

アプリケーションがデプロイされたら、Webブラウザーを開いて以下のURLにアクセスしてください。
http://localhost:8080/brokerage

JBoss環境でアプリケーションをテストした時に使ったユーザー名、パスワードと同じものを指定してログインします。

Back to Top

サマリー

この記事ではO/Rマッピング層にHibernateを使っているサンプル・アプリケーションをJBossからApache Geronimoへ移行する方法を紹介しました。JBossで提供されている機能・フィーチャーの全てが現時点のGeronimoでは実装されているわでけはないので、結果的に若干の追加のコーディングが必要でしたが、全体的な移行の難易度は低いものでした。

Back to Top

4.7. JBoss to Geronimo - JDBC の移行

This page last changed on 4 21, 2008 by JAGUG.

この文章は、JBoss v4 向けに開発された JDBC アプリケーションを Apache Geronimo へ移行する手助けとなります。この文章は、アプリケーション移行の様々な種類をカバーする一連の文章の一部です。

この文章は、JBoss v4 から Apache Geronimo へ JDBC アプリケーションを移行する順をおった手順の詳細だけでなく、2つのアプリケーション・サーバーの JDBC 実装の違いの詳細も書かれています。一般的な出発点（ソース環境）として、この文章には JBoss ソース環境に Online Brokerage サンプル・アプリケーションをデプロイする手順があります。次に Apache Geronimo へアプリケーションを移行、デプロイする手順を解説します。

この文章は、以下のセクションで構成されています。

JDBC 実装の解析

JDBC 実装は各ベンダーごとにさまざまです。このセクションの目的は、JBoss と Apache Geronimo との間で JDBC 固有の機能の違いを説明することで、違いが明確になり、移行前にそれ相応に計画することができるようになります。

JBoss も Geronimo も JDBC リソースへの接続には J2CA コネクターを利用しているので、プラットフォーム固有の JDBC 機能を比較してみると、いくつかの J2CA 機能は重複しています。

メモ: Geronimo はまだ生まれたばかりなので、JBoss が提供するいくつかの機能はまだ Geronimo に実装されていないかもしれません。

機能	Apache Geronimo	JBoss v4
JDBC 接続	Geronimo は JDBC による直接統合を有していませんが、一般的な J2CA フレームワークを通じての接続をサポートしています。TranQL プロジェクトは各種データベースの J2CA アダプターです。	JBoss での JDBC 接続は、JCA 仕様に基づく JDBC コネクターを通じて行われます。
JCA 実装	Geronimo は JCA 1.5 仕様をサポートし、JCA 1.0 仕様と互換性があります。	JBoss AS 4.0 は JCA (Java Connector Architecture) 1.5 仕様を実装しています。JBoss AS 3.2 では JCA 1.0 でした。
データ・ソース・サポート	TranQL はデータソース、コネクション・プール・データソース、XA データソース向けの包括的なラッパーです。	以下の5種類のデータソースをサポートしています。 • no-tx-datasource • local-tx-datasource • xa-datasource • ha-local-tx-datasource • ha-xa-datasource
データソース・フェイルオーバー	TranQL は特定のデータベース（Apache Derby、Oracle や、もうすぐ DB2 も含まれます）向けに特化したドライバーで、ドライバーの拡張機能との密接な統合をもたらします。 この改装では、ロード・バランシングやフェイル・オーバーといった機能が提供	JBoss は2つのデータ・ソースの実装により、データ・ソースのフェイル・オーバーが可能になります。 1) ha-local-tx-datasource - ローカル・トランザクション向け 2) ha-za-datasource - 分散トランザクション向け

	されます。また、データベースのクラスタリングやフェイル・オーバーのために C-JDBC ラッパーも利用できます。	
XA のサポート	XA トランザクション、ローカル・トランザクション、トランザクションなしをサポートしています。	XA トランザクション、ローカル・トランザクション、トランザクションなしをサポートしています。
コネクション・マネージャーの構成可能性	J2CA フレームワークは各種コネクション・フレームワークの違う部分を組み込む異ができるインターセプターです。現在はまだカスタムのコネクション・マネージャーを組み込むことはできませんが、すぐにこの機能は追加されます。	必要に応じてカスタムのコネクション・マネージャーを組み込むことができます。
JTA の実装	トランザクションのサポートは JOTM と HOWL によって与えられます。	JBossJTA によって JTA のすべてをサポートしています。
コネクション・プーリングと管理	Geronimo のコードと TranQL がコネクション・プーリングと管理に利用されます。	JBossCX フレームワークがコネクション・プーリングと管理に利用されます。
以前のドライバーのサポート	Geronimo は TranQL コネクターの JDBC - JCA ラッパーを通じて以前のドライバーをサポートします。JDBC 3.0 と 2.1 をサポートしています。	JBoss は、まだ JCA- JDBC ドライバーが実装されていない RDBMS への接続を JDBC ドライバーの JCA ラッパーによって行います。

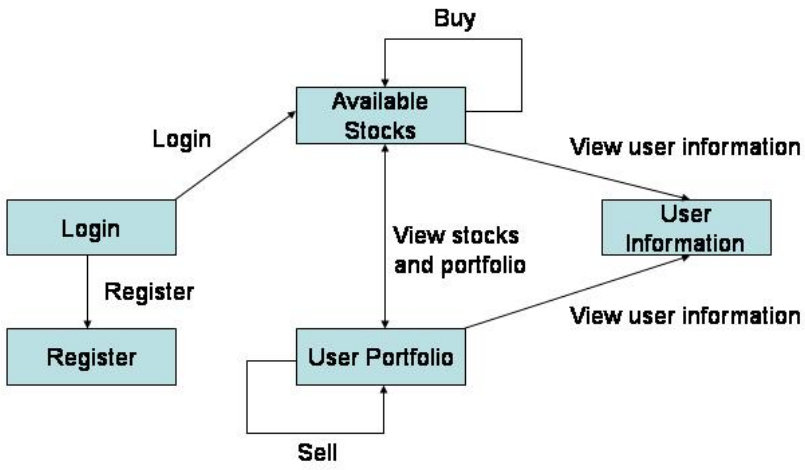
Back to Top

サンプル・アプリケーション

この文章には、JBoss から Geronimo へアプリケーションの移行をデモンストレーションするサンプル・アプリケーションがあります。名前は Online Brokerage です。オンライン取引でユーザーが株の売買を行うシナリオです。このアプリケーションには以下の5つのページがあります。

- Login ページ
- Registration ページ
- User Information ページ
- Available Stocks ページ
- User Portfolio ページ

以下の図は、アプリケーションのフローです。



最初に、ユーザーは Login ページに接続します。ログインページからユーザーはユーザー名とパスワードを入力します。ユーザー名またはパスワードが不正の場合は、アプリケーションはエラーメッセージを表示し、そのログインを拒否します。ユーザー名とパスワードが正確であれば、すべての株についてその時点の株価を見ることができる Available Stocks ページを開きます。

ユーザーは口座にある利用可能な金額に応じて、購入したい株を Buy ボタンをクリックすることで選択します。

取引が無事完了すると、Available Stocks ページ が表示され、必要ならばさらに株が購入できます。

株の購入に資金が不足していたら、アプリケーションはエラーとなり取引を進めません。エラーメッセージが Available Stocks ページの上部に表示されます。このページには User Info ボタンがあります。これをクリックすると、User Info ページが表示され、ユーザーの詳細情報が表示されます。

Available Stocks ページには、ユーザーが所有するすべての株を確認できる Portfolio へのリンクがあります。このページでは、ユーザーは株と株数を選んで売却することができます。また、User Cash 欄に現在の利用可能な現金の学が表示されています。所有している数より多くの株を売ろうとすると、アプリケーションはエラーになります。エラーメッセージが同じページ上に表示されます。売却が成功すると、売却金額がユーザーの現金に加算されます。

quantity テキストボックスにはユーザーが持っている特定の会社の株の数量が表示されています。Quantity to Sell 欄では、その会社の株の売りたい数量を入力することができます。売却と購入にはラジオボタンをチェックします。このことは数値を入力した後になされます。もし quantity to sell テキストボックスが入力されていないか、ラジオボタンがチェックされずに sell ボタンを押すと、必須入力欄が空であるという JavaScript のアラートが表示されます。数量欄に数値以外の文字が入っていると、別のアラートが表示されます。この動き方は、Available Stocks Page と同じようなものです。

新規ユーザーは、login ページで Register ボタンをクリックすることで登録できます。Registration ページでは、ユーザーID、ユーザー名、パスワード、住所と利用可能現金を入力することができます。

Back to Top

アプリケーション・クラスと JSP ページ

オンライン取引サンプル・アプリケーションには、以下のパッケージが含まれます。

- com.dev.trade.bo
  - Stock - 会社の株を表す
  - User - ユーザーを表す
- com.dev.trade.dao
  - TradeDAO - すべてのデータベース接続メソッドを含む
- com.dev.trade.exception
  - DBException - すべてのデータベースの例外を発生させるカスタム例外
- com.dev.trade.servlet
  - TradeDispatcherServlet - すべてのリクエストを、要求されたデータベース機能を実行した後に JSP へ送る処理

オンライン取引には、以下の JSP ページを含みます。

- login.jsp - アプリケーションのログインページ
- error.jsp - アプリケーションのデフォルトのエラーページ
- register.jsp - ユーザー登録のページ
- stocks.jsp - ユーザーが株を購入できる Available Stocks ページ
- userstocks.jsp - ユーザーの保持する株を表示する portfolio ページで、ユーザはここで株を売却する

Back to Top

利用するツール

オンライン取引アプリケーションの開発とビルドのツールは以下の通りです。

Eclipse

Eclipse IDE はサンプル・アプリケーションの開発に利用されています。とても強力で、人気のあるオープンソースの開発ツールです。プラグインの統合によって、JBoss と Geronimo の両方で利用可能です。Eclipse は以下の URL からダウンロードできます。
http://www.eclipse.org

Apache Ant

Ant はピュア Java のビルド・ツールです。war ファイルをビルドし、オンライン取引アプリケーション向けデータベースを作成するために利用されています。Ant は以下の URL からダウンロードできます。
http://ant.apache.org

Back to Top

サンプル・データベース

このアプリケーションのデモンストレーションに利用するデータベースは MySQL です。サンプル・データベース名は tradedb です。STOCKS、USERS、TRADINGACCOUNT の3つのテーブルも持ちます。各テーブルは以下のフィールドを持ちます。

Table Name	Fields
STOCKS	ID (PRIMARY KEY) NAME PRICE
USERS	USERID (PRIMARY KEY) NAME PASSWORD ADDRESS CASH
TRADINGACCOUNT	USERID STOCKID QUANTITY

TRADINGACCOUNT テーブルは、各ユーザが所有する株を保存するために利用されます。USER と STOCKS テーブルはユーザーと株の詳細情報を保存するために利用されます。これは単にサンプル・アプリケーションですので、ユーザーの持つ金額の量は、ユーザー登録時にユーザー自身に入力してもらいます。

このデータベース生成用の DDL は db.sql です。brokerage/sql ディレクトリーにあります。

Back to Top

JBoss 環境

このセクションでは、サンプルの JBoss 環境がどのように、どこにインストールされるかを示すので、貴方はこのシナリオを貴方の実装に合わせることができます。

JBoss のインストール、構成と管理の詳細な説明については、JBoss のドキュメントにあります。JBoss のウェブ・サイトで最新のドキュメントをチェックしてください。

以下のリストはインストールと初期環境の構成を完了させ、サンプル・アプリケーションのデプロイの準備を完了させるまでに必要な一般的なタスクを示します。

1. 製品の説明書にしたがって、JBoss v4 をダウンロード、インストールしてください。以下、インストール・ディレクトリーを <jboss_home> として参照します。
2. デフォルトの JBoss v4 アプリケーション・サーバーのコピーを作ってください。<jboss_home>¥server¥default ディレクトリーを再帰的に <jboss_home>¥server¥<your_server_name> へコピーしてください。
3. <jboss_home>¥bin ディレクトリーで `run.sh -c <your_server_name>` コマンドを入力し、新しいサーバーを始動してください。
4. サーバーが始動すると、実行中かどうかの確認はウェブ・ブラウザーで http://localhost:8080を開くことでできます。JBoss Welcome ウィンドウが表示され、JBoss コンソールへ接続することができます。
5. アプリケーション・サーバーが起動し、実行されたら、次はサンプル・アプリケーションに必要なソフトウェアをインストールし、構成するステップです。このステップは以下のセクションで説明します。

Back to Top

事前に必要なソフトウェアのインストールと構成

この文章でのオンライン取引アプリケーションのビルドと実行のために、ビルド・ツールとアプリケーションが利用するデータベースをインストールし、構成する必要があります。

データベースのインストール

前述のとおり、このアプリケーションは MySQL データベースを利用しています。次の URL からダウンロードできます。
http://www.mysql.com
MySQL のインストールと構成は、かなり直感的です。MySQL リファレンス・マニュアルは次の URL からダウンロードできます。

http://dev.mysql.com/doc/mysql/en

メモ：簡単に構成するために、私はセキュリティ設定を変更し、root ユーザーに "password" というパスワードをつけました。このユーザー ID とパスワードは後ほどサンプル・アプリケーションからデータベースへ接続する際に利用されます。

サンプル・データベースの作成

MySQL のインスタンスが構成されたら、オンライン取引アプリケーションが利用するサンプル・データベースを作ります。コマンド・ラインから以下のコマンドを入力し、MySQL モニターを始動してください。

```
mysql -u root -ppassword
```

-p フラグとパスワードとの間にブランク文字がないことに注意してください。

MySQL コマンド・インターフェースが起動すると、以下のような表示になります。

MySQL monitor interface

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 4.1.14-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

MySQL コマンド・インターフェースから、以下のコマンドを入力し、tradedb サンプル・データベースを作成してください。

```
mysql> create database tradedb;
```

Ant の構成

前述のとおり、オンライン取引アプリケーションのビルドには Apache Ant を利用しています。もしまだ Ant をインストールしていなかったら、そろそろインストールして、<ant_home>¥bin ディレクトリーをシステムの path 変数に追加してください。

Apache Ant は以下の URL からダウンロードできます。

http://ant.apache.org

Back to Top

リソースの構成

オンライン取引アプリケーションとデータベースのビルドとデプロイを成功させるために、まずはいくつかのリソースを構成する必要があります。このセクションでは、JBoss での MySQL データ・ソースの構成方法を説明します。

データ・ソース

オンライン取引アプリケーションはウェブ・アプリケーションです。データ・ソースとしては MySQL データベースに接続します。JBoss を MySQL と一緒に利用するために、MySQL のドライバーを JBoss サーバーのクラスパスにコピーする必要があります。ドライバーはデータベース・ベンダーから提供され、Connector/J 3.0 と呼ばれます。以下の URL からダウンロードできます。
http://dev.mysql.com/downloads/connector/j/3.1.html

ドライバーをダウンロードし、解凍したあと、mysql-connector-java-3.1.14.jar ファイルを <jboss_home>¥server ¥<your_server_name>¥lib ディレクトリーにコピーしてください。

JBoss でのデータ・ソースの構成は、XML 構成ファイルによって行われます。JBoss のインストールには MySQL を含む多くの人気のあるデータベース向けのサンプル構成ファイルを含みます。

MySQL 向けのサンプル・データ・ソース構成ファイルは mysql-ds.xml です。<jboss_home>¥docs¥examples¥jca ディレクトリーにあります。

JBoss サーバーをデプロイしたディレクトリー (<jboss_home>\server\<your_server_name>\deploy) に mysql-ds.xml ファイルをコピーして、以下のように編集してください。

Update the mysql-ds.xml file

```
<jndi-name>jdbc/TradeDB</jndi-name>
<connection-url>jdbc:mysql://localhost:3306/tradedb</connection-url>
<driver-class>com.mysql.jdbc.Driver</driver-class>
<user-name>root</user-name>
<password>password</password>
<exception-sorter-class-name>
org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorter
</exception-sorter-class-name>
```

<local-tx-datasource> タグの中にある他のすべてのタグを削除して、このファイルを保存してください。JBoss が自動的にデータ・ソースをデプロイします。

Back to Top

サンプル・アプリケーションのビルド

この文章でのオンライン取引アプリケーションには、アプリケーションのビルドとデータベースの生成に利用できる Ant スクリプトが含まれています。オンライン取引アプリケーションを以下の URL からダウンロードしてください。

Online Brokerage

zip ファイルを解凍すると、brokerage ディレクトリーが作られます。このディレクトリーで build.properties ファイルを開き、環境に合わせてプロパティを編集してください。以下に例を示します。

Update the build.properties file

```
# Replace server.name with either jboss or geronimo depending on which server to deploy.
server.name=jboss
#Replace <JAVA_HOME> with your JDK home directory
java.home=<JAVA_HOME>
#Replace <JBOSS_HOME> with the root directory for your specific JBoss server <jboss_home>\servers
\<server_name>
jboss.home=<JBOSS_HOME>
#Replace <GERONIMO_HOME> with the root directory for Geronimo
geronimo.home=<GERONIMO_HOME>
#fully qualified name of the JDBC driver class
db.driver=com.mysql.jdbc.Driver
#database URL
db.url=jdbc:mysql://localhost:3306/tradedb
#database userId
db.userid=root
#database password
db.password=password
#script files for creating the tables
sql.file=sql/db.sql
#location of the jdbc driver jar.
driver.classpath=<jboss_home>/server/<your_server_name>/lib/mysql-connector-java-3.1.14-bin.jar
```

build.properties ファイルの driver.classpath を設定する際に、スラッシュ "/" を利用することに注意してください。そうしないと jar ファイルが見つかりません。

コマンド・プロンプトやシェルで、brokerage ディレクトリーに移動して、ant を実行してください。そうすると、war ファイルをビルドし、brokerage ディレクトリー直下にファイルを配置します。ant によってビルドされた war は、JBoss 固有のデプロイメント・ディスクリプターである jboss-web.xml ファイルを WEB-INF ディレクトリーに含みます。以下に例を示します。

JBoss deployment descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web>
<context-root>/brokerage</context-root>
<resource-ref>
<res-ref-name>jdbc/TradeDB</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<jndi-name>java:jdbc/TradeDB</jndi-name>
</resource-ref>
</jboss-web>
```

resource-ref 要素が、web.xml 内の jdbc/TradeDB という名前で参照されるリソースと、MySQL データ・ソースのような java:jdbc/TradeDB という JNDI 名のリソースとを結びつけます。

Back to Top

サンプル・アプリケーションのデプロイ

オンライン取引アプリケーションを JBoss にデプロイするには、先ほど Ant でビルドした brokerage.war ファイルを以下に示すディレクトリにコピーします。

```
<jboss_home>\server\<your_server_name>\deploy
```

もし JBoss がすでに始動していたら、自動的にアプリケーションがデプロイされ、始動されます。そうでなければ、アプリケーションは次回始動時にデプロイされ、始動されます。

Back to Top

サンプル・アプリケーションのテスト

アプリケーションのテストには、ウェブ・ブラウザーを開き、以下の URL に接続してください。

http://localhost:8080/brokerage

オンライン取引アプリケーションのログイン画面が表示されます。user name 欄に j2ee 、password 欄に password を入力し、login をクリックしてください。以下の図のような available stocks ページが表示されます。アプリケーションは構成され、実行されています。

!availableStocks.jpg!

> Pro Tip
> もしデータベースの root 用に他のパスワードを利用したいなら、`password' という記述にあるパスワードを書き換えてください。変更するのは以下の3箇所です。
>
> 1. MySQL インスタンス構成
> 2. データ・ソース構成ファイル
> 3. Ant ビルド・スクリプトが利用する build.properties ファイル

Back to Top

Geronimo 環境

以下の URL から Geronimo をダウンロードし、インストールしてください。

http://geronimo.apache.org/downloads.html

そこにあるリリース・ノートには、システム要件とインストール、始動方法がが明示されています。以下、この文章では Geronimo のインストール・ディレクトリーを <geronimo_home> とします。

TCP/IP ポートの衝突

もし JBoss と Geronimo を同じマシンで実行しようとすれば、少なくともどちらかのサーバではデフォルトのサービス・ポートを変更する必要があります。

Back to Top

リソースの構成

Geronimo では、JDBC リソースは J2EE コネクターを利用して実装されています。したがって、JDBC データ・ソースのデプロイのために Geronimo 固有のデプロイメント・プランを記述する必要があります。加えて、デプロイには tranql-connector-ra-1.3.rar ファイルをデプロイヤーに与える必要があります。この RAR ファイルには、Geronimo のコネクション・プールの管理ロジックを含んでいます。

Geronimo でオンライン取引アプリケーションを実行するために、JBoss で利用したものと同じ MySQL データベース を利用します。Geronimo 環境に準備のためにするべき作業は、データ・ソースを構成するだけです。

データ・ソースの構成

最初に、データ・ソースのデプロイメント・プランが参照できるように Geronimo リポジトリーに MySQL データベースのドライバーをコピーします。

Geronimo リポジトリーは <geronimo_home>/repository にあります。このディレクトリーの中に、mysql/mysql-connector-java/3.1.14-bin というディレクトリーを作成し、そこに mysql-connector-java-3.1.14-bin.jar ファイルをコピーしてください。次は、データ・ソースのデプロイメント・プランを作成します。

データ・ソースの種類の違い
Geronimo がサポートするデータ・ソースは以下の3種類あります。
グローバル・データ・ソース - Geronimo インスタンスにあるすべてのアプリケーションから参照できます。アプリケーション・スコープ・データ・ソース - 定義をした単一のアプリケーションから参照できます。モジュール・スコープ・データ・ソース - 定義した単一のモジュールから参照できます。
この移行の演習ではグローバル・データ・ソースを利用します。他のデータ・ソース2種類の構成と利用については、以下の developerWorks の文章を参照してください。
http://www.ibm.com/developerworks/opensource/library/os-ag-jdbc/

Back to Top

データ・ソースのデプロイメント・プランの作成

まず、mysql-geronimo-plan.xml という xml ファイルを作成してください。以下に示すプランをコピーして、この xml ファイルへ貼り付けてください。

Data source deployment plan

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.2">

<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.1">
<dep:moduleId>
<dep:groupId>user</dep:groupId>
<dep:artifactId>jdbcdatasource</dep:artifactId>
<dep:version>2.0</dep:version>
<dep:type>car</dep:type>
</dep:moduleId>
<dep:dependencies>
<dep:dependency>
<dep:groupId>mysql</dep:groupId>
<dep:artifactId>mysql-connector-java</dep:artifactId>
<dep:version>3.1.14-bin</dep:version>
<dep:type>jar</dep:type>
</dep:dependency>
</dep:dependencies>
```

```
</dep:environment>
<resourceadapter>
<outbound-resourceadapter>
<connection-definition>
<connectionfactory-interface>
javax.sql.DataSource
</connectionfactory-interface>
<connectiondefinition-instance>
<name>TradeDS</name>
<config-property-setting name="UserName">
root
</config-property-setting>
<config-property-setting name="Password">
password
</config-property-setting>
<config-property-setting name="Driver">
com.mysql.jdbc.Driver
</config-property-setting>
<config-property-setting name="ConnectionURL">
jdbc:mysql://localhost:3306/tradedb
</config-property-setting>
<config-property-setting name="CommitBeforeAutocommit">
false
</config-property-setting>
<config-property-setting name="ExceptionSorterClass">
org.tranql.connector.NoExceptionsAreFatalSorter
</config-property-setting>
<connectionmanager>
<local-transaction/>
<single-pool>
<max-size>10</max-size>
<min-size>0</min-size>
<blocking-timeout-milliseconds>
5000
</blocking-timeout-milliseconds>
<idle-timeout-minutes>
30
</idle-timeout-minutes>
<match-one/>
</single-pool>
</connectionmanager>
</connectiondefinition-instance>
</connection-definition>
</outbound-resourceadapter>
</resourceadapter>
</connector>
```

上記リストに示すプランは JDBC でデータベースに接続するための J2EE コネクターをデプロイするものです。ルート要素は <connector> 要素です。この要素は、以下の4つの属性（訳注：3つしかありませんが...）を持ちます。

- xmlns - XML ネームスペースを定義します。
- configId - Geronimo がコネクターを内部で参照する際の名称を定義します。
- parentId - この構成の親構成を定義します。

他の重要な要素や属性は以下の通りです。

- connectionfactory-interface - おそらく javax.sql.DataSource になります。
- connectiondefinition-instance - データベースへの接続に必要な構成プロパティを含みます。
- connectionmanager - geronimo 接続マネージャーの構成設定を含みます。
- config-property-setting - ひとつのコネクターの構成プロパティを記述します。
- name - データ・ソースの名称です。

- local-transaction - この要素の存在によって、リソース・アダプターがローカル・トランザクションをサポートすることができるようになります。もし 'local-transaction' の代わりに 'no-transaction' 要素があれば、リソース・アダプターはトランザクションをサポートしなくなります。もし 'local-transaction' の代わりに 'xa-transaction' 要素があれば、リソース・アダプターは XA トランザクションをサポートするようになります。

- single-pool - max-size や min-size などのシングル・プール向けコネクション・プールの構成プロパティを含みます。もし 'single-pool' のかわりに 'no-pool' 要素があれば、コネクションはプールされません。もし 'single-pool' のかわりに 'partitioned-pool' 要素があれば、パーティション・タイプのコネクション・プールがいくつか作成されます。
- max-size - 最大の同時接続数です。
- min-size - 最小のコネクション・プール数です。
- blocking-timeout-milliseconds - プールにあるすべてのコネクションが利用中のときに、例外を発生させる前にどれだけ待機するかの時間です。
- idle-timeout-minutes - コネクションがアイドル状態のときに、タイムアウトさせるまでの時間を分単位で指定します。

以下の表に、config-property-setting 要素で記述する構成プロパティを示します。

Configuration Property	Meaning
Driver	JDBC ドライバーの完全修飾クラス名です。
ConnectionURL	JDBC 接続の URL です。ドライバーがサポートするパラメータを含めることができます。例えば、MySQL では JDBC 接続 URL に ? `socketFactory=com.mysql.jdbc.NamedPipeSocketFactory` を付加すると、TCP/IP の代わりに名前つきパイプを利用できます。
UserName	データベースへの接続に必要なユーザー名です。
Password	データベースへの接続に必要なパスワードです。
CommitBeforeAutocommit	`setAutoCommit` (true) が呼び出されたときにコミットしていない保留のデータがある場合、データが変更されます。
ExceptionSorterClass	発生した例外が致命的かそうでないかの情報を与えます。

MySQL のデプロイメント・プランを作成したあと、デプロイヤー・ツールを実行してコネクション・プール・データ・ソースをデプロイする必要があります。

デプロイヤー・ツールを利用するには、サーバーが起動し、実行されている必要があるので、Geronimo のインスタンスを始動してください。

コマンドラインで <geronimo_home>/bin ディレクトリーに移動し、以下のコマンドを実行してください。

```
./deployer.cmd --user system --password manager deploy <brokerage_home>\plan\mysql-geronimo-plan.xml <geronimo_home>\repository\org\tranql\tranql-connector-ra\1.3tranql-connector-ra-1.3.rar
```

このとき、brokerage_home というのは brokerage ディレクトリーのパスです。このコマンドでデータ・ソースがデプロイされます。これで移行の準備が整いました。

Back to Top

移行の手順

Geronimo は以下のような実行時の JNDI プロパティの設定による外部のリソース（JDBC データ・ソース）への接続はサポートしていません。

```
Context ctx = null;
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "Geronimo specific factory");
env.put(Context.PROVIDER_URL, "provider URL for geronimo");
ctx = new InitialContext(env)
```

これは、デプロイ時に JSR-77 に基づくコンポーネントのオブジェクト名参照の解析を行うためです。オンライン取引アプリケーションを Geronimo に移行するために、まず jboss-web.xml ファイルを Geronimo 固有の記述ファイルをである geronimo-web.xml ファイルに置き換える必要があります。

geronimo-web.xml ファイルは <brokerage_home>¥web¥descriptors¥geronimo ディレクトリー内にあります。以下のような内容です。

geronimo-web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1" xmlns:naming="http://
geronimo.apache.org/xml/ns/naming-1.1">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.1">
<dep:moduleId>
<dep:groupId>BrokerageApp</dep:groupId>
<dep:artifactId>MySqlDS</dep:artifactId>
<dep:version>2.0</dep:version>
<dep:type>car</dep:type>
</dep:moduleId>

<dep:dependencies>
<dep:dependency>
<dep:groupId>user</dep:groupId>
<dep:artifactId>jdbcdatasource</dep:artifactId>
<dep:version>2.0</dep:version>
<dep:type>car</dep:type>
</dep:dependency>
</dep:dependencies>

</dep:environment>
<context-root>/brokerage</context-root>

<resource-ref>

<ref-name>jdbc/TradeDB</ref-name>

<resource-link>TradeDS</resource-link>

</resource-ref>

</web-app>
```

例に示す通り、この構成の親は MySQL J2EE コネクターです。naming:resource-ref 要素が TradeDS データ・ソースと jdbc/TradeDB という名前を結びつけています。context-root 要素はアプリケーションのコンテキスト・ルートを定義します。

このアプリケーションを Geronimo で実行するには、たったこれだけの変更が必要となります。

Geronimo 向けに .war ファイルをビルドするために、サンプルアプリケーションにある build.properties ファイルを変更し、jboss-web.xml を取り除き、geronimo-web.xml を入れてください。server.name プロパティを geronimo に変更すれば、作業完了です。また、ビルド・スクリプト内にある geronimo.home プロパティの設定も必要です。

変更する必要のある2つの値は、以下の通りです。

```
server.name=geronimo
geronimo.home=<geronimo_home>
```

build.properties はビルド・スクリプトによって利用され、Geronimo 向けの J2EE 仕様を拾い出します。もし jboss.home プロパティがまだ有効であれば、このプロパティを設定する必要がありません。このプロパティが設定されていないか jboss.home が有効でなければ、編集エラーになります。また、あたらしいドライバーの jar を指し示す driver.classpath も設定する必要があります。コマンド・プロンプトから brokerage ディレクトリーに移動し、ant を実行してください。これで war ファイルが brokerage ディレクトリーに作成されます。

Back to Top

移行されるサンプル・アプリケーションのデプロイ

移行されるオンライン取引アプリケーションのデプロイには、Geronimo サーバーが起動し、実行中である必要があります。

コマンドラインで <geronimo_home> ディレクトリー（訳注：<geronimo_home>/bin の誤り？）に移動し、以下のコマンドを入力してください。

```
./deployer --user system --password manager deploy <brokerage_home>/brokerage.war
```

アプリケーションがデプロイされたら、ウェブ・ブラウザーを開き、以下の URL へ接続してください。

http://localhost:8080/brokerage

前に JBoss でテストしたときと同じユーザー名とパスワードでログインしてください。

Back to Top

まとめ

この文章では、サンプル・アプリケーションを JBoss から Apache Geronimo アプリケーション・サーバーへ移行する方法を紹介しました。手順にしたがって順にアプリケーションをビルド、デプロイし、実行し、それから Geronimo 環境へ移行しました。

以下のリストでは、このサンプル・アプリケーションの移行によって見つかった大きな違いをまとめています。

1. Geronimo 固有のデプロイメント記述ファイルではリソース参照名がデータ・ソース名に結び付けられます。JBoss 固有のデプロイメント記述ではリソース名はデータ・ソースの JNDI 名に結びつけられます。
2. JBoss では、データ・ソースのデプロイはただ deploy ディレクトリに構成ファイルをコピーするだけでよいのですが、Geronimo ではデプロイヤー・ツールを利用する必要があります。
3. 現在では、Geronimo は実行時に JNDI プロパティの設定による外部のリソース（JDBC データ・ソースなど）への接続はサポートしていません。このようなリソースは、デプロイ時に指定されている必要があります。

Back to Top

4.8. JBoss to Geronimo - Security Migration

This page last changed on 4 20, 2008 by JAGUG.

4.9. JBoss to Geronimo - サーブレットとJSPの移行

This page last changed on 4 21, 2008 by JAGUG.

この記事では、JBoss v4にデプロイされたサーブレットとJSPをApache Geronimoに移行する際の一助となります。この記事は様々な種類のアプリケーションの移行に関する記事のシリーズのひとつです。

この記事はJ2EEの最も基礎的な側面のひとつ、すなわちサーブレットとJSPの移行をカバーしています。この移行の練習で使用するサンプル・アプリケーションは College Fest で、制御の流れを司るサーブレットとJSPだけを含んでいます。College Festサンプル・アプリケーションはデータベースは一切使用していません。JDBCアプリケーションの移行の詳細については 4.7. JBoss to Geronimo - JDBC の移行 の記事を参照してください。

この記事を読み終えれば、貴方はJBossのビルド・ファイルとデプロイメント記述子を再構成し、移行先であるApache Geronimo向けにセットアップしてシンプルなWebアプリケーションとしてデプロイできるようになります。

この記事は以下のセクションから構成されています。

サーブレットとJSPの実装の分析

サーブレットとJSPの実装はアプリケーション・サーバー毎に異なっているかもしれません。このセクションの目的はサーブレットおよびJSPの個別の機能をJBossとApache Geronimoの間で比較して、移行前に相違点を明確にして、計画を策定できるようにすることです。

Apache GeronimoはJ2EE Webアプリケーションをサポートするwebアプリケーション・コンテナを含んでいます。Webコンテナそれ自体はネットワーク・ポートやSSLオプションのような基本的な構成をサポートしており、Webアプリケーションは Geronimo固有の構成情報を含んでいる場合があります。Webアプリケーションは Geronimoのセキュリティ機構に参加しているので、Webアプリケーションを認証することでEJBやコネクターにセキュアにアクセスすることができます。

Apache Geronimoは現在2つのWebコンテナ、すなわち Jetty と Tomcat をサポートしています。

Jetty

Jettyは100% Javaで書かれたHTTPサーバーであり、サーブレット・コンテナです。ということは、サーブレットやJSPを使用して動的なコンテンツを生成する際に、別のWebサーバーを構成し、動かしておく必要がないということを意味します。Jettyは静的なコンテンツと動的なコンテンツの両方のための、完全な機能を持つWebサーバーです。

サーバーとコンテナを分離するソリューションと違って、JettyのWebサーバーとWebアプリケーションは、通信のオーバーヘッドや複雑な構成をとらずに、同一のプロセスの内部で動きます。さらに、Jettyはpure Javaのコンポーネントなので、デモ、配布、デプロイ用に貴方のアプリケーションに含めてしまうこともできます。

GeronimoではWebコンテナに接続を試みるブラウザーの経路を明示的に構成する必要があります。デフォルトのWebコンテナがJettyの場合には、この経路はコネクターと呼ばれます。標準的な構成ではJettyコネクターはHTTPをポート8080番で、HTTPSをポート8443番でサポートするように構成されています。

Jettyコネクター各々はGBeanなので、Jettyコネクターを構成する際にはGBeanの構成作業が必要です。

Apache Tomcat

Apache TomcatはJavaサーブレットとJSPテクノロジーの公式なリファレンス実装として使用されるサーブレット・コンテナです。

http://java.sun.com/products/servlet
http://java.sun.com/products/jsp

相違点

JBoss v4はデフォルトのWebコンテナとしてTomcat5.5しかサポートしていません。埋め込まれたTomcatサービスはdeployディレクトリーに存在する　　jbossweb-tomcat55.sar という拡張SARです。

HTTPコネクターはポート8080番の上に構成されており、もしApache HTTPのような分離されたWebサーバー経由で接続したい場合は8009番が使用されます。

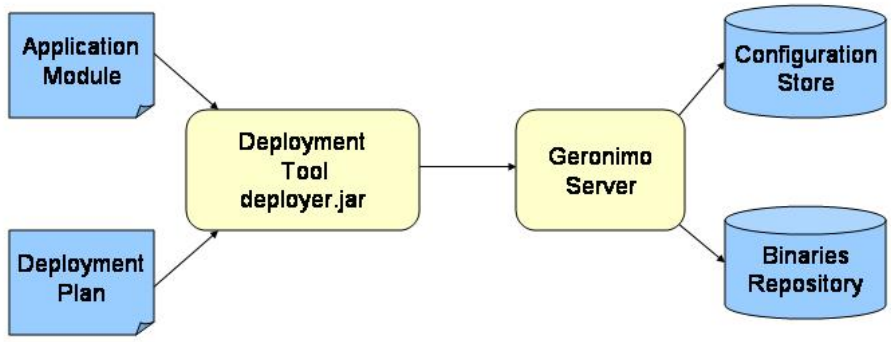
デフォルトのWebコンテナの次の、二番目の大きな相違点はデプロイメント・プランです。Geronimoのデプロイメント・プランは特定のアプリケーションやサービスの構成情報を格納しているXMLファイルを含んでいるという意味で、J2EEのデプロイメント記述子に似ています。

セキュリティやリソース参照が不要な、非常にシンプルなアプリケーションでは、Geronimoのでプロイメント・プランは必要なく、デプロイ時に自動的にデフォルトのコンテキスト・ルートと依存性の定義が提供されます。

GeronimoのWebアプリケーションのデプロイメント・プランは geronimo-web.xml です。JBossで相当するデプロイメント記述子は jboss-web.xml です。より詳細なGeronimoのデプロイメント・アーキテクチャーについては Understand Geronimo's deployment architecture の記事をご参照ください。

サーブレットとJSPにおけるもうひとつの違いは、Webアプリケーションのデプロイの方法です。Geronimoでは、アプリケーションのパッケージ(ear, war, rar またはjar)は <geronimo_home>/bin ディレクトリーにある deployer.jar というデプロイ・ツールを使ってデプロイします。

deployer.jar は（もしプランが提供されていれば）デプロイメント・プランの中で提供された情報に基づいてアプリケーション・モジュールをGeronimoサーバーにデプロイします。その後、サーバーはメタ・データを構成ストアに、実行イメージはバイナリー・リポジトリーに各々保管します。以下の図はデプロイメント・ツールの動きを図解したものです。



JBossでは、Webアプリケーションはアプリケーション・パッケージ(ear, war, rar or jar)は、単純に<jboss_home>/server/<your_server_name>/deploy　ディレクトリーにコピーするだけでサーバーはそれを検知し、適切にデプロイします。

以下の表はJBossとGeronimoの相違点のサマリーです。

機能	JBoss v4	Apache Geronimo
デプロイメント記述子/プラン	jboss.xml	geronimo-web.xml
デプロイの方法	パッケージ(ear, war, rar or jar)をJBossサーバーの<jboss_home>/server/<your_server_name>/deployフォルダーにコピー	サーバーのbinディレクトリー<geronimo_home>/binの - 1.9.3. Deployer tool - デプロイヤー・ツールを利用可能。 デプロイは1.6. Geronimo 管理コンソールでも可能。3番目の方法は Hot deployment で、JBossの機能に相当。
Web コンテナ	Apache Tomcat 5.5	Jetty または Apache Tomcat

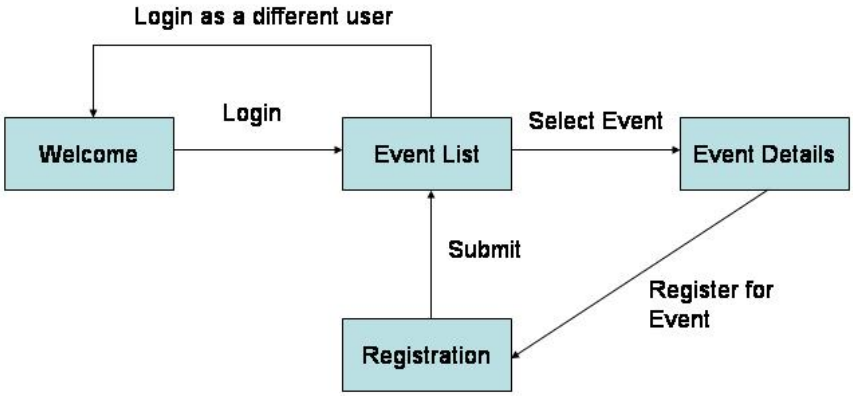
Back to Top

サンプル・アプリケーション

College Fest アプリケーションは大学祭のイベントの登録申し込みを扱います。これはとてもシンプルなアプリケーションで、一切データベースは使いません。College Fest アプリケーションには以下の4種類のページがあります。

- Welcome(ようこそ)ページ
- Event List (イベント・リスト)ページ
- Event Details (イベントの詳細)ページ
- Registration (参加登録)ページ

以下の図でアプリケーションの流れを説明します:



ユーザーはWebページにアクセスしてユーザー名とパスワードを入力します。その後の画面で現在有効なイベントのリストが表示されます。利用者はリスト中のイベントをクリックすれば、その詳細を見ることができます。イベントの詳細ページでは、その特定のイベントへの参加申し込みを行えます。

アプリケーションのクラスとJSPページ

大学祭サンプル・アプリケーションは以下の2つのサーブレットから成り立っています。

- WelcomeServlet - ユーザーのログインを処理し、ユーザー名を取得してから次のサーブレットへ引き渡します。
- PersonalServlet - そのユーザー向けにページを処理してから制御をwelcome.jspに戻します。

大学祭サンプル・アプリケーションには以下のJSPが含まれています。

- welcome.jsp - イベントのリストをユーザーに表示して、どのイベントに参加するかを選択できるようにします。
- dc.jsp - Dumb Charades イベントの詳細を表示します。
- pp.jsp - Pot Potpourri イベントの詳細を表示します。
- wtgw.jsp - What's The Good Word イベントの詳細を表示します。
- gq.jsp - General Quiz イベントの詳細を表示します。
- team_reg.jsp - イベントの参加登録を処理します。

使用するツール

大学祭サンプル・アプリケーションを開発・ビルドするのに使用するツールは下記のとおりです。

Eclipse

サンプル・アプリケーションの開発にはEclipse IDEを使います。これはとてもパワフルで有名なオープンソースの開発ツールです。JBossとGeronimoの統合用のプラグインが利用できます。Eclipseは http://www.eclipse.org からダウンロードできます。

Apache Ant

Antはpure Javaのビルド・ツールです。大学祭アプリケーションのwarファイルをビルドするために使います。Antはhttp://ant.apache.orgからダウンロードできます。

Back to Top

JBoss環境

このセクションではサンプルのJBossの環境のインストールの方法やインストールの場所を示し、貴方の環境に於けるシナリオと対比できるようにします。

JBossのインストール、構成、管理の詳細な方法は、製品ドキュメントで提供されています。製品のWebサイトで最新のドキュメントを確認してください。

以下のリストがサンプル・アプリケーションをデプロイする際の出発点として、初めの環境のインストール・構成を完了する際に必要となるタスクの概要のハイライトです。

1. 製品ドキュメントのガイドに従ってJBoss v4をダウンロードし、インストールします。以降、導入ディレクトリーは <jboss_home> と記載します。
2. デフォルトのJBoss v4アプリケーション・サーバーのコピーを作成します。<jboss_home>¥server¥default から下をすべて <jboss_home>¥server¥<your_server_name> へコピーします。
3. <jboss_home>¥bin ディレクトリーから `run.sh -c <##########>` コマンドで新しいサーバーを開始します。
4. サーバーが開始したら、WebブラウザーでURL: http://localhost:8080 を指定して、JBossが稼動していることを確認します。
5. アプリケーション・サーバーが開始して稼動したら、次のステップとして、サンプル・アプリケーションに必要な残りの前提ソフトウエアをインストール・構成します。

前にも述べましたが、大学祭アプリケーションのバイナリーをビルドするにはApache Antを利用します。もし貴方がAntをインストールしていなければ、これが丁度よい機会です。インストール後、システムのpath変数に <ant_home>¥bin ディレクトリーが追加されていることを確認してください。
Apache Antは以下のURLからダウンロードできます。

http://ant.apache.org

サンプル・アプリケーションのビルド

この記事で扱う大学祭アプリケーションにはアプリケーションをビルドする際に使用するAntスクリプトが含まれています。以下のリンクから大学祭アプリケーションをダウンロードしてください。

College Fest

Zipファイルを解凍すると、college_festディレクトリーが作られます。そのディレクトリーで build.properties ファイルを開き、以下の例に従って貴方の環境に一致するように属性を編集してください。

build.propertiesファイルの更新

```
#java.home####JDK###############
java.home=<JAVA_HOME>
#j2ee.home#lib/j2ee.jar#################
j2ee.home=<jboss_home>/server/<your_server_name>
#jboss###################
jboss.server=<jboss_home>/server/<your_server_name>
#Geronimo############
geronimo.home=<geronimo_home>
```

college_festディレクトリーに更に2つのビルド・ファイル　build.xmlとjboss-build.xmlもあります。build.xmlはデフォルトのビルド・ファイルで、もし貴方が ant とだけタイプしたら、アプリケーションのビルドにはこのファイルが使われます。今回のサンプル・アプリケーション用にはjboss-build.xml　が提供されています。

college_festディレクトリーのコマンド行から、以下のコマンドをタイプしてください。

ant -f jboss-build.xml clean deploy

このコマンドを入力すると、antはjboss-build.xmlファイル中に定義してあるターゲットを使って大学祭アプリケーションをビルドし、JBossサーバーにデプロイします。<target name="deploy" ...>, にご注目ください。jboss-build.xml中のこの箇所でWARファイルをデプロイする場所を指定しています。以下の例でjboss-build.xmlファイル中の定義を示します。

jboss-build.xml

```
<?xml version="1.0"?>
<!-- build file for building a war -->

<project name="build" default="war" basedir=".">

<property file="build.properties"/>
<property name="src.dir" value="src"/>
<property name="dest.dir" value="bin"/>

<target name="clean" description="Delete all generated files">
<echo message="Deleting bin directory" />
<delete dir="bin" />
</target>

<target name="compile">
<mkdir dir="${dest.dir}"/>
<javac srcdir="${src.dir}" destdir="${dest.dir}">
<classpath path="${java.home}/lib/tools.jar"/>
<classpath path="${j2ee.home}/lib/j2ee.jar"/>
</javac>
</target>

<target name="war" depends="compile">
<war destfile="college_fest.war" webxml="WEB-INF/web.xml">
<zipfileset dir="jsp" prefix="jsp"/>
<zipfileset dir="pix" prefix="pix"/>
<classes dir="${dest.dir}"/>
<webinf dir="WEB-INF" />
</war>
</target>

<target name="deploy" depends="war">
<copy file="college_fest.war" todir="${jboss.server}/deploy"/>
</target>

<target name="undeploy">
<delete file="${jboss.server}/deploy/college_fest.war"/>
</target>
</project>
```

Antのビルドで作成されたwarはJBoss固有のデプロイメント記述子を含んでいます。下記にWARのWEB-INFディレクトリー中のjboss-web.xml を示します。

JBoss deployment descriptor

```
<?xml version="1.0" encoding="UTF-8"?>

<jboss-web>

<context-root>college_fest</context-root>
<context-priority-classloader>
false
</context-priority-classloader>
</jboss-web>
```

Back to Top

サンプル・アプリケーションのデプロイ

ここまでのステップでは、カスタマイズされたjboss-build.xmlファイルを使って、ビルド時にアプリケーションをデプロイする方法をご覧頂きました。もし貴方がデフォルトのbuild.xmlを使ってしまった場合は、大学祭アプリケーションをご自分の手でデプロイせねばなりません。大学祭アプリケーションをJBossにデプロイするには、Antでビルドした college_fest.war ファイルを以下のディレクトリーにコピーしてください。

<jboss_home>¥server¥<your_server_name>¥deploy

既にJBossが開始しているなら、自動的にデプロイされ、アプリケーションが開始します。JBossが開始していなければ、次回の始動時にデプロイされ、開始されます。

Back to Top

アプリケーションのテスト

アプリケーションをテストするには、Webブラウザーを開いて以下のURLへアクセスしてください。

http://localhost:8080/college_fest

Welcome画面が表示され、貴方の名前と大学を指定すればログインできます。貴方の名前と大学を入力してSubmit をクリックすると、ページの末尾のメッセージに貴方の名前と、サイトに入るための"Click here" リンクが表示されます。サイトを閲覧でき、オプションをチェックできれば、大学祭アプリケーションは構成され稼動しています。

Back to Top

Geronimo環境

以下のURLからGeronimoをダウンロードしてインストールしてください。

http://geronimo.apache.org/downloads.html

URLにあるリリース・ノートには、システム要件、Geronimoのインストールや構成のための方法が正確に記載されています。以降、当記事では以降、Geronimoの導入ディレクトリーを <geronimo_home> と表記します。

> TCP/IPポートの競合
> もしJBossとGeronimoを同じマシンで動かそうとしているなら、いずれかのサーバーのデフォルトのポート番号を変更することを検討する必要があります。

Back to Top

ステップ・バイ・ステップの移行

大学祭アプリケーションをGeronimoに移行するには、jboss-web.xmlファイルをGeronimo固有の記述子ファイルである geronimo-web.xml に置換する必要があります。geronimo-web.xmlファイルはcollege_festディレクトリー構造のWEB-INFディレクトリーにあります。Geronimoのデプロイメント・プランgeronimo-web.xmlを以下の例で図示します。

> 以下のgeronimo-web.xmlはJ2Gツールで生成されたものです。

Geronimo specific deployment plan geronimo-web.xml

```
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1"
xmlns:naming="http://geronimo.apache.org/xml/ns/naming-1.1"
xmlns:security="http://geronimo.apache.org/xml/ns/security-1.1"
xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.1">
<sys:environment>
<sys:moduleId>
<sys:groupId>j2g</sys:groupId>
<sys:artifactId>web-module</sys:artifactId>
<sys:version>1.0</sys:version>
```

```
<sys:type>war</sys:type>
</sys:moduleId>
<sys:dependencies/>
</sys:environment>
<context-root>college_fest</context-root>
</web-app>
```

大学祭アプリケーションはシンプルなアプリケーションなので、Geronimoのデプロイメント・プランもとてもシンプルなものです。このアプリケーションではデータベースも使っていませんし、セキュリティの構成も行っていないことはお忘れなく。他の「JBossからGeronimoへ」の移行関連の記事を読むと、サンプル・アプリケーションの様々な移行シナリオでの複雑さが増すにつれ、デプロイメント・プランの複雑さも増すことに気づかれることでしょう。

この記事の初めの方で、デプロイメント・ツールの振る舞いについて説明しました。デプロイメントの際には、アプリケーション・モジュールとデプロイメント・プランを指定します。サンプルの大学祭アプリケーションはとてもシンプルなので、わざわざデプロイメント・プランを指定せず、デフォルトの設定値(例えばコンテキスト・ルートなど)を使ってGeronimoにデプロイを任せてしまってもかまいません。

Antを使ってサンプルの大学祭アプリケーションをビルドした際には、デフォルトのbuild.xmlの代わりにjboss-build.xml　を使用しました。以下がデフォルトのbuild.xmlの例です。

build.xml

```
<?xml version="1.0"?>
<!-- build file for building a war -->

<project name="build" default="war" basedir=".">

<property file="build.properties"/>
<property name="src.dir" value="src"/>
<property name="dest.dir" value="bin"/>


<target name="clean" description="Delete all generated files.">
<echo message="Deleting bin folder" />
<delete dir="bin"/>
</target>

<target name="compile">
<mkdir dir="${dest.dir}"/>

<javac srcdir="${src.dir}" destdir="${dest.dir}">
<classpath path="${java.home}/lib/tools.jar"/>
<!--classpath path="${j2ee.home}/lib/j2ee.jar"/-->
<classpath path="${geronimo.home}/repository/org/apache/geronimo/specs/geronimo-
servlet_2.5_spec/1.1/geronimo-servlet_2.5_spec-1.1.jar" />
</javac>
</target>

<target name="war" depends="compile">
<war destfile="college_fest.war" webxml="WEB-INF/web.xml">
<zipfileset dir="jsp" prefix="jsp"/>
<zipfileset dir="pix" prefix="pix"/>
<classes dir="${dest.dir}"/>
<webinf dir="WEB-INF" />
</war>
</target>


</project>
```

移行後のアプリケーションをビルドするには、コマンド行で追加のパラメーターの指定を一切せずに、ant コマンドを実行してください。大学祭アプリケーションのディレクトリー構造のルート・ディレクトリーにcollege_fest.warファイルが作成されます。

生成されたファイルを削除して、クリーンにビルドするには ant の次に ant clean を実行してください。
Back to Top

移行後のサンプル・アプリケーションのデプロイ

移行後の大学祭サンプル・アプリケーションをデプロイするために、Geronimoサーバーが起動していることを確認してください。

次に、コマンド行で **<geronimo_home>/bin** ディレクトリーに移動して、下記のコマンドを入力してください。

```
deploy --user system --password manager deploy <college_fest_home>/college_fest.war
```

アプリケーションがデプロイされたら、Webブラウザーを開き、以下のURLへアクセスします。

http://localhost:8080/college_fest/

そしてJBoss環境での アプリケーションのテスト で行ったステップを繰り返してください。

Back to Top

サマリー

この記事では、シンプルなサーブレットとJSPをJBossからGeronimoアプリケーション・サーバーに移行する方法を示しました。アプリケーションをビルド、デプロイ、実行してからGeronimo環境へ移行する手順を順番に行いました。

当記事を読み終えた方への特記事項:

- Apache GeronimoはJettyとApache Tomcatの2種類のコンテナを提供しています。
- Geronimoへアプリケーションをデプロイするためのデプロイヤー・ツールの使用法を学びました。
- 常にデプロイメント・プランが必要というわけではありません。アプリケーションがリソース参照を使用していないなら、Geronimoのデフォルトのデプロイメントを受け入れることもできます。

4.a. JBoss to Geronimo - Web Services Migration

This page last changed on 4 20, 2008 by JAGUG.

5. クイック・スタート - いますぐ始めたい人の Apache Geronimo

This page last changed on 4 21, 2008 by JAGUG.

とにかく一刻も早く Apache Geronimo を動かしたい方のために、ここではダウンロードや（必要に応じ）ビルドについての基本的なステップをご紹介します。10分にも満たない時間でサーバーを稼動させることができます。

ソフトウェアの入手*

現在2つの方法でリリース済みのソフトウェアを入手することができます。一つはApache SVN リポジトリーからリトリーブする方法ですが、より早くて簡単な方法はバイナリーを直接 Apache Geronimo サイトからダウンロードすることです。ここでは手っ取り早く Apache Geronimo を稼動させるためにも、後者のバイナリー・ダウンロードについて扱います。

事前準備*

この導入パスが最速とはいえ、Geronimo 以前に導入を要するソフトウェアがいくつかあります。

J2SE 1.5

Apache Geronimo v2.0 は、Sun JDK 5.0+ (J2SE 1.5.0+)　のフル・サポートを提供しています。下記 URL に J2SE 1.5 のダウンロード / 導入に関する詳細が記載されていますので、未導入の場合は J2SE 1.5 環境を導入・設定してください。

http://java.sun.com

バイナリーのダウンロード

Apache Geronimo を導入し実行するプラットフォームごとに適切な導入イメージを選択します。Web ブラウザを開き、下記 URL に接続すると、ダウンロード・パッケージ（バイナリーおよびソース・コード）が入手可能です。

http://geronimo.apache.org/downloads.html

ご自身の環境にあった圧縮形式(zip, tar, gz)のファイルを選択し、ダウンロードしたバイナリー・ファイルを適切なディレクトリーに展開します。もしz:/Geronimo に展開したのであれば、以降 <geronimo_home> と読み替えます。

Apache Geronimo の導入は、zip, tar, gz ファイルを解凍するだけです。次にサーバーを始動します。

サーバーの始動*

Apache Geronimo が導入されたら、コマンド・ライン・コンソールを開き、ディレクトリーを <geronimo_home>/bin　に変更し以下のコマンドを実行します。

`geronimo run`

このスクリプトによりコマンドを実行した端末上でサーバーが開始されます。サーバー始動後に下記のような画面が出力されます。

```
D:\geronimo-jetty6-jee5-2.0\bin>geronimo run
Using GERONIMO_BASE:   D:\geronimo-jetty6-jee5-2.0
Using GERONIMO_HOME:   D:\geronimo-jetty6-jee5-2.0
Using GERONIMO_TMPDIR: var\temp
Using JRE_HOME:        C:\Java\jdk1.5.0_06\jre
Booting Geronimo Kernel (in Java 1.5.0_06)...
Starting Geronimo Application Server v2.0
[*******************************] 100%  40s Startup complete
```

```
Listening on Ports:
1050 127.0.0.1 CORBA Naming Service
1099 0.0.0.0   RMI Naming
1527 0.0.0.0   Derby Connector
2001 127.0.0.1 OpenEJB ORB Adapter
4201 0.0.0.0   OpenEJB Daemon
6882 127.0.0.1 OpenEJB ORB Adapter
8009 0.0.0.0   Jetty Connector AJP13
8080 0.0.0.0   Jetty SelectChannel Connector HTTP
8443 0.0.0.0   Jetty Select Channel Connector HTTPS
9999 0.0.0.0   JMX Remoting Connector
61613 0.0.0.0   ActiveMQ Transport Connector
61616 0.0.0.0   ActiveMQ Transport Connector

Started Application Modules:
EAR: org.apache.geronimo.configs/webconsole-jetty6/2.0/car
RAR: org.apache.geronimo.configs/activemq-ra/2.0/car
RAR: org.apache.geronimo.configs/system-database/2.0/car
WAR: org.apache.geronimo.configs/dojo-jetty6/2.0/car
WAR: org.apache.geronimo.configs/remote-deploy-jetty/2.0/car
WAR: org.apache.geronimo.configs/welcome-jetty/2.0/car

Web Applications:
/
/console
/console-standard
/dojo
/remote-deploy

Geronimo Application Server started
```

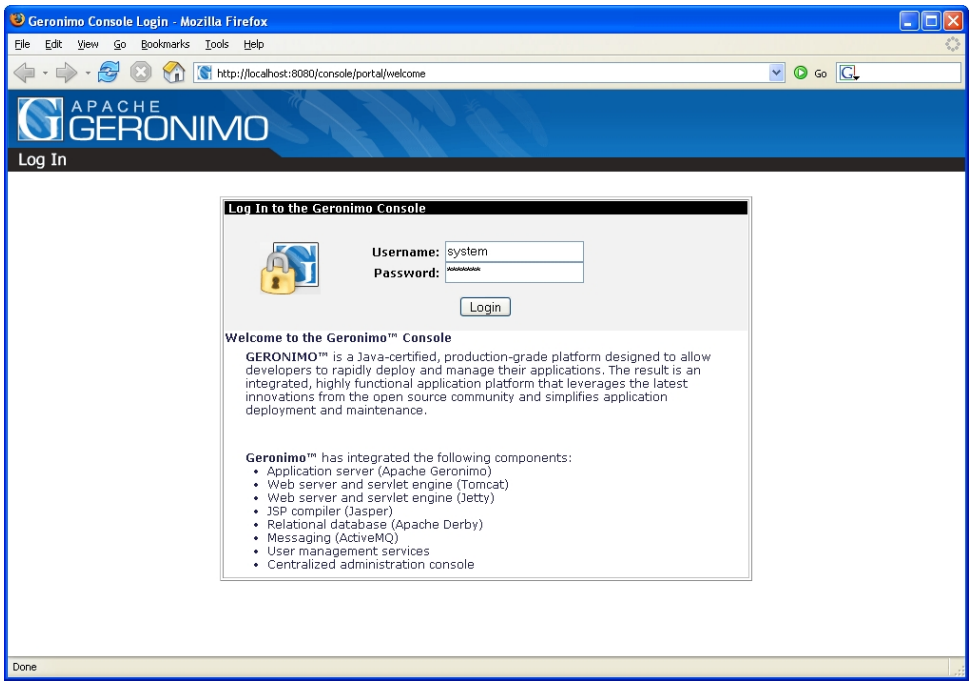
上記画面はサンプルですので、"..."という文字列で切れていたりなど、実際の表示とは若干異なりますのでご留意ください。

Apache Geronimo サーバーをテストしてみましょう。Web ブラウザで下記 URL をクリックし、Geronimo コンソールをテストします。Geronimo 管理コンソールが最初にテスト可能なアプリケーションです。都合よく、フルスタックのJ2EE 認定ディストリビューションとしてデプロイされています。

http://localhost:8080/console

ユーザー名にはsystem 、パスワードには manager をそれぞれ入力します。

下図が Web コンソールのイメージです。



この時点ですでにApache Geronimo を動かしていますが、ここまでに費やした時間はとても短いです。次のセクションではとても基本的な JSP サンプルを通して、どうやってアプリケーションのデプロイと実行を行うか、をご紹介します。

サンプル・アプリケーションの作成とデプロイ*

こういった場合のサンプルとして皆さんよくご存知の HelloWorld のサンプル JSP を使ってテストしてみましょう。新規のディレクトリーを作成し、アプリケーション関連ファイルをそこに入れることにします。以降そのディレクトリーを*<app_home>* と記載します。

<app_home> ディレクトリーにプレーンなテキスト・ファイルで HelloWorld.jsp　を作成し、下記例のコンテンツをコピーします。

HelloWorld.jsp

```
<html>
<head>
<jsp:useBean id="datetime" class="java.util.Date"/>
<title>
Basic HelloWorld JSP
</title>
</head>
<body bgcolor="#909DB8">
<h1>
<font face="tahoma" color="white">
Hello world from GERONIMO!
</font>
</h1>
<font face="tahoma" color="white">on ${datetime}</font>
</body>
</html>
```

Geronimo 特有の「デプロイメント・プラン」を作成し、それが何なのかをつかみましょう。WEB-INF ディレクトリーを <app_home> ディレクトリーの下に作成します。

<app_home>/WEB-INF ディレクトリー内にプレーンなテキスト・ファイルを作成し、geronimo-web.xml というファイル名で下記内容をコピーします。

Geronimo deployment plan geronimo-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.2">
<environment>
```

```
<moduleId>
<groupId>sample.applications</groupId>
<artifactId>HelloWorldApp</artifactId>
<version>2.0</version>
<type>war</type>
</moduleId>
</environment>
<context-root>/hello</context-root>
</web-app>
```

同様に、<app_home>/WEB-INF ディレクトリー内にプレーンなテキスト・ファイルを作成し、web.xml というファイル名で下記内容をコピーします。

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"

xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

<welcome-file-list>
<welcome-file>HelloWorld.jsp</welcome-file>
</welcome-file-list>

</web-app>
```

ここでご紹介した HelloWorld (HelloWorld_2.0.zip もしくは HelloWorld_2.0.war) サンプル・アプリケーションがダウンロード可能ですので、あわせてご利用ください。

アプリケーションのデプロイとテスト*

ここでは、アプリケーションのパッケージングの解説は行いません。その代わり、デプロイヤー・ツールの -inPlace オプションを使用します。それによって、どのディレクトリーからもアプリケーションをデプロイすることが可能になります。

ディレクトリーを <geronimo_home>/bin に変更し、下記コマンドを実行します。

deploy --user system --password manager deploy --inPlace <app_home>

アプリケーションのデプロイが成功すると、下記メッセージが出力されます。

```
D:\geronimo-jetty6-jee5-2.0\bin>deploy --user system --password manager deploy --inPlace
\HelloWorld_2.0
Using GERONIMO_BASE:   D:\geronimo-jetty6-jee5-2.0
Using GERONIMO_HOME:   D:\geronimo-jetty6-jee5-2.0
Using GERONIMO_TMPDIR: var\temp
Using JRE_HOME:        C:\Java\jdk1.5.0_06\jre
Deployed sample.applications/HelloWorldApp/2.0/war @ /hello
Deployed org.apache.geronimo.configs/axis/2.0/car
Deployed org.apache.geronimo.configs/cxf/2.0/car
```

下記 URL をブラウザーに入力し、アプリケーションをテストしてみましょう。

http://localhost:8080/hello

サマリー

ここまでで Apache Geronimo の環境が簡単に構築できることをご理解いただけたかと思います。前提となるソフトウェアが導入されてさえいれば、ほんの数分でサーバーが実行できてしまいます。加えて、シンプルな JSP によるサンプル・アプリケーションの作成/デプロイ/テストまで数分でできてしまうこともお分かりいただけたことと思います。

6. README.txt

This page last changed on 4 23, 2008 by JAGUG.

README.txt

```
=====================================================
Apache Geronimo v2.0.1  ( 8#, 2007 )

http://geronimo.apache.org/
-----------------------------------------------------


_____________________
##########
======================
## README ## 5 ## Geronimo ##################
#########################


-  http://geronimo.apache.org/documentation.html


_____________________
########
======================
##############################################
###################################


_____________________
##
======================
################################Geronimo #########


Geronimo ###################################

<geronimo_home>/var/config/config-substitutions.properties


Windows #############
=========================

Windows ##################255#######################
"My Documents" # "Program Files" ##############################
#################################
#################### <geronimo_home> ################


_____________________
#####
======================
Geronimo ####################/###################
######## geronimo.bat (win) ### geronimo.sh (linux)
####################
##### JAVA_HOME ###### Sun 5 JDK/JRE ######################


################################

<geronimo_home>/bin/geronimo.sh
or ###
<geronimo_home>\bin\geronimo.bat

######################Geronimo #######/################
```

```
########################## <geronimo_home> ####################


java -Djava.endorsed.dirs=lib/endorsed -javaagent:bin/jpa.jar -jar bin/server.jar


############################"welcome" ################

http://localhost:8080/


Geronimo ###############################

http://localhost:8080/console/

############ "system" ######"manager"###

_____________________
####
======================
Geronimo # J2EE ################################################
################################### executable jar ###########


<geronimo_home>/bin/java -jar deployer.jar deploy my-web-app.war [deploy plan]


#####################################

<geronimo_home>/bin/deploy deploy my-web-app.war [deploy plan]


#########################"system" #####"manager" #######


############################################
############################################################

############################################
Geronimo ####### <geronimo_home>/bin: #########

java -jar deployer.jar help [command]


#####################################
<geronimo_home>/deploy/ ###################
#################################

############################################
############################################

###### Geronimo #############GUI #############
####################

http://localhost:8080/console/


_____________________
####
======================

##############Geronimo ###############
Jira ######################

################
```

```
http://mail-archives.apache.org/mod_mbox/geronimo-user/

####################

user-subscribe@geronimo.apache.org

Jira:
http://issues.apache.org/jira/secure/BrowseProject.jspa?id=10220


---------------------------------------
################
=======================================

##############################
####################################
####################
##########################################
################################################
######### http://www.wassenaar.org/ ########

####### Bureau of Industry and Security (BIS) #
Export Commodity Control Number(ECCN) 5D002.C.1###
#################5D002.C.1 ############
################################
## Apache Software Foundation ####################
####################License Exception ENC Technology
Software Unrestricted (TSU)  ###############
(#### the BIS Export Administration Regulations, Section 740.13)

############################

#################geronimo-util ########
artifact ##############################

http://svn.apache.org/repos/asf/geronimo/server/

## URL #################### Apache Geronimo Server #
#################
```

7. RELEASE-NOTES-2.0.1.TXT


```
Release Notes -- Apache Geronimo -- Version 2.0.1

Geronimo URLs
-------------
Home Page:     http://geronimo.apache.org/
Downloads:     http://geronimo.apache.org/downloads.html
Documentation: http://geronimo.apache.org/documentation.html
Mailing Lists: http://geronimo.apache.org/mailing.html
Source Code:   http://geronimo.apache.org/svn.html
Bug Tracking:  http://issues.apache.org/jira/browse/GERONIMO
Wiki:          http://cwiki.apache.org/geronimo

Note
----
Why is 2.0.1 out when I never saw a 2.0?

During the release of Geronimo 2.0 we discovered a security related issue that exposed a running
server to unauthorized
users being able to conduct a remote deploy on a running server.  The problem was discovered during
the release process
and the project decided to not continue the release.  Given that we were far enough into the
release process it was not
appropriate to re-use the 2.0 version number despite the fact that we had not "officially"
completed the release.  For
this reason 2.0.1 is the first 2.0 version number to be released.

Updated Information
-------------------
Please see http://cwiki.apache.org/GMOxDOC20/release-notes-201txt.html for the
latest information on this release.


System Requirements
-------------------
You need a platform that supports the Sun JDK 5.0+ (J2SE 1.5.0+).  Other Java VMs
should work as well.

Most testing has been done on Linux, Mac OS X, and Windows.


Significant Changes in the 2.0 Release
--------------------------------------

- Implements the Java Enterprise Edition 5.0 (Java EE 5) specification
- JAX-WS Web Services
- Two JAX-WS engines integrated: Apache Axis2 and Apache CXF
- Users can configure which JAX-WS engine to use.


Certification Status
--------------------
Apache Geronimo v2.0 have passed 100% SUN's Java Enterprise Edition 5.0
Certification Test Suite. See "Distributions" for further details.


Distributions
-------------
Apache Geronimo v2.0 is available in five distributions so you can pick the
one that better fits your environment.

The available distributions are as follows:

Certified distributions:
```

```
- Apache Geronimo with Tomcat web container, AXIS2 for Web Services and OpenJPA
for persistence.
- Apache Geronimo with Jetty web container, CXF for Web Services and OpenJPA
for persistence.

Non-Certified distributions:
- Little-G with Tomcat web container, minimal configuration.
- Little-G with Jetty web container, minimal configuration.
- Micro-G, stripped down Geronimo pluggable framework.

Note: Non-Certified distributions do not contain a complete JavaEE5 stack and so
cannot be certified.  Certified distributions can be reconfigured by the user
(such as Tomcat web container with CXF for Web Services).

Supported features
------------------
All programming model elements of the Java EE 5.0 Specification are available.
Some of the non-specification related elements such as clustering are still
being worked on.

Installing & Starting Geronimo
------------------------------
To install, simply unpack the .zip (Windows) or tar.gz (Unix) file containing
Geronimo.

If you wish to modify the default ports that Geronimo will use, edit the file
<geronimo_home>/var/config/config-substitutions.properties

Geronimo comes with batch and script files to control server start and stop
functions.  To see usage examples simply type geronimo.bat or geronimo.sh
command as appropriate for your platform.  It is necessary to set JAVA_HOME to
the copy of your Sun 5 JDK/JRE prior to executing the command.

Here is an example to set JAVA_HOME:

export JAVA_HOME=<JDK/JRE_home>

To see the available command options type:

<geronimo_home>/bin/geronimo.sh
or
<geronimo_home>\bin\geronimo.bat

The command will display help text instructing you as to how to start and stop
the Geronimo server.

If you prefer to start the server without a script file you can simply type the
following command from <geronimo_home> directory:

java -Djava.endorsed.dirs=lib/endorsed -javaagent:bin/jpa.jar -jar bin/server.jar

Once the server has started, you can access the Geronimo Administration Console
at http://localhost:8080/console/ . The default user name is "system" and the
default password is "manager".

Note for Windows platforms:
Windows users keep in mind the directory path length limitation of 255 characters.
Defaulting installation to predefined directories such as "My Documents" or
"Program Files" may cause the installation or the server start up to fail. Try a
<geronimo_home> at a root level instead.

Administration Console Security Configuration
---------------------------------------------
The default administration user/password for the Geronimo Administration Console
and command line deployment tool is system/manager.  You can change these defaults
directly from the Geronimo Administration Console by accessing Security -> Console
```

Realm and change the user name and password from the Console Realm Users portlet.

As an alternative, you can make the same changes by editing the
<geronimo_home>/var/security/users.properties and
<geronimo_home>/var/security/groups.properties files.

Deploying Applications

Geronimo comes with deploy scripts and batch files to deploy J2EE modules or
applications. You can use the scripts or simply invoke the executable jar by
running the following command (note that you need to start Geronimo first):

<geronimo_home>/bin/java -jar deployer.jar deploy my-web-app.war [deploy plan]

You will need to use the username "system" and password "manager" unless you
customized those as described above. The deployment plan argument is
optional -- you can pack a deployment plan into the application module, provide
it on the command line, or in some cases omit it entirely.

You can also use the "Login" command to avoid entering a user name and password
every time you use the deploy tool

For more information on the commands and options supported by the deploy tool,
run from within the Geronimo directory <geronimo_home>/bin:

java -jar deployer.jar help [command]

You can also graphically deploy applications and resources via the Geronimo
Administration Console available at http://localhost:8080/console/

Other Deployment Options

As an alternative to the command-line deployer, you can copy application
modules into the <geronimo_home>/deploy/ directory and the
hot deployer service will deploy them automatically. The command-line deployer
has some advantages, as it will output any
deployment errors to its own console rather than just the server log.

Additionally, Geronimo provides a Maven plugin that can deploy applications to
Geronimo as part of a Maven build.

Configuration

Most configuration attributes can be updated in the
<geronimo_home>/var/config/config.xml file. The attributes most likely to be
changed are already included in the supplied config.xml file, while others may
need to be added manually.

Known Issues and Limitations

- If Geronimo is configured with CXF as its JAX-WS engine, some Web Services
might not work correctly on non-Sun JDKs. That is because CXF uses Glassfish
SAAJ implementation which has dependencies on Sun proprietary classes
available only on Sun JDKs.
To workaround this issue and if Sun JDK is not available on your platform
configure Geronimo to use Axis2 as its JAX-WS engine.

Release Notes - Geronimo - Version 2.0

** Sub-task

* [GERONIMO-3040] - Implement process to ensure all annotations are discovered and processed

** Bug
* [GERONIMO-807] - Better handling for system log viewer portlet render requests
* [GERONIMO-1786] - JMS Listeners for protocols activeio, peer and openwire fail to start
* [GERONIMO-2286] - app client plan still uses Strings for dependency Module IDs
* [GERONIMO-2534] - Security realms portlet should validate the realm-name for duplicate name
* [GERONIMO-2699] - Module was not a war
* [GERONIMO-2771] - GlassFish specs are being included in 2.0-M2 builds
* [GERONIMO-2878] - JVM stats exposed through JMX are incorrect and missing init/max heap size
* [GERONIMO-2885] - JSF is leaking application ClassLoaders
* [GERONIMO-2950] - Build test failure in geronimo-naming
* [GERONIMO-3196] - ConcurrentModificationException in ?wsdl processing
* [GERONIMO-3270] - Avoid CMP foreign key violations
* [GERONIMO-3305] - Out Of Memory Error when Running DayTrader in WebServices Mode
* [GERONIMO-3322] - web apps with login config still object to server-side attribute.
* [GERONIMO-3332] - POJO Web-services under Geronimo fail with arrays as method call arguments
* [GERONIMO-3337] - Deploy of servlet-examples-tomcat-2.0-SNAPSHOT.car fails due to existing config.ser
* [GERONIMO-3341] - Slim down geronimo by removing unneeded jars from apps and repository
* [GERONIMO-3343] - Out Of Memory Error when deploying and undeploying applications
* [GERONIMO-3346] - Unable to render Plugins portlet in Jetty
* [GERONIMO-3349] - broken link in console when on the Apache HTTP page
* [GERONIMO-3350] - Console needs to administer all the web connectors. Needs WebManager change.
* [GERONIMO-3355] - unnecessary warning message when trying to create a new group in console realms
* [GERONIMO-3357] - <run-as> role is ignored in web.xml
* [GERONIMO-3358] - LocalAttributeManager needs write permissions even if readOnly is true
* [GERONIMO-3361] - Tomcat is using the wrong classloader to create LoginContext/LoginModules
* [GERONIMO-3362] - Enable Tomcat context level cluster
* [GERONIMO-3365] - config.xml condition vs. load attribute issues
* [GERONIMO-3366] - Dependency needed in openejb config to start OpenEjbSystem before the EntityManagerRegistry
* [GERONIMO-3368] - Axis2ClientConfigurationFactory not always being used in certain platforms
* [GERONIMO-3369] - MultiParentClassLoader does not check for hidden resources in recursiveFind()
* [GERONIMO-3370] - Create local build of OpenJPA in our local repo
* [GERONIMO-3375] - Upgrade to released xmlbeans-maven-plugin
* [GERONIMO-3387] - Unable to deploy an ear file (that contains a war file) if web.xml isn't supplied in the war file.
* [GERONIMO-3411] - Injection target entries are not added to existing env-entry entries.

** Improvement
* [GERONIMO-1874] - synthetic ears should allow use of modules outside the g repository
* [GERONIMO-2438] - adding CRAM-MD5 and DIGEST-MD5 authentication to POP3
* [GERONIMO-2498] - Geronimo should use the full javamail uber jar instead of just the spec jar + provider jar.
* [GERONIMO-2607] - GoperationInfo should include the return type for the operation
* [GERONIMO-2944] - Replace ancient ActiveIO usage in geronimo-security module
* [GERONIMO-3059] - CLIs refactoring - options and arguments parsing should be done prior the boot of a Kernel to provide a quicker feedback to users if they are invalid
* [GERONIMO-3194] - Upgrading documentation for deployment plan-geronimo-application.xml
* [GERONIMO-3344] - Split out non-kernel-dependent transaction and connector classes for use by openejb, jencks, etc etc.
* [GERONIMO-3352] - Upgrade to Jetty 6.1.5 (just released) for Geronimo 2.0

** New Feature
* [GERONIMO-1638] - Multiple servers sharing the same repo and config store

** Task
* [GERONIMO-2581] - Update Version Numbers of Dependent Projects for Geronimo Version 2.0
* [GERONIMO-2700] - JSR-88 1.2 Tasklist (JEE5 Deployment)
* [GERONIMO-3353] - Adjust server log4j files to use a default logging level of error instead of info before releasing.
* [GERONIMO-3364] - Finalize License/Notice files for 2.0 release
* [GERONIMO-3371] - Create local build of Axis2 in our local repo

Previous Milestones release notes:

- 7.1. RELEASE-NOTES-2.0-M1.TXT
- 7.2. RELEASE-NOTES-2.0-M2.TXT
- 7.3. RELEASE-NOTES-2.0-M3.TXT
- 7.4. RELEASE-NOTES-2.0-M4.TXT
- 7.5. RELEASE-NOTES-2.0-M5.TXT
- 7.6. RELEASE-NOTES-2.0-M6.TXT
- 7.7. What's new in Geronimo v2-M2

7.1. RELEASE-NOTES-2.0-M1.TXT


```
Release Notes -- Apache Geronimo -- Version 2.0 - Milestone 1

Geronimo URLs
-------------
Home Page:     http://geronimo.apache.org/
Downloads:     http://geronimo.apache.org/downloads.html
Documentation: http://geronimo.apache.org/documentation.html
Mailing Lists: http://geronimo.apache.org/mailing.html
Source Code:   http://geronimo.apache.org/svn.html
Bug Tracking:  http://issues.apache.org/jira/browse/GERONIMO
Wiki:          http://cwiki.apache.org/geronimo


IMPORTANT
---------
This is a Milestone release, that means that is not the final version of
Apache Geronimo v2.0 Take a look at "Known Issues and Limitations" section for
furhter details.


System Requirements
-------------------
You need a platform that supports the Sun JDK 5.0+ (J2SE 1.5.0+).

Most testing has been done on Linux, Mac OS X, and Windows.


Significant Changes in the 2.0 Release
--------------------------------------
Apache Geronimo v2.0 includes the following features:

- Full Sun JDK 5.0+ (J2SE 1.5.0+)
- Servlet 2.5 (Tomcat)
- JSP 2.1 (Tomcat)
- JSP Debug 1.0 (Tomcat)
- Servlet 2.5 (Jetty)
- JSP 2.1 (Jetty - via Jasper)
- JSP Debug 1.0 (Jetty)
- JSF 1.2
- JSTL 1.2
- Common Annotations 1.0
- JAF 1.1
- JavaMail 1.4
- EJB 3.0 (JPA only)
- JTA 1.1
- JMS 1.1
- JACC 1.1


Installing & Starting Geronimo
------------------------------
To install, simply unpack the .zip (Windows) or tar.gz (Unix) file containing
Geronimo.

If you wish to modify the default ports that Geronimo will use, edit the file
<geronimo_home>/var/config/config.xml

Geronimo comes with batch and script files to control server start and stop
functions.  To see usage examples simply type geronimo.bat or geronimo.sh
command as appropriate for your platform.  It is necessary to set JAVA_HOME to
the copy of your Sun 5 JDK/JRE prior to executing the command.

Here is an example to set JAVA_HOME:
```

```
export JAVA_HOME=<JDK/JRE_home>

To see the available command options type:

<geronimo_home>/bin/geronimo.sh
or
<geronimo_home>/bin/geronimo.bat

The command will display help text instructing you as to how to start and stop
the Geronimo server.

If you prefer to start the server without a script file you can simply type:

java -jar <geronimo_home>/bin/server.jar

Once the server has started, you can access the Geronimo Administration Console
at http://localhost:8080/console/ . The default user name is "system" and the
default password is "manager".


Administration Console Security Configuration
---------------------------------------------
The default administration user/password for the Geronimo Administration Console
and deployment tool is system/manager.  You can change these defaults directly
from the Geronimo Administration Console by accessing Security -> Console Realm
and change the user name and password from the Console Realm Users portlet.

As an alternative, you can make the same changes by editing the
<geronimo_home>/var/security/users.properties and
<geronimo_home>/var/security/groups.properties files.


Deploying Applications
----------------------
Geronimo comes with deploy scripts and batch files to deploy J2EE modules or
applications. You can use the scripts or simply invoke the executable jar by
running the following command (note that you need to start Geronimo first):

<geronimo_home>/bin/java -jar deployer.jar deploy my-web-app.war [deploy plan]

You will need to use the username "system" and password "manager" unless you
customized those during the install process.  The deployment plan argument is
optional -- you can pack a deployment plan into the application module, provide
it on the command line, or in some cases omit it entirely.

You can also use the "Login" command once and avoid to enter user name and
password every time you use the deploy tool

For more information on the commands and options supported by the deploy tool,
run from within the Geronimo directory <geronimo_home>/bin:

java -jar deployer.jar help [command]

You can also graphically deploy applications and resources via the Geronimo
Administration Console available at http://localhost:8080/console/


Other Deployment Options
------------------------
As an alternative to the command-line deployer, you can copy application
modules into the <geronimo_home>/deploy/ directory and the
hot deployer service will deploy them automatically.  The command-line deployer
has some advantages, as it will output any
deployment errors to its own console rather than just the server log.
```

Additionally, Geronimo provides a Maven plugin that can deploy applications to
Geronimo as part of a Maven build.

Configuration

Most configuration attributes can be updated in the
<geronimo_home>/var/config/config.xml file. The attributes most likely to be
changed are already included in the supplied config.xml file, while others may
need to be added manually.

Certification Status

Apache Geronimo v2.0-M1, being a MILESTONE release is not a certified yet certified.

Known Issues and Limitations

<!-- open Bugs for 1.2 + 2.* -->

* GERONIMO-2657   Persistence-* builders need improvements
* GERONIMO-2644   Fix leaking ClassLoaders
* GERONIMO-2643   Stack trace (due to amq) while shutting down Geronimo
* GERONIMO-2642   welcome app not included in the jetty assembly.
* GERONIMO-2622   Implement PolicyContextHandlerSOAPMessage
* GERONIMO-2605   NPE if exporting plugin for module having dependency on module with no groupId
* GERONIMO-2598   Deploy tool prints useless message if configuration start fails
* GERONIMO-2551   Plugin hits NPE if maven-metadata listed artifact doesn't exist or JAR artifact
maven-metadata doesn't exist
* GERONIMO-2534   Security realms portlet should validate the realm-name for duplicate name
* GERONIMO-2491   Hibernate passes connections between servlets which we don't support
* GERONIMO-2481   WebServers portlet: Create/Edit Tomcat Connectors should support editing of all
supported connector attributes
* GERONIMO-2480   Plugin installer status sticks on "Searching for X at Y" while downloading
* GERONIMO-2416   ProxyMethodInterceptor should work with classes that have start,stop methods
* GERONIMO-2290   Percent complete goes over 100% when installing configurations
* GERONIMO-2289   GeronimoAsMavenServlet.java generates wrong default-repository element
* GERONIMO-2288   Abstract/Maven repositories install modules incorrectly
* GERONIMO-2286   app client plan still uses Strings for dependency Module IDs
* GERONIMO-2283   Common libs portlet guesses wrong group ID, gives no usage advice
* GERONIMO-2246   Why resource-env-ref:admin-object-module?
* GERONIMO-2082   [m2] stax dependencies are all wrong
* GERONIMO-2028   Plugin export errors don't stop process, but cause it to fail much later
* GERONIMO-2025   Undeploy and redeploy with no version leaves dangling entries in config.xml
* GERONIMO-1917   repository doesn't deal well with case insensitive file systems
* GERONIMO-1786   JMS Listeners for protocols activeio, peer and openwire fail to start
* GERONIMO-1761   Change geronimo-util module to geronimo-crypto, give credit where credit is due
* GERONIMO-1631   NoSuchConfigException when restarting app after undeploying
* GERONIMO-1285   Deployer does not list all modules that have been stopped
* GERONIMO-603   IllegalArgumentException when deploying WebApp containing a url-pattern of /* in
security-constraint
* GERONIMO-268   Connection Error handling problems
* GERONIMO-250   Connector tries to commit after connection error

Specific Issues, Features and Improvements fixed in Version 2.0-M1

Release Notes - Geronimo - Version 2.0-M1

<!-- closed JIRAs for 2.* -->

** Bug
* GERONIMO-2592   TSSLink doStart() method is not getting called.
* GERONIMO-2630   sun j2ee schemas are being redistributed in jsp and servlet specs

* GERONIMO-1135   Keystore password in System.properties
* GERONIMO-2522   Hot deployer makes app hangs
* GERONIMO-2402   Redeployment fails after third iteration.
* GERONIMO-2646   WAR without a geronimo-web.xml deploys to the wrong context
* GERONIMO-2560   Realm added using SecurityRealm portlet does not work
* GERONIMO-1657   CommandSupport doesn't bubble up the exception. Prints stacktrace.
* GERONIMO-2236   keystore portlet - providing a null or incorrect password on edit unlock can only be recovered with server recycle
* GERONIMO-2350   CertificateChainCallbackHandler willfully conceals causes of failure
* GERONIMO-2363   Console: create new pool using wizard, cannot use "show plan" button for any XA database, even derby
* GERONIMO-2458   MapEditor does not work
* GERONIMO-2459   Connector deployer needs to add dependency to j2ee-server for J2EEServer gbean reference
* GERONIMO-2479   j2ee remote ejb clients should look for "localhost" not 0.0.0.0 by default
* GERONIMO-2533   Password setup forms should use a confirmation field
* GERONIMO-2548   GBeanInfo should exclude attributes and operations of java.lang.Object class
* GERONIMO-2549   NullPointerException: CommandListConfigurations
* GERONIMO-2559   cannot stop activemq via kernel shutdown
* GERONIMO-2566   Creating new listeners for ActiveMQ from JMS Server portlet fails
* GERONIMO-2580   CorbaRefBuilder inserts ref for java:comp/CORBA that fails when corba gbean is not present.
* GERONIMO-2584   Hot deploy module/server restart, throws IllegalArgumentException if application deployed using hotdeployment
* GERONIMO-2585   KeystorePortlet: Lock keystore throws NullPointerException
* GERONIMO-2586   KeystorePortlet: Unlock keystore for availability shows key aliases only when keystore is unlocked for edit
* GERONIMO-2587   FileKeystoreInstance.loadKeystoreData() results in inconsistent state if wrong password is supplied
* GERONIMO-2588   KeyStorePortlet: Locking and unlocking could use some error and info messages
* GERONIMO-2591   Database Pools portlet: Create new pool dependency jar selection problems
* GERONIMO-2602   default compile scope is overloaded with 2 meanings in PlanProcessorMojo
* GERONIMO-2603   Building 1.2 if there are 2.0 artifacts in the repo results in mostly 2.0 artifacts in the server.
* GERONIMO-2611   Configuration should not have duplicates in allServiceParents
* GERONIMO-2615   Not enough info when a gbean ref can't be verified during deployment
* GERONIMO-2623   Infinite loop in the SMTPTransport code when a socket factory class is used.
* GERONIMO-2624   Offline deployer busted
* GERONIMO-2625   Geronimo Console: login page prevents using username, password longer than 25 characters for login
* GERONIMO-2627   jsr88 classpath is all messed up
* GERONIMO-2631   jetty5 builder needs to parameterize the jsp servlet class
* GERONIMO-2632   Connection errors can result in infinite loop
* GERONIMO-2456   NOTICE.txt is missing some notices added during 1.1.1 development
* GERONIMO-2377   deploying a new datasource with the same name does not indicate any problem in the console
* GERONIMO-2652   XmlBeans is having trouble validating some xml from substitution groups
* GERONIMO-2555   Windows scripts don't work when used from different drive
* GERONIMO-2656   for jetty, in servlet-mapping, url patterns must be trimmed.
* GERONIMO-2437   Empty dirs and config.xml entries left behind after undeploy
* GERONIMO-2599   deploying RAR leads to message that Geronimo can't find web.xml
* GERONIMO-2619   Javamail 1.4 spec needs to be using the JAF 1.1 version.
* GERONIMO-1982   server try to deploy the modules in hot deployment directory again during server startup
* GERONIMO-1519   ResourceException.toString() can return null
* GERONIMO-646   Servlet calling HttpServletRequest.isUserInRole(null) causes NPE using Jetty container

** Improvement
* GERONIMO-931   Rename administrative security realm
* GERONIMO-1396   Provide consistent look and feel for table views in the web console across all portlets
* GERONIMO-1880   To Allow configurable password digests during REALM Deployment.
* GERONIMO-2392   Too hard to tell why a gbean doesn't start
* GERONIMO-2499   Generalize NamingBuilder so it can handle more than just jndi building
* GERONIMO-2589   Generate explicit-versions.properties transitvily

```
* GERONIMO-2597    Make web service builder optional
* GERONIMO-2604    Create a specs pom
* GERONIMO-2608    JACC 1.1 support (jsr-115 MR4)
* GERONIMO-2634    Switch to activemq 4.1.0-incubator: include apache incubator repo in our repo
list.
* GERONIMO-2616    Move jee5 work from sandbox to trunk

** New Feature
* GERONIMO-2460    JPA container managed persistence support
* GERONIMO-2636    Update Jetty 6 assembly to use Jasper 6 for JSP 2.1 support
* GERONIMO-2536    Support Java EE 5 JavaServerPages Standard Tag Library in Geronimo (JSTL, JSR
52)
* GERONIMO-2535    Support Java EE 5 Common Annotation Spec in Geronimo (JSR 250)

** Task
* GERONIMO-2537    All Geronimo source files must be brought in line with the new ASF source header
and copyright notice policy
* GERONIMO-2601    Remove the "Old Keystore" portlet
* GERONIMO-2639    Upgrade dojo to 0.4.1

** Sub Task
* GERONIMO-1722    "GERONIMO-851 Module migration to Maven 2: activemq-embedded-rar"

** Test
* GERONIMO-2620    Need to create javamail 1.4 versions of the provider and mail jars
```

7.2. RELEASE-NOTES-2.0-M2.TXT


```
Release Notes -- Apache Geronimo -- Version 2.0 - Milestone 2

Geronimo URLs
-------------
Home Page:     http://geronimo.apache.org/
Downloads:     http://geronimo.apache.org/downloads.html
Documentation: http://geronimo.apache.org/documentation.html
Mailing Lists: http://geronimo.apache.org/mailing.html
Source Code:   http://geronimo.apache.org/svn.html
Bug Tracking:  http://issues.apache.org/jira/browse/GERONIMO
Wiki:          http://cwiki.apache.org/geronimo


IMPORTANT
---------
This is a Milestone release, that means that is not the final version of
Apache Geronimo v2.0 Take a look at "Known Issues and Limitations" section for
further details.

System Requirements
-------------------
You need a platform that supports the Sun JDK 5.0+ (J2SE 1.5.0+).

Most testing has been done on Linux, Mac OS X, and Windows.

Significant Changes in the 2.0-M2 Release
-----------------------------------------
- Java EE management 1.1
- EJB 3.0 support (via OpenEJB integration)
- Web Services technologies:
- STAX 1.0 Streaming API for XML (JSR 173)
- JAXB 2.0 - Java Architecture for XML Binding
- JAX-WS 2.0 support is provided in this milestone. Only POJO based JAX-WS services are supported.
(CXF integration)
- JAX-RPC 1.1 (POJO Only)
- JAXR 1.0
- Addition to admin console that allows users to graphically view classloader hierarchy, JNDI tree
and module dependencies


*See "Functional Characteristics" section below for more details on the new functions.


Significant Changes in the 2.0-M1 Release (prior milestone)
-----------------------------------------------------------

- Full Sun JDK 5.0+ (J2SE 1.5.0+)
- Servlet 2.5 (Tomcat)
- JSP 2.1 (Tomcat)
- JSP Debug 1.0 (Tomcat)
- Servlet 2.5 (Jetty)
- JSP 2.1 (Jetty - via Jasper)
- JSP Debug 1.0 (Jetty)
- JSF 1.2 (JSF applications won't execute)
- JSTL 1.2
- Common Annotations 1.0
- JAF 1.1
- JavaMail 1.4
- EJB 3.0 (JPA only)
- JTA 1.1
- JMS 1.1
- JACC 1.1
```

Functional Characteristics for 2.0-M2

- EJB 3.0 (via OpenEJB project)
Supported:
- JPA (Custom Provider, App-managed, Container-managed) (Also supported in 2.0-M1)
- POJO as a Business Interface (both local and remote)
- POJO Session EJBs
- geronimo-openejb.xml file not required unless you have geronimo specific information to configure
- Deployment of annotated beans (@Stateful and @Stateless)
- Injection via deployment descriptor
- @Resource injection for env-entries, resource-refs, message-destinations, service-refs, most resource-env-refs
- @EJB injection of ejb-refs and ejb-local-refs (Tomcat)
- @PersistenceContext injection
- @PersistenceUnit injection
- JNDI references to the ejb
- JNDI references from the ejb
- Transaction support
- Legacy component (i.e. home) interfaces on a Pojo session bean
- Xml-based *and* annotation-based injection for ejbs, except for message-destinations, or SessionContext when the field or setter is not named setSessionContext
- References to business interfaces, local or remote, from a servlet or an ejb
- References to home interfaces, local or remote, from a servlet or an ejb
- Extended JNDI and DI types
- Deploy and Undeploy

Simple EJB 3.0 example application available at:
http://cwiki.apache.org/confluence/display/GMOxDOC20/Using+some+of+EJB+3.0+functionalities

Limitations:
- No support for MDBs.
- @EJB injection of ejb-refs and ejb-local-refs (Jetty)

- Web Services (CXF)
Supported:
- Deployng a JAX-WS based POJO service that leverages annotations to simplify the creation of the Web service.
- Deploying a traditional JAX-RPC based POJO service
- @Resource Annotations are fully supported provided they are also defined in the web.xml deployment descriptor
- The @Resource WebServiceContext annotation is fully supported.

Simple Web Services example available at: http://cwiki.apache.org/GMOxDOC20/simple-web-service-with-jax-ws.html

Limitations:
- No EJB support for Web Services in 2.0-M2
- <service-ref> elements in the deployment descriptor or @WebServiceRef annotations are not processed.
- Dynamically generated WSDL returned via ?wsdl request might be missing some information.
- Dynamic clients (obtained using the javax.xml.ws.Service API) might not always work.

- JSTL 1.2
Applications that use JSTL must add a dependency in their deployment plan to the jstl jar.
Specifically:
<dep:dependencies>
<dep:dependency>
<dep:groupId>jstl</dep:groupId>
<dep:artifactId>jstl</dep:artifactId>
</dep:dependency>
</dep:dependencies>
Alternatively, the jstl jar can be included in the application's WEB-INF/lib directory.

- JSF 1.2 - Not yet supported
-JSF applications will deploy and start but will not execute yet. An updated MyFaces package will remedy this in the near future.

Installing & Starting Geronimo

To install, simply unpack the .zip (Windows) or tar.gz (Unix) file containing Geronimo.

If you wish to modify the default ports that Geronimo will use, edit the file <geronimo_home>/var/config/config.xml

Geronimo comes with batch and script files to control server start and stop functions. To see usage examples simply type geronimo.bat or geronimo.sh command as appropriate for your platform. It is necessary to set JAVA_HOME to the copy of your Sun 5 JDK/JRE prior to executing the command.

Here is an example to set JAVA_HOME:

```
export JAVA_HOME=<JDK/JRE_home>
```

To see the available command options type:

```
<geronimo_home>/bin/geronimo.sh
or
<geronimo_home>\bin\geronimo.bat
```

The command will display help text instructing you as to how to start and stop the Geronimo server.

If you prefer to start the server without a script file you can simply type:

```
java -jar <geronimo_home>/bin/server.jar
```

Once the server has started, you can access the Geronimo Administration Console at http://localhost:8080/console/ . The default user name is "system" and the default password is "manager".

Administration Console Security Configuration

The default administration user/password for the Geronimo Administration Console and command line deployment tool is system/manager. You can change these defaults directly from the Geronimo Administration Console by accessing Security -> Console Realm and change the user name and password from the Console Realm Users portlet.

As an alternative, you can make the same changes by editing the <geronimo_home>/var/security/users.properties and <geronimo_home>/var/security/groups.properties files.

Deploying Applications

Geronimo comes with deploy scripts and batch files to deploy J2EE modules or applications. You can use the scripts or simply invoke the executable jar by running the following command (note that you need to start Geronimo first):

```
<geronimo_home>/bin/java -jar deployer.jar deploy my-web-app.war [deploy plan]
```

You will need to use the username "system" and password "manager" unless you customized those as described above. The deployment plan argument is optional -- you can pack a deployment plan into the application module, provide it on the command line, or in some cases omit it entirely.

You can also use the "Login" command to avoid entering a user name and password
every time you use the deploy tool

For more information on the commands and options supported by the deploy tool,
run from within the Geronimo directory <geronimo_home>/bin:

java -jar deployer.jar help [command]

You can also graphically deploy applications and resources via the Geronimo
Administration Console available at http://localhost:8080/console/

Other Deployment Options

As an alternative to the command-line deployer, you can copy application
modules into the <geronimo_home>/deploy/ directory and the
hot deployer service will deploy them automatically. The command-line deployer
has some advantages, as it will output any
deployment errors to its own console rather than just the server log.

Additionally, Geronimo provides a Maven plugin that can deploy applications to
Geronimo as part of a Maven build.

Configuration

Most configuration attributes can be updated in the
<geronimo_home>/var/config/config.xml file. The attributes most likely to be
changed are already included in the supplied config.xml file, while others may
need to be added manually.

Certification Status

Apache Geronimo v2.0-M2, being a MILESTONE release is not yet certified.

Known Issues and Limitations

<!-- open Bugs for 2.* -->
GERONIMO-2779 JNDI, Dependency, and ClassLoader views don't work with Safari browser
GERONIMO-2778 Deployer should always fill in required <local> or <remote> interfaces elements in
<ejb-ref>
GERONIMO-2775 Enabling web statistics collection for jetty fails from the admin console
GERONIMO-2773 cannot create a new jetty http connector from console
GERONIMO-2771 GlassFish specs are being included in 2.0-M2 builds
GERONIMO-2768 geronimo-j2ee-management_1.1_spec is still using geronimo-ejb_2.1_spec instead of
new ejb_3.0_spec
GERONIMO-2766 cannot restart a tomcat http connector from the console
GERONIMO-2763 JACC URLPatternSpec does not accept *do:login.do as it should
GERONIMO-2754 Server wont start on Revision: 497496
GERONIMO-2749 AbstractWebModuleBuilder needs to support HTTP extension methods per jacc 1.1
GERONIMO-2747 Error while shutting down Geronimo
GERONIMO-2744 App client needs to be able to log out
GERONIMO-2741 Info messages are being displayed in the console output
GERONIMO-2737 Server shutdown time has increased significantly because of ActiveMQ v4
GERONIMO-2734 Client container does not terminate
GERONIMO-2730 JAX-RPC EJB Web Services support appears to be broken
GERONIMO-2699 Module was not a war
GERONIMO-2696 SQL Exception: Failed to start database
GERONIMO-2694 STATUS file sent out as the Geronimo Weekly Status email needs updating
GERONIMO-2688 no-pool and xa-transaction with no caching result in connections being closed before
the tx ends.
GERONIMO-2687 All "default" Subjects should be obtained by logging in to a realm, not constructed
explicitly
GERONIMO-2682 Sending a message throws a SendFailedException

GERONIMO-2681 IOException when reading content of a message
GERONIMO-2680 mod_jk configfilegenerator produces unusable configentries
GERONIMO-2676 Web Application without required jars throws NullPointer
GERONIMO-2675 Fix geronimo assemblies
GERONIMO-2673 WARN message from Global JNDI
GERONIMO-2668 axis deployer still assumes it's the only one
GERONIMO-2666 NPE in JavaComponentContext
GERONIMO-2664 Servlet Filter Error
GERONIMO-2657 Persistence-* builders need improvements
GERONIMO-2655 Jetty 6 assemblies do not include sample AJP Connectors
GERONIMO-2654 Enabling Welcome App (context "/") on geronimo-jetty6-jee5 assembly breaks web-console (context "/console")
GERONIMO-2650 JSP 2.1 error in Jetty/Tomcat
GERONIMO-2643 Stack trace (due to amq) while shutting down Geronimo
GERONIMO-2626 Building geronimo-kernel leaves 46 files in java.io.tmpdir
GERONIMO-2622 Implement PolicyContextHandlerSOAPMessage
GERONIMO-2605 NPE if exporting plugin for module having dependency on module with no groupId
GERONIMO-2598 Deploy tool prints useless message if configuration start fails
GERONIMO-2481 WebServers portlet: Create/Edit Tomcat Connectors should support editing of all supported connector attributes
GERONIMO-1939 Server Info portlet doesn't display the 'Server Memory Usage' live graph on Internet Explorer

Specific Issues, Features and Improvements fixed in Version 2.0-M2

<!-- closed/fixed JIRAs for 2.* -->
GERONIMO-2746 use dependencymanagement for some axis2 dependent modules FIXED
GERONIMO-2752 Axis2 integration displays invalid information for URL requests FIXED
GERONIMO-2764 Clustered HttpSessions are not properly destroyed during eviction FIXED
GERONIMO-2679 ArrayIndexOutOfBoundsException when getting Mails from Pop3-Account with geronimo-javamail_1.4_mail-1.0.jar FIXED
GERONIMO-2774 Server memory usage graph does not work FIXED
GERONIMO-2706 Unable to create database pool DUPLICATE
GERONIMO-2728 Exception while removing directory module from the console FIXED
GERONIMO-1519 ResourceException.toString() can return null FIXED
GERONIMO-2653 Inconsistencies between jetty6-builder and tomcat6-builder plan.xml files FIXED
GERONIMO-1657 CommandSupport doesn't bubble up the exception. Prints stacktrace. FIXED
GERONIMO-2769 EarConfigBuilder ignores geronimo plan if no application.xml present FIXED
GERONIMO-2714 FileAuditLoginModule fails to write login attempts FIXED
GERONIMO-2745 NPE at org.apache.geronimo.security.SubjectId.hashCode(SubjectId.java:79) FIXED
GERONIMO-2027 Mismatched passwords when editing user in web console FIXED
GERONIMO-2751 [BUILD BREAK] Module geronimo-axis2 FIXED
GERONIMO-2753 the "namingProviderUrl" element does *not* appear to control which IP addresses the RMI service binds to FIXED
GERONIMO-2642 welcome app not included in the jetty assembly. FIXED
GERONIMO-2683 Multiple versions of org.codehaus.wadi.wadi-tribes listed in server/trunk/pom.xml FIXED
GERONIMO-2685 database pool type emty in pool creation wizard - Jetty FIXED
GERONIMO-2740 Filter out the Tomcat debug log msg - java.lang.ClassNotFoundException: org.apache.tomcat.util.net.puretls.PureTLSImplementation FIXED
GERONIMO-2686 database creation pool wizzard fails to deploy FIXED
GERONIMO-2560 Realm added using SecurityRealm portlet does not work FIXED
GERONIMO-2748 DB Manager portlets (DB Viewer and Run SQL) not working FIXED
GERONIMO-2579 TestNG testcases in testsuite broken. JUnit expected. Move to JUnit 4.0 ? WON'T FIX
GERONIMO-2580 CorbaRefBuilder inserts ref for java:comp/CORBA that fails when corba gbean is not present. FIXED
GERONIMO-2587 FileKeystoreInstance.loadKeystoreData() results in inconsistent state if wrong password is supplied FIXED
GERONIMO-2586 KeystorePortlet: Unlock keystore for availability shows key aliases only when keystore is unlocked for edit FIXED
GERONIMO-2585 KeystorePortlet: Lock keystore throws NullPointerException FIXED
GERONIMO-2588 KeyStorePortlet: Locking and unlocking could use some error and info messages FIXED
GERONIMO-2592 TSSLink doStart() method is not getting called. FIXED

GERONIMO-2590 The TSSLinkBuilder is not included in the list of loaded builders. FIXED
GERONIMO-2591 Database Pools portlet: Create new pool dependency jar selection problems FIXED
GERONIMO-2602 default compile scope is overloaded with 2 meanings in PlanProcessorMojo FIXED
GERONIMO-2603 Building 1.2 if there are 2.0 artifacts in the repo results in mostly 2.0 artifacts in the server. FIXED
GERONIMO-2692 Current specs/trunk fails to build, due to missing modules FIXED
GERONIMO-2611 Configuration should not have duplicates in allServiceParents FIXED
GERONIMO-2709 Database creation pool wizard fails in Jetty FIXED
GERONIMO-2615 Not enough info when a gbean ref can't be verified during deployment FIXED
GERONIMO-2619 Javamail 1.4 spec needs to be using the JAF 1.1 version. FIXED
GERONIMO-2722 enable commons-logging in tomcat module FIXED
GERONIMO-2623 Infinite loop in the SMTPTransport code when a socket factory class is used. FIXED
GERONIMO-2624 Offline deployer busted FIXED
GERONIMO-2625 Geronimo Console: login page prevents using username, password longer than 25 characters for login FIXED
GERONIMO-2627 jsr88 classpath is all messed up FIXED
GERONIMO-2631 jetty5 builder needs to parameterize the jsp servlet class FIXED
GERONIMO-2684 Upgrade server trunk (2.0) to use the latest released geronimo-spec versions FIXED
GERONIMO-2632 Connection errors can result in infinite loop FIXED
GERONIMO-2630 sun j2ee schemas are being redistributed in jsp and servlet specs FIXED
GERONIMO-2641 Missing license headers FIXED
GERONIMO-2644 Fix leaking ClassLoaders INVALID
GERONIMO-2646 WAR without a geronimo-web.xml deploys to the wrong context FIXED
GERONIMO-2649 Insert of new EJB does not appear to be occurring FIXED
GERONIMO-2652 XmlBeans is having trouble validating some xml from substitution groups FIXED
GERONIMO-2656 for jetty, in servlet-mapping, url patterns must be trimmed. FIXED
GERONIMO-2659 jspc plugin busted by spec >> provided change FIXED
GERONIMO-2663 Build Break ! at configs/client-deployer CANNOT REPRODUCE
GERONIMO-2669 fix o.a.g.j.ClusteredSessionManager to match changes in Jetty AbstractSessionManager FIXED
GERONIMO-2720 tomcat https connector needs additional param FIXED
GERONIMO-1585 Web app security on /* causes deployment exception FIXED
GERONIMO-2674 The basic CXF integration is not working FIXED
GERONIMO-2732 J2EE Application Client deployment fails with NPE FIXED
GERONIMO-2711 specs/trunk fails to build FIXED
GERONIMO-2721 Unable to deploy anything FIXED
GERONIMO-2713 LDAP Realm fails to test and deploy FIXED
GERONIMO-2105 When redeploying with no version number, new entries in config.xml break FIXED
GERONIMO-2689 New View for JNDI name in all the contexts FIXED
GERONIMO-2691 New view for the hierarchical modules and linked dependencies FIXED
GERONIMO-2750 Remove Depricated Code from Axis2 Integration FIXED
GERONIMO-2719 Use released Jetty 6.1 in Geronimo 2.0-M2 FIXED
GERONIMO-2600 Upgrade to Derby 10.2.2.0 FIXED
GERONIMO-2589 Generate explicit-versions.properties transitvily FIXED
GERONIMO-2594 Add xalan to endorsed directory FIXED
GERONIMO-2716 Create javaee 5 test jars for j2ee-builder tests FIXED
GERONIMO-2597 Make web service builder optional FIXED
GERONIMO-2604 Create a specs pom FIXED
GERONIMO-2608 JACC 1.1 support (jsr-115 MR4) FIXED
GERONIMO-2616 Move jee5 work from sandbox to trunk FIXED
GERONIMO-2629 Upgrade to J2EE Management 1.1 (JSR77) FIXED
GERONIMO-2634 Switch to activemq 4.1.0-incubator: include apache incubator repo in our repo list. FIXED
GERONIMO-2635 Upgrade to JavaMail 1.4 and JavaBeans Activation Framework 1.1 FIXED
GERONIMO-2761 Upgrade to Log4J 1.2.14 maintenance release FIXED
GERONIMO-2648 Integrate JSF 1.2 into 2.0-M1 FIXED
GERONIMO-2662 remove jetty5 support from trunk and 2.0-m1 FIXED
GERONIMO-2783 CXF-based WebServices support: webservices.xml file is no longer required FIXED
GERONIMO-2725 Remove geronimo-qname_1.1_spec usage FIXED
GERONIMO-2726 Update JAXB and StAX implementation versions FIXED
GERONIMO-2762 Updated JAX-WS tests FIXED
GERONIMO-2690 New view for all the classloaders and classes loaded in it FIXED
GERONIMO-2628 Upgrade to tomcat 6.0.2 beta FIXED
GERONIMO-2770 Ejb Deployment with no ejb-jar.xml FIXED
GERONIMO-2718 Upgrade to tomcat 6.0.7 beta FIXED
GERONIMO-2772 Support for EJB 3 descriptors and previous FIXED

GERONIMO-2777 Expose JAX-WS mandatory MessageContext properties FIXED
GERONIMO-2636 Update Jetty 6 assembly to use Jasper 6 for JSP 2.1 support FIXED
GERONIMO-2535 Support Java EE 5 Common Annotation Spec in Geronimo (JSR 250) FIXED
GERONIMO-2667 Streaming API for XML integration DUPLICATE
GERONIMO-2671 Streaming API for XML and JAXB integration FIXED
GERONIMO-2738 JAXB, STAX support for client applications FIXED
GERONIMO-2727 Enable axis2-deployer in jetty and tomcat jee5 assemblies FIXED
GERONIMO-2536 Support Java EE 5 JavaServerPages Standard Tag Library in Geronimo (JSTL, JSR 52)
FIXED
GERONIMO-2729 JAX-RPC POJO WS tests FIXED
GERONIMO-2756 Basic @Resource injection for CXF web services FIXED
GERONIMO-2781 Improved CXF-based POJO WebService support FIXED
GERONIMO-2601 Remove the "Old Keystore" portlet FIXED
GERONIMO-2639 Upgrade dojo to 0.4.1 FIXED
GERONIMO-2658 Add 2.0-M1 Release Notes FIXED
GERONIMO-2670 Update geronimo plugin repository version FIXED
GERONIMO-2760 Upgrade tomcat to version 6.0.8a FIXED
GERONIMO-2755 JSP tests for web-testsuite FIXED
GERONIMO-2620 Need to create javamail 1.4 versions of the provider and mail jars FIXED

7.3. RELEASE-NOTES-2.0-M3.TXT


```
Release Notes -- Apache Geronimo -- Version 2.0 - Milestone 3

Geronimo URLs
-------------
Home Page:     http://geronimo.apache.org/
Downloads:     http://geronimo.apache.org/downloads.html
Documentation: http://geronimo.apache.org/documentation.html
Mailing Lists: http://geronimo.apache.org/mailing.html
Source Code:   http://geronimo.apache.org/svn.html
Bug Tracking:  http://issues.apache.org/jira/browse/GERONIMO
Wiki:          http://cwiki.apache.org/geronimo


IMPORTANT
---------
This is a Milestone release, that means that is not the final version of
Apache Geronimo v2.0 Take a look at "Known Issues and Limitations" section for
further details.

Updated Information
-------------------
Please see http://cwiki.apache.org/GMOxDOC20/release-notes-20-m3txt.html for the latest information
on this release.

System Requirements
-------------------
You need a platform that supports the Sun JDK 5.0+ (J2SE 1.5.0+).

Most testing has been done on Linux, Mac OS X, and Windows.

Significant Changes in the 2.0-M3 Release
-----------------------------------------
Overall this new milestone release is more stable and includes additional and enhanced support
to the features integrated in the previous releases.

Some of the new features added in this release include:

- JavaServer Faces 1.2 (via Apache MyFaces v1.2)
- Common Annotations for the Java Platform (partial support)
- Enterprise JavaBeans 3.0 (via Apache OpenEJB 3.0, partial support)
- Web Services Implementation (partial support)

*See "Functional Characteristics" section below for more details on the new functions.

Functional Characteristics for 2.0-M3
-------------------------------------

- Web Services (Apache Axis2)
- JAX-WS 2.0 limited support (POJO only)
- Web Services Metadata 2.0 limited support

- Web Services (Apache CXF)
- JAX-WS 2.0 support for POJO and EJB
- Web Services Metadata 2.0 limited support

- Limitations:
- Redeploy does not work on Windows. Undeploy leaves behind files in the $geronimo_home/repository
which are locked.
This prevents the same configuration from being deployed again. The server has to be stopped
and the files deleted before the configuration can be deployed again.

Features and functions supported in the 2.0-M2 Release
------------------------------------------------------
```

- EJB 3.0 (via Apache OpenEJB project)
Supported:
- JPA (Custom Provider, App-managed, Container-managed) (Also supported in 2.0-M1)
- POJO as a Business Interface (both local and remote)
- POJO Session EJBs
- geronimo-openejb.xml file not required unless you have geronimo specific information to configure
- Deployment of annotated beans (@Stateful and @Stateless)
- Injection via deployment descriptor
- @Resource injection for env-entries, resource-refs, message-destinations, service-refs, most resource-env-refs
- @EJB injection of ejb-refs and ejb-local-refs (Tomcat)
- @PersistenceContext injection
- @PersistenceUnit injection
- JNDI references to the ejb
- JNDI references from the ejb
- Transaction support
- Legacy component (i.e. home) interfaces on a Pojo session bean
- Xml-based *and* annotation-based injection for ejbs, except for message-destinations, or SessionContext when the field or setter is not named setSessionContext
- References to business interfaces, local or remote, from a servlet or an ejb
- References to home interfaces, local or remote, from a servlet or an ejb
- Extended JNDI and DI types
- Deploy and Undeploy

Simple EJB 3.0 example application available at:
http://cwiki.apache.org/GMOxDOC20/using-some-of-ejb-30-functionalities.html

Limitations:
- No support for MDBs.
- @EJB injection of ejb-refs and ejb-local-refs (Jetty)

- Web Services (Apache CXF)
Supported:
- Deployng a JAX-WS based POJO service that leverages annotations to simplify the creation of the Web service.
- Deploying a traditional JAX-RPC based POJO service
- @Resource Annotations are fully supported provided they are also defined in the web.xml deployment descriptor
- The @Resource WebServiceContext annotation is fully supported.

Simple Web Services example available at: http://cwiki.apache.org/GMOxDOC20/simple-web-service-with-jax-ws.html

Limitations:
- No EJB support for Web Services in 2.0-M2
- <service-ref> elements in the deployment descriptor or @WebServiceRef annotations are not processed.
- Dynamically generated WSDL returned via ?wsdl request might be missing some information.
- Dynamic clients (obtained using the javax.xml.ws.Service API) might not always work.

- JSTL 1.2
Applications that use JSTL must add a dependency in their deployment plan to the jstl jar.
Specifically:
<dep:dependencies>
<dep:dependency>
<dep:groupId>jstl</dep:groupId>
<dep:artifactId>jstl</dep:artifactId>
</dep:dependency>
</dep:dependencies>
Alternatively, the jstl jar can be included in the application's WEB-INF/lib directory.

- JSF 1.2 - Not yet supported

-JSF applications will deploy and start but will not execute yet. An updated MyFaces package
will remedy this in the near future.

Features and functions supported in the 2.0-M1 Release

- Full Sun JDK 5.0+ (J2SE 1.5.0+)
- Servlet 2.5 (Tomcat)
- JSP 2.1 (Tomcat)
- JSP Debug 1.0 (Tomcat)
- Servlet 2.5 (Jetty)
- JSP 2.1 (Jetty - via Jasper)
- JSP Debug 1.0 (Jetty)
- JSF 1.2 (JSF applications won't execute)
- JSTL 1.2
- Common Annotations 1.0
- JAF 1.1
- JavaMail 1.4
- EJB 3.0 (JPA only)
- JTA 1.1
- JMS 1.1
- JACC 1.1

Installing & Starting Geronimo

To install, simply unpack the .zip (Windows) or tar.gz (Unix) file containing
Geronimo.

If you wish to modify the default ports that Geronimo will use, edit the file
<geronimo_home>/var/config/config.xml

Geronimo comes with batch and script files to control server start and stop
functions. To see usage examples simply type geronimo.bat or geronimo.sh
command as appropriate for your platform. It is necessary to set JAVA_HOME to
the copy of your Sun 5 JDK/JRE prior to executing the command.

Here is an example to set JAVA_HOME:

export JAVA_HOME=<JDK/JRE_home>

To see the available command options type:

<geronimo_home>/bin/geronimo.sh
or
<geronimo_home>\bin\geronimo.bat

The command will display help text instructing you as to how to start and stop
the Geronimo server.

If you prefer to start the server without a script file you can simply type:

java -jar <geronimo_home>/bin/server.jar

Once the server has started, you can access the Geronimo Administration Console
at http://localhost:8080/console/ . The default user name is "system" and the
default password is "manager".

Administration Console Security Configuration

The default administration user/password for the Geronimo Administration Console
and command line deployment tool is system/manager. You can change these defaults
directly from the Geronimo Administration Console by accessing Security -> Console
Realm and change the user name and password from the Console Realm Users portlet.

As an alternative, you can make the same changes by editing the

<geronimo_home>/var/security/users.properties and
<geronimo_home>/var/security/groups.properties files.

Deploying Applications

Geronimo comes with deploy scripts and batch files to deploy J2EE modules or applications. You can use the scripts or simply invoke the executable jar by running the following command (note that you need to start Geronimo first):

<geronimo_home>/bin/java -jar deployer.jar deploy my-web-app.war [deploy plan]

You will need to use the username "system" and password "manager" unless you customized those as described above. The deployment plan argument is optional -- you can pack a deployment plan into the application module, provide it on the command line, or in some cases omit it entirely.

You can also use the "Login" command to avoid entering a user name and password every time you use the deploy tool

For more information on the commands and options supported by the deploy tool, run from within the Geronimo directory <geronimo_home>/bin:

java -jar deployer.jar help [command]

You can also graphically deploy applications and resources via the Geronimo Administration Console available at http://localhost:8080/console/

Other Deployment Options

As an alternative to the command-line deployer, you can copy application modules into the <geronimo_home>/deploy/ directory and the hot deployer service will deploy them automatically. The command-line deployer has some advantages, as it will output any deployment errors to its own console rather than just the server log.

Additionally, Geronimo provides a Maven plugin that can deploy applications to Geronimo as part of a Maven build.

Configuration

Most configuration attributes can be updated in the <geronimo_home>/var/config/config.xml file. The attributes most likely to be changed are already included in the supplied config.xml file, while others may need to be added manually.

Certification Status

Apache Geronimo v2.0-M3, being a MILESTONE release is not yet certified.

Known Issues and Limitations

** Bug
* [GERONIMO-2643] - Stack trace (due to amq) while shutting down Geronimo
* [GERONIMO-2650] - JSP 2.1 error in Jetty/Tomcat
* [GERONIMO-2664] - Servlet Filter Error
* [GERONIMO-2728] - Exception while removing directory module from the console
* [GERONIMO-2771] - GlassFish specs are being included in 2.0-M2 builds
* [GERONIMO-2780] - Tomcat fails to accept certificate alias
* [GERONIMO-2785] - Sun changed the (final?) javaee 5 schemas and we need to generate new xmlbeans classes for them

* [GERONIMO-2788] - Temporarily remove yoko jars from the lib/endorsed directory until we have corba support
* [GERONIMO-2792] - Assembly plans still contain references to non-existant corba-sun configs
* [GERONIMO-2793] - Build failure in geronimo-openejb-builder
* [GERONIMO-2795] - TransactionSynchronizationRegistry needs to register lots of interposed syncs per tx.
* [GERONIMO-2800] - Connector Lazy Activation leaks managed connections
* [GERONIMO-2803] - JAXWS build failure due to missing axis2-jaxws-api-SNAPSHOT.jar
* [GERONIMO-2819] - openejb client ejb ref lookup don't use the right classloader.
* [GERONIMO-2823] - System class loader should not be searched for resource META-INF/geronimo-dependency.xml by AbstractRepository.getDependencies
* [GERONIMO-2829] - car plugin needs to start a new kernel for each invocation
* [GERONIMO-2832] - Need to work harder to register our corba UtilDelegateImpl
* [GERONIMO-2834] - ejb method intf names have wrong case for jacc EjbMethodPermission constructor
* [GERONIMO-2855] - Missing persistence-context-ref while deploying DayTrader on 2.0-trunk
* [GERONIMO-2867] - javax.mail.Service.connect() not using port from transport URLName()
* [GERONIMO-2873] - OOME's and file-locking problems on Windows
* [GERONIMO-2881] - Client-Transaction explicit-versions.properties does not include all dependencies.
* [GERONIMO-2892] - Undeploy locks files on Windows

** Improvement
* [GERONIMO-1716] - Add usage of SimpleEncryption to PropertiesFileLoginModule and Admin Console
* [GERONIMO-2454] - Upgrade xerces to version 2.8.1
* [GERONIMO-2758] - Upgrade to Howl 1.0.2
* [GERONIMO-2862] - Detect Yoko classes are not endorsed
* [GERONIMO-2865] - EJB WS update: pass env. in standard way

** New Feature
* [GERONIMO-2651] - Servlet annotations are not supported

** RTC
* [GERONIMO-2638] - Improve ModuleBuilder and ConfigurationBuilder interfaces to replace use of JarFile

** Task
* [GERONIMO-2804] - implement jsf 1.2 support
* [GERONIMO-2805] - add tribes clustering support for tomcat

** Wish
* [GERONIMO-2178] - JSP Taglibs accessed via a global context

7.4. RELEASE-NOTES-2.0-M4.TXT

This page last changed on 4 21, 2008 by JAGUG.

```
Release Notes -- Apache Geronimo -- Version 2.0 - Milestone 4

Geronimo URLs
-------------
Home Page:     http://geronimo.apache.org/
Downloads:     http://geronimo.apache.org/downloads.html
Documentation: http://geronimo.apache.org/documentation.html
Mailing Lists: http://geronimo.apache.org/mailing.html
Source Code:   http://geronimo.apache.org/svn.html
Bug Tracking:  http://issues.apache.org/jira/browse/GERONIMO
Wiki:          http://cwiki.apache.org/geronimo


IMPORTANT
---------
This is a Milestone release, that means that is not the final version of
Apache Geronimo v2.0 Take a look at "Known Issues and Limitations" section for
further details.

Updated Information
-------------------
Please see http://cwiki.apache.org/GMOxDOC20/release-notes-20-m3txt.html for the latest information
on this release.

System Requirements
-------------------
You need a platform that supports the Sun JDK 5.0+ (J2SE 1.5.0+).

Most testing has been done on Linux, Mac OS X, and Windows.

Significant Changes in the 2.0-M4 Release
-----------------------------------------


Functional Characteristics for 2.0-M4
-------------------------------------


Significant Changes in the 2.0-M3 Release
-----------------------------------------
Overall this new milestone release is more stable and includes additional and enhanced support
to the features integrated in the previous releases.

Some of the new features added in this release include:

- JavaServer Faces 1.2 (via Apache MyFaces v1.2)
- Common Annotations for the Java Platform (partial support)
- Enterprise JavaBeans 3.0 (via Apache OpenEJB 3.0, partial support)
- Web Services Implementation (partial support)

*See "Functional Characteristics" section below for more details on the new functions.

Functional Characteristics for 2.0-M3
-------------------------------------

- Web Services (Apache Axis2)
- JAX-WS 2.0 limited support (POJO only)
- Web Services Metadata 2.0 limited support

- Web Services (Apache CXF)
- JAX-WS 2.0 support for POJO and EJB
- Web Services Metadata 2.0 limited support
```

- Limitations:
- Redeploy does not work on Windows. Undeploy leaves behind files in the $geronimo_home/repository which are locked.
This prevents the same configuration from being deployed again. The server has to be stopped and the files deleted before the configuration can be deployed again.

Features and functions supported in the 2.0-M2 Release

- EJB 3.0 (via Apache OpenEJB project)
Supported:
- JPA (Custom Provider, App-managed, Container-managed) (Also supported in 2.0-M1)
- POJO as a Business Interface (both local and remote)
- POJO Session EJBs
- geronimo-openejb.xml file not required unless you have geronimo specific information to configure
- Deployment of annotated beans (@Stateful and @Stateless)
- Injection via deployment descriptor
- @Resource injection for env-entries, resource-refs, message-destinations, service-refs, most resource-env-refs
- @EJB injection of ejb-refs and ejb-local-refs (Tomcat)
- @PersistenceContext injection
- @PersistenceUnit injection
- JNDI references to the ejb
- JNDI references from the ejb
- Transaction support
- Legacy component (i.e. home) interfaces on a Pojo session bean
- Xml-based *and* annotation-based injection for ejbs, except for message-destinations, or SessionContext when the field or setter is not named setSessionContext
- References to business interfaces, local or remote, from a servlet or an ejb
- References to home interfaces, local or remote, from a servlet or an ejb
- Extended JNDI and DI types
- Deploy and Undeploy

Simple EJB 3.0 example application available at:
http://cwiki.apache.org/GMOxDOC20/using-some-of-ejb-30-functionalities.html

Limitations:
- No support for MDBs.
- @EJB injection of ejb-refs and ejb-local-refs (Jetty)

- Web Services (Apache CXF)
Supported:
- Deployng a JAX-WS based POJO service that leverages annotations to simplify the creation of the Web service.
- Deploying a traditional JAX-RPC based POJO service
- @Resource Annotations are fully supported provided they are also defined in the web.xml deployment descriptor
- The @Resource WebServiceContext annotation is fully supported.

Simple Web Services example available at: http://cwiki.apache.org/GMOxDOC20/simple-web-service-with-jax-ws.html

Limitations:
- No EJB support for Web Services in 2.0-M2
- <service-ref> elements in the deployment descriptor or @WebServiceRef annotations are not processed.
- Dynamically generated WSDL returned via ?wsdl request might be missing some information.
- Dynamic clients (obtained using the javax.xml.ws.Service API) might not always work.

- JSTL 1.2
Applications that use JSTL must add a dependency in their deployment plan to the jstl jar.
Specifically:
<dep:dependencies>
<dep:dependency>

```
<dep:groupId>jstl</dep:groupId>
<dep:artifactId>jstl</dep:artifactId>
</dep:dependency>
</dep:dependencies>
Alternatively, the jstl jar can be included in the application's WEB-INF/lib directory.


- JSF 1.2 - Not yet supported
-JSF applications will deploy and start but will not execute yet. An updated MyFaces package
will remedy this in the near future.

Features and functions supported in the 2.0-M1 Release
------------------------------------------------------

- Full Sun JDK 5.0+ (J2SE 1.5.0+)
- Servlet 2.5 (Tomcat)
- JSP 2.1 (Tomcat)
- JSP Debug 1.0 (Tomcat)
- Servlet 2.5 (Jetty)
- JSP 2.1 (Jetty - via Jasper)
- JSP Debug 1.0 (Jetty)
- JSF 1.2 (JSF applications won't execute)
- JSTL 1.2
- Common Annotations 1.0
- JAF 1.1
- JavaMail 1.4
- EJB 3.0 (JPA only)
- JTA 1.1
- JMS 1.1
- JACC 1.1

Installing & Starting Geronimo
------------------------------
To install, simply unpack the .zip (Windows) or tar.gz (Unix) file containing
Geronimo.

If you wish to modify the default ports that Geronimo will use, edit the file
<geronimo_home>/var/config/config.xml

Geronimo comes with batch and script files to control server start and stop
functions.  To see usage examples simply type geronimo.bat or geronimo.sh
command as appropriate for your platform.  It is necessary to set JAVA_HOME to
the copy of your Sun 5 JDK/JRE prior to executing the command.

Here is an example to set JAVA_HOME:

export JAVA_HOME=<JDK/JRE_home>

To see the available command options type:

<geronimo_home>/bin/geronimo.sh
or
<geronimo_home>\bin\geronimo.bat

The command will display help text instructing you as to how to start and stop
the Geronimo server.

If you prefer to start the server without a script file you can simply type:

java -jar <geronimo_home>/bin/server.jar

Once the server has started, you can access the Geronimo Administration Console
at http://localhost:8080/console/ . The default user name is "system" and the
default password is "manager".
```

Administration Console Security Configuration

The default administration user/password for the Geronimo Administration Console and command line deployment tool is system/manager. You can change these defaults directly from the Geronimo Administration Console by accessing Security -> Console Realm and change the user name and password from the Console Realm Users portlet.

As an alternative, you can make the same changes by editing the <geronimo_home>/var/security/users.properties and <geronimo_home>/var/security/groups.properties files.

Deploying Applications

Geronimo comes with deploy scripts and batch files to deploy J2EE modules or applications. You can use the scripts or simply invoke the executable jar by running the following command (note that you need to start Geronimo first):

```
<geronimo_home>/bin/java -jar deployer.jar deploy my-web-app.war [deploy plan]
```

You will need to use the username "system" and password "manager" unless you customized those as described above. The deployment plan argument is optional -- you can pack a deployment plan into the application module, provide it on the command line, or in some cases omit it entirely.

You can also use the "Login" command to avoid entering a user name and password every time you use the deploy tool

For more information on the commands and options supported by the deploy tool, run from within the Geronimo directory <geronimo_home>/bin:

```
java -jar deployer.jar help [command]
```

You can also graphically deploy applications and resources via the Geronimo Administration Console available at http://localhost:8080/console/

Other Deployment Options

As an alternative to the command-line deployer, you can copy application modules into the <geronimo_home>/deploy/ directory and the hot deployer service will deploy them automatically. The command-line deployer has some advantages, as it will output any deployment errors to its own console rather than just the server log.

Additionally, Geronimo provides a Maven plugin that can deploy applications to Geronimo as part of a Maven build.

Configuration

Most configuration attributes can be updated in the <geronimo_home>/var/config/config.xml file. The attributes most likely to be changed are already included in the supplied config.xml file, while others may need to be added manually.

Certification Status

Apache Geronimo v2.0-M4, being a MILESTONE release is not yet certified.

Known Issues and Limitations

7.5. RELEASE-NOTES-2.0-M5.TXT

This page last changed on 4 21, 2008 by JAGUG.

```
Release Notes -- Apache Geronimo -- Version 2.0 - Milestone 5

Geronimo URLs
-------------
Home Page:     http://geronimo.apache.org/
Downloads:     http://geronimo.apache.org/downloads.html
Documentation: http://geronimo.apache.org/documentation.html
Mailing Lists: http://geronimo.apache.org/mailing.html
Source Code:   http://geronimo.apache.org/svn.html
Bug Tracking:  http://issues.apache.org/jira/browse/GERONIMO
Wiki:          http://cwiki.apache.org/geronimo


IMPORTANT
---------
This is a Milestone release, that means that is not the final version of
Apache Geronimo v2.0 Take a look at "Known Issues and Limitations" section for
further details.

Updated Information
-------------------
Please see http://cwiki.apache.org/GMOxDOC20ja/release-notes-20-m5txt.html for the latest
information
on this release.

System Requirements
-------------------
You need a platform that supports the Sun JDK 5.0+ (J2SE 1.5.0+).  Other Java VMs
should work as well.

Most testing has been done on Linux, Mac OS X, and Windows.

Significant Changes in the 2.0-M5 Release
-----------------------------------------
Added annotations:
@Resource WebServiceContext injection in OpenEJB

Injection of Message Destination References:
ENC: resource-env-ref: javax.jms.Queue
ENC: resource-env-ref: javax.jms.Topic
ENC: message-destination-ref: javax.jms.Queue
ENC: message-destination-ref: javax.jms.Topic
ENC: resource-ref: java.net.URL
ENC: resource-ref: javax.mail.Session
javax.ejb.ActivationConfigProperty
javax.ejb.Timeout
javax.annotation.Resource: for message-destination-ref


Supported features
------------------
Most of the Java EE functionality is complete.  We are continuing to test the server as
well as related components.  Certain portions of the WebServices stacks are not fully
functional and are continuing.

Installing & Starting Geronimo
------------------------------
To install, simply unpack the .zip (Windows) or tar.gz (Unix) file containing
Geronimo.

If you wish to modify the default ports that Geronimo will use, edit the file
<geronimo_home>/var/config/config.xml
```

Geronimo comes with batch and script files to control server start and stop functions. To see usage examples simply type geronimo.bat or geronimo.sh command as appropriate for your platform. It is necessary to set JAVA_HOME to the copy of your Sun 5 JDK/JRE prior to executing the command.

Here is an example to set JAVA_HOME:

```
export JAVA_HOME=<JDK/JRE_home>
```

To see the available command options type:

```
<geronimo_home>/bin/geronimo.sh
or
<geronimo_home>\bin\geronimo.bat
```

The command will display help text instructing you as to how to start and stop the Geronimo server.

If you prefer to start the server without a script file you can simply type:

```
java -jar <geronimo_home>/bin/server.jar
```

Once the server has started, you can access the Geronimo Administration Console at http://localhost:8080/console/ . The default user name is "system" and the default password is "manager".

Administration Console Security Configuration

The default administration user/password for the Geronimo Administration Console and command line deployment tool is system/manager. You can change these defaults directly from the Geronimo Administration Console by accessing Security -> Console Realm and change the user name and password from the Console Realm Users portlet.

As an alternative, you can make the same changes by editing the <geronimo_home>/var/security/users.properties and <geronimo_home>/var/security/groups.properties files.

Deploying Applications

Geronimo comes with deploy scripts and batch files to deploy J2EE modules or applications. You can use the scripts or simply invoke the executable jar by running the following command (note that you need to start Geronimo first):

```
<geronimo_home>/bin/java -jar deployer.jar deploy my-web-app.war [deploy plan]
```

You will need to use the username "system" and password "manager" unless you customized those as described above. The deployment plan argument is optional -- you can pack a deployment plan into the application module, provide it on the command line, or in some cases omit it entirely.

You can also use the "Login" command to avoid entering a user name and password every time you use the deploy tool

For more information on the commands and options supported by the deploy tool, run from within the Geronimo directory <geronimo_home>/bin:

```
java -jar deployer.jar help [command]
```

You can also graphically deploy applications and resources via the Geronimo Administration Console available at http://localhost:8080/console/

Other Deployment Options

As an alternative to the command-line deployer, you can copy application
modules into the <geronimo_home>/deploy/ directory and the
hot deployer service will deploy them automatically. The command-line deployer
has some advantages, as it will output any
deployment errors to its own console rather than just the server log.

Additionally, Geronimo provides a Maven plugin that can deploy applications to
Geronimo as part of a Maven build.

Configuration

Most configuration attributes can be updated in the
<geronimo_home>/var/config/config.xml file. The attributes most likely to be
changed are already included in the supplied config.xml file, while others may
need to be added manually.

Certification Status

There are current efforts for attaining certification, however this milestone release is not yet
certified.

Known Issues and Limitations

Release Notes - Geronimo - Version 2.0-M5

** Bug
* [GERONIMO-250] - Connector tries to commit after connection error
* [GERONIMO-348] - Invalid module path or references in plan should result in failed deployment or
warning
* [GERONIMO-603] - IllegalArgumentException when deploying WebApp containing a url-pattern of /* in
security-constraint
* [GERONIMO-807] - Better handling for system log viewer portlet render requests   *
[GERONIMO-1076] - Deployment of an unpacked EAR defining an unpacked nested module does not work
* [GERONIMO-1082] - Deployment doesn't ensure queries defined for ejbSelect methods
* [GERONIMO-1285] - Deployer does not list all modules that have been stopped
* [GERONIMO-1631] - NoSuchConfigException when restarting app after undeploying
* [GERONIMO-1657] - CommandSupport doesn't bubble up the exception. Prints stacktrace.
* [GERONIMO-1658] - NPE in the ConnectionTrackingCoordinator with Roller
* [GERONIMO-1761] - Change geronimo-util module to geronimo-crypto, give credit where credit is due
* [GERONIMO-1786] - JMS Listeners for protocols activeio, peer and openwire fail to start
* [GERONIMO-1917] - repository doesn't deal well with case insensitive file systems
* [GERONIMO-1945] - Console's web application list does not show WARs that are deployed inside EARs
* [GERONIMO-2025] - Undeploy and redeploy with no version leaves dangling entries in config.xml
* [GERONIMO-2028] - Plugin export errors don't stop process, but cause it to fail much later
* [GERONIMO-2105] - When redeploying with no version number, new entries in config.xml break
* [GERONIMO-2114] - Timer already cancelled error when attempting to grab a connection from a pool
* [GERONIMO-2176] - deploy time validations lost in openejb rewrite
* [GERONIMO-2236] - keystore portlet - providing a null or incorrect password on edit unlock can
only be recovered with server recycle
* [GERONIMO-2238] - Can't copy a MultiParentClassLoader
* [GERONIMO-2246] - Why resource-env-ref:admin-object-module?
* [GERONIMO-2286] - app client plan still uses Strings for dependency Module IDs
* [GERONIMO-2290] - Percent complete goes over 100% when installing configurations
* [GERONIMO-2339] - Empty auth-constraint tag in web app security-constraint does not prevent
access to resource
* [GERONIMO-2363] - Console: create new pool using wizard, cannot use "show plan" button for any XA
database, even derby
* [GERONIMO-2377] - deploying a new datasource with the same name does not indicate any problem in
the console
* [GERONIMO-2416] - ProxyMethodInterceptor should work with classes that have start,stop methods
* [GERONIMO-2458] - MapEditor does not work
* [GERONIMO-2480] - Plugin installer status sticks on "Searching for X at Y" while downloading

* [GERONIMO-2481] - WebServers portlet: Create/Edit Tomcat Connectors should support editing of all supported connector attributes
* [GERONIMO-2489] - Client builder bug is blocking usage of Daytrader AppClient
* [GERONIMO-2491] - Hibernate passes connections between servlets which we don't support
* [GERONIMO-2501] - Unable to deploy database pools
* [GERONIMO-2533] - Password setup forms should use a confirmation field
* [GERONIMO-2534] - Security realms portlet should validate the realm-name for duplicate name
* [GERONIMO-2548] - GBeanInfo should exclude attributes and operations of java.lang.Object class
* [GERONIMO-2549] - NullPointerException: CommandListConfigurations
* [GERONIMO-2551] - Plugin hits NPE if maven-metadata listed artifact doesn't exist or JAR artifact maven-metadata doesn't exist
* [GERONIMO-2555] - Windows scripts don't work when used from different drive
* [GERONIMO-2559] - cannot stop activemq via kernel shutdown
* [GERONIMO-2560] - Realm added using SecurityRealm portlet does not work
* [GERONIMO-2566] - Creating new listeners for ActiveMQ from JMS Server portlet fails
* [GERONIMO-2579] - TestNG testcases in testsuite broken. JUnit expected. Move to JUnit 4.0 ?
* [GERONIMO-2580] - CorbaRefBuilder inserts ref for java:comp/CORBA that fails when corba gbean is not present.
* [GERONIMO-2585] - KeystorePortlet: Lock keystore throws NullPointerException
* [GERONIMO-2586] - KeystorePortlet: Unlock keystore for availability shows key aliases only when keystore is unlocked for edit
* [GERONIMO-2587] - FileKeystoreInstance.loadKeystoreData() results in inconsistent state if wrong password is supplied
* [GERONIMO-2588] - KeyStorePortlet: Locking and unlocking could use some error and info messages
* [GERONIMO-2591] - Database Pools portlet: Create new pool dependency jar selection problems
* [GERONIMO-2592] - TSSLink doStart() method is not getting called.
* [GERONIMO-2598] - Deploy tool prints useless message if configuration start fails
* [GERONIMO-2599] - deploying RAR leads to message that Geronimo can't find web.xml
* [GERONIMO-2602] - default compile scope is overloaded with 2 meanings in PlanProcessorMojo
* [GERONIMO-2603] - Building 1.2 if there are 2.0 artifacts in the repo results in mostly 2.0 artifacts in the server.
* [GERONIMO-2605] - NPE if exporting plugin for module having dependency on module with no groupId
* [GERONIMO-2609] - On linux and maybe other platform, the geronimo.out log file contains invalid characters
* [GERONIMO-2611] - Configuration should not have duplicates in allServiceParents
* [GERONIMO-2615] - Not enough info when a gbean ref can't be verified during deployment
* [GERONIMO-2619] - Javamail 1.4 spec needs to be using the JAF 1.1 version.
* [GERONIMO-2622] - Implement PolicyContextHandlerSOAPMessage
* [GERONIMO-2623] - Infinite loop in the SMTPTransport code when a socket factory class is used.
* [GERONIMO-2624] - Offline deployer busted
* [GERONIMO-2625] - Geronimo Console: login page prevents using username, password longer than 25 characters for login
* [GERONIMO-2627] - jsr88 classpath is all messed up
* [GERONIMO-2630] - sun j2ee schemas are being redistributed in jsp and servlet specs
* [GERONIMO-2631] - jetty5 builder needs to parameterize the jsp servlet class
* [GERONIMO-2632] - Connection errors can result in infinite loop
* [GERONIMO-2653] - Inconsistencies between jetty6-builder and tomcat6-builder plan.xml files
* [GERONIMO-2656] - for jetty, in servlet-mapping, url patterns must be trimmed.
* [GERONIMO-2657] - Persistence-* builders need improvements
* [GERONIMO-2659] - jspc plugin busted by spec >> provided change
* [GERONIMO-2668] - axis deployer still assumes it's the only one
* [GERONIMO-2674] - The basic CXF integration is not working
* [GERONIMO-2688] - no-pool and xa-transaction with no caching result in connections being closed before the tx ends.
* [GERONIMO-268] - Connection Error handling problems
* [GERONIMO-2693] - Application classloader contains a massive number of duplicate classpath entries
* [GERONIMO-2694] - STATUS file sent out as the Geronimo Weekly Status email needs updating
* [GERONIMO-2699] - Module was not a war
* [GERONIMO-2707] - Cannot deploy app client from ear
* [GERONIMO-2721] - Unable to deploy anything
* [GERONIMO-2737] - Server shutdown time has increased significantly because of ActiveMQ v4
* [GERONIMO-2741] - Info messages are being displayed in the console output
* [GERONIMO-2747] - Error while shutting down Geronimo
* [GERONIMO-2751] - [BUILD BREAK] Module geronimo-axis2
* [GERONIMO-2754] - Server wont start on Revision: 497496

```
* [GERONIMO-2768] - geronimo-j2ee-management_1.1_spec is still using geronimo-ejb_2.1_spec instead
of new ejb_3.0_spec
* [GERONIMO-2771] - GlassFish specs are being included in 2.0-M2 builds
* [GERONIMO-2778] - Deployer should always fill in required <local> or <remote> interfaces elements
in <ejb-ref>
* [GERONIMO-2778] - Deployer should always fill in required <local> or <remote> interfaces elements
in <ejb-ref>
* [GERONIMO-2806] - mail.null.host property not resolved by SMTPTransport class
* [GERONIMO-2809] - TransactionManagerImpl.getTransactionStatus() shouldn't throw on no tx (with
patch)
* [GERONIMO-2811] - got runtimeexception when processing soap request

** Improvement
* [GERONIMO-795] - Extend Portlet skin capabilities to support minimize and maximize
* [GERONIMO-931] - Rename administrative security realm
* [GERONIMO-1265] - Preserve comments added by users in the config.xml file
* [GERONIMO-1277] - Change group-id to org.apache.geronimo
* [GERONIMO-1314] - Provide documentation that helps users make an informed decision as whether to
use Tomcat or Jetty Web Container
* [GERONIMO-1396] - Provide consistent look and feel for table views in the web console across all
portlets
* [GERONIMO-1418] - allow user to specify deployment targets by "nickname"
* [GERONIMO-1431] - Make deploy tool and hot deploy directory work better together
* [GERONIMO-1470] - Our context root settings should take precedence over those from
application.xml
* [GERONIMO-1471] - Connector dependencies
* [GERONIMO-1642] - Deployment plan namespace validation
* [GERONIMO-1716] - Add usage of SimpleEncryption to PropertiesFileLoginModule and Admin Console
* [GERONIMO-1749] - Server Logs portlet - Web Access Log Viewer improvements
* [GERONIMO-1807] - Remove uses of ObjectName from core server
* [GERONIMO-1808] - Replace AbstractName with URI
* [GERONIMO-1829] - Service Plans should allow GBean references by interface (vs. by name)
* [GERONIMO-1842] - Dynamically load jars from the WEB-INF/lib directory
* [GERONIMO-1880] - To Allow configurable password digests during REALM Deployment.
* [GERONIMO-1907] - Deploy command should redeploy if the app is already deployed
* [GERONIMO-1980] - Move Plugin Installer from rmi-naming to j2ee-system
* [GERONIMO-2015] - Let's replace JKS to PKCS12 key store type
* [GERONIMO-2045] - Plugin prerequisites: for DB pools, support DB type/version and table
validation
* [GERONIMO-2127] - Expose the ability to use Select for Update on CMP entity beans
* [GERONIMO-2128] - Allow user to specify the Isolation Level for a CMP bean's SQL access
* [GERONIMO-2387] - Server life cycle log entries should be generated by the server
* [GERONIMO-2401] - Upgrade commons-logging to 1.1
* [GERONIMO-2424] - Add config.xml support for ConfigurationAwareReference
* [GERONIMO-2485] - PersistenceUnitGBean needs a NamespaceDrivenDeployer
* [GERONIMO-2542] - Writing XML schema documentation for geronimo-web in distribution
* [GERONIMO-2589] - Generate explicit-versions.properties transitvily
* [GERONIMO-2597] - Make web service builder optional
* [GERONIMO-2604] - Create a specs pom
* [GERONIMO-2608] - JACC 1.1 support (jsr-115 MR4)
* [GERONIMO-2616] - Move jee5 work from sandbox to trunk
* [GERONIMO-2621] - Exception handling in Console
* [GERONIMO-2634] - Switch to activemq 4.1.0-incubator: include apache incubator repo in our repo
list.
* [GERONIMO-2635] - Upgrade to JavaMail 1.4 and JavaBeans Activation Framework

** New Feature
* [GERONIMO-1035] - tomcat integration should wrap each servlet indiviudally
* [GERONIMO-1293] - Provide tomcat statistics
* [GERONIMO-1488] - externalize sensitive data out of the deployment plans
* [GERONIMO-1638] - Multiple servers sharing the same repo and config store
* [GERONIMO-1686] - Servlet 2.5 and JSP 2.1 api jars for JavaEE 5 work
* [GERONIMO-2153] - Global JNDI
* [GERONIMO-2399] - Support both jetty 5 and jetty 6
* [GERONIMO-2493] - Integrate Axis2 webservices
* [GERONIMO-2530] - Enable configuration of Axis global request/response handlers
```

* [GERONIMO-2640] - Expose "development" and "modificationTestInterval" attributes in Jasper as configuration attributes for webcontainer
* [GERONIMO-2727] - Enable axis2-deployer in jetty and tomcat jee5 assemblies
* [GERONIMO-2777] - Expose JAX-WS mandatory MessageContext properties
* [GERONIMO-2814] - Add a second repository to Geronimo
* [GERONIMO-2908] - Deploy JAX-WS services without webservices.xml
* [GERONIMO-2943] - Add ModuleBuilderExtensions to web module builders
* [GERONIMO-2985] - webservice: provide test cases for different protocol combinations
* [GERONIMO-2988] - Axis2: needs to support optional wsdl file
* [GERONIMO-2990] - Axis2: Supports OASIS XML Catalogs 1.1 specification to be used to resolve web service description document (wsdl/xsd)
* [GERONIMO-2991] - Axis2: supports service-ref overwrite
* [GERONIMO-2992] - Axis2: invoke the EJB3 container with the invoking method and method parameters

** Task
* [GERONIMO-2581] - Update Version Numbers of Dependent Projects for Geronimo Version 2.0
* [GERONIMO-2601] - Remove the "Old Keystore" portlet
* [GERONIMO-2670] - Update geronimo plugin repository version
* [GERONIMO-2700] - JSR-88 1.2 Tasklist (JEE5 Deployment)
* [GERONIMO-2703] - JSR-252 Tasklist (JSF 1.2)
* [GERONIMO-2710] - Annotations Support Tasklist
* [GERONIMO-2827] - Java EE 5 application client support
* [GERONIMO-2889] - add support for resource injection in JSF Managed Beans
* [GERONIMO-2920] - Upgrade to tomcat 6.0.10 (stable)
* [GERONIMO-2936] - Complete hookup of CORBA TSS-link elements.
* [GERONIMO-2948] - Update yoko version to the 1.0-incubating-SNAPSHOT
* [GERONIMO-2953] - Remove obsolete TSSLinkBuilder references from deployer plans.
* [GERONIMO-3036] - upgrade to dwr 1.1.3
* [GERONIMO-3039] - upgrade to Scout snapshot

** Test
* [GERONIMO-2620] - Need to create javamail 1.4 versions of the provider and mail jars
* [GERONIMO-2755] - JSP tests for web-testsuite
* [GERONIMO-2824] - port the handler tests from CXF to Axis2
* [GERONIMO-2954] - Minor update to jaxws-ejb test so that tomcat request.getMethod() returns the correct value (per G2841)
* [GERONIMO-2986] - Provide a testcase to test optional web.xml with WebServices.
* [GERONIMO-2987] - Webservice: need different test for POJO

** Wish
* [GERONIMO-1354] - The var/config.xml file is always re-written even if no attribute changes are made by the user

7.6. RELEASE-NOTES-2.0-M6.TXT

This page last changed on 4 21, 2008 by JAGUG.

```
Release Notes -- Apache Geronimo -- Version 2.0 - Milestone 6

Geronimo URLs
-------------
Home Page:     http://geronimo.apache.org/
Downloads:     http://geronimo.apache.org/downloads.html
Documentation: http://geronimo.apache.org/documentation.html
Mailing Lists: http://geronimo.apache.org/mailing.html
Source Code:   http://geronimo.apache.org/svn.html
Bug Tracking:  http://issues.apache.org/jira/browse/GERONIMO
Wiki:          http://cwiki.apache.org/geronimo


IMPORTANT
---------
This is a Milestone release, that means that is not the final version of
Apache Geronimo v2.0 Take a look at "Known Issues and Limitations" section for
further details.

Updated Information
-------------------
Please see http://cwiki.apache.org/GMOxDOC20/release-notes-20-m6txt.html for the latest information
on this release.

System Requirements
-------------------
You need a platform that supports the Sun JDK 5.0+ (J2SE 1.5.0+).  Other Java VMs
should work as well.

Most testing has been done on Linux, Mac OS X, and Windows.

Significant Changes in the 2.0-M6 Release
-----------------------------------------
All function is complete, starting with 2.0-M6-r1 Geronimo has passed all CTS tests in the Java EE
5.0
Compatibility Test Suite.  Although we have some additional work to do before we release the
official Geronimo v2.0
release this milestone represents a complete and validated Java EE 5.0 platform.


Supported features
------------------
All programming model elements of the Java EE 5.0 Specification are available.  Some of the non-
specification related
elements such as clustering are still being worked on.

Installing & Starting Geronimo
------------------------------
To install, simply unpack the .zip (Windows) or tar.gz (Unix) file containing
Geronimo.

If you wish to modify the default ports that Geronimo will use, edit the file
<geronimo_home>/var/config/config.xml

Geronimo comes with batch and script files to control server start and stop
functions.  To see usage examples simply type geronimo.bat or geronimo.sh
command as appropriate for your platform.  It is necessary to set JAVA_HOME to
the copy of your Sun 5 JDK/JRE prior to executing the command.

Here is an example to set JAVA_HOME:

export JAVA_HOME=<JDK/JRE_home>
```

To see the available command options type:

<geronimo_home>/bin/geronimo.sh
or
<geronimo_home>\bin\geronimo.bat

The command will display help text instructing you as to how to start and stop
the Geronimo server.

If you prefer to start the server without a script file you can simply type:

java -Djava.endorsed.dirs=<geronimo_home>/lib/endorsed -javaagent:<geronimo_home>/bin/jpa.jar -jar
<geronimo_home>/bin/server.jar

Once the server has started, you can access the Geronimo Administration Console
at http://localhost:8080/console/ . The default user name is "system" and the
default password is "manager".

Administration Console Security Configuration

The default administration user/password for the Geronimo Administration Console
and command line deployment tool is system/manager. You can change these defaults
directly from the Geronimo Administration Console by accessing Security -> Console
Realm and change the user name and password from the Console Realm Users portlet.

As an alternative, you can make the same changes by editing the
<geronimo_home>/var/security/users.properties and
<geronimo_home>/var/security/groups.properties files.

Deploying Applications

Geronimo comes with deploy scripts and batch files to deploy J2EE modules or
applications. You can use the scripts or simply invoke the executable jar by
running the following command (note that you need to start Geronimo first):

<geronimo_home>/bin/java -jar deployer.jar deploy my-web-app.war [deploy plan]

You will need to use the username "system" and password "manager" unless you
customized those as described above. The deployment plan argument is
optional -- you can pack a deployment plan into the application module, provide
it on the command line, or in some cases omit it entirely.

You can also use the "Login" command to avoid entering a user name and password
every time you use the deploy tool

For more information on the commands and options supported by the deploy tool,
run from within the Geronimo directory <geronimo_home>/bin:

java -jar deployer.jar help [command]

You can also graphically deploy applications and resources via the Geronimo
Administration Console available at http://localhost:8080/console/

Other Deployment Options

As an alternative to the command-line deployer, you can copy application
modules into the <geronimo_home>/deploy/ directory and the
hot deployer service will deploy them automatically. The command-line deployer
has some advantages, as it will output any
deployment errors to its own console rather than just the server log.

Additionally, Geronimo provides a Maven plugin that can deploy applications to
Geronimo as part of a Maven build.

Configuration

Most configuration attributes can be updated in the
<geronimo_home>/var/config/config.xml file. The attributes most likely to be
changed are already included in the supplied config.xml file, while others may
need to be added manually.

Certification Status

Apache Geronimo v2.0-M6 (Tomcat assembly with CXF for WebServices and
OpenJPA for persistence) have passed 100% SUN's Java Enterprise Edition 5.0
Certification Test Suite.

We continue to strive towards certification on the other assemblies.

Known Issues and Limitations

Release Notes - Geronimo - Version 2.0-M6

** Sub-task
* [GERONIMO-2702] - Annotation changes
* [GERONIMO-2704] - Testcase changes
* [GERONIMO-2705] - Testcase changes
* [GERONIMO-2837] - @Resource/@Resources annotation support
* [GERONIMO-2901] - Annotation support for ResourceRefBuilder Naming Builder
* [GERONIMO-2934] - Support annotation processing for all Module/Naming builders
* [GERONIMO-2958] - MyFaces annotation support

** Bug
* [GERONIMO-348] - Invalid module path or references in plan should result in failed deployment or
warning
* [GERONIMO-1285] - Deployer does not list all modules that have been stopped
* [GERONIMO-1413] - Console needs to set JSP and Servlet contentType to UTF-8
* [GERONIMO-1767] - bad group for console has no error page that allows user logout/correction
* [GERONIMO-2598] - Deploy tool prints useless message if configuration start fails
* [GERONIMO-2655] - Jetty 6 assemblies do not include sample AJP Connectors
* [GERONIMO-2744] - App client needs to be able to log out
* [GERONIMO-2768] - geronimo-j2ee-management_1.1_spec is still using geronimo-ejb_2.1_spec instead
of new ejb_3.0_spec
* [GERONIMO-2773] - cannot create a new jetty http connector from console
* [GERONIMO-2841] - Valve reports request method as GET even though POST request was made
* [GERONIMO-3038] - h.tld file not getting properly generated to conform to the latest web-
jsptaglibrary_2_1.xsd schema
* [GERONIMO-3073] - More security bugs in openejb integration
* [GERONIMO-3099] - Check if <web-service-address> for ejb starts with "/"
* [GERONIMO-3103] - GBeanOverride.writeXml parseException
* [GERONIMO-3114] - Update setup of java.endorsed.dirs to always have ours before the JVM
* [GERONIMO-3120] - Sample Java EE 5 applications fail to deploy without application.xml file
* [GERONIMO-3121] - JPA Queries created outside a tx need to know about and close their
EntityManagers
* [GERONIMO-3124] - Reminder: Switch Axis2 stack back to axis2-saaj from Sun SAAJ Impl
* [GERONIMO-3125] - AdminObjectRefBuilder does not handle UserTransaction resource-env-refs
* [GERONIMO-3127] - ERROR [DriverDownloader] Unable to download driver properties
* [GERONIMO-3128] - Error in Geronimo.bat blocks its usage for Minimal assemblies, due to no bin
\jpa.jar
* [GERONIMO-3132] - jpa system needs the urls of the jars/dirs with enhanceable classes.
* [GERONIMO-3133] - jpa system needs to be able to set default properties in deployed persistence
units.
* [GERONIMO-3134] - jpa jndi references can refer to gbeans in the web module, not just in parent
modules. This needs to work.
* [GERONIMO-3135] - Allow enlisting into tx marked for rollback only
* [GERONIMO-3136] - App client needs jpa

* [GERONIMO-3137] - Servlets must rollback uncommitted transaction when the service method exits
* [GERONIMO-3138] - @Resource refs to ORBs doesn't seem to work
* [GERONIMO-3139] - Update Copyright year in Admin Console and Welcome app
* [GERONIMO-3141] - potential deadlock in TransformerCollection
* [GERONIMO-3142] - Geronimo / Jetty fails to startup under Sun Java 1.6 update 1
* [GERONIMO-3145] - Sample EJB 3.0 application deployment fails
* [GERONIMO-3146] - run-as support for tomcat servlets
* [GERONIMO-3147] - Unable to deploy anything from the command line (trunk)
* [GERONIMO-3151] - DeploymentUtil.recursiveDelete(File,Collection) is returning a collection of File objects, whereas most builders are expecting Strings
* [GERONIMO-3152] - Cannot redeploy or undeploy/deploy webconsole-tomcat car on geronimo-tomcat6-jee5
* [GERONIMO-3153] - Minimal assembly is missing corba spec classes
* [GERONIMO-3155] - geronimo-tomcat6-minimal assembly has doubled in size from 17MB to 35MB in the past month
* [GERONIMO-3156] - Web security parsing can result in wrong unchecked WebResourcePermissions
* [GERONIMO-3163] - Class-Path in MANIFEST.MF causes DayTrader deployment failed
* [GERONIMO-3165] - Extended persistence contexts need to use some openejb facilities so they can get created when a stateful bean instance is created
* [GERONIMO-3167] - When JSF renders, fields are shifted
* [GERONIMO-3169] - Several configs are using a hardcoded schema version=1.1, instead of ${geronimoSchemaVersion}
* [GERONIMO-3170] - User supplied JPDA_OPTS are never used in the geronimo script for Windows
* [GERONIMO-3172] - remove obsolete JDB_OPTS reference from geronimo.bat
* [GERONIMO-3173] - port conflict with multiple server instances
* [GERONIMO-3174] - Can't start server with eclipse plugin
* [GERONIMO-3176] - ArrayIndexOutOfBound happens for WebServices
* [GERONIMO-3177] - exclude-unlisted-classes needs to prevent adding any path info to the persistenceUnitInfo
* [GERONIMO-3179] - Allow specifying default datasources for legacy cmp
* [GERONIMO-3180] - Override xml parsing function in JRE
* [GERONIMO-3181] - service endpoint "all" permissions are computed from the local home, not the service endpoint interface
* [GERONIMO-3183] - fix offline deployment in minimal configurations
* [GERONIMO-3185] - PostConstruct/PreDeploy specified in xml can have class default to component's type.
* [GERONIMO-3188] - Annotation scanning needs to look at superclasses
* [GERONIMO-3189] - Add missing geronimo-web.xml for jsp-examples and servlet-examples
* [GERONIMO-3191] - Can not login the remote Geronimo server on Linux
* [GERONIMO-3195] - OpenJPA graduated from incubator, move to 1.0.0-SNAPSHOT
* [GERONIMO-3197] - Axis2: don't call wsgen tool if BindingType is HTTP_Binding
* [GERONIMO-3198] - Revert back to xalan 2.6.0 to correct a TCK issue
* [GERONIMO-3201] - Tomcat doesn't need to process the bits of web.xml for jndi and security, we handle that.
* [GERONIMO-3203] - Wrong Main Class fro "geronimo debug"
* [GERONIMO-3205] - Fix JSP compiler errors during build
* [GERONIMO-3206] - Create and include a new patched version of Tomcat
* [GERONIMO-3208] - In-place deployment fails when renaming file
* [GERONIMO-3220] - In-place deployment fails during discovery of web services
* [GERONIMO-3235] - Javascript error when creating database pools using the admin console wizard

** Improvement
* [GERONIMO-1431] - Make deploy tool and hot deploy directory work better together
* [GERONIMO-2485] - PersistenceUnitGBean needs a NamespaceDrivenDeployer
* [GERONIMO-3011] - Adjust port configuration for running multiple instances of geronimo
* [GERONIMO-3034] - Use jettys NIO based connectors by default.
* [GERONIMO-3123] - Create a template/var directory for creating additional instances of geronimo
* [GERONIMO-3126] - Upgrade to released OpenJPA 0.9.7 which passed JPA TCK
* [GERONIMO-3164] - Axis2: support bindingtype overwrite from wsdl to annotation
* [GERONIMO-3171] - java 1.6 compile fix for geronimo-naming mock DataSource class
* [GERONIMO-3186] - Show schema names along with table names in the db portlet

** New Feature
* [GERONIMO-2735] - Add property substitution capability in the config.xml and plan files

** Task

```
* [GERONIMO-2703] - JSR-252 Tasklist  (JSF 1.2)
* [GERONIMO-2710] - Annotations Support Tasklist
* [GERONIMO-3207] - Created 2.0-M6 Branch

** Test
* [GERONIMO-3143] - Testcase changes for CORBA
```

7.7. What's new in Geronimo v2-M2

This page last changed on 4 21, 2008 by JAGUG.

This section provides a quick overview of the are several additions and fixes included in this milestone release.

More integration

With Apache Geronimo you get to choose the Web container (Tomcat or Jetty), the Web services implementation (Axis2 or CXF) and the persistence framework (OpenJPA or Cayene).

EJB 3

One of the major additions in this new milestone release is the initial integration of EJB 3 (JSR220) through the Apache OpenEJB project. The EJB 3 specification now offers a totally redesigned Enterprise JavaBeans architecture, reduced complexity and simplified APIs to make the enterprise application development easier.

Some remarks about this new specification are:

- Simpler to use compared to the previous specification.
- Metadata annotations targeted to reduce the number of classes and interfaces needed as well as to eliminate the need for EJB deployment descriptors.
- Default configuration and behaviors now available, "configuration by exception" approach taken whenever possible.
- Dependency injection.
- EJB components interfaces and home interfaces are no longer required for session beans.
- Business interfaces (plain Java interfaces) can used instead of EJBObject, EJBLocalObject or java.rmi.Remote interfaces for session beans.
- Simplified bean types.
- Simplified Java Persistence API.
- Persistent entities no longer require interfaces.
- Dynamic query capabilities and support for native SQL queries.

Refer to the JSR 220: Enterprise JavaBeans™ 3.0 Specification Request for further details.

Integration in Geronimo

As mentioned earlier, this specification makes it to Geronimo via the Apache OpenEJB project. The current development status of the OpenEJB project can be seen at OpenEJB 3 RoadMap page.

Web Services Technologies

The following Web Services technologies have been integrated in this milestone release:

- Streaming API for XML (JSR 173)
  - Woodstox implementation
- Java Architecture for XML Binding (JAXB) 2.0
  - GlassFish implementation

Also, a basic JAX-WS 2.0 support is provided in this milestone. Only Servlet-based JAX-WS services are supported. The following limitations exist at the time of this release:

- <service-ref> elements in the deployment descriptor or @WebServiceRef annotations are not processed.
- @Resource annotations (except for WebServiceContext type) are not processed.
- Dynamically generated WSDL returned via ?wsdl request might be missing some information.
- Dynamic clients (obtained using the javax.xml.ws.Service API) might not always work.

The JAX-WS support in this release is provided by the Apache CXF project. Apache Axis2 integration is in progress and will be provided in later releases.

8. RELEASE-NOTES-2.0.2.TXT


```
######## -- Apache Geronimo -- 2.0.2 #####

Geronimo URLs
-------------
######:     http://geronimo.apache.org/
######:     http://geronimo.apache.org/downloads.html
######: http://geronimo.apache.org/documentation.html
########: http://geronimo.apache.org/mailing.html
######:   http://geronimo.apache.org/svn.html
########:  http://issues.apache.org/jira/browse/GERONIMO
Wiki:          http://cwiki.apache.org/geronimo

####
-------------------
####################http://cwiki.apache.org/GMOxDOC20/release-notes-202txt.html ##########

######
-------------------
Sun JDK 5.0 ## (J2SE 1.5.0 ##) ######################### Java VM #############

########Linux#Mac OS X#####Windows ##########


2.0.2 ############
---------------------------------------

- MEJB #############
* Management EJB (MEJB) ########admin (######)#####mejb-admin (##/######) #################

- EJB ### JNDI ####
* EJB ### JNDI #############
Geronimo 2.0.1 ############ JNDI ############Geronimo 2.0.2 #########
################ http://cwiki.apache.org/GMOxDEV/client-jndi-names.html ##########

- CA (###) ################
* Internet Explorer ###################################

- #######
* ###### Geronimo 2.0.2 ########################################

####
--------------------
Apache Geronimo v2.0 ##SUN # Java Enterprise Edition 5.0 Certification Test Suite #########
############"############" ###############

############
-------------
Apache Geronimo v2.0 ##5#################################################

##########################:

################:
- Apache Geronimo with Tomcat Web ##### ( AXIS2 Web #########OpenJPA #######)#
- Apache Geronimo with Jetty Web ######( CXF Web #########OpenJPA #######)#

###############:
- Little-G with Tomcat Web ##### (######)#
- Little-G with Jetty Web ##### (######)#
- Micro-G (Geronimo ############################)#
##: #################Java EE 5 ##########################################(CXF Web #### ##### Tomcat
Web #########) ###############

##########
```

```
-----------------
Java EE 5.0 ############################
####################################

Geronimo ####
-----------------
##### Geronimo 2.0.2 ##############http://cwiki.apache.org/GMOxDEV/building-apache-geronimo.html #
#########

Geronimo ####### & ##
----------------------------
###########Geronimo ### .zip (Windows) #### tar.gz (Unix) #################

Geronimo ######################## <geronimo_home>/var/config/config-substitutions.properties ######
####

Geronimo ############################################
###################### geronimo.bat #### geronimo.sh #############
##############JAVA_HOME ### Sun JDK/JRE 5 ##################

####JAVA_HOME ##########:

export JAVA_HOME=<JDK/JRE_home>

############################################:

<geronimo_home>/bin/geronimo.sh
####
<geronimo_home>\bin\geronimo.bat

########Geronimo ##########################################

##########################<geronimo_home> ########################

java -Djava.endorsed.dirs=lib/endorsed -javaagent:bin/jpa.jar -jar bin/server.jar

#############Geronimo ####### http://localhost:8080/console/ ##########
####### "system"############# "manager" ###

Windows ########:
Windows #################255########################
"My Documents" # "Program Files" ###############################################################
###################### <geronimo_home> ################

#########
----------------------
Geronimo ########################################/#######system/manager ###
Geronimo ##########Security -> Console Realm #######Console Realm Users ##########################
############################

###########
<geronimo_home>/var/security/users.properties
###
<geronimo_home>/var/security/groups.properties
############################

var/security ###############Geronimo ####################################

users.properties ########################
#######Geronimo ######################
########################
#############users.properties ################
########################

############################################################
```

```
#############
---------------------
Geronimo ###J2EE ###################################################
################################## jar ###############(##: ### Geronimo ###############):

<geronimo_home>/bin/java -jar deployer.jar deploy my-web-app.war [###########]

################################ "system" ### ##### "manager" ##############
####################### -- ###########################################################################
##########

########################################"Login" ############

##############################################Geronimo directory <geronimo_home>/bin ###############
###:

java -jar deployer.jar help [####]

http://localhost:8080/console/ ####### Geronimo ##########################################

#################
----------------------
#######################<geronimo_home>/deploy/ ###########################################################
###############
#########################################################################

####Geronimo ##Maven ################### Geronimo ############# Maven ##############

##
-------------
############<geronimo_home>/var/config/config.xml ##############
######################### config.xml ##############################################

MEJB ################## mejb-admin #################
####'system' #######/###############<geronimo_home>/var/security/groups.properties ###############
###:

mejb-admin=system


##############
---------------------------

EAR ##### lib ############### Spring # jar ####### EAR ############ Web ####################
##############Web ################ Spring ####################Web ######################## Spring #
jar #################

#######
---------------

######
[GERONIMO-3461] - Disable MEJB gbean in the default assemblies until G3456 is fixed

##
[GERONIMO-1746] - Cannot override default configurationFile value in Log4jService GBean
[GERONIMO-2188] - When oracle wrapper is used, commits are not immediately committed to oracle
database
[GERONIMO-2567] - Remote admin of server using deployer.jar fails to connect
[GERONIMO-2775] - Enabling web statistics collection for jetty fails from the admin console
[GERONIMO-2851] - LDAP view in the console
[GERONIMO-2884] - JNDI Name Not Correct
[GERONIMO-2925] - Key used for encryption same for all server instances
[GERONIMO-3208] - In-place deployment fails when renaming file
[GERONIMO-3248] - Extraneous WARN messages during deployment of resource-env-refs in EJB jar
[GERONIMO-3265] - Spring stale version in 2.0-M6-rc1
[GERONIMO-3310] - Geronimo is not finding message-destination elements in ejb-jar.xmls
```

[GERONIMO-3317] - "has not been enhanced" error when invoking an EJB 2.1 Entity Bean
[GERONIMO-3324] - Could not scan module for TLD files: illegal char (space)
[GERONIMO-3326] - ClassLoader memory leak caused by OpenJPA
[GERONIMO-3329] - javaVirtualMachine param for StartServerMojo in geronimo-maven-plugin doesn't work
[GERONIMO-3348] - java.lang.NoSuchMethodError in org.springframework.context.i18n.LocaleContextHolder
[GERONIMO-3363] - ArrayList thread safe problem in OpenJPA
[GERONIMO-3376] - can't customize tomcat's session manager behavior per web context
[GERONIMO-3380] - Derby embedded database pool created from console doesn't work
[GERONIMO-3383] - "Connect" button on LDAP Viewer portlet just listens on port "1389" ,not 10389
[GERONIMO-3386] - Nothing returned when clicking on "Save" button at creating Tomcat NIO HTTPs Connector
[GERONIMO-3388] - LoginKerberosTest relies on kerberos not being installed... so it can fail on windows?
[GERONIMO-3390] - Server Logs portlet - Web Access Log Viewer is broken
[GERONIMO-3405] - Generating WSDL files are runtime leaves files in use on Windows
[GERONIMO-3409] - Several geronimo-web.xml files still use a 2.0-SNAPSHOT depend. Upgrade to geronimo.components vers=2.0.1
[GERONIMO-3411] - injection target entries are not added to existing env-entry entries
[GERONIMO-3412] - Upgrade geronimo.components to released 2.0.1 levels
[GERONIMO-3414] - servlet.destroy() not called on POJOWebServiceServlet
[GERONIMO-3420] - Remote deploy of an EAR without an application.xml plan fails
[GERONIMO-3421] - ClassFinder classloader problems cause deployer to hang
[GERONIMO-3426] - The Holder object contains duplicated injections
[GERONIMO-3435] - Axis2: unable to create a service-ref without wsdl using a generic Service class
[GERONIMO-3437] - Axis2: serviceimplClass being null caused NPE at invoke in JavaBeanDispatcher
[GERONIMO-3439] - geronimo-openejb-2.0.xsd not packaged under schema directory!
[GERONIMO-3446] - Improperly configured JRE_HOME or JAVA_HOME environment variables can cause server failure
[GERONIMO-3452] - Stateless Session EJBs cannot contain a remove() method
[GERONIMO-3456] - Make MEJB security configurable
[GERONIMO-3458] - include Dojo information in LICENSE and NOTICE files
[GERONIMO-3459] - Build break due to OpenEJB Rev572863 changes
[GERONIMO-3464] - A webapp with <init-param> fails deployment
[GERONIMO-3465] - Default log level needs to be tweaked to provide more useful information to users
[GERONIMO-3473] - CA Helper app should support submitting Certificate Requests from Internet Explorer
[GERONIMO-3474] - Use released openjpa 1.0.0
[GERONIMO-3475] - expose thread pool size in config.xml/config-substitutions.properties
[GERONIMO-3477] - Transaction recovery broken for resource adapter
[GERONIMO-3481] - Offline deployer throws BIND Exception when port 1099 is in use
[GERONIMO-3484] - openejb-deployer should not require openejb to be running
[GERONIMO-3489] - Deployment problems caused by file deletion failures
[GERONIMO-3494] - HTTP GET is not handled correctly for JAX-WS services
[GERONIMO-3499] - Expose ConnectionTimeout configuration in tomcat web connector to the user
[GERONIMO-3505] - JettyAJP13Connector in geronimo-jetty6-minimal isn't using config-substitutions.properties
[GERONIMO-3508] - Still cannot build branches/2.0 without building OpenEJB first, due to XBean depends
[GERONIMO-3514] - NullPointerException in EjbModuleBuilder.addResourceAdapterMDBInfos

##
[GERONIMO-2964] - Cannot specify the Tomcat work directory for a web application
[GERONIMO-3378] - Enhance debugability of Deployer by logging errors and more data on debug
[GERONIMO-3391] - Use the config-substitutions.properties for the Deployer remoteDeployAddress settings
[GERONIMO-3401] - Stop of module from "System Modules" in console should warn user of destructive action
[GERONIMO-3423] - Ensure that users can change the ejb jndi name format
[GERONIMO-3507] - Expose all TomcatConnector attributes in config.xml
[GERONIMO-3511] - Upgrade to TranQL 1.1 released MySQL and PostgresSQL
[GERONIMO-3513] - Update XBean versions to new 3.2 release

###
[GERONIMO-3521] - plugin catalog for 2.0.2

9. サンプル・アプリケーション

This page last changed on 4 23, 2008 by JAGUG.

さらなるサンプルは こちら からどうぞ。

This page last changed on 4 23, 2008 by JAGUG.

Harmony 上での Geronimo の実行

この文章では、Apache Harmony を JVM として利用して Geronimo v2.0.2 を実行する方法と、既知の問題について説明します。

この文章の構成は以下のとおりです。

Geronimo の構成

JNDI 向けに Harmony の RMI レジストリー・プロバイダーを利用 するための config.xml の調整

Harmony では `org.apache.harmony.jndi` に JNDI providers のパッケージ・ツリーがあり、Sun Java の `com.sun.jndi` のものとは違います。ですので、Geronimo に RMI レジストリー・プロバイダーがどこにあり、どのようにアクセスするのかを指定しなければいけません。

そのために、`var/config/config.xml` ファイルを編集し、NamingProperties GBean の構成を以下のとおりにしてください。

```
<gbean name="NamingProperties">
<attribute
name="namingFactoryInitial">org.apache.harmony.jndi.provider.rmi.registry.RegistryContextFactory</
attribute>
<attribute name="namingFactoryUrlPkgs">org.apache.harmony.jndi.provider</attribute>
<attribute name="namingProviderUrl">rmi://${ServerHostname}:${NamingPort + PortOffset}</attribute>
</gbean>
```

JNDI 向けに Harmony の RMI レジストリー・プロバイダーを利用 するためのデプロイヤーの構成

Geronimo のデプロイヤーは `var/config/config.xml` ファイルを利用しないので (上記を確認してください)、JNDI の構成にシステム・プロパティを与える必要があります。例えば以下のとおりです。

```
java -
Djava.naming.factory.initial=org.apache.harmony.jndi.provider.rmi.registry.RegistryContextFactory
-Djava.naming.factory.url.pkgs=org.apache.harmony.jndi.provider -jar bin/deployer.jar ...
```

Harmony の構成

DRLVM の利用

DRLVM (Harmony VM) を利用して Geronimo を実行します。

Harmony はまた、IBM J9 VM もサポートしていますが、これでは Geronimo は動きません。それは、Geronimo は `sun.misc.Unsafe` の実装クラスとしての `java.util.concurrent` パッケージをあてにしていますが、IBM J9 VM にはこれが無いからです。

その他の問題

GERONIMO-2014 - Geronimo uses outdated version of ApacheDS
GERONIMO-2015 - Let's replace JKS to PKCS12 key store type
GERONIMO-2113 - Geronimo doesn't start if restarted using another JDK
GERONIMO-2128 - Allow user to specify the Isolation Level for a CMP bean's SQL access

MX4J/commons-logging の問題

GERONIMO-2595 - Hardcoded MX4J logger in org.apache.geronimo.kernel.log.GeronimoLogging class
HARMONY-1259 - NoClassDefFoundError while working with MX4J loggers

この問題はもう発生しないように見えます。

テストで発生した問題

GERONIMO-1805 - org.apache.geronimo.directory.RunningTest hangs on BEA Jrockit VMs
GERONIMO-1826 - Naming tests might not work on non-Sun VMs
GERONIMO-1832 - Non-public Sun classes dependencies in tests
GERONIMO-1833 - Non-public Sun classes dependencies in tests
GERONIMO-1840 - NamingPropertiesTest is not compatible with non-Sun VMs
GERONIMO-2055 - RunningTest is not compatible with non-Sun VMs

9.2. 新しいサンプルの生成

This page last changed on 4 23, 2008 by JAGUG.

簡単確実にGeronimoのサンプル・アプリケーションを作るために、先ずmvn アークタイプを使ってプロジェクトの雛型を作ります。mvn はApache Manvenプロジェクトのコマンドです。サンプルをつくるためには、Mavenのダウンロードと構成が必要です。Mavenはインターネット接続を必要とするので、mven_home/conf/setting.xmlのプロキシーの設定に注意して下さい。

この雛型を使ってサンプルを作る手順は下記の通りです。

1. samples-pom.zip をダウンロードして、ディレクトリに保存します。以下、samples-project ディレクトリに保存したものとします。このディレクトリに、zipファイルを解凍します。
2. samples-project/samples ディレクトリにて下記のコマンドを実行し、新しいサンプルを作ります。mySample はあなたの作るサンプルの名前に変更して下さい。
   `mvn -Pcreate -DsampleName=mySample -DsampleVersion=myVersionNumber`
   おめでとうございます !!! これでmySampleというサンプル の雛型ができました。

   > 必要に応じて samples/mySample のmvn モジュールを追加/削除して下さい。

3. このサンプルをビルドするには、samples/mySample ディレクトリにて下記のコマンドを実行します。

   ```
   mvn install
   mvn install site
   ```

   これでサンプルのバイナリーが samples/mySample/mySample-ear-myVersion.ear というファイル名で生成されます。samples/mySample/docs にはJavadocとsource xrefが生成されます。
4. このバイナリーは稼働中のGeronimoにデプロイできます。
5. あなたのサンプルに合わせて、http://cwiki.apache.org/GMOxDOC20/sample-applications.htmlのWikiを編集して、ページを追加して下さい。
6. samples/mySample/docs/wiki に、wikiのhtmlファイルを保存して下さい。
7. mySample ディレクトリにて、`mvn clean` を実行後、mySample ディレクトリをzipで圧縮。このzipファイルには、サンプルのソース、Javadoc, source xref、wiki のドキュメントが格納されます。
8. このzipファイルをwikiに添付して下さい。

This page last changed on 4 23, 2008 by JAGUG.

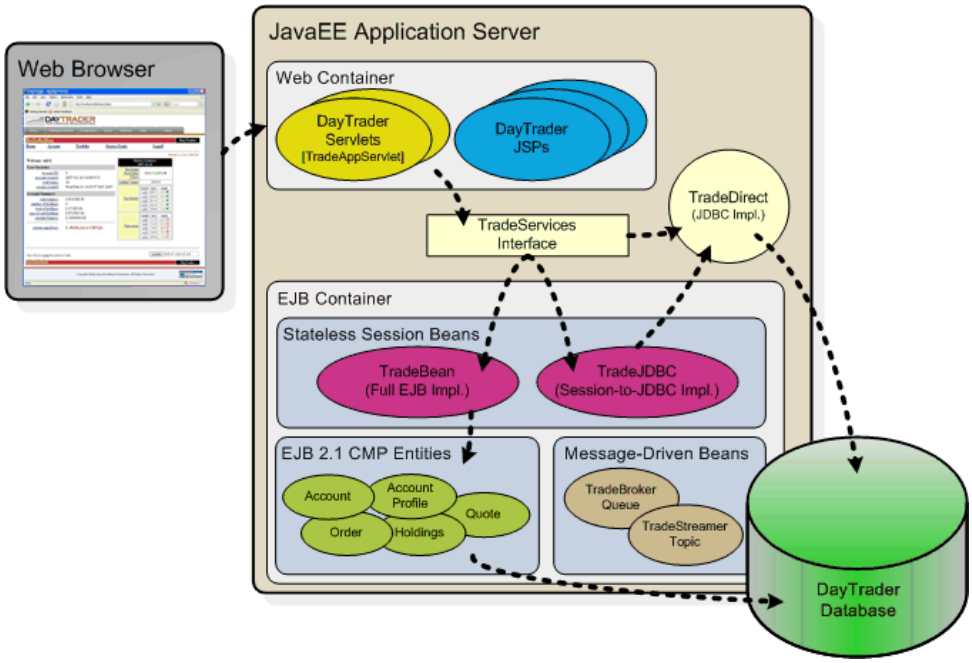
Apache DayTrader ベンチマークサンプル

DayTrader とは

DayTrader は株式のオンライン・トレード・システムを基にしたベンチマーク・アプリケーションです。元々は IBM によって開発された Trade Performance Benchmark Sample で、2005年に Apache Geronimo コミュニティに寄贈されました。DayTrader には、ログインやポートフォリオの表示、株価情報の検索、株の売買の機能があります。Mercury LoadRunner や Rational Performance Tester、Apache JMeter といった Web の負荷ツールで DayTrader に現実世界に近い負荷をかけることで、様々なベンダから提供されている Java Platform, Enterprise Edition (Java EE) アプリケーションサーバの性能の計測や比較をすることが出来ます。負荷の測定に加え、DayTrader には Java EE のコンポーネントやデザインパターンの機能や性能のテストに使えるような、基本的な機能が備わっています。

アプリケーション・アーキテクチャ

DayTrader は Java EE の中核となる技術によって構成されています。それは、プレゼンテーション層のためのサーブレットや JavaServer Pages（JSP）、バックエンドのビジネス層や永続化層のための Java database connectivity（JDBC）、Java Message Service（JMS）、Enterprise JavaBean（EJB）、Message-Driven Bean（MDB） といった技術です。以下の図は、DayTrader のアーキテクチャの概要を表しています。



プレゼンテーション層

プレゼンテーション層は、いくつかのサーブレットと JSP からなり、ほぼ Model-View-Controller (MVC) パターンに沿っています。TradeAppServlet はクライアントからのリクエストを処理する中心的なサーブレットであり、目的のビジネスロジックを呼び出し、適切な JSP に要求を転送します。それ以外のいくつかのサーブレットと JSP は、DayTrader のランタイムオプションの構成やサポート用データベースのメンテナンスに利用します。

ビジネス層と永続化層

ビジネス層と永続化層は、DayTrader の大部分を構成します。TradeServices インターフェースは、業務処理の中心部分を定義します。例えば、register や login、getHoldings、buy、completeOrder、logout などです。DayTrader は3つの異なる実装を提供しており、これらはよく使われる3つの Java EE デザインパターンに対応しています。これらの実装について、以下で説明します。ユーザは、構成用のページで Runtime Mode を変更することで、これらを切り替えられれます。

実装	詳細
TradeDirect (デフォルト)	Pattern: Servlet-to-JDBC Runtime Mode: Direct TradeDirect クラスは、サポートするデータベースに対するCRUD（create、read、update、delete）操作を、JDBCのコードを使って直接行います。データベース接続やコミット、ロールバックの処理はコード内で手動で管理されます。2フェーズコミットを行う場合は、JTAのユーザトランザクションを使って手動で行います。
TradeJDBC	Pattern: Servlet-to-SessionBean-to-JDBC Runtime Mode: Session Direct TradeJDBC ステートレスセッションビーンは、TradeDirectクラスのラッパーです。TradeDirect がJDBCの処理や接続を処理するのに対し、セッションビーンは全てのトランザクション処理を制御します。この実装はJava EE のデザインパターンとして非常に一般的なものです。
TradeBean	Pattern: Servlet-to-SessionBean-to-EntityBean Runtime Mode: EJB TradeBean ステートレスセッションビーンは Caontainer Managed Persistence (CMP) エンティティビーンを使ってビジネスオブジェクトを処理します。これらのオブジェクトの状態は、アプリケーションサーバのEJBコンテナによって完全に管理されます。

ビジネス層のその他のコンポーネントとしては、Java Messaging Service (JMS) があります。DayTrader において、JMS は2つの目的で使われています。非同期での株式の売買と株価の更新処理です。これらの処理について、以下の表で詳細を説明します。

処理	詳細
非同期の注文処理	株の売買の操作が行われる際、クライアント接続から TradeBroker JMS キューに注文の要求が投入されます。TradeBrokerMDB は、キューからメッセージを受信し、売買処理を行います。
株価の更新	株が売買されると、株価が更新されて、それが JMS のトピックに配信されます。TradeStreamerMDB は価格更新のメッセージを処理しますが、それ以上のことは行いません。DayTrader に含まれている TradeStreamer Java EE クライアントを使うことで、株価の動きをリアルタイムに表示することが出来ます。

ビジネスオブジェクトとリレーションシップ

次の図は、データベースのスキーマとビジネスオブジェクトとの関係を表しています。Container managed relationship (CMR) についても、表されています。

Create diagram and add here

ビジネス処理（TradeServices で定義）

前述のように、DayTrader アプリケーションの主な機能は全て、TradeServices インターフェースで定義されます。これらの機能の詳細を、次の表で説明します。

処理	詳細

login	
logout	
buy	
sell	
getMarketSummary	
queueOrder	
completeOrder	
cancelOrder	
orderCompleted	
getOrders	
getClosedOrders	
createQuote	
getQuote	
getAllQuotes	
updateQuotePriceVolume	
getHoldings	
getHolding	
getAccountData	
getAccountProfileData	
updateAccountProfile	
register	
resetTrade	

ユーザインターフェースの操作

DayTrader のJSPとサーブレットによるWebインターフェースで、たいていの株取引とポートフォリオ管理用のアプリケーションに備わっている基本的な操作が行えます。それらの操作を契機に、ビジネスロジックと永続化層内の対応する処理が起動されます。次の表は、それぞれのユーザの操作とそれに対する業務処理をまとめたものです。

UIの操作	ビジネス処理のフロー
Register 登録	
Login ログイン	
View Account アカウントの表示	
View Account Profile アカウントプロファイルの表示	
Update Account Profile アカウントプロファイルの更新	

View Portfolio ポートフォリオの表示	
Sell Holding 売却	
View Quotes 株の表示	
Buy Stock 購入	
Logout ログアウト	

ソースの入手方法

http://svn.apache.org/repos/asf/geronimo/daytrader/trunk

9.4. DBプールをテストするサンプル・アプリケーション

This page last changed on 4 23, 2008 by JAGUG.

J2EEアプリケーションで、アプリケーション・サーバー独自の機能にアクセスして活用すれば、単にJ2EEの機能を利用する以上の力を手に入れられます。アプリケーション・サーバーの拡張機能を作ることさえ可能になります。

このサンプル・アプリケーションはGeronimoに定義されている全てのデータベース・コネクションをリストするものです。既存のスキーマやテーブルもリストできますし、コネクションのいずれかを選んでデータベースへの接続をテストすることもできます。更に、リストされたテーブルのレコードを見ることもできます。

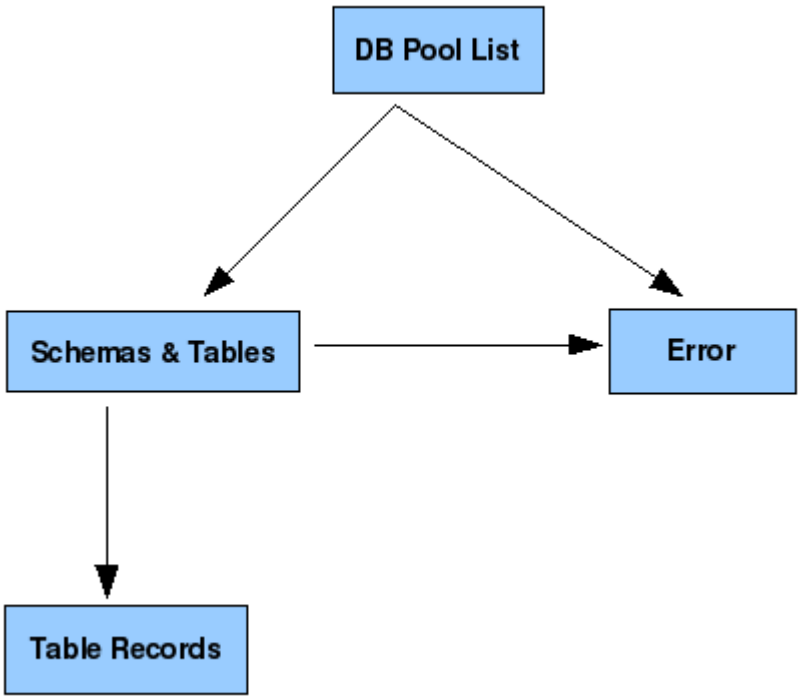
この記事を読み終えれば、貴方はアプリケーションからGeronimo固有のリソースにアクセスし、効率的にそれらを使用できるようになります。

この記事は以下のセクションから構成されています。

アプリケーションの概要

この記事で扱うサンプル・アプリケーションを使えば、Geronimoサーバーにデプロイされたデータベースのコネクション・プールをテストすることができます。この機能はGeronimo管理コンソールの拡張機能と見做すこともできます。なぜなら、現在のバージョンでは、データベース・プールがデプロイされた後にコネクションをテストする機能は含まれていないからです。

以下の図でアプリケーションの流れを説明します。



アプリケーションのウエルカム・ページはgeronimoアプリケーション・サーバーにデプロイされているデータベースのコネクション・プールのリストを表示する掲示板のような役割を果たします。一番目のページからコネクション・プールのテストをすることができます。特定のコネクション・プールがコネクションを取得するためにユーザー名とパスワードを要求する場合は、表示されるポップアップ・ウインドウに詳細を入力してください。データベース中のコネクション・プールに関連付けられたのスキーマとテーブルのリストはSchemas and Tables ページに表示されます。そのページからテーブルの中身にアクセスすることができます。

アプリケーションのコンテンツ

インベントリー・アプリケーションは以下のパッケージから構成されています。

- org.apache.geronimo.samples.dbtester.beans
  - DBManagerBean - （Geronimoカーネルへのアクセスを含む）殆どのアプリケーション・ロジックを扱うアプリケーションの心臓部です。
- org.apache.geronimo.samples.dbtester.web
  - ContentTableServlet - データベースのテーブルからコンテンツを入手し、それらをプレゼンテーション層に渡します。
  - ListTablesServlet - データベース・プールに関連付けられているスキーマとテーブル群のリストを入手します。

下記はこのアプリケーションのwebアプリケーション・ファイルのリストです。

```
|- jsp
|- common_error.jsp
|- popup.jsp
|- table_content.jsp
|- table_list.jsp
|- WEB-INF
|- geronimo-web.xml
|- web.xml
|- index.jsp
```

geronimo-web.xml はWebアプリケーションのクラスローダーへロードされる際の依存関係のリストを定義するものです。今回は、特に依存関係はありません。特定のプロジェクトに関する情報（例えば、モジュールのユニークな識別子や、何らかの依存関係など）は<environment>タグの内側に記述します。後で他のデプロイされるアプリケーションから参照しやすくするために、このモジュールにユニークな識別子のようなものを付与するのは良い考えです。このモジュールはorg.apache.geronimo.samples.dbtester グループに所属しています。<context-root>タグで記述されたパスがこのWebアプリケーションをアクセスする際のURLの最初のセグメントになります。つまり、このアプリケーションへアクセスするためのURLはhttp://<hostname>:<port>/dbtester となります。

geronimo-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.2">

<environment>
<moduleId>
<groupId>${pom.groupId}</groupId>
<artifactId>${pom.artifactId}</artifactId>
<version>${version}</version>
<type>war</type>
</moduleId>
</environment>

<context-root>/dbtester</context-root>

</web-app>
```

最終的なWARの構造は下記のようになります。

```
|- WEB-INF
|- classes
|- org
|- apache
|- geronimo
|- samples
|- dbtester
|- web.xml
|- geronimo-web.xml
```

web.xml でプレゼンテーション層とサービス層を結ぶコントロール層として機能する2つのサーブレットを定義しています。

web.xml

```
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd">

<description>dbtester Servlet Sample</description>
<servlet>
<display-name>ContentTableServlet</display-name>
<servlet-name>ContentTableServlet</servlet-name>
<servlet-class>org.apache.geronimo.samples.dbtester.web.ContentTableServlet</servlet-class>
</servlet>

<servlet>
<display-name>ListTablesServlet</display-name>
<servlet-name>ListTablesServlet</servlet-name>
<servlet-class>org.apache.geronimo.samples.dbtester.web.ListTablesServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>ContentTableServlet</servlet-name>
<url-pattern>/listContent</url-pattern>
</servlet-mapping>

<servlet-mapping>
<servlet-name>ListTablesServlet</servlet-name>
<url-pattern>/listTables</url-pattern>
</servlet-mapping>

<welcome-file-list>
<welcome-file>index.html</welcome-file>
</welcome-file-list>

</web-app>
```

このアプリケーションでの重要なポイントは、どのようにしてGeronimo カーネルへアクセスしデプロイされているデータベース・プールのリストを入手するか、です。このタスクはDBManagerBean クラスで実装されています。

DBManagerBean.java

```
private void init(){
Kernel kernel = KernelRegistry.getSingleKernel();
Set cfList = kernel.listGBeans(new AbstractNameQuery(ConnectionFactorySource.class.getName()));

for(Iterator iterator = cfList.iterator();iterator.hasNext();){

AbstractName name = (AbstractName)iterator.next();
try {
Object rs = kernel.invoke(name, "$getResource", new Object[]{}, new String[]{});

if(rs instanceof javax.sql.DataSource){
DataSource ds = (DataSource)rs;
poolMap.put(name.getArtifact().getArtifactId(), ds);
}
} catch (GBeanNotFoundException e) {
e.printStackTrace();
} catch (NoSuchOperationException e) {
e.printStackTrace();
} catch (InternalKernelException e) {
e.printStackTrace();
} catch (Exception e) {
e.printStackTrace();
}
}
}
```

アプリケーションは、JDBCドライバーが提供しているメタデータを利用すればスキーマとそのスキーマに所属するテーブルのリストを入手することができます。データに関係するレコードを入手し、データベースのコンテンツを表示するためにはResultSetメタデータを使います。以下のコード抜粋はDataSourceからスキーマとテーブルを入手する方法を示しています。

DBManagerBean.java

```
public Map getTableList(String poolName) throws SQLException {

tableMap = new HashMap();

if(poolMap.containsKey(poolName)){
DataSource ds = (DataSource)poolMap.get(poolName);
Connection con = null;
try {

con = ds.getConnection();

DatabaseMetaData metaData = con.getMetaData();
String[] tableTypes = {"TABLE"};
ResultSet rs = metaData.getTables(null, null, null, tableTypes);

while(rs.next()){
String schemaName = rs.getString("TABLE_SCHEM");
String tableName = rs.getString("TABLE_NAME");
ArrayList tableList = null;

if(tableMap.containsKey(schemaName)){
tableList = (ArrayList)tableMap.get(schemaName);
tableList.add(tableName);
}else {
tableList = new ArrayList();
tableList.add(tableName);
}
tableMap.put(schemaName, tableList);
}
} catch (SQLException e) {
throw e;
}finally {
if(con != null){
try {
con.close();
} catch (SQLException e) {
e.printStackTrace();
}
}
}
}

return tableMap;
}
```

利用ツール

このDB Listサンプル・アプリケーションを開発・ビルドするために使用したツールは以下の通りです。

Eclipse

このサンプル・アプリケーションの開発にはEclipse IDEを使用しました。Eclipse IDEはとてもパワフルで広く知られたオープンソースの開発ツールです。Geronimoアプリケーション・サーバー用の統合プラグインも利用可能になっています。Eclipseは http://www.eclipse.org からダウンロードできます。

Apache Maven 2

Mavenはエンタープライズ Javaプロジェクト向けのとてもよく知られたオープンソースのビルドツールで、ビルド・プロセスでの様々な面倒な作業を任せられるように設計されています。Mavenはプロジェクト構造とその中身を記述するという宣言的アプローチを採用しており、その点がAntや従来型のmakeで採用しているタスク・ベースのアプローチとは異なります。この特長により全社的な開発標準の適用を徹底しやすいですし、スクリプトを作成・保守する手間も減らせます。Maven1で採用されていた宣言的なライフサイクル・ベースのアプローチは、伝統的なビルド方法からラジカルに決別するものでしたが、Maven2はこの点を更に推し進めたものになっています。Maven2は http://maven.apache.org からダウンロードできます。

Back to Top

サンプル・アプリケーションのビルドとデプロイ

dbtesterアプリケーションを以下のリンクからダウンロードしてください。

dbtester

上記のzipファイルを解凍すると dbtester ディレクトリーが作られます。

ソース・コード

このサンプルのソースコードはSVNからチェックアウトできます。

svn checkout http://svn.apache.org/repos/asf/geronimo/samples/trunk/samples/dbtester

ビルド

dbtester アプリケーションはユーザーがソースコードからのビルドを行いやすくするためにMaven2の `pom.xml` と共に配布されています。コマンド・プロンプトのウインドウを開き、dbtester ディレクトリーに移動して mvn clean install コマンドを入力してください。すると dbtester の下の target フォルダーに dbtester-war-2.0-SNAPSHOT.war ファイルが作られます。これで dbtester Webアプリケーションを Geronimoアプリケーション・サーバーにデプロイする準備ができました。

デプロイ

サンプル・アプリケーションのデプロイは非常に簡単で、Geronimo管理コンソールから行います。

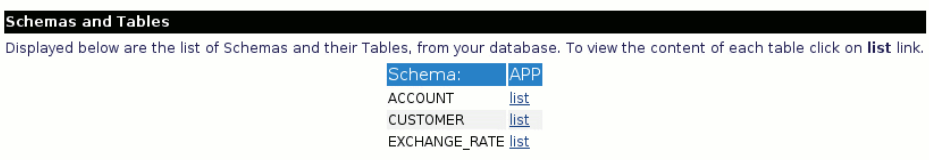
1. Console Navigation パネルから Deploy New リンクをたどります。
2. Archive の入力ボックスで dbtester/target フォルダーの dbtester-war-2.0-SNAPSHOT.war ファイルを指定します。
3. Install ボタンを押し、サーバーにアプリケーションをデプロイします。

Back to Top

サンプル・アプリケーションのテスト

サンプル・アプリケーションをテストするには、ブラウザーを開いて http://localhost:8080/dbtester とタイプします。dbtesterアプリケーションのインデックス・ページがロードされ、Geronimoにデプロイされているデータベース・プールのリストが一覧形式で表示されます。

このサンプル・アプリケーションはデータベース・プールを直接テストすることができます。更に、データベースの中身をリストすることもできます。



サマリー

この記事ではJ2EEアプリケーションからGeronimo関連の機能にアクセスする方法を示しました。これらの機能を作り上げるためのサンプル・アプリケーションのビルド、デプロイ、テストの方法を、順を追って試してみました。このサンプル・アプリケーションはGeronimoにデプロイされたデータベース・コネクションの確認用に使うこともできます。

9.5. EJB サンプル・アプリケーション

This page last changed on 4 23, 2008 by JAGUG.

EJBはJ2EE仕様の重要な技術の一つです。J2EE 1.5 認証サーバーとしてApache Geronimo はEJBコンテナーであるOpenEJBを利用し、幅広くEJB をサポートします。通常の(非EJBの)Javaオブジェクトでビジネスロジックを実装する事も可能ですがEJBを利用すると、そこで発生する課題(スケーラビリティ、ライフサイクル管理、状態管理 etc)EJBが対応してくれます。本稿では、簡単なデータベース・アプリケーションを拡張して、クライアントのアプリケーションがEJBをローカル参照とリモート参照の両方で利用する方法を紹介します。当該アプリケーションは、Apache Derbyを利用します。この記事を通じて、エンタープライズアプリケーションの開発作業を簡単にする方法を学ぶ事ができます。

バンキング・アプリケーションは、2種類のクライアント "バンキング リモート" と"バンキングWeb"から利用されます。これらのクライアントは、それぞれEJBのリモート参照とローカル参照のデモです。両方とも、セッション・ビーンとエンティティ・ビーンにより実装された、共通のビジネス・レイヤーを参照します。ステートレス・セッション・ビーンは、アプリケーション・クライアントとビジネス・エンティティを接続するビジネスサービス・インターフェースとなります。アプリケーション層の全てのビジネス・インタフェースは、エンティティー・ビーンとして実装されています。

本稿を読んだ後、EJBの定義、EJBと異なる参照方法との関連を管理する方法を学事ができ、GeronimoのEJB機能を最大限に利用する事ができるようになります。

本章は、以下のセクションから構成されています。

EJB 機能の概要

EJBの実装はベンダー毎に異なっているかも知れません。下記のリストが、J2EE コンテナーとしてのApache Geornimoの機能です。

- ステートフルとステートレスのセッション・ビーン
- エンティティー・ビーン
- メッセージ・ドリブン・ビーン (MDB)
- RMI-IIOP と、JAXRPCを利用した相互運用性
- Web Serviceとして外部から利用可能な、ステートレスセッション・ビーンとMDB
- Web Service経由のメッセージ送受信のサポート
- EJBとJMXベースのWebサービスの、簡単な準備とホット・デプロイ
- 外部 CORBA オブジェクトからのEJBの利用

アプリケーションの概要

バンキング・アプリケーションについて簡単に説明すると、2種類のクライアン・アプリケーショントがあり、それぞれのアプリケーションの概要は以下の通りです。

1. バンキング・リモート
   簡単なswingのクライアント・アプリケーションで、銀行の管理者向けです。限られた銀行の行員だけが、このアプリケーションを利用可能です。銀行口座の預金残高の照会と更新をすることができます。
2. バンキングWeb
   このアプリケーションは顧客に開放されています。顧客が自分の口座情報を照会する事ができます。加えて、銀行の為替レートを照会する事ができます。Geronimoの環境ではセキュリティ機能を簡単に実装する事ができますが、単純可のために、それぞれのアプリケーションのセキュリティ機能は省略しています。

各クライアントは、共通のビジネス・サービス層を利用します。ビジネス・サービス層の背後には、バンキング・アプリケーションで使われる3つの共通ビジネスエンティティ(Account, Customer, ExchangeRate)が有ります。顧客は一つ以上の口座を持ち、口座は一人の顧客によって所有されます。ExchangeRate は、銀行によって決められる対USドルの為替レートを表現します。

アプリケーションの内容

バンキング・アプリケーションには以下のパッケージとクラスが有ります。

- org.apache.geronimo.samples.bank.ejb
  - Account　- エンティティ・ビーン,DBに格納されている account テーブルの情報を表します。
  - BankManagerFacadeBean - ステートレスセッション・ビーン。クライアント・アプリケーションから利用されるサービスクラスです。
  - Customer -　エンティティ・ビーン, DBに格納されている Customer テーブルの情報を表します。
  - ExchangeRate - エンティティ・ビーン, 対USドルの為替レートを表します。
- org.apache.geronimo.samples.bank.web
  - CustomerServiceServlet - 顧客の口座残高照会のWebリクエストを受け取って、サービスレイヤーを呼び出します。
  - CommonServiceServlet - 為替レート照会のWebリクエストを受け取って、サービスレイヤーを呼び出します。

最終的に、バンキング・アプリケーションはEARファイルとして、アプリケーション・サーバーにデプロイされます。EARファイルの内容は、下記の通りです。

```
|-Bank.ear
|-BankEJB.jar
|-META-INF
|- persistence.xml
|- openejb-jar.xml
|-BankWeb.war
|-jsp
|- customer_info.jsp
|- customer_main.jsp
|- error.jsp
|- exchange_rates.jsp
|- index.jsp
|-WEB-INF
|- web.xml
|- geronimo-web.xml
|- classes
|-META-INF
|- application.xml
|- geronimo-application.xml
```

最初に、アプリケーションのビジネス・サービス層がEJBを利用してどの様に実装されているか、見てみましょう

openejb-jar.xml の定義は、Geronimo 独自のEJB機能と一致します。アプリケーションと共にデータベース・プールをデプロイするので、既存のデータベース・プールへの依存関係は不要です。

openejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<openejb-jar xmlns="http://www.openejb.org/xml/ns/openejb-jar-2.1">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2">
<dep:moduleId>
<dep:groupId>${pom.groupId}</dep:groupId>
<dep:artifactId>${pom.artifactId}</dep:artifactId>
<dep:version>${version}</dep:version>
<dep:type>jar</dep:type>
</dep:moduleId>
<dep:dependencies>
<dep:dependency>
<dep:groupId>org.apache.geronimo.configs</dep:groupId>
<dep:artifactId>openjpa</dep:artifactId>
<dep:type>car</dep:type>
</dep:dependency>
</dep:dependencies>
<dep:hidden-classes/>
<dep:non-overridable-classes/>
</dep:environment>
</openejb-jar>
```

persistence.xmlは、永続化の定義で、BankPool経由でBankDBに接続するためにEntityMangerFactoryによって使われます。<persistent-unit>の名前が、EJBのアノテーションによって参照されます。SynchronizeMappings属性の構成により、データベースに保存されている既存のデータを更新しない様に指定する事ができます。これはデータを削除したくない場合には、重要な指定です。jta-data-source と non-jta-data-source には同じ値を指定する必要が有ります。

persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence         xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/
persistence_1_0.xsd">
<persistence-unit name="BankPU">
<description>Entity Beans for Bank</description>
<provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider>
<class>org.apache.geronimo.samples.bank.ejb.Account</class>
<class>org.apache.geronimo.samples.bank.ejb.Customer</class>
<class>org.apache.geronimo.samples.bank.ejb.ExchangeRate</class>
<properties>
<property name="openjpa.jdbc.SynchronizeMappings" value="false" />
</properties>
<jta-data-source>BankPool</jta-data-source>
<non-jta-data-source>BankPool</non-jta-data-source>
</persistence-unit>
</persistence>
```

web.xml には、特別な指定は何も要りません。web-apにサーブレットの定義と、各サーブレットのマッピングが有るだけです。

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_5.xsd"
version="2.5">

<welcome-file-list>
<welcome-file>/index.html</welcome-file>
</welcome-file-list>

<servlet>
<display-name>CustomerServiceServlet</display-name>
<servlet-name>CustomerServiceServlet</servlet-name>
<servlet-class>org.apache.geronimo.samples.bank.web.CustomerServiceServlet</servlet-class>
</servlet>

<servlet>
<display-name>CommonServiceServlet</display-name>
<servlet-name>CommonServiceServlet</servlet-name>
<servlet-class>org.apache.geronimo.samples.bank.web.CommonServiceServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>CustomerServiceServlet</servlet-name>
<url-pattern>/customer_info</url-pattern>
</servlet-mapping>

<servlet-mapping>
<servlet-name>CommonServiceServlet</servlet-name>
<url-pattern>/exchange_rates</url-pattern>
</servlet-mapping>
</web-app>
```

geronimo-web.xml は、is Geronimo 独自のデプロメイント・プランです。通常、モジュール情報とコンテキスト・ルートを指定します。これで、このアプリケーションを表示するためにURLは、http://localhost:8080/Bank になります。

geronimo-web.xml

```
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1" xmlns:naming="http://
geronimo.apache.org/xml/ns/naming-1.1">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.1">
<dep:moduleId>
<dep:groupId>${pom.groupId}</dep:groupId>
<dep:artifactId>${pom.artifactId}</dep:artifactId>
<dep:version>${version}</dep:version>
<dep:type>war</dep:type>
</dep:moduleId>
<dep:dependencies/>
<dep:hidden-classes/>
<dep:non-overridable-classes/>
</dep:environment>

<context-root>/Bank</context-root>
</web-app>
```

BankPool.xml には、データベースの接続情報を記述します。今回は、Geronimoにビルトインされている、Derbyデータベースを利用します。依存関係には、org.apache.geronimo.configs/system-database//carを指定する必要が有ります。私の場合、これを違う物にしてしまい障害が発生しました。下記のデータベース接続プールのプランは、管理コンソールにより生成されたものです。私が実施した変更は、依存関係を 稼働中のDerbyエンジンであるsystem-databaseにした事だけです。このデータベース・プールの情報は、EARアプリケーションの一部としてデプロイされます。

BankPool.xml

```
<connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.1">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.1">
<dep:moduleId>
<dep:groupId>console.dbpool</dep:groupId>
<dep:artifactId>BankPool</dep:artifactId>
<dep:version>1.0</dep:version>
<dep:type>rar</dep:type>
</dep:moduleId>
<dep:dependencies>
<dep:dependency>
<dep:groupId>org.apache.geronimo.configs</dep:groupId>
<dep:artifactId>system-database</dep:artifactId>
<dep:type>car</dep:type>
</dep:dependency>
</dep:dependencies>
</dep:environment>
<resourceadapter>
<outbound-resourceadapter>
<connection-definition>
<connectionfactory-interface>javax.sql.DataSource</connectionfactory-interface>
<connectiondefinition-instance>
<name>BankPool</name>
<config-property-setting name="Driver">org.apache.derby.jdbc.EmbeddedDriver</config-property-
setting>
<config-property-setting name="UserName">app</config-property-setting>
<config-property-setting name="ConnectionURL">jdbc:derby:BankDB</config-property-setting>
<connectionmanager>
<local-transaction/>
<single-pool>
<max-size>10</max-size>
<min-size>0</min-size>
<match-one/>
</single-pool>
</connectionmanager>
</connectiondefinition-instance>
</connection-definition>
</outbound-resourceadapter>
</resourceadapter>
</connector>
```

geronimo-application.xml には、データベースプールがあるアプリケーションがあり、これもデプロイする必要がある。データベースプールはBankPool.xmlに定義され、デプロイをするために必要なドライバーはtranql-connector-ra-1.3.rar ファイルです。これら2つのファイルはEARファイルのトップレベルに配置します。

geronimo-application.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://geronimo.apache.org/xml/ns/j2ee/application-1.2">
<environment xmlns="http://geronimo.apache.org/xml/ns/deployment-1.2">
<moduleId>
<groupId>${pom.groupId}</groupId>
<artifactId>${pom.artifactId}</artifactId>
<version>${version}</version>
<type>ear</type>
</moduleId>
</environment>
<module>
<connector>tranql-connector-ra-1.3.rar</connector>
<alt-dd>BankPool.xml</alt-dd>
</module>
</application>
```

サンプル・データベース

今回のデモのサンプル・データベースは、Geronimo内蔵のDerbyデータベースを利用します。サンプル・データベースの名前は、BankDBで、CUSTOMER, ACCOUNT, EXCHANGE_RATEという3テーブルから構成されます。各テーブルの定義は以下の通りです。

Table Name	Fields
CUSTOMER	customerId (PRIMARY KEY) name
ACCOUNT	accountNumber (PRIMARY KEY) accountType balance customerId
EXCHANGERATE	rateId (PRIMARY KEY) currency rate

CUSTOMERテーブルは、顧客の情報を保存します。IDと名前を管理するだけのテーブルです。ACCOUNT テーブルは、ユニークな口座番号が主キーです。口座の種類と残高を管理します。Accountテーブルのcustomeridは、Customerテーブルへの外部キー で、口座の所有者の情報を保持します。EXCHANGERATE テーブルは rateId が主キーです。EXCHANGERATEの各レコードは、通過名と支払い時に利用される為替レートを保持します。

利用ツール

バンキング・アプリケーションのビルドとデプロイで利用されるツールは、

Apache Derby

Apache DBサブプロジェクトのApache DerbyはJavaで実装されたリレーショナル・データベースです。フットプリントが非常に小さいので、Javaベースのあらゆるソリューションに容易に埋め込めます。更にこのような埋め込み型のフレームワークだけではなく、Derby Network Serverを使えば、一般的なクライアントーサーバー型のフレームワークをサポートできます。

http://db.apache.org/derby/index.html

Apache Maven 2

Mavenはエンタープライズ Javaプロジェクト向けに広く使われているオープンソースのビルド・ツールで、ビルド作業に伴う大変な仕事を肩代わりしてくれます。MavenはAntやmakeのようなタスク・ベースのアプローチではなく、プロジェクトの構造やコンテンツを記述するという宣言的アプローチを採用しています。これにより全社的な開発標準の遵守が容易になり、ビルド・スクリプトの開発と保守の時間を削減できます。Maven1は宣言的でライフサイクル・ベースのアプローチを採用したことで従来のトラディショナルなビルド・テクニックから劇的に飛躍しました。Maven2では、この点が更に推し進められています。Maven2は以下のURLからダウンロードできます。
http://maven.apache.org

Back to Top

サンプル・アプリケーションの構成、ビルドとデプロイ

以下からバンキング・アプリケーションをダウンロードします。
Bank

圧縮ファイルを復元すると、bank というディレクトリーが作られます。

ソースコード

当サンプルのソースコードはSVNからチェックアウトできます。

svn checkout http://svn.apache.org/repos/asf/geronimo/samples/trunk/samples/bank

構成

当アプリケーションでの構成作業は、データベースの作成とデータベースにアクセスするためのコネクション・プールの定義です。

データベースの作成と操作

Bank DB を作成するには、Apache Geronimoを開始してコンソールにログインしてから、以下のステップを実行してください。

BankDB.sql

```
CREATE TABLE Customer(
customerId VARCHAR(255) PRIMARY KEY,
name VARCHAR(255)
);

CREATE TABLE Account(
accountNumber VARCHAR(255) PRIMARY KEY,
accountType VARCHAR(255),
balance DOUBLE,
customerId VARCHAR(255),
FOREIGN KEY (customerId) REFERENCES customer
);

CREATE TABLE ExchangeRate(
rateId VARCHAR(255) PRIMARY KEY,
currency VARCHAR(255),
rate DOUBLE
);

INSERT INTO Customer(customerId, name) VALUES('12345','John Doe');
INSERT INTO Account(accountNumber, accountType, balance, customerId)
VALUES('1234567890','Savings',1005.35,'12345');
INSERT INTO Account(accountNumber, accountType, balance, customerId)
VALUES('2345678901','Current',999.95,'12345');

INSERT INTO ExchangeRate(rateId,currency,rate) VALUES('001','EURO',0.812);
```

```
INSERT INTO ExchangeRate(rateId,currency,rate) VALUES('002','YEN',111.15);
INSERT INTO ExchangeRate(rateId,currency,rate) VALUES('003','SLR',99.18);
```

1. 左側のConsole Navigationから DB Manager のリンクを選択
2. database nameに BankDB と入力して Create ボタンを押す
3. Use DBフィールドで BankDB を選択
4. テキスト･エディターでbankディレクトリーにある BankDB.sqlを開く
5. BankDB.sql の中身を　SQL Commands テキスト･エリアに貼り付け、Run SQL ボタンを押す

ビルド

コマンド･プロンプトで bank ディレクトリーに移動して、mvn install site コマンドをそのまま入力するとビルドが始まります。結果、bankフォルダーの下に bank-ear-2.0-SNAPSHOT.ear が作成されます。さあ、これでbankアプリケーションをGeronimoアプリケーション･サーバーにデプロイする準備が整いました。

アプリケーションのデプロイ

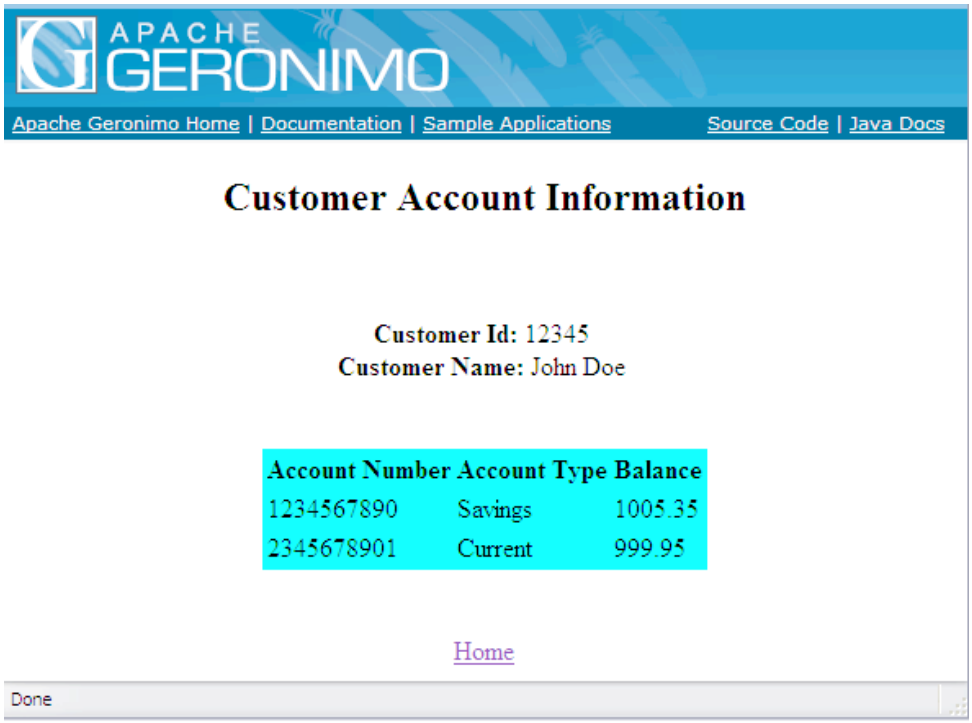
サンプルアプリケーションのデプロイは管理コンソールを使えば非常に簡単です。

1. Console Navigation パネルで Deploy New までスクロール･ダウン
2. Archive 入力ボックスに bank フォルダー上の bank-ear-2.0-SNAPSHOT.ear を指定
3. Install ボタンを押すとアプリケーションがサーバーにデプロイされる.

Back to Top

Banking Web アプリケーション

サンプルのWebアプリケーションをテストするには、ブラウザーを開いて http://localhost:8080/Bank とタイプします。すると、bankアプリケーションのインデックス･ページが表示され、そこには顧客と交換レートの情報を表示するためのダイレクト･リンクがあります。顧客の個々の口座の情報を見るには、DB上で該当する顧客idを入力します。交換レート･ページは交換レート･テーブル上の全ての通貨のリストを表示します。('Bank'はケース･センシティブであることに注意ください)



サマリー

当記事ではApache　GeroimoのEJB機能の使用方法を紹介しました。各種の機能を理解いただくため、アプリケーションをビルド、デプロイし、実行するところまでをステップ・バイ・ステップでご紹介しました。

以下が当記事のハイライトです。

- Apache Geronimo は J2EE 1.5 準拠のアプリケーション・サーバーであり、エンタープライズ・アプリケーション用に必要な全ての機能を持っています。
- セッション・ビーンとエンティティ・ビーンの作成と構成
- 様々な種類のクライアントから、エンタープライズ・レベルのサービスへのアクセス定義

9.6. Geronimo 2.0 でのJNDIの利用方法

This page last changed on 4 23, 2008 by JAGUG.

イントロダクション

Java Naming and Directory Interface (JNDI) はApache Geronimoサーバーのコネクション・プールとのインターフェースです。開発者はこのインターフェースを通してEnterprise Java Beans (EJBs)を含むすべてのJavaオブジェクトにアクセスできます。

Apache Geronimo JNDI naming and Java resource connection pools, Part 1: Data source connections. URL: http://www-128.ibm.com/developerworks/opensource/library/os-ag-jndi1の記事は、JNDIを使ってデータソースのコネクション・プールやJava Messaging Services (JMS)、メール・セッションやURLコネクションへアクセスするための豊富なドキュメンテーションを提供してくれます。

Geronimo 2.0でのJNDIの利用方法

上記は良い記事なのですが、Geronimo 1.x向けに書かれたものでありGeronimo 2.0向けとはいえません。

サンプルのJ2EEエンタープライズ・アプリケーションを動かすためには、記事の中では言及されていない、いくつかの追加の構成作業が必要でした。
注記:この情報の確認環境はWindows XP Professional, Version 5.1.2600 Service Pack 2 Build 2600です。

Customer Service J2EEエンタープライズ・アプリケーションを成功裡にデプロイするには、以下の手順に従ってください。アプリケーションは以下からダウンロードできます。
http://www.ibm.com/developerworks/views/download.jsp?contentid=175237&filename=CustomerService-part1.zip&method=http&locale=worldwide

1) j2ee.jarのclasspathへの追加

WebSphere やWebLogicといったアプリケーション・サーバーのlibディレクトリーに存在するj2ee.jar ファイルをclasspathに追加する必要があります。さもないと、antビルド・スクリプトを実行しようとした際に下記のエラーが発生することでしょう。

```
compile:
[javac] Compiling 9 source files to E:\DOWNLOADS\GERONIMO\CustomerService\build\class
[javac] E:\DOWNLOADS\GERONIMO\CustomerService\src\com\service\customer\ejb\Customer.java:6: package
javax.ejb does not exist
[javac] import javax.ejb.EJBObject;
[javac] ^
```

2) Geronimo/Tomcatディストリビューションの利用

Tomcatディストリビューションを使用してください。というのも、現在のJettyディストリビューション(geronimo-jetty6-jee5-2.0-M1)ではGeronimo Server Console | Services | Database Pools | Database　ウイザードで "Database Types" ドロップダウン・リスト中にアイテムが何も表示されないからです。

3) データベース・プール (CustomerServicePool)の作成
Tomcatディストリビューションであっても、新しいデータベース・プールを作成する際に問題が起きます。例えば新しいデータベース・プールを作成しようとする際に、データベースのタイプで"Derby embedded"を選択してテストしてみます。この操作は動くのですが、実際は新しいデータベース・プールは作成されません。<GERONIMO_HOME>¥var¥log¥geronimo.logファイルには以下のようなエントリーが出力します。

```
14:20:19,687 ERROR [DatabasePoolPortlet] Unable to save connection pool
javax.enterprise.deploy.spi.exceptions.InvalidModuleException: Not supported
at
org.apache.geronimo.deployment.plugin.jmx.JMXDeploymentManager.createConfiguration(JMXDeploymentManager.java:29
at
org.apache.geronimo.console.databasemanager.wizard.DatabasePoolPortlet.save(DatabasePoolPortlet.java:880)
at
org.apache.geronimo.console.databasemanager.wizard.DatabasePoolPortlet.processAction(DatabasePoolPortlet.java:3
at org.apache.pluto.core.PortletServlet.dispatch(PortletServlet.java:229)
at org.apache.pluto.core.PortletServlet.doPost(PortletServlet.java:163)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:713)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:806)
at org.apache.pluto.core.PortletServlet.service(PortletServlet.java:153)
```

```
at
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:290)
at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:206)
at org.apache.catalina.core.ApplicationDispatcher.invoke(ApplicationDispatcher.java:683)
at org.apache.catalina.core.ApplicationDispatcher.doInclude(ApplicationDispatcher.java:585)
at org.apache.catalina.core.ApplicationDispatcher.include(ApplicationDispatcher.java:505)
at org.apache.pluto.invoker.impl.PortletInvokerImpl.invoke(PortletInvokerImpl.java:120)
at org.apache.pluto.invoker.impl.PortletInvokerImpl.action(PortletInvokerImpl.java:68)
at org.apache.pluto.PortletContainerImpl.processPortletAction(PortletContainerImpl.java:164)
at
org.apache.pluto.portalImpl.core.PortletContainerWrapperImpl.processPortletAction(PortletContainerWrapperImpl.j
at org.apache.pluto.portalImpl.Servlet.doGet(Servlet.java:227)
at org.apache.pluto.portalImpl.Servlet.doPost(Servlet.java:267)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:713)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:806)
at
org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:290)
at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:206)
at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:228)
at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:175)
at org.apache.geronimo.tomcat.valve.DefaultSubjectValve.invoke(DefaultSubjectValve.java:56)
at org.apache.catalina.authenticator.AuthenticatorBase.invoke(AuthenticatorBase.java:525)
at org.apache.geronimo.tomcat.GeronimoStandardContext
$SystemMethodValve.invoke(GeronimoStandardContext.java:325)
at
org.apache.geronimo.tomcat.valve.GeronimoBeforeAfterValve.invoke(GeronimoBeforeAfterValve.java:47)
at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:128)
at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:105)
at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:109)
at org.apache.catalina.valves.AccessLogValve.invoke(AccessLogValve.java:542)
at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:212)
at org.apache.coyote.http11.Http11Processor.process(Http11Processor.java:818)
at org.apache.coyote.http11.Http11Protocol$Http11ConnectionHandler.process(Http11Protocol.java:624)
at org.apache.tomcat.util.net.JIoEndpoint$Worker.run(JIoEndpoint.java:445)
at java.lang.Thread.run(Thread.java:595)
```

3.a) デプロイメント・テスト・プランの手動での作成とデプロイ

Geronimoは"Derby embedded"コネクション・プール用のデプロイメント・プランを作成してくれていません。そのため、以下のように自分の手でデプロイメント・プランを作成してデプロイする必要があります。

以下のデプロイメント・プラン・ファイル(CustomerServicePool_test_plan.xml)を作成してください。

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.2">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2">
<dep:moduleId>
<dep:groupId>console.dbpool</dep:groupId>
<dep:artifactId>CustomerServicePool</dep:artifactId>
<dep:version>2.0</dep:version>
<dep:type>rar</dep:type>
</dep:moduleId>
<dep:dependencies>
<dep:dependency>
<dep:groupId>org.apache.derby</dep:groupId>
<dep:artifactId>derby</dep:artifactId>
<dep:version>10.1.3.1</dep:version>
<dep:type>jar</dep:type>
</dep:dependency>
</dep:dependencies>
</dep:environment>
<resourceadapter>
<outbound-resourceadapter>
<connection-definition>
<connectionfactory-interface>javax.sql.DataSource</connectionfactory-interface>
<connectiondefinition-instance>
```

```
<name>CustomerServicePool</name>
<config-property-setting name="Driver">org.apache.derby.jdbc.EmbeddedDriver</config-property-
setting>
<config-property-setting name="ConnectionURL">jdbc:derby:CustomerServiceDatabase</config-property-
setting>
<connectionmanager>
<local-transaction/>
<single-pool>
<max-size>10</max-size>
<min-size>0</min-size>
<match-one/>
</single-pool>
</connectionmanager>
</connectiondefinition-instance>
</connection-definition>
</outbound-resourceadapter>
</resourceadapter>
</connector>
```

<GERONIMO_HOME>¥bin ディレクトリーで以下のコマンドを実行します。

```
deploy --verbose --user system --password manager deploy D:\appsData\GERONIMO\JNDIApp
\CustomerService.ear
```

3.b) CustomerServicePool用のデプロイメント・バージョンの更新

前述のdeployコマンドを実行したら、さらに別の構成作業を行わなくてはなりません。

```
Using GERONIMO_BASE:   C:\GERONIMO
Using GERONIMO_HOME:   C:\GERONIMO
Using GERONIMO_TMPDIR: C:\GERONIMO\var\temp
Using JRE_HOME:        C:\Java\Java150_10\jre
Error: Unable to distribute CustomerService.ear: Unable to create
configuration for deployment

org.apache.geronimo.common.DeploymentException: Unable to create
configuration for deployment

at
org.apache.geronimo.deployment.DeploymentContext.createTempConfiguration(DeploymentContext.java:121)

at
org.apache.geronimo.deployment.DeploymentContext.<init>(DeploymentContext.java:101)

at
org.apache.geronimo.deployment.DeploymentContext.<init>(DeploymentContext.java:81)

at
org.apache.geronimo.j2ee.deployment.EARContext.<init>(EARContext.java:65)

at
org.apache.geronimo.j2ee.deployment.EARConfigBuilder.buildConfiguration(EARConfigBuilder.java:475)

at
org.apache.geronimo.j2ee.deployment.EARConfigBuilder$$FastClassByCGLIB$
$38e56ec6.invoke(<generated>)

at net.sf.cglib.reflect.FastMethod.invoke(FastMethod.java:53)

at
org.apache.geronimo.gbean.runtime.FastMethodInvoker.invoke(FastMethodInvoker.java:38)

at
org.apache.geronimo.gbean.runtime.GBeanOperation.invoke(GBeanOperation.java:124)

at
org.apache.geronimo.gbean.runtime.GBeanInstance.invoke(GBeanInstance.java:820)
```

```
at
org.apache.geronimo.gbean.runtime.RawInvoker.invoke(RawInvoker.java:57)

at
org.apache.geronimo.kernel.basic.RawOperationInvoker.invoke(RawOperationInvoker.java:35)

at
org.apache.geronimo.kernel.basic.ProxyMethodInterceptor.intercept(ProxyMethodInterceptor.java:96)

at
org.apache.geronimo.j2ee.deployment.CorbaGBeanNameSource$$EnhancerByCGLIB$
$44342279.buildConfiguration(<generated>)

at
org.apache.geronimo.deployment.Deployer.deploy(Deployer.java:302)

at
org.apache.geronimo.deployment.Deployer.deploy(Deployer.java:124)

at
org.apache.geronimo.deployment.Deployer$$FastClassByCGLIB$$734a235d.invoke(<generated>)

at net.sf.cglib.reflect.FastMethod.invoke(FastMethod.java:53)

at
org.apache.geronimo.gbean.runtime.FastMethodInvoker.invoke(FastMethodInvoker.java:38)

at
org.apache.geronimo.gbean.runtime.GBeanOperation.invoke(GBeanOperation.java:124)

at
org.apache.geronimo.gbean.runtime.GBeanInstance.invoke(GBeanInstance.java:855)

at
org.apache.geronimo.kernel.basic.BasicKernel.invoke(BasicKernel.java:239)

at
org.apache.geronimo.kernel.KernelGBean.invoke(KernelGBean.java:338)

at
org.apache.geronimo.kernel.KernelGBean$$FastClassByCGLIB$$1cccefc9.invoke(<generated>)

at net.sf.cglib.reflect.FastMethod.invoke(FastMethod.java:53)

at
org.apache.geronimo.gbean.runtime.FastMethodInvoker.invoke(FastMethodInvoker.java:38)

at
org.apache.geronimo.gbean.runtime.GBeanOperation.invoke(GBeanOperation.java:124)

at
org.apache.geronimo.gbean.runtime.GBeanInstance.invoke(GBeanInstance.java:855)

at
org.apache.geronimo.kernel.basic.BasicKernel.invoke(BasicKernel.java:239)

at
org.apache.geronimo.system.jmx.MBeanGBeanBridge.invoke(MBeanGBeanBridge.java:168)

at
com.sun.jmx.mbeanserver.DynamicMetaDataImpl.invoke(DynamicMetaDataImpl.java:213)

at
com.sun.jmx.mbeanserver.MetaDataImpl.invoke(MetaDataImpl.java:220)
```

```
at
com.sun.jmx.interceptor.DefaultMBeanServerInterceptor.invoke(DefaultMBeanServerInterceptor.java:815)

at
com.sun.jmx.mbeanserver.JmxMBeanServer.invoke(JmxMBeanServer.java:784)

at
javax.management.remote.rmi.RMIConnectionImpl.doOperation(RMIConnectionImpl.java:1408)

at
javax.management.remote.rmi.RMIConnectionImpl.access$100(RMIConnectionImpl.java:81)

at
javax.management.remote.rmi.RMIConnectionImpl$PrivilegedOperation.run(RMIConnectionImpl.java:1245)

at java.security.AccessController.doPrivileged(Native Method)

at
javax.management.remote.rmi.RMIConnectionImpl.doPrivilegedOperation(RMIConnectionImpl.java:1348)

at
javax.management.remote.rmi.RMIConnectionImpl.invoke(RMIConnectionImpl.java:782)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)

at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)

at java.lang.reflect.Method.invoke(Method.java:585)

at
sun.rmi.server.UnicastServerRef.dispatch(UnicastServerRef.java:294)

at sun.rmi.transport.Transport$1.run(Transport.java:153)

at java.security.AccessController.doPrivileged(Native Method)

at sun.rmi.transport.Transport.serviceCall(Transport.java:149)

at
sun.rmi.transport.tcp.TCPTransport.handleMessages(TCPTransport.java:466)

at
sun.rmi.transport.tcp.TCPTransport$ConnectionHandler.run(TCPTransport.java:707)

at java.lang.Thread.run(Thread.java:595)

Caused by: org.apache.geronimo.kernel.config.LifecycleException:
load of default/CustomerService/1.0/ear failed

at
org.apache.geronimo.kernel.config.SimpleConfigurationManager.loadConfiguration(SimpleConfigurationManager.java:

at
org.apache.geronimo.deployment.DeploymentConfigurationManager.loadConfiguration(DeploymentConfigurationManager.

at
org.apache.geronimo.kernel.config.SimpleConfigurationManager.loadConfiguration(SimpleConfigurationManager.java:

at
org.apache.geronimo.deployment.DeploymentConfigurationManager.loadConfiguration(DeploymentConfigurationManager.

at
```

```
org.apache.geronimo.deployment.DeploymentContext.createTempConfiguration(DeploymentContext.java:118)

... 50 more

Caused by: org.apache.geronimo.kernel.config.InvalidConfigException:
Error starting configuration gbean default/CustomerService/1.0/ear

at
org.apache.geronimo.kernel.config.SimpleConfigurationManager.load(SimpleConfigurationManager.java:347)

at
org.apache.geronimo.deployment.DeploymentConfigurationManager.load(DeploymentConfigurationManager.java:119)

at
org.apache.geronimo.kernel.config.SimpleConfigurationManager.loadConfiguration(SimpleConfigurationManager.java:

... 54 more

Caused by:
org.apache.geronimo.kernel.repository.MissingDependencyException:
Unable to resolve dependency
console.dbpool/CustomerServicePool/1.0/rar

at
org.apache.geronimo.kernel.config.ConfigurationResolver.resolve(ConfigurationResolver.java:112)

at
org.apache.geronimo.kernel.config.Configuration.buildClassPath(Configuration.java:402)

at
org.apache.geronimo.kernel.config.Configuration.createConfigurationClasssLoader(Configuration.java:324)

at
org.apache.geronimo.kernel.config.Configuration.<init>(Configuration.java:268)

at
org.apache.geronimo.kernel.config.SimpleConfigurationManager.load(SimpleConfigurationManager.java:343)

... 56 more
```

ここに注目してください。

```
Caused by:
org.apache.geronimo.kernel.repository.MissingDependencyException:
Unable to resolve dependency
console.dbpool/CustomerServicePool/1.0/rar
```

このメッセージにより、記事の中のファイルが以前の(1.0の)the CustomerServicePool (a Derby embedded database pool)向けに作成されていることがわかります。

3.c) Dependencyのバージョン番号の更新

この事象は、以下のようにCustomerEJB-openejb.xmlファイルのdependenciesセクションの"dep:version"タグを"1.0" から"2.0"へ変更すれば解消できます。

```
<?xml version="1.0" encoding="UTF-8"?>

<openejb-jar xmlns="http://www.openejb.org/xml/ns/openejb-jar-2.1"
xmlns:naming="http://geronimo.apache.org/xml/ns/naming-1.1">

<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.1">
<dep:moduleId>
<dep:groupId>default</dep:groupId>
<dep:artifactId>CustomerEJB</dep:artifactId>
<dep:version>1.0</dep:version>
<dep:type>jar</dep:type>
</dep:moduleId>
```

```
<dep:dependencies>
<dep:dependency>
<dep:groupId>console.dbpool</dep:groupId>
<dep:artifactId>CustomerServicePool</dep:artifactId>
<dep:version>2.0</dep:version>
<dep:type>rar</dep:type>
</dep:dependency>
</dep:dependencies>         . . .
```

（訳者注）

さらに、Apache Derby への接続記述を以下のとおり書き換えてください。

・変更前

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.2">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2">
<dep:dependencies>
<dep:dependency>
<dep:groupId>org.apache.derby</dep:groupId>
<dep:artifactId>derby</dep:artifactId>
<dep:version>10.1.3.1</dep:version>
<dep:type>jar</dep:type>
</dep:dependency>
</dep:dependencies>
</dep:environment>
... ##### ...
</connector>
```

・変更後

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.2">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2">
<dep:dependencies>
<dep:dependency>
<dep:groupId>org.apache.geronimo.configs</dep:groupId>
<dep:artifactId>system-database</dep:artifactId>
<dep:type>car</dep:type>
</dep:dependency>
</dep:dependencies>
</dep:environment>
... ##### ...
</connector>
```

もう一度、<GERONIMO_HOME>¥binディレクトリーでコマンド・プロンプトから以下を実行してください。

deploy --verbose --user system --password manager deploy D:\appsData\GERONIMO\JNDIApp\CustomerService.ear

3.d) デプロイメント記述子の更新

application.xmlのデプロイメント記述子をGeronimo 2.0の要件と互換性を保つように変更します。

前述の delpoy コマンドを実行しても、まだ geronimo.log 中に別のエラーが見つかることでしょう。

```
Deployer operation failed: Could not parse application.xml
org.apache.geronimo.common.DeploymentException: Could not parse application.xml
at org.apache.geronimo.j2ee.deployment.EARConfigBuilder.getEarPlan(EARConfigBuilder.java:314)
at
org.apache.geronimo.j2ee.deployment.EARConfigBuilder.getDeploymentPlan(EARConfigBuilder.java:261)
at org.apache.geronimo.j2ee.deployment.EARConfigBuilder$$FastClassByCGLIB$
$38e56ec6.invoke(<generated>)
at net.sf.cglib.reflect.FastMethod.invoke(FastMethod.java:53)
at org.apache.geronimo.gbean.runtime.FastMethodInvoker.invoke(FastMethodInvoker.java:38)
```

```
at org.apache.geronimo.gbean.runtime.GBeanOperation.invoke(GBeanOperation.java:124)
at org.apache.geronimo.gbean.runtime.GBeanInstance.invoke(GBeanInstance.java:820)
at org.apache.geronimo.gbean.runtime.RawInvoker.invoke(RawInvoker.java:57)
at org.apache.geronimo.kernel.basic.RawOperationInvoker.invoke(RawOperationInvoker.java:35)
at
org.apache.geronimo.kernel.basic.ProxyMethodInterceptor.intercept(ProxyMethodInterceptor.java:96)
at org.apache.geronimo.j2ee.deployment.CorbaGBeanNameSource$$EnhancerByCGLIB$
$44342279.getDeploymentPlan(<generated>)
at org.apache.geronimo.deployment.Deployer.deploy(Deployer.java:232)
at org.apache.geronimo.deployment.Deployer.deploy(Deployer.java:124)
at org.apache.geronimo.deployment.Deployer$$FastClassByCGLIB$$734a235d.invoke(<generated>)
at net.sf.cglib.reflect.FastMethod.invoke(FastMethod.java:53)
at org.apache.geronimo.gbean.runtime.FastMethodInvoker.invoke(FastMethodInvoker.java:38)
at org.apache.geronimo.gbean.runtime.GBeanOperation.invoke(GBeanOperation.java:124)
at org.apache.geronimo.gbean.runtime.GBeanInstance.invoke(GBeanInstance.java:855)
at org.apache.geronimo.kernel.basic.BasicKernel.invoke(BasicKernel.java:239)
at
org.apache.geronimo.deployment.plugin.local.AbstractDeployCommand.doDeploy(AbstractDeployCommand.java:114)
at org.apache.geronimo.deployment.plugin.local.DistributeCommand.run(DistributeCommand.java:60)
at java.lang.Thread.run(Thread.java:595)
Caused by: org.apache.xmlbeans.XmlException: Invalid deployment descriptor: [error: cvc-complex-
type.2.4a: Expected element 'module@http://java.sun.co
m/xml/ns/j2ee' instead of 'description@http://java.sun.com/xml/ns/j2ee' here in element
application@http://java.sun.com/xml/ns/j2ee]
Descriptor: <application xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/
xml/ns/j2ee/application_1_4.xsd" version="1.4" xmlns=
"http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<display-name>Customer Service utility</display-name>
<description>Customer Service utility</description>
<module>
<web>
<web-uri>CustomerService-web.war</web-uri>
<context-root>/service</context-root>
</web>
</module>
<module>
<ejb>CustomerEJB.jar</ejb>
</module>
<module>
<ejb>ProcessCustomerEJB.jar</ejb>
</module>
</application>
at org.apache.geronimo.deployment.xmlbeans.XmlBeansUtil.validateDD(XmlBeansUtil.java:213)
at
org.apache.geronimo.j2ee.deployment.EARConfigBuilder.convertToApplicationSchema(EARConfigBuilder.java:420)
at org.apache.geronimo.j2ee.deployment.EARConfigBuilder.getEarPlan(EARConfigBuilder.java:312)
... 21 more
```

application.xmlファイルから"description"と"display-name"タグを除去し、CustomerService.ear 中のapplication.xmlファイルに再度保管してください。

```
<?xml version="1.0" ?>
<application xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/
application_1_4.xsd"
version="1.4">
<display-name>Customer Service utility</display-name>

<module>
<web>
<web-uri>CustomerService-web.war</web-uri>
<context-root>/service</context-root>
</web>
</module>
```

```
<module>
<ejb>CustomerEJB.jar</ejb>
</module>

<module>
<ejb>ProcessCustomerEJB.jar</ejb>
</module>
</application>
```

今度はCustomerService.ear　ファイルはうまくデプロイできることでしょう。

```
deploy --verbose --user system --password manager deploy D:\appsData\GERONIMO\JNDIApp
\CustomerService.ear
```

4) アプリケーションのデプロイとテスト

- Geronimo Server Console | Applications | Deploy Newウインドウを開きます。
- <Browse>ボタンをクリックしてCustomerService.ear ファイルを選択します。
- <Install>をクリックします。
- http://localhost:8080/service/customers/で、デプロイされたアプリケーションをテストしてください。

サマリー

以下の手順に従い、動いているGeronimo 2.0インスタンスへJ2EEエンタープライズ・アプリケーションを正常にデプロイできました。

1) j2ee.jarのclasspathへの追加
2) Geronimo/Tomcatディストリビューションの利用
3) データベース・プール (CustomerServicePool)の作成
a. デプロイメント・テスト・プランの手動での作成とデプロイ
b. CustomerServicePool用のデプロイメント・バージョンの更新
c. Dependencyのバージョン番号の更新
d. デプロイメント記述子の更新
4) アプリケーションのデプロイとテスト

9.7. インバウンド JCA の例

This page last changed on 4 25, 2008 by JAGUG.

J2EE コネクター・アーキテクチャー (JCA) は、多くのアプリケーション・コンテナーと今日の企業情報システム (EIS) との間の接続に関する問題解決に寄与する Java の技術です。JCA 1.5 仕様では、インバウンド通信モデルとして定義されていて、そのことによって EIS は EJB アプリケーションに対してすべての通信を始めることができます。このメカニズムではインバウンド・リソース・アダプターがエンタープライズ Java Bean を呼び出すことになります。

概略

この文章では EJB アプリケーションをデプロイして、JCA アダプターからインバウンド・イベントを受け取ることができるようにする方法を説明します。最初に、スタンドアロンのリソース・アダプターをデプロイする Geronimo 固有のプラン記述だけではなく、インバウンド通信モデルに付随するリソース・アダプターの重要な部分を説明します。その後、EIS によって実行されるエンタープライズ・アプリケーションのインバウンド通信メカニズムを説明します。

リソース・アダプター

このセクションで説明するコードは、デプロイされるリソース・アダプターの一部分です。多くの部分は Geronimo 固有ではありませんが、このセクションの最後では Geronimo コンテナーへリソース・アダプターをデプロイする方法を示します。

JCA 1.5 仕様では、インバウンド・アダプターはカスタマイズされたメッセージ・フォーマットのサポートが可能です。したがって、メッセージ駆動 Bean (MDB) がインバウンド・リソース・アダプターによって定義されたどんなインターフェースでも実装できます。以前の JCA 仕様では、Java メッセージ・サービス (JMS) のメッセージに javax.jms.MessageListener インターフェースを実装した MDB を必要とするようなインバウンド通信のメカニズムでした。JCA 1.5 では、どのようなインターフェースの定義でも MDB としてインバウンド・リクエストを扱えるようになりました。

インターフェースの定義

この例では、com.sample.connector.EventListener インターフェースが定義されています。このインターフェースはどのような MDB にも実装でき、この MDB はインバウンド・リソース・アダプターによって呼び出されるようになります。

EventListener.java

```
package com.sample.connector;

import java.util.List;
import javax.resource.ResourceException;

/**
* An interface allowing Events to be Handled.  Message-driven Beans implement
* this interface, in order to receive inbound-events from our EIS via the JCA adapter.
*/
public interface EventListener {

/**
* Method to be called when a Event is handled.
*
* @param eventName the name of the event
* @param paramList the parameters sent to the event
*
* @throws ResourceException if an error occurs
*/
void handleEvent(String eventName, List paramList) throws ResourceException;
}
```

ActivationSpec の実装

インバウンド・リソース・アダプターは javax.resource.spi.ActivationSpec インターフェースを実装しています。インターフェース自身にはメソッドはありませんが、このインターフェースを実装するクラスは、

- JavaBean でなければなりません。getter と setter メソッドを bean のフィールドに用意してください。
- シリアライズ可能でなければなりません。

API ドキュメントによれば、「ActivationSpec 実装は、メッセージのエンド・ポイントに関する情報をアクティベーションの構成情報として持つことになります」。今回の場合では、メッセージのエンド・ポイントは今回のエンタープライズ・アプリケーション内にある MDB です。今回の ActivationSpec を実装したクラスは com.sample.connector.EventActivationSpec です。今回の例での ActivationSpec は通信を開始するために必要なすべてのデータを持っているので、リモートの EIS システム用にアプリケーション・コンテナーに読み込むことができます（マシン名、通信ポート、ユーザー名、ユーザーのパスワード、イベント・パターン）。

EventActivationSpec.java

```
/*
* EventActivationSpec.java
*/

package com.sample.connector;

import java.io.Serializable;
import javax.resource.spi.ActivationSpec;
import javax.resource.spi.InvalidPropertyException;
import javax.resource.spi.ResourceAdapter;

/**
* Implementation of the <code>ActivationSpec</code> interface, which allows EIS Events
* to be exposed to message-drive beans.  This <code>ActivationSpec</code> is used to
* support inbound connection from our EIS to a message-driven bean.  This will open
* an EIS connection and use its event mechanism provided by the EIS-specific connection
* class.  The connection to the EIS will be held open while the application containing
* the message-driven bean is available.
*/
public class EventActivationSpec implements ActivationSpec, Serializable  {

private ResourceAdapter resourceAdapter;     private String serverName;
private Integer portNumber;
private String userName;
private String password;
private String eventPatterns;//A comma-delimeted string of patterns

/**
* Creates a new instance of the EventActivationSpec class.
*/
public EventActivationSpec() {
}

/**
* No validation is performed
*/
public void validate() throws InvalidPropertyException {
}

//javadoc inherited
public ResourceAdapter getResourceAdapter() {
return resourceAdapter;
}

//javadoc inherited
public void setResourceAdapter(ResourceAdapter resourceAdapter) {
this.resourceAdapter = resourceAdapter;
}     /**
* Getter method for the ServerName attribute.  This allows the server name
```

```
* to be defined against a Connection pool
*
* @return a <code>String</code> containing the server name value
*/
public String getServerName() {
return serverName;
}

/**
* Setter method for the ServerName attribute.  This allows the server name to be
* defined against a connection pool/.
*
* @param serverName the <code>String</code> that will be defined against this attribute
*/
public void setServerName(String serverName) {
this.serverName = serverName;
}

/**
* Getter method for the PortNumber attribute.  This allows the port number
* to be defined against a Connection pool
*
* @return a <code>Integer</code> containing the server port value
*/
public Integer getPortNumber() {
return portNumber;
}     /**
* Setter method for the PortNumber attribute.  This allows the server port to be
* defined against a connection pool/.
*
* @param portNumber the <code>Integer</code> that will be defined against this attribute
*/
public void setPortNumber(Integer portNumber) {
this.portNumber = portNumber;
}

/**
* Getter method for the UserName attribute.  This allows the user name
* to be defined against a Connection pool
*
* @return a <code>String</code> containing the user name value
*/
public String getUserName() {
return userName;
}

/**
* Setter method for the UserName attribute.  This allows the user name to be
* defined against a connection pool/.
*
* @param userName the <code>String</code> that will be defined against this attribute
*/
public void setUserName(String userName) {
this.userName = userName;
}
/**
* Getter method for the Password attribute.  This allows the password
* to be defined against a Connection pool
*
* @return a <code>String</code> containing the password value
*/
public String getPassword() {
return password;
}

/**
```

```
* Setter method for the Password attribute.  This allows the password to be
* defined against a connection pool/.
*
* @param password the <code>String</code> that will be defined against this attribute
*/
public void setPassword(String password) {
this.password = password;
}

/**
* Returns the event patterns that will be used when subscribing to Events.  This string can contain
* comma-delimited value, in order to subscribe to multiple Events.
*
* @return a <code>String</code> containing the event patterns that will be subscribed
*/
public String getEventPatterns() {
return eventPatterns;
}

/**
* Defines the event patterns that will be subscribed to by the message-driven beans.  The value can
* contain comma-delimited patterns, such that multiple events can be subscribed to in the single
bean.
*
* @param eventPatterns a <code>String</code> containing the single pattern or comma-delimited
patterns
*/
public void setEventPatterns(String eventPatterns) {
this.eventPatterns = eventPatterns;
}
}
```

一般的なデプロイメント記述

JCA デプロイメント記述 (ra.xml) には、アダプターを ActivationSpec を実装したインターフェースに結びつけるためのセクションがあります。これが必要なのは、アダプターによって呼び出される MDB に定義されたプロパティをアプリケーション・コンテナーに伝えるためです。

ra.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/
connector_1_5.xsd"
version="1.5">
<display-name>EIS Connector</display-name>
<vendor-name>My Component</vendor-name>
<eis-type>My Remote Server</eis-type>
<resourceadapter-version>1.0</resourceadapter-version>
<resourceadapter>
<resourceadapter-class>com.sample.connector.MyResourceAdapter</resourceadapter-class>
<!-- snip -->
<inbound-resourceadapter>
<messageadapter>
<messagelistener>
<messagelistener-type>com.sample.connector.EventListener</messagelistener-type>
<activationspec>
<activationspec-class>com.sample.connector.EventActivationSpec</activationspec-class>
<required-config-property>
<config-property-name>ServerName</config-property-name>
</required-config-property>
<required-config-property>
<config-property-name>PortNumber</config-property-name>
</required-config-property>
```

```
<required-config-property>
<config-property-name>UserName</config-property-name>
</required-config-property>
<required-config-property>
<config-property-name>Password</config-property-name>
</required-config-property>
<required-config-property>
<config-property-name>EventPatterns</config-property-name>
</required-config-property>
</activationspec>
</messagelistener>
</inbound-resourceadapter>
<security-permission>
<description>Permissions allowed to the EIS Connector</description>
<security-permission-spec/>
</security-permission>
</resourceadapter>
</connector>
```

ActivationSpec が実装されると、リソース・アダプターの endpointActivation メソッドが更新されます。MDB が環境にデプロイされた際に、アプリケーション・コンテナーはこのメソッドをリソース・アダプター上で呼び出します。このことによって、インバウンド・アダプターは通信を開始することができるようになります。このメソッドのシグネチャーは次のとおりです。

```
public void endpointActivation(MessageEndpointFactory endpointFactory, ActivationSpec spec) throws
ResourceException
```

endpointFactory によって MDB の新しいインスタンスを作成できるようになり、spec は必要なすべてのプロパティを含んでいる ActivationSpec の実装になります。このメソッドを実装することで、EIS を利用するメカニズムによるメッセージのサポートが可能になります。通常、リソース・アダプターは javax.resource.spi.work.Work インターフェースの実装が定義されていて、新しいスレッドを作ることなくインバウンド・リクエストを処理するようになります。

同じように重要なことは endpointDeactivation メソッドの実装です。エンド・ポイントが活性化された時に作成されたすべてのリソースを、アダプターから開放することができるようになります。

Geronimo のデプロイメント記述

最後に、Geronimo 固有の仕様の詳細です。Geronimo デプロイメント記述はインバウンド・リソース・アダプターの仕様が定義されているセクションです。

geronimo-ra.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.2">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2">
<dep:moduleId>
<dep:groupId>com.sample</dep:groupId>
<dep:artifactId>myConnector</dep:artifactId>
<dep:version>1.0</dep:version>
<dep:type>rar</dep:type>
</dep:moduleId>
<dep:dependencies/>
<dep:hidden-classes/>
<dep:non-overridable-classes/>
</dep:environment>
<resourceadapter>
<resourceadapter-instance>
<resourceadapter-name>MyInboundEvents</resourceadapter-name>
<workmanager>
<gbean-link>DefaultWorkManager</gbean-link>
</workmanager>
</resourceadapter-instance>
</resourceadapter>
</connector>
```

上記デプロイメント・プランによってアダプターの RAR ファイルをデプロイすると、MyInboundEvents という名前のインバウンド・アダプターがひとつ作られます。

EJB アプリケーション

ここまで、Geronimo サーバーのインスタンスにリソース・アダプターを定義し、デプロイしてきました。では、リモートの EIS からメッセージを受信した時にインバウンド・リソース・アダプターによって呼び出されるメッセージ駆動 Bean を持っている EJB アプリケーションを作成し、デプロイします。

この例では、メッセージ駆動 Bean はリソース・アダプターで定義された EventListener インターフェースを実装しています。

EventBean.java

```
package com.sample.eventDemo.message;

import com.sample.connector.EventListener;
import java.util.List;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.resource.ResourceException;

/**
* Message-driven Bean, listening for Events
*/
@MessageDriven(
name="EventBean",
messageListenerInterface=EventListener.class,
activationConfig={
@ActivationConfigProperty(propertyName="ServerName",propertyValue="fozzie"),
@ActivationConfigProperty(propertyName="PortNumber",propertyValue="40100"),
@ActivationConfigProperty(propertyName="UserName",propertyValue="user"),
@ActivationConfigProperty(propertyName="Password",propertyValue="pwd"),
@ActivationConfigProperty(propertyName="EventPatterns",propertyValue="myPattern")
}
)
public class EventBean implements EventListener {

/**
* Method that will be called by the inbound Event handler
*/
public void handleEvent(String eventName, List paramList) throws ResourceException {
System.out.println("In MDB: " + eventName + " with params " + paramList.toString());
}
}
```

この bean は多くのことをするわけではないのが実行されることは見せられます。ActivationConfigProperty アノテーションを利用してインバウンド・リソース・アダプターに必要とされるプロパティを定義します。これらの値は対応する ActivateSpec（今回では EventActivationSpec）向けに定義され、リソースアダプターの endpointActivation メソッドへ渡します。

Geronimo のデプロイメント記述

最後に、MDB と、リソース・アダプターを Geronimo コネクターにデプロイした時に定義したインバウンド・リソース・アダプターの MyInboundEvents とを結びつけます。これは EJB の Geronimo デプロイメント記述である openejb-jar.xml によって行います。

openejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<openejb-jar
xmlns="http://www.openejb.org/xml/ns/openejb-jar-2.1"
xmlns:naming="http://geronimo.apache.org/xml/ns/naming-1.1"
xmlns:security="http://geronimo.apache.org/xml/ns/security-1.1"
xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2">
<sys:environment>
<sys:moduleId>
<sys:groupId>com.sample.connectorDemo</sys:groupId>
<sys:artifactId>SampleEventHandler</sys:artifactId>
<sys:version>1.1</sys:version>
```

```
<sys:type>car</sys:type>
</sys:moduleId>
<sys:dependencies>
<sys:dependency>
<sys:groupId>com.sample</sys:groupId>
<sys:artifactId>myConnector</sys:artifactId>
<sys:version>1.0</sys:version>
<sys:type>rar</sys:type>
</sys:dependency>
</sys:dependencies>
<sys:hidden-classes/>
<sys:non-overridable-classes/>
</sys:environment>
<enterprise-beans>
<message-driven>
<ejb-name>EventBean</ejb-name>
<resource-adapter>
<resource-link>MyInboundEvents</resource-link>
</resource-adapter>
</message-driven>
</enterprise-beans>
</openejb-jar>
```

このファイルでは、EventBean と MyInboundEvents アダプターとを結びつけています。

This page last changed on 4 25, 2008 by JAGUG.

紹介

同じ Ear の中に含まれていない他の Jar にある bean を参照することは、ひとつの Ear に含まれる場合ほど単純ではありません。目的達成には（私の知る限り）3つの方法があります。

参照される bean を持つ Jar

Java のコード

```
@Stateless(name="JmsDispatcherGate")
public class JmsDispatcherGateImpl implements DispatcherGateLocal, DispatcherGateRemote {

@Local
public interface DispatcherGateLocal {

@Remote
public interface DispatcherGateRemote {
```

Geronimo 固有の XML

```
<?xml version="1.0" encoding="UTF-8"?>
<openejb-jar xmlns="http://www.openejb.org/xml/ns/openejb-jar-2.1" xmlns:pkgen="http://
www.openejb.org/xml/ns/pkgen-2.0" xmlns:sec="http://geronimo.apache.org/xml/ns/security-2.0"
xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2">
<sys:environment>
<sys:moduleId>
<sys:groupId>my.app.dispatcher</sys:groupId>
<sys:artifactId>Dispatcher</sys:artifactId>
<sys:version>1.0</sys:version>
<sys:type>jar</sys:type>
</sys:moduleId>
</sys:environment>
</openejb-jar>
```

@EJB(mappedName="...") + openejb-jar.xml による参照

```
package my.app.services;

@Stateless(name = "Sender")
public class SenderImpl {

@EJB(mappedName="Dispatcher/JmsDispatcherGate")
private DispatcherGateRemote dispatcherGate = null;
```

```
<openejb-jar xmlns="http://www.openejb.org/xml/ns/openejb-jar-2.1"
xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2"
xmlns:pkgen="http://www.openejb.org/xml/ns/pkgen-2.0"
xmlns:sec="http://geronimo.apache.org/xml/ns/security-2.0"
xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2">
<sys:environment>
<sys:moduleId>
<sys:groupId>my.app.services</sys:groupId>
<sys:artifactId>Services</sys:artifactId>
<sys:version>1.0</sys:version>
<sys:type>jar</sys:type>
</sys:moduleId>
```

```
<!-- only dependency is required -->
<sys:dependencies>
<sys:dependency>
<sys:groupId>my.app.dispatcher</sys:groupId>
<sys:artifactId>Dispatcher</sys:artifactId>
<sys:version>1.0</sys:version>
<sys:type>jar</sys:type>
</sys:dependency>
</sys:dependencies>
</sys:environment>

</openejb-jar>
```

@EJB(name="") + openejb-jar.xml による参照

```
package my.app.services;

@Stateless(name = "Sender")
public class SenderImpl {

@EJB(name="dispatcher")
private DispatcherGateRemote dispatcherGate = null;
```

```
<openejb-jar xmlns="http://www.openejb.org/xml/ns/openejb-jar-2.1"
xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2"
xmlns:pkgen="http://www.openejb.org/xml/ns/pkgen-2.0"
xmlns:sec="http://geronimo.apache.org/xml/ns/security-2.0"
xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2">
<sys:environment>
<sys:moduleId>
<sys:groupId>my.app.services</sys:groupId>
<sys:artifactId>Services</sys:artifactId>
<sys:version>1.0</sys:version>
<sys:type>jar</sys:type>
</sys:moduleId>
<sys:dependencies>
<sys:dependency>
<sys:groupId>my.app.dispatcher</sys:groupId>
<sys:artifactId>Dispatcher</sys:artifactId>
<sys:version>1.0</sys:version>
<sys:type>jar</sys:type>
</sys:dependency>
</sys:dependencies>
</sys:environment>
<enterprise-beans>
<session>
<ejb-name>Sender</ejb-name>
<ejb-ref>
<!-- @EJB(name="dispatcher") DispatcherGateRemote dispatcherGateRemote; -->
<ref-name>dispatcher</ref-name>
<nam:pattern xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.1">
<nam:artifactId>Dispatcher</nam:artifactId>
<nam:name>JmsDispatcherGate</nam:name>
</nam:pattern>
</ejb-ref>
</session>
</enterprise-beans>

</openejb-jar>
```

@EJB(name なし) + openejb-jar.xml による参照

```
package my.app.services;
```

```
@Stateless(name = "Sender")
public class SenderImpl {

@EJB
private DispatcherGateRemote dispatcherGate = null;
```

```
<openejb-jar xmlns="http://www.openejb.org/xml/ns/openejb-jar-2.1"
xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2"
xmlns:pkgen="http://www.openejb.org/xml/ns/pkgen-2.0"
xmlns:sec="http://geronimo.apache.org/xml/ns/security-2.0"
xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2">
<sys:environment>
<sys:moduleId>
<sys:groupId>my.app.services</sys:groupId>
<sys:artifactId>Services</sys:artifactId>
<sys:version>1.0</sys:version>
<sys:type>jar</sys:type>
</sys:moduleId>
<sys:dependencies>
<sys:dependency>
<sys:groupId>my.app.dispatcher</sys:groupId>
<sys:artifactId>Dispatcher</sys:artifactId>
<sys:version>1.0</sys:version>
<sys:type>jar</sys:type>
</sys:dependency>
</sys:dependencies>
</sys:environment>
<enterprise-beans>
<session>
<ejb-name>Sender</ejb-name>
<ejb-ref>
<!-- @EJB DispatcherGateRemote dispatcherGateRemote; -->
<ref-name>my.app.services.SenderImpl/dispatcherGate</ref-name>
<nam:pattern xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.1">
<nam:artifactId>Dispatcher</nam:artifactId>
<nam:name>JmsDispatcherGate</nam:name>
</nam:pattern>
</ejb-ref>
</session>
</enterprise-beans>
</openejb-jar>
```

9.9. JMS と MDB のサンプル・アプリケーション

エンタープライズ・メッセージングは信頼性のあるエンタープライズ・フレームワークに緩やかに結びつけられたコンポーネントとして重要性が増しています。これはエンタープライズ・アプリケーションや異種のエンタープライズ・リソースの急増に重要な要素であり、これらのアプリケーションを密接にまとまったシステムとして統合するニーズを増大させています。長年に渡って、メッセージングやメッセージング指向ミドルウェア (MOM) はこの統合に対して独占的な方法となっていました。標準化された Java メッセージング・サービス (JMS) の導入は、独占的な MOM ベースの製品の多くの不便さを解消します。加えて、J2EE アプリケーション・サーバーにエンタープライズ Java Beans 2.0 から導入されたメッセージ駆動 Beans (MDBs) は投資削減に役立ちます。現在のほとんどの J2EE アプリケーション・サーバーは、いっさいの価値を JMS に加えられた MOM として実行します。J2EE 1.4 認定のアプリケーション・サーバーとして、Apache Geronimo はオープン・ソースでよく成長したメッセージング・フレームワークの一つである ActiveMQ を統合して JMS のサポートをしています。この文章では Geronimo と ActiveMQ を使ってローカルとリモート両方の環境におけるエンタープライズ・アプリケーションのシナリオでの JMS の利用方法を示します。

このサンプル・アプリケーションでの会社は、ある物品を小売市場、卸売市場という違う市場の両方で販売しています。このアプリケーションでのすべての注文は顧客への配送前に会社の営業担当社員によって認証されます。卸売市場では、この会社はすべての国に渡って仲介者へ販売します。彼らは一度にまとめて注文を送ってきます。これを委託販売品と呼びます。エンド・ユーザーはこの会社のウェブ・サイトを利用して物品を注文し、仲介者は委託販売品を店舗でインストールした特別なソフトウェアを利用して送付します。すべての委託販売品は営業担当社員が扱う前にこの会社のゼネラル・マネージャーによって承認されます。

これは、委託販売品と注文の依頼を非同期で処理する典型的な JMS を利用したアプリケーションです。

この文章を読むと、メッセージ・キュー、Geronimo/ActiveMQ 環境でのコネクション・ファクトリー、簡単にエンタープライズ・アプリケーションにおける異なる種類のアプリケーションを利用しているメッセージの送受信の定義ができるようになるでしょう。

この文章は以下のセクションで構成されています。

- Geronimo/ActiveMQ 環境での JMS の概略
- アプリケーションの概略
- サンプル・アプリケーションの構成、ビルド、デプロイ
- サンプル・アプリケーションのテスト
- まとめ

Geronimo/ActiveMQ 環境での JMS の概略

Geronimo サーバーは JMS サーバーとアプリケーション・コンポーネントを持っていて、JMS リソースにコネクション・ファクトリーやトピックス、キューなどのように接続できます。また、この JMS サーバーはメッセージ・ブローカーとしても知られています。Geronimo がデフォルトでサポートしているメッセージ・ブローカーは ActiveMQ です。これは十分に成熟した機能豊富な JMS プロダクトですので、通常は変更する必要はありません。この実装では組み込みの Derby データベースをメッセージ永続化機能のために利用しています。

ActiveMQ はさまざまなトランスポート（例えば TCP、SSL、UDP、マルチキャスト、内部 JVM、NIOなど）とクライアント相互処理（例えばプッシュ、プル、パブリッシュ/サブスクライブ）をサポートします。Geronimo での ActiveMQ は、JMS メッセージを処理する EJBs である MDBs をサポートします。ActiveMQ は Geronimo の J2EE 仕様の機能を JMS アプリケーションに保持させ、例えば JSP やサーブレットや EJB に JMS を利用させることができるようになります。Geronimo はこの JMS API を抽象レイヤーに実装しているので、いずれの JMS プロバイダーもサポートします。J2EE コネクター (JCA) 仕様をサポートすることでこの機能が含まれています。JCA 1.5 の詳細仕様により、アプリケーション・サーバーとドライバーの間に ActiveMQ（リソース・アダプター）による決まりごとがあります。Geronimo にデプロイしたアプリケーションは、このリソース・アダプター (RA) を通じてのみ ActiveMQ メッセージ・ブローカーへ接続します。

アプリケーションの概略

Order processing アプリケーションは2つのメッセージ・キューを定義して注文や委託販売品を受信しています。注文リクエストはこの会社のウェブ・アプリケーション経由で生成され、送信されます。注文リクエストが注文キューで受信されると、MDB が処理開始となります。これらのリクエストをサーバー・リポジトリーに保存することによって、注文リクエストは次の段階へすすみます。これらの保存された注文リクエストは後でその会社の社員によって処理されます。

この会社の販売仲介者は委託販売品送信アプリケーションを利用して、委託販売品注文（まとまった注文）を自分の所から送信します。まず、XML ファイルに注文情報を準備し、これをアプリケーションのパラメーターとして渡します。委託販売品送信アプリケーションは XML ファイルの中身（注文依頼）を読み込み、委託販売品キューに送信します。この会社のゼネラル・マネージャーは委託販売品受注アプリケーションを利用して委託販売品依頼を確認します。委託販売品の依頼を委託販売品キューが受信すると、委

託販売品受信リスナー・アプリケーションがゼネラル・マネージャーのコンピューターにこれらのリクエストをダウンロードします。彼はこれを承認し、営業担当社員に回して次の段階へ進みます。

アプリケーションの内容

注文処理アプリケーションのコアは、アプリケーション・サーバーに　EAR としてデプロイされます。以下に EAR の内容の概略を示します。

```
|-Order.ear
|- geronimo-activemq-ra-2.0-SNAPSHOT.rar
|- jms-mdb-sample-ejb-2.0-SNAPSHOT.jar
|-META-INF
|- openejb-jar.xml
|- jms-mdb-sample-ejb-2.0-SNAPSHOT.war
|- index.jsp
|- error.jsp
|- WEB-INF
|- web.xml
|- classes
|- META-INF
|- application.xml
|- geronimo-application.xml
```

MDB の実装

ejb-jar.xml ファイル内に MDB の宣言をする代わりに、メッセージ駆動 Bean には @MessageDriven アノテーションを利用します。さらなる情報をアノテーションに与えることによって、送信先（今回はキューに入れる）を探して処理する方法がわかります。この MDB は 'OrderQueue' にメッセージを送信する処理をします。終了結果として画面にこのメッセージを表示します。

OrderRecvMDB.java

```
//
// MessageDrivenBean that listens to items on the
// 'OrderQueue' queue and processes them accordingly.
//
@MessageDriven(activationConfig = {
@ActivationConfigProperty(propertyName="destinationType", propertyValue="javax.jms.Queue"),
@ActivationConfigProperty(propertyName="destination", propertyValue="OrderQueue")
})
public class OrderRecvMDB implements MessageListener{

private static final String ORDER_MGMT_INFO = "order_mgmt.properties";
private static final String ORDER_REPO = "order.repo";

public OrderRecvMDB() {
}

public void onMessage(Message message) {
TextMessage textMessage = (TextMessage) message;
try {
System.out.println("Order Received \n"+ textMessage.getText());
} catch (JMSException e) {
e.printStackTrace();
}
}
}
```

このアプリケーションには、OrderQueue でリッスンする MDB があります。openejb-jar.xml では Geronimo に MDB が jms-resources という JMS リソース・グループに所属していることを通知します。これは OrderRecvMDB と OrderQueue を CommonConnectionFactory 経由でリンクしています。

openejb-jar.xml

```
<openejb-jar
xmlns="http://www.openejb.org/xml/ns/openejb-jar-2.1"
```

```
xmlns:naming="http://geronimo.apache.org/xml/ns/naming-1.1"
xmlns:security="http://geronimo.apache.org/xml/ns/security-1.1"
xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2">
<sys:environment>
<sys:moduleId>
<sys:groupId>${pom.groupId}</sys:groupId>
<sys:artifactId>${pom.artifactId}</sys:artifactId>
<sys:version>${version}</sys:version>
<sys:type>jar</sys:type>
</sys:moduleId>
<sys:dependencies>
<sys:dependency>
<sys:groupId>org.apache.geronimo.configs</sys:groupId>
<sys:artifactId>activemq-broker</sys:artifactId>
<sys:type>car</sys:type>
</sys:dependency>
</sys:dependencies>
<sys:hidden-classes/>
<sys:non-overridable-classes/>
</sys:environment>
<enterprise-beans>
<message-driven>
<ejb-name>OrderRecvMDB</ejb-name>
<resource-adapter>
<resource-link>jms-resources</resource-link>
</resource-adapter>
</message-driven>
</enterprise-beans>
</openejb-jar>
```

geronimo-application.xml と application.xml は EAR の主たるコンポーネントを定義しています。EJB コンポーネントと Web アーカイブ情報は通常これらのファイルにかかれます。この geronimo-application.xmlはまた、JMS キューとそれへ接続するための一般的なキューのコネクション・ファクトリーを定義するセクションも含みます。これは ear に組み込まれた geronimo-activemq-ra.rar のデプロイに利用されます。

geronimo-application.xml

```
<application xmlns="http://geronimo.apache.org/xml/ns/j2ee/application-1.1">
<environment xmlns="http://geronimo.apache.org/xml/ns/deployment-1.2">
<moduleId>
<groupId>${pom.groupId}</groupId>
<artifactId>${pom.artifactId}</artifactId>
<version>${version}</version>
<type>ear</type>
</moduleId>
</environment>
<module>
<connector>geronimo-activemq-ra-2.0-SNAPSHOT.rar</connector>
<connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.2">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2">
<dep:moduleId>
<dep:groupId>${pom.groupId}</dep:groupId>
<dep:artifactId>jms-resources</dep:artifactId>
<dep:version>${version}</dep:version>
<dep:type>rar</dep:type>
</dep:moduleId>
<dep:dependencies>
<dep:dependency>
<dep:groupId>org.apache.geronimo.configs</dep:groupId>
<dep:artifactId>activemq-broker</dep:artifactId>
<dep:type>car</dep:type>
</dep:dependency>
</dep:dependencies>
</dep:environment>
<resourceadapter>
<resourceadapter-instance>
```

```
<resourceadapter-name>jms-resources</resourceadapter-name>
<nam:workmanager xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.2">
<nam:gbean-link>DefaultWorkManager</nam:gbean-link>
</nam:workmanager>
</resourceadapter-instance>
<outbound-resourceadapter>
<connection-definition>
<connectionfactory-interface>javax.jms.ConnectionFactory</connectionfactory-interface>
<connectiondefinition-instance>
<name>CommonConnectionFactory</name>
<implemented-interface>javax.jms.QueueConnectionFactory</implemented-interface>
<implemented-interface>javax.jms.TopicConnectionFactory</implemented-interface>
<connectionmanager>
<xa-transaction>
<transaction-caching/>
</xa-transaction>
<single-pool>
<match-one/>
</single-pool>
</connectionmanager>
</connectiondefinition-instance>
</connection-definition>
</outbound-resourceadapter>
</resourceadapter>
<adminobject>
<adminobject-interface>javax.jms.Queue</adminobject-interface>
<adminobject-class>org.apache.activemq.command.ActiveMQQueue</adminobject-class>
<adminobject-instance>
<message-destination-name>OrderQueue</message-destination-name>
<config-property-setting name="PhysicalName">OrderQueue</config-property-setting>
</adminobject-instance>
</adminobject>
<adminobject>
<adminobject-interface>javax.jms.Topic</adminobject-interface>
<adminobject-class>org.apache.activemq.command.ActiveMQTopic</adminobject-class>
</adminobject>
</connector>
</module>
</application>
```

application.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/
application_5.xsd" version="5">
<description>Geronimo Sample EAR for jms-mdb-sample</description>
<display-name>Geronimo Sample EAR for jms-mdb-sample</display-name>
<module>
<connector>geronimo-activemq-ra-2.0-SNAPSHOT.rar</connector>
</module>
<module>
<ejb>jms-mdb-sample-ejb-2.0-SNAPSHOT.jar</ejb>
</module>
<module>
<web>
<web-uri>jms-mdb-sample-war-2.0-SNAPSHOT.war</web-uri>
<context-root>/order</context-root>
</web>
</module>
</application>
```

クライアントの実装

OrderSenderServlet.java サーブレットはウェブのフォームを分析し、メッセージを生成し、CommonConnectoryFactory 経由で OrderQueue へメッセージを送信します。

> ⚠ Geronimo は @Resource の 'mappedName' 属性を無視することに注意してください。その代わりにアノテーションは 'name' を使ってください。

OrderSenderServlet.java

```
public class OrderSenderServlet extends HttpServlet {

@Resource(name="CommonConnectionFactory")
private ConnectionFactory factory;

@Resource(name="OrderQueue")
private Queue receivingQueue;

public void init() throws ServletException {
super.init();
}

protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,
IOException {
manageOrders(req,res);
}

protected void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException,
IOException {
doGet(req,res);
}

private void manageOrders(HttpServletRequest req, HttpServletResponse res) throws ServletException,
IOException{
String path = "/error.jsp";
Connection connection = null;
MessageProducer messageProducer = null;
Session sess = null;
try
{
String customerId = req.getParameter("customerId");
String orderId = req.getParameter("orderId");
String qty = req.getParameter("quantity");
String model = req.getParameter("model");

if(!customerId.equals("") && !orderId.equals("") && !qty.equals("")){
System.out.println("Start Sending Order Request");
// creating online order request
String orderRequest = "<Order orderId=\""+orderId+"\" custId=\""+customerId+"\" qty=
\""+qty+"\" model=\""+model+"\"/>" ;
connection = factory.createConnection();
sess = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
path = "/index.jsp";
TextMessage msg = sess.createTextMessage("<OrderId=" + orderId + " CustomerId=" + customerId
+ " Quantity=" + qty + " Model=" + model + ">" );
messageProducer = sess.createProducer(receivingQueue);
messageProducer.send(msg);
System.out.println("Order Request Send");
} else{
String error = "";

if(customerId.equals("")){
error = "Customer Id Cannot be Empty";
}else if(orderId.equals("")){
error = "Order Id Cannot be Empty";
```

```
}else if(qty.equals("")){
error = "Quantity Cannot be Empty";
}
req.setAttribute("error",error);
}
} catch (Exception e)
{
System.out.println("Error "+e);
e.printStackTrace();
} finally {
try {
if(messageProducer != null) messageProducer.close();
if(sess != null)sess.close();
if(connection!= null)connection.close();
} catch (JMSException e) {
e.printStackTrace();
}
}
getServletContext().getRequestDispatcher(path).forward(req,res);
}
}
```

このアーカイブの web.xml では、キューのコネクション・ファクトリーとキューとの関連する構成を記述しています。これはリソースをローカル環境で参照するためには必須です。

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd">
<description>JMS Servlet Sample</description>
<servlet>
<servlet-name>OrderSenderServlet</servlet-name>
<servlet-class>org.apache.geronimo.samples.order.OrderSenderServlet</servlet-class>
<load-on-startup>0</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>OrderSenderServlet</servlet-name>
<url-pattern>/order</url-pattern>
</servlet-mapping>

<resource-ref>
<res-ref-name>CommonConnectionFactory</res-ref-name>
<res-type>javax.jms.QueueConnectionFactory</res-type>
<res-auth>Container</res-auth>
<es-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>

<message-destination-ref>
<message-destination-ref-name>OrderQueue</message-destination-ref-name>
<message-destination-type>javax.jms.Queue</message-destination-type>
<message-destination-usage>Produces</message-destination-usage>
<message-destination-link>OrderQueue</message-destination-link>
</message-destination-ref>

<welcome-file-list>
<welcome-file>/index.jsp</welcome-file>
</welcome-file-list>

</web-app>
```

このウェブ・アプリケーションはサーブレット 2.5 仕様をサポートしていることに注意してください。古いバージョン（2.4）での構成のいくつかは上記 web.xml とは若干違います。

アノテーションがキューやコネクション・ファクトリーの参照解決に利用されるので、geronimo-web.xml は今回は不要です。

ツールの利用

この注文アプリケーションのデプロイとビルドには以下のツールを利用しています。

Apache Maven 2

Maven はエンタープライズ Java プロジェクト向けの有名なオープンソースのビルドツールです。ビルド作業の負荷を軽減できるように設計されました。Maven では Ant や他の伝統的な make ファイルなどで利用されるタスク・ベースの手法ではなく、宣言的手法を利用して、プロジェクトの構成や内容が定義されます。このことにより、企業全体で開 発標準を適用することや、ビルド用スクリプトの記述やメンテナンスに必要な時間を削ることの助けとなります。宣言的で、ライフサイクルに基づく手法を使っていた Maven 1 は、伝統的なビルド方法よりも、多くの人にとって、根本的な発展となり、さらに Maven 2 はこの点を高めました。Maven 2 は次の URL からダウンロードできます。
http://maven.apache.org

サンプル・アプリケーションの構成、ビルド、デプロイ

以下のリンクから Time Reporting アプリケーションをダウンロードしてください。
jms-mdb-sample

ファイルを解凍すると、jms-mdb-sample ディレクトリーが作られます。

ソース・コード

SVN からサンプルのソースコードをチェックアウトすることができます。

svn checkout http://svn.apache.org/repos/asf/geronimo/samples/trunk/samples/jms-mdb-sample

ビルド

jms-mdb-sample にはすでにデプロイできる ear ファイルが含まれています。ですが、ソースからビルドすることもできます。

コマンド・プロンプトを利用して jms-mdb-sample ディレクトリーへ移動し、mvn install site コマンドを入力するとビルドされます。jms-mdb-sample フォルダーの下に jms-mdb-sample-ear-2.0-SNAPSHOT.ear が作られます。

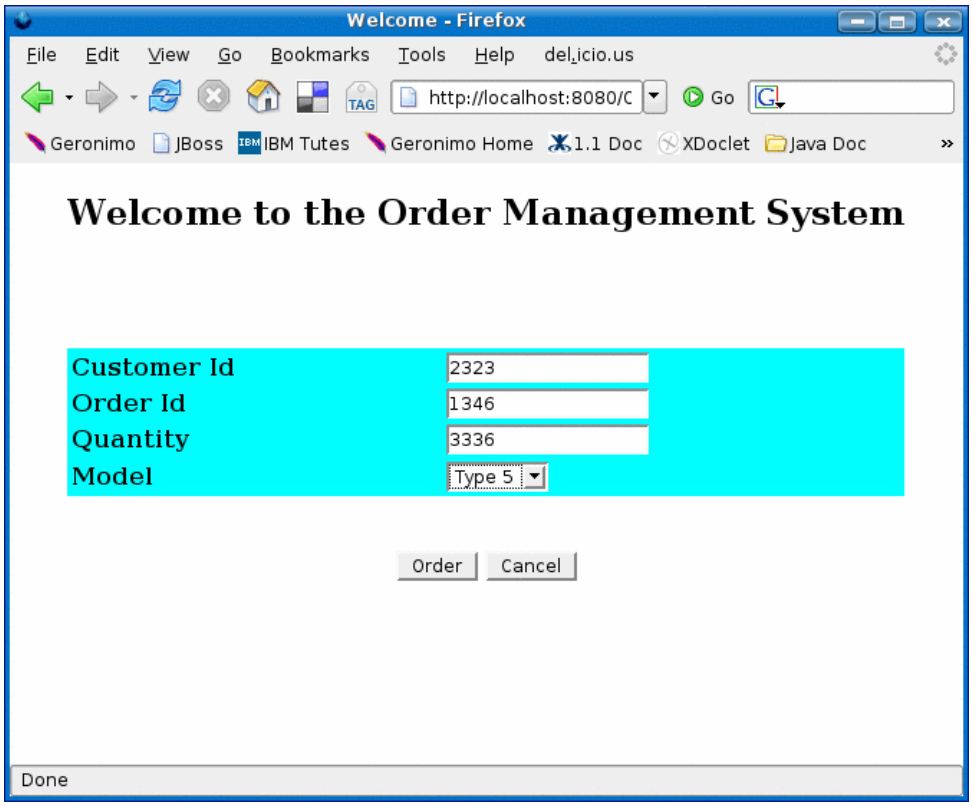
デプロイ

Order processing サンプル・アプリケーションのデプロイは、JMS リソースのデプロイとほとんど同じです。

1. Console Navigation パネルから Deploy New を選択してください。
2. Archive 入力欄に jms-mdb-sample フォルダーの jms-mdb-sample-ear-2.0-SNAPSHOT.ear を読み込んでください。
3. Install ボタンを押してアプリケーションをサーバーへデプロイしてください。

サンプル・アプリケーションのテスト

サンプル・ウェブ・アプリケーションをテストするには、ブラウザーを開いて http://localhost:8080/order を入力してください。注文管理の Welcome ページが表示されます。ユーザーは注文に必要な情報を入力し、サブミットします。



注文処理の後、コンソール上にメッセージが表示されているでしょう。

まとめ

この文章では ActiveMQ JMS サーバーを持つ Apache Geronimo の JMS 機能の利用をデモンストレーションしました。JMS 機能を広く利用する仮定的な例となります。

この文章での重要なことは以下の通りです。

- Geronimo 環境における JMS コネクション・ファクトリーと関連するキューの定義
- メッセージ駆動 Beans は J2EE コンテナーによって与えられた JMS キュー上でコンポーネントのリスニングをしている

9.a. LDAP サンプル・アプリケーション

This page last changed on 4 25, 2008 by JAGUG.

Geronimo はディレクトリー・サービスとして Apache Directory Server を利用しています。これは Apache Directory Project に含まれています。Geronimo は ApacheDS プロジェクトの内、以下の2つのプロジェクトを実装しています。

- ApacheDS Core
  すべてのバックエンドのサブ・システムを含むサーバーのコア部分です。プロトコルに依存し、seda を利用して LDAP リクエストのサービスを行います。コアには JNDI プロバイダー、インターセプター・フレームワーク、インターセプター・サービス、スキーマ・サブ・システムとデータベース・サブ・システムを含みます。

- ApacheDS Shared
  コアと maven プラグインとのプロジェクトの循環的依存性を排除するものです。一般的にモジュールを横断的に共有するどのようなコードもここで実行することによって、他のモジュールに依存しないようになります。

これら2つのプロジェクトの詳細については、以下の ApacheDS プロジェクトの URL を参照してください。

http://directory.apache.org/subprojects/apacheds/projects/index.html

今時点では、Geronimo は LDAP の参照機能のみを持っていて、編集することはできません。これは次の Geronimo のリリースで機能追加が計画されています。ldapbrowser/editor、jxplorer や gq などの外部 LDAP クライアントを利用して Geronimo のディレクトリー・サーバーの構成の編集を行ってください。

この文章は以下のセクションから構成されています。

LDAP サーバーの始動

今回の Geronimo のリリースでは、Apache Directory v0.92 が配布物の中にすでに含まれていますが、デフォルトでは始動していません。コマンド・ラインでデプロイヤー・ツールを利用するか、Geronimo 管理コンソールを通じて始動することができます。

管理コンソールを利用して、左側の navigation メニューにある System Modules をクリックし、org.apache.geronimo.configs/directory という名前のコンポーネントを Installed System Modules ポートレットから探し出してください。各コンポーネントの現在の状況と利用可能なコマンドが表示されています。

既に述べたように、このコンポーネントはデフォルトでは停止しています。Start をクリックすると、このサービスが利用できるようになります。

また、もし貴方が Geronimo のカスタム・バージョンを自身でビルドしているなら、Geronimo プラグインを通じて Apache Directory を後からインストールすることができます。

LDAP サンプル・アプリケーション

貴方に都合が良いように、サンプル・アプリケーションとデプロイメント・プランを zip ファイルにまとめました。
以下の URL からサンプル・アプリケーションをダウンロードしてください。

ldap-sample-app

zip ファイルを展開すると ldap-sample-app が作られます。このディレクトリーが <ldap_home> として参照されます。

ここでは、貴方がすでに LDAP クライアントをインストールし、ディレクトリー・サーバーへの **`.ldif`** ファイルのエクスポート/インポートができるものとします。

ソース・コード

SVN からこのサンプルのソースコードをチェックアウトすることができます。

svn checkout http://svn.apache.org/repos/asf/geronimo/samples/trunk/samples/ldap-sample-app

Back to Top

LDAP エントリーの追加

Geronimo が起動し、ディレクトリー・サービスが始動していることを確認してください。LDAP クライアントを始動し、接続プロファイルを以下の値で作成してください。

Host:	<localhost>
Port:	10389
Base DN:	ou=system
User DN:	uid=admin,ou=system
Password:	secret

Geronimo ディレクトリー・サーバーへ接続すると、初期構成が表示されます。この構成はバックアップとして ldif ファイルをエクスポートできます。LDAP クライアントによってエクスポート/インポート手順には違いがあります。例えば、ldapsearch ツールを利用した初期構成のエクスポートは以下のコマンドで実行されます。

`ldapsearch -h localhost -p 10389 -b "ou=system" -D "uid=admin,ou=system" -w secret -x "(objectclass=``)"*`

初期構成のエクスポートをすると、以下の例のような内容をもつ ldif ファイルが得られます。

export.ldif

```
dn: ou=system
ou: system
objectClass: organizationalUnit
objectClass: top

dn: uid=admin, ou=system
displayName: Directory Superuser
uid: admin
userPassword:: c2VjcmV0
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
objectClass: top
sn: administrator
cn: system administrator

dn: ou=users, ou=system
ou: users
objectClass: organizationalUnit
objectClass: top

dn: ou=groups, ou=system
ou: groups
objectClass: organizationalUnit
objectClass: top

dn: ou=configuration, ou=system
ou: configuration
```

```
objectClass: organizationalUnit
objectClass: top

dn: ou=partitions, ou=configuration, ou=system
ou: partitions
objectClass: organizationalUnit
objectClass: top

dn: ou=services, ou=configuration, ou=system
ou: services
objectClass: organizationalUnit
objectClass: top

dn: ou=interceptors, ou=configuration, ou=system
ou: interceptors
objectClass: organizationalUnit
objectClass: top

dn: prefNodeName=sysPrefRoot, ou=system
objectClass: extensibleObject
prefNodeName: sysPrefRoot
```

Back to Top

次に、サンプル・アプリケーションの実行に必要なエントリーをインポートします。サンプル・アプリケーションに、そのエントリーのサンプルの .ldif ファイルが含まれています。このファイルは <ldap_home>/ldap-sample.ldif にあります。ldapmodify ツールでデータをインポートするには、以下のコマンドを実行してください。

`ldapmodify -h localhost -p 10389 -D "uid=admin,ou=system" -w secret -x -a -f <ldap_home>/ldap-sample.ldif`

以下の例に、ldap-sample.ldif ファイルの内容を示します。

ldap-sample.ldif

```
# User: system

dn: uid=system,ou=users,ou=system
cn: John Doe
sn: Doe
givenname: John
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
ou: Human Resources
ou: People
l: Las Vegas
uid: system
mail: system@apachecon.comm
telephonenumber: +1 408 555 5555
facsimiletelephonenumber: +1 408 555 5556
roomnumber: 4613
userPassword: manager

# User: user1

dn: uid=user1,ou=users,ou=system
cn: User
sn: One
givenname: User1
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
ou: Human Resources
ou: People
```

```
l: Las Vegas
uid: user1
mail: user1@apachecon.comm
telephonenumber: +1 408 555 5555
facsimiletelephonenumber: +1 408 555 5556
roomnumber: 4613
userPassword: p1

# User: user2

dn: uid=user2,ou=users,ou=system
cn: User
sn: Two
givenname: User2
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
ou: Human Resources
ou: People
l: Las Vegas
uid: user2
mail: user2@apachecon.comm
telephonenumber: +1 408 555 5555
facsimiletelephonenumber: +1 408 555 5556
roomnumber: 4613
userPassword: p2

# Group: admin

dn: cn=admin,ou=groups,ou=system
objectClass: groupOfUniqueNames
uniqueMember: uid=system,ou=users,ou=system
uniqueMember: uid=user2,ou=users,ou=system
cn: admin

# Group: guest

dn: cn=guest,ou=groups,ou=system
objectClass: groupOfUniqueNames
uniqueMember: uid=user1,ou=users,ou=system
cn: guest
```

ファイルをインポートすると、5つのエントリーが無事追加されたことが通知されます。

Back to Top

LDAP レルムのデプロイ

LDAP サンプル・アプリケーションには、アプリケーション自身のデプロイ前にデプロイしておく必要のあるセキュリティ・レルムがあります。このレルムは <ldap_home>/ldap_realm.xml にあり、以下のような内容です。

ldap-realm.xml

```
<module xmlns="http://geronimo.apache.org/xml/ns/deployment-1.2">
<environment>
<moduleId>
<groupId>console.realm</groupId>
<artifactId>LDAP_Sample_Realm</artifactId>
<version>1.0</version>
<type>car</type>
</moduleId>
<dependencies>
<dependency>
<groupId>org.apache.geronimo.configs</groupId>
```

```
<artifactId>j2ee-security</artifactId>
<type>car</type>
</dependency>
</dependencies>
</environment>
<gbean name="LDAP_Sample_Realm" class="org.apache.geronimo.security.realm.GenericSecurityRealm"
xsi:type="dep:gbeanType" xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<attribute name="realmName">LDAP_Sample_Realm</attribute>
<reference name="ServerInfo">
<name>ServerInfo</name>
</reference>
<xml-reference name="LoginModuleConfiguration">
<log:login-config xmlns:log="http://geronimo.apache.org/xml/ns/loginconfig-1.2">
<log:login-module control-flag="REQUIRED" wrap-principals="false">
<log:login-domain-name>LDAP_Sample_Realm</log:login-domain-name>
<log:login-module-class>org.apache.geronimo.security.realm.providers.LDAPLoginModule</log:login-
module-class>
<log:option name="initialContextFactory">com.sun.jndi.ldap.LdapCtxFactory</log:option>
<log:option name="connectionURL">ldap://localhost:10389</log:option>
<log:option name="connectionUsername">uid=admin,ou=system</log:option>
<log:option name="connectionPassword">secret</log:option>
<log:option name="authentication">simple</log:option>
<log:option name="userBase">ou=users,ou=system</log:option>
<log:option name="userSearchMatching">uid={0}</log:option>
<log:option name="userSearchSubtree">false</log:option>
<log:option name="roleBase">ou=groups,ou=system</log:option>
<log:option name="roleName">cn</log:option>
<log:option name="roleSearchMatching">(uniqueMember={0})</log:option>
<log:option name="roleSearchSubtree">false</log:option>
</log:login-module>
<log:login-module control-flag="OPTIONAL" wrap-principals="false">
<log:login-domain-name>LDAP_Sample_Realm-Audit</log:login-domain-name>
<log:login-module-class>org.apache.geronimo.security.realm.providers.FileAuditLoginModule</
log:login-module-class>
<log:option name="file">var/log/login-attempts.log</log:option>
</log:login-module>
</log:login-config>
</xml-reference>
</gbean>
</module>
```

このデプロイメント・プランは、Geronimo に対してすべての接続や検索に LDAP データベースが必要であることが記述されています。また、このプランではすべてのログイン処理についても `login-attempts.log` というログファイルに記録するように記述されています。

ldap-realm.xml をデプロイするために、以下のコマンドを <geronimo_home>/bin ディレクトリーから実行してください。

`java -jar deployer.jar --user system --password manager deploy <ldap_home>/ldap-realm.xml`

デプロイすると、以下のような確認メッセージが表示されるでしょう。

```
D:\geronimo-tomcat6-jee5-2.0\bin>deploy deploy \samples\2.0\ldap-sample-app\ldap-realm.xml
Using GERONIMO_BASE:   D:\geronimo-tomcat6-jee5-2.0
Using GERONIMO_HOME:   D:\geronimo-tomcat6-jee5-2.0
Using GERONIMO_TMPDIR: D:\geronimo-tomcat6-jee5-2.0\var\temp
Using JRE_HOME:        C:\Java\jdk1.5.0_06\\jre
Deployed console.realm/LDAP_Sample_Realm/1.0/car
```

Back to Top

詳しくは 1.1.3.3.3. LDAPレルム セクションを参照してください。

デプロイメント・プラン

デプロイメント・プランは <ldap_home>/WEB-INF ディレクトリーにあります。geronimo-web.xml は Geronimo 固有のデプロイメント・プランです。何のセキュリティ・レルムを使うのかの詳細が記述され、セキュリティ構成における一意のエレメントを使った Geronimo 固有のネーム・スペースと同様に、ユーザー・ロールのマッピングが記述されます。一般的な他の種類のアプリケーションでは、まさにセキュリティというわけではありませんが、デプロイメント・プランにはそのプランの主たるネーム・スペースが与えられ、モジュールの一意名（任意です）、親モジュール構成 ID（これも任意です）、コンテキスト・ルートが記述されます。以下に例として Geronimo 仕様のデプロイメント・プランを示します。

geronimo-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.2">
<environment>
<moduleId>
<groupId>samples</groupId>
<artifactId>LDAP_Sample</artifactId>
<version>1.2</version>
</moduleId>
</environment>
<context-root>/LDAP_Sample</context-root>

<security-realm-name>LDAP_Sample_Realm</security-realm-name>
<security>
<default-principal realm-name="LDAP_Sample_Realm">
<principal class="org.apache.geronimo.security.realm.providers.GeronimoUserPrincipal"
name="system"/>
</default-principal>
<role-mappings>
<role role-name="content-administrator">
<realm realm-name="LDAP_Sample_Realm">
<principal class="org.apache.geronimo.security.realm.providers.GeronimoGroupPrincipal" name="admin"
designated-run-as="true"/>
<principal class="org.apache.geronimo.security.realm.providers.GeronimoUserPrincipal"
name="system"/>
</realm>
</role>
<role role-name="guest">
<realm realm-name="LDAP_Sample_Realm">
<principal class="org.apache.geronimo.security.realm.providers.GeronimoGroupPrincipal" name="guest"
designated-run-as="true"/>
<principal class="org.apache.geronimo.security.realm.providers.GeronimoUserPrincipal" name="user1"/
>
</realm>
</role>
</role-mappings>
</security>
</web-app>
```

デプロイメント・プランの最初の部分はわかりやすいのですが、セキュリティ構成は手が込んでいます。<security-realm-name> は <security> 要素内の一連の <realms> 要素の定義に利用されています。

web.xml のセキュリティ・ロール（security roles）の記述によって、geronimo-web.xml はユーザーやグループがどの Geronimo セキュリティー・レルムに所属しているかを割り当てます。ログインしていないユーザーには、<default-principal> 要素によって定義された realm が付与されます。

本プロジェクトには 内容管理者（content-administrator） と ゲスト（guest） という2つの principal があります。ともに GeronimoGroupPrincipal と GeronimoUserPrincipal の2つの要素を持っています。principal には designated-run-as フラグがオンになっている要素がものがあり、これらは web.xml によって run-as ロールが付与されます。

これらのロール・マッピングは LDAP サーバーに実際に定義されているロール（どのユーザーがどのグループに所属するか）によって上書きされることに注意してください。結局レルムは、アプリケーションのデプロイメント・プランがどの検証方法を用いるかを定義しています。とはいえ、今回の例では XML スキーマ に従って principal とロール・マッピングを定義する必要があります。

Back to Top

以下にファイル位置によるセキュリティ制限を追加した web.xml デプロイメント記述の例（また、<ldap_home>/WEB-INF ディレクトリーに位置していること）を示します。

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd"
version="2.4">

<welcome-file-list>
<welcome-file>index.html</welcome-file>
</welcome-file-list>

<security-constraint>
<web-resource-collection>
<web-resource-name>Admin Role</web-resource-name>
<url-pattern>/protect/*</url-pattern>
</web-resource-collection>
<auth-constraint>
<role-name>content-administrator</role-name>
</auth-constraint>
</security-constraint>

<security-constraint>
<web-resource-collection>
<web-resource-name>No Access</web-resource-name>
<url-pattern>/forbidden/*</url-pattern>
</web-resource-collection>
<auth-constraint/>
</security-constraint>

<login-config>
<auth-method>FORM</auth-method>
<realm-name>ldap-realm-1</realm-name>
<form-login-config>
<form-login-page>/auth/logon.html?param=test</form-login-page>
<form-error-page>/auth/logonError.html?param=test</form-error-page>
</form-login-config>
</login-config>

<security-role>
<role-name>content-administrator</role-name>
</security-role>

</web-app>
```

Back to Top

サンプル・アプリケーションのパッケージ

すべての要素が一意になっていて、ウェブ・アプリケーション・アーカイブ (.war) にサンプル・アプリケーションにパッケージされる必要があります。コマンド・ライン・ウィンドウを開き、<ldap_home> ディレクトリーに移動し、以下のコマンドを実行してください。

jar -cvf ldap-demo.war

このコマンドで <ldap_home> にあるすべてのファイルをパッケージします。.war ファイル内に必要はありませんが、ldap-realm.xml と ldap-sample.ldif ファイルも含まれます。

Back to Top

アプリケーションのデプロイとテスト

LDAP サンプル・アプリケーションをデプロイするために、Geronimo サーバーが始動していることを確認してください。コマンド・ライン・ウィンドウを開き、<geronimo_home>/bin ディレクトリーに移動し、以下のコマンドを実行してください。

```
java -jar deployer.jar --user system --password manager deploy <ldap_home>/ldap-demo.war
```

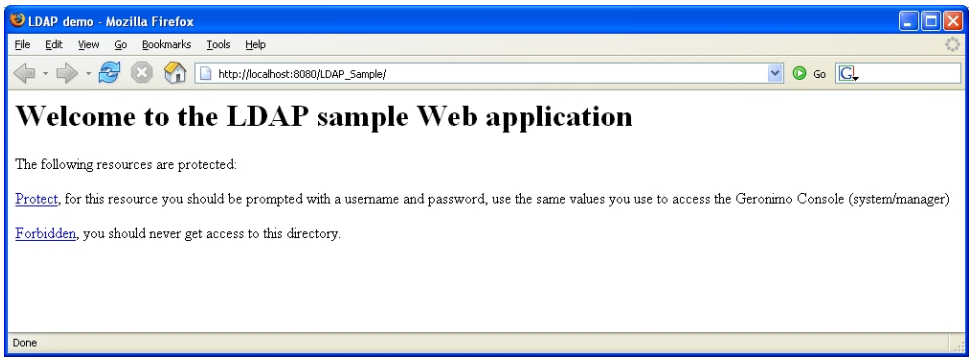
ウェブ・アプリケーションのデプロイが成功すると、以下に似たようなメッセージが表示されるでしょう。

```
D:\geronimo-tomcat6-jee5-2.0\bin>deploy deploy \samples\2.0\ldap-sample-app\ldap-demo.war
Using GERONIMO_BASE:   D:\geronimo-tomcat6-jee5-2.0
Using GERONIMO_HOME:   D:\geronimo-tomcat6-jee5-2.0
Using GERONIMO_TMPDIR: D:\geronimo-tomcat6-jee5-2.0\var\temp
Using JRE_HOME:        C:\Java\jdk1.5.0_06\\jre
Deployed samples/LDAP_Sample/1.2/war @
http://localhost:8080/LDAP_Sample
```

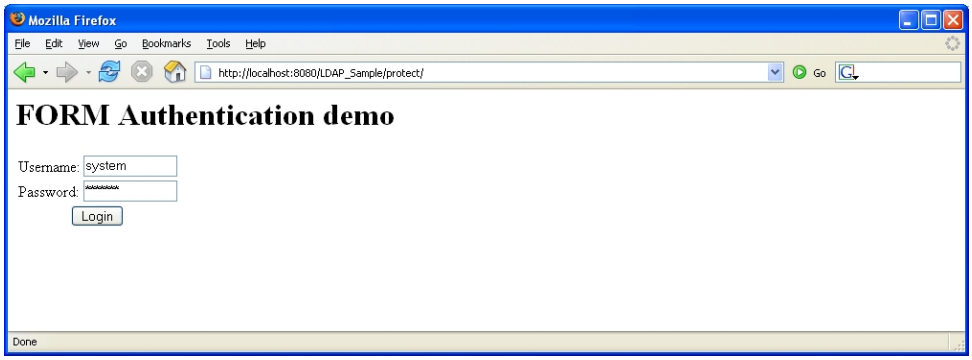
LDAP アプリケーションのテストは、ウェブ・ブラウザーを開き、以下の URL へ接続してください。

http://localhost:8080/LDAP_Sample

LDAP サンプル・アプリケーションの Welcome ページが表示されます。



Protect をクリックして、LDAP ディレクトリー・サーバーの認証を受けてください。



ユーザー名に system パスワードに manager を入力して Login をクリックしてください。ここで入力するユーザー名とパスワードは Geronimo ウェブ・コンソールへの接続時に利用するものと同じで、ディレクトリー・サーバーのデータベースに保存されています。ログインすると、以下のような画面が表示されます。

ここでは、前に LDAP レルムに与えたセキュリティ構成に基づいて LDAP ディレクトリー・サーバーでユーザー名とパスワードの検証が行われます。さて次は、welcome ページに戻り、Forbidden をクリックしてください。以下のような 403 - Forbidden HTTP エラーが表示されるでしょう。

利用しているウェブ・コンテナー (Jetty または Tomcat) に応じて、この画面は少し違うかもしれません。

この例でさらなるテストをするには、**`ldap-sample.ldif`** に与えられている違うユーザで試してみてください。LDAP クライアントを利用して、違うグループのユーザーを追加/削除してください。変更はすぐに反映されます。(ウェブ・ブラウザーを閉じる必要があるかもしれません)

Back to Top

9.b. データベース接続の簡単なサンプル・アプリケーション

This page last changed on 4 25, 2008 by JAGUG.

この文章は Apache Geronimo アプリケーション・サーバーの持つ JDBC 機能を解説します。JDBC 機能のデモンストレーションでは、JSP、サーブレットによりウェブ関連の機能や組み込みデータベースの Derby を扱う単純な Inventory アプリケーションを利用します.

Inventory アプリケーションはデータベースへの接続方法にサーバー・プロバイダー・インターフェース (SPI) を利用します。この方法では、アプリケーションはデータベースとの接続確立のために JDBC データソース・インターフェースを利用します。これが J2EE アプリケーションの接続方法としてふさわしいのは、以下のいくつかの理由によります。

- プログラム・コードは全体的にデータベース非依存となります。ドライバー情報、データベースの位置情報、構成パラメーターが J2EE サーバー内に保持されます。
- コネクション・プーリングの利用が可能です。J2EE サーバー接続マネージャーは効果的に接続を処理し、性能や拡張性を改善します。

- J2EE サーバーの一部としてビジネス・ロジックを実装しているエンタープライズ JavaBeans (EJB) を利用してデータベースが利用できるようになります。必須ではありませんが、EJB 層の実装によりアプリケーションの構造に対して高い拡張性あたえる基礎となります。

この文章を読み終えると、Geronimo の JDBC 機能、たとえばデータベース・プールの定義やデータソースを使ってのデータベース接続などの機能を最大限活用することができるようになるでしょう。

この文章は以下のセクションで構成されています。

JDBC 機能の概略

アプリケーション・サーバーへの JDBC の実装は、アプリケーション・サーバーごとに異なります。以下の表は Apache Geronimo の JDBC 機能一覧です。

Feature	Description
JDBC access	Geronimo は JDBC を直接統合していませんが、総括的な J2CA フレームワークを通しての接続をサポートしています。TranQL プロジェクトは様々なデータベース用 J2CA アダプターです。
JCA implementation	Geronimo は JCA 1.5 仕様をサポートしています。これは JCA 1.0 仕様と下位互換です。
Data sources supported	TranQL は各種データソースの総括的ラッパーです。
Data source failover	TranQL は特定のデータベース (Apache Derby、Oracle、DB2 を含みます) 向けに特化されたドライバーを持っていますので、それらドライバーの拡張機能と密接に統合されます。 例えばロード・バランシングやフェイル・オーバーなどの機能が提供されます。クラスタリングやフェイル・オーバーのために C-JDBC ラッパーを利用することも可能です。
XA support	XA トランザクション、ローカル・トランザクション、トランザクション無しをサポートしています。
Connection Manager Configurability	J2CA フレームワークは異なるコネクション・フレームワークの組み込みを可能にするインターセプターを持ちます。

JTA implementation	トランザクションのサポートは Geronimo Specific Transaction Managing フレームワーク と HOWL により与えられます。
Connection pooling and management	カスタムの Geronimo コードと TranQL はコネクション・プーリングと管理に利用されます。
Legacy driver support	Geronimo JDBC の TranQL−コネクター を通じて Geronimo のサポートする JDBC 3.0 と 2.1 の JCA ラッパーが与えられます。

アプリケーションの概略

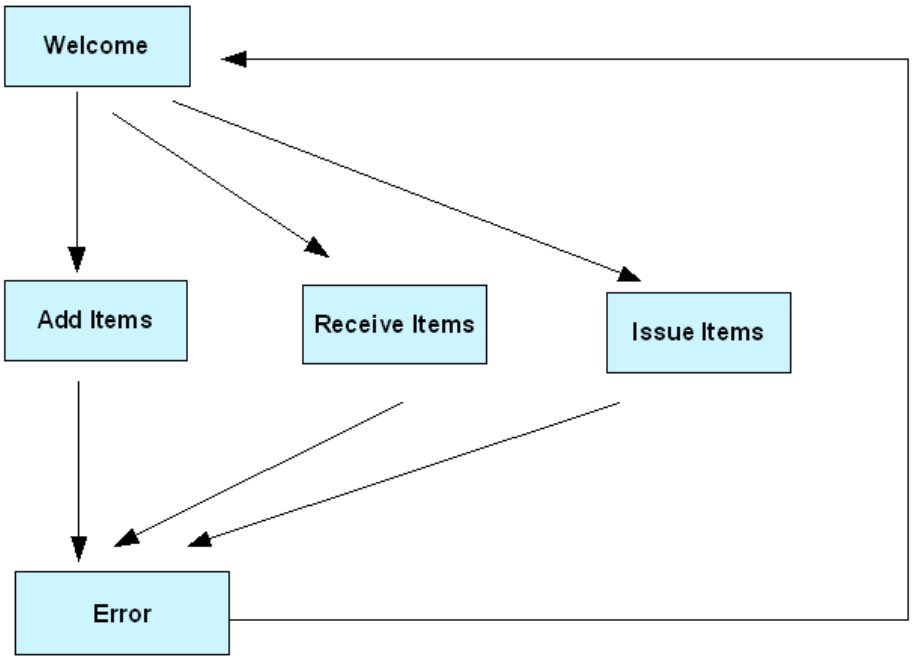
この文章にある Inventory アプリケーションは、この手のアプリケーションの利用場面のうち、基本的な3つについてのみサポートします。

1. アイテムを追加して在庫とする
2. アイテムを入荷する
3. アイテムを出荷する
   アプリケーションのワークフローはアイテム情報を追加して在庫とするところから始まります。その後、物の入荷、出荷情報を受け付けます。これらの更新された情報はすべて組み込みの Derby データベースに保存されます。

Inventory ウェブ・アプリケーションは以下のページを持ちます。

- Welcome
- Add Item
- Receive Goods
- Issue Goods

次の図はアプリケーションのフロー図です。



アプリケーションの Welcome ページは現在の各アイテムの在庫数を表示します。Welcome ページを通してユーザはアイテムの追加、物の入荷、または物の出荷ページへ遷移できます。それらの処理が成功すると、在庫情報が更新された Welcome ページにリダイレクトされます。アイテムの追加ページではアイテムの在庫数を定義することができますので、在庫数0としてアイテム情報を保持することができます。物の入荷および出荷ページはそれぞれアプリケーションの入荷と出荷処理の役割をもちます。

アプリケーションの内容

Inventory アプリケーションは以下のパッケージとクラスを含みます。

- org.apache.geronimo.samples.inventory
  - Item - Inventory でのアイテムを表します。
- org.apache.geronimo.samples.inventory.services
  - InventoryManager - Inventory のもつ一連のサービスを表します。
- org.apache.geronimo.samples.inventory.dao
  - ItemDAO - データベース接続方法をすべて含みます。
- org.apache.geronimo.samples.inventory.exception
  - DuplicateItemIdException - アイテムの ID の重複を扱うために用意した例外です。
  - NotSufficientQuantityException - 十分な在庫量が無い状況を扱うために用意した例外です。
- org.apache.geronimo.samples.inventory.util
  - DBManager - データベース接続などを扱うデータベース関連処理です。
- org.apache.geronimo.samples.inventory.web
  - AddItemServlet - サービス層にアイテム情報の追加機能を持たせます。
  - IssueingServlet - サービス層にアイテムの出荷機能を持たせます。
  - RecievingServlet - サービス層にアイテムの入荷機能を持たせます。

ウェブ・アプリケーションのファイルは以下のリストのとおりです。

```
|- jsp
|- add.jsp
|- error.jsp
|- issue.jsp
|- recv.jsp
|- WEB-INF
|- geronimo-web.xml
|- web.xml
|- welcome.jsp
```

geronimo-web.xml と web.xml ファイルによって、アプリケーションはデータソースを定義します。geronimo-web.xml は EAR ファイルに含まれ、データベース・プールへのリンクを持ちます。

geronimo-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1">

<environment>
<moduleId>
<groupId>${pom.groupId}</groupId>
<artifactId>${pom.artifactId}</artifactId>
<version>${version}</version>
<type>war</type>
</moduleId>
<dependencies></dependencies>
</environment>

<context-root>/inventory</context-root>

<!-- define a reference name to the db pool-->
<resource-ref>
<ref-name>jdbc/InventoryDS</ref-name>
<resource-link>InventoryPool</resource-link>
</resource-ref>
</web-app>
```

以下に Inventory アプリケーションの web.xml を示します。データソースを作成する際に利用される geronimo-web.xml と同じ名前を利用しています。

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
version="2.4">

<welcome-file-list>
<welcome-file>index.html</welcome-file>
</welcome-file-list>

<servlet>
<display-name>AddItemServlet</display-name>
<servlet-name>AddItemServlet</servlet-name>
<servlet-class>org.apache.geronimo.samples.inventory.web.AddItemServlet</servlet-class>
</servlet>
<servlet>
<display-name>IssueingServlet</display-name>
<servlet-name>IssueingServlet</servlet-name>
<servlet-class>org.apache.geronimo.samples.inventory.web.IssueingServlet</servlet-class>
</servlet>
<servlet>
<display-name>RecievingServlet</display-name>
<servlet-name>RecievingServlet</servlet-name>
<servlet-class>org.apache.geronimo.samples.inventory.web.RecievingServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>AddItemServlet</servlet-name>
<url-pattern>/add_item</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>IssueingServlet</servlet-name>
<url-pattern>/issue</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>RecievingServlet</servlet-name>
<url-pattern>/recv</url-pattern>
</servlet-mapping>

<!-- reference name exposed as a datasource -->
<resource-ref>
<res-ref-name>jdbc/InventoryDS</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
<res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
</web-app>
```

geronimo-application.xml には、アプリケーションにデプロイする必要のあるデータベース・プールがあることを示します。データベース・プールは InventoryPool.xml に定義されています。また、ドライバーとして tranql-connector-ra-3.3.rar ファイルをデプロイする必要があることが記述されています。これら2つのファイルは生成された EAR ファイル内の最上位階層に置きます。

```
<application xmlns="http://geronimo.apache.org/xml/ns/j2ee/application-1.1">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.1">
<dep:moduleId>
<dep:groupId>${pom.groupId}</dep:groupId>
<dep:artifactId>${pom.artifactId}</dep:artifactId>
<dep:version>${version}</dep:version>
<dep:type>ear</dep:type>
</dep:moduleId>
</dep:environment>
<module>
<connector>tranql-connector-ra-1.3.rar</connector>
<alt-dd>InventoryPool.xml</alt-dd>
</module>
</application>
```

次の重要なアプリケーションの記述はソースコードから定義されたデータソースへ接続するものです。これは DBManager クラスとして扱われます。

DBManager.java

```
public static Connection getConnection(){
Connection con = null;
try {
Context context = new InitialContext();
DataSource ds = (DataSource)context.lookup("java:comp/env/jdbc/InventoryDS");
con = ds.getConnection();
} catch (NamingException e) {
e.printStackTrace();
} catch (SQLException e) {
e.printStackTrace();
}
return con;
}
```

サンプル・データベース

このサンプル・データベースは組み込みの Derby データベースを利用しています。サンプル・データベースの名前は InventoryDB で、2つの表を持ちます。ITEM 表と ITEM_MASTER 表です。それぞれの表の列は以下に記述します。

Table Name	Fields
ITEM	ITEM_ID (PRIMARY KEY) ITEM_NAME DESCRIPTION
ITEM_MASTER	ITEM_ID (PRIMARY KEY) QUANTITY

ITEM 表は ITEM_MASTER 表の各アイテムの在庫数量を保持します。

ツールの利用

nventory サンプル・アプリケーションの開発、ビルドは次のツールを利用しています。

Apache Derby

Apache Derby は Apache DB サブ・プロジェクトであり、Java で実装されたリレーショナル・データベースです。サイズが小さく、Java に基づくソリューションに簡単に組み込めるものです。組み込みフレームワークに加え、Derby は Derby ネットワーク・サーバを利用することで、よくあるクライアント／サーバ・フレームワークもサポートします。
http://db.apache.org/derby/index.html

Apache Maven 2

Maven はエンタープライズ Java プロジェクト向けの有名なオープンソースのビルド・ツールです。ビルド作業の負荷を軽減できるように設計されました。Maven では Ant や他の伝統的な make ファイルなどで利用されるタスク・ベースの手法ではなく、宣言的手法を利用して、プロジェクトの構成や内容が定義されます。このことにより、企業全体で開 発標準を適用することや、ビルド用スクリプトの記述やメンテナンスに必要な時間を削ることの助けとなります。宣言的で、ライフサイクルに基づく手法を使っていた Maven 1 は、伝統的なビルド方法よりも、多くの人にとって、根本的な発展となり、さらに Maven 2 はこの点を高めました。Maven 2 は次の URL からダウンロードできます。
http://maven.apache.org

Back to Top

サンプル・アプリケーションの構成、ビルド、デプロイ

以下のリンクから Inventory アプリケーションをダウンロードしてください。
Inventory

ファイルを解凍すると、inventory ディレクトリーが作られます。

ソース・コード

SVN からサンプルのソースコードをチェックアウトすることができます。

svn checkout http://svn.apache.org/repos/asf/geronimo/samples/trunk/samples/inventory

構成

アプリケーションの構成は、データベースの作成とデータベースへ接続するコネクション・プールの定義を含んでいます。

データベースの作成とデータ追加

Apache Geronimo サーバーを始動した後、Geronimo コンソールへログインし、以下の手順により InventoryDB を作成してください。

InventoryDB.sql

```
CREATE TABLE item(
item_id VARCHAR(10) PRIMARY KEY,
item_name VARCHAR(25),
description VARCHAR(100)
);

CREATE TABLE item_master(
item_id VARCHAR(10) PRIMARY KEY,
quantity INTEGER
);

INSERT INTO item VALUES('001', 'Item 1', 'Test Item 1');
INSERT INTO item VALUES('002', 'Item 2', 'Test Item 2');
INSERT INTO item VALUES('003', 'Item 3', 'Test Item 3');
INSERT INTO item VALUES('004', 'Item 4', 'Test Item 4');


INSERT INTO item_master VALUES('001', 12);
INSERT INTO item_master VALUES('002', 8);
INSERT INTO item_master VALUES('003', 49);
INSERT INTO item_master VALUES('004', 34);
```

1. 左側の Console Navigation から DB Manager リンクを選択してください。
2. データベース名として InventoryDB を入力し、Create ボタンをクリックしてください。
3. Use DB 欄で InventoryDB を選択してください。
4. テキストエディタで inventory/inventory-ear/src/main/resources ディレクトリーから InventoryDB.sql を開いてください。
5. SQL Commands のテキストエリアに InventoryDB.sql の内容を貼りつけ、Run SQL ボタンを押してください。

ビルド

Inventory アプリケーションはソースコードからのビルドに利用できる pom.xml スクリプトが含まれます。コマンド・プロンプトを利用して inventory ディレクトリーへ移動し、mvn install コマンドを入力するとビルドされます。inventory フォルダーの下に inventory-ear-2.0-SNAPSHOT.ear が作られます。これで Geronimo アプリケーション・サーバーへ Inventory アプリケーションをデプロイする準備ができました。

デプロイ

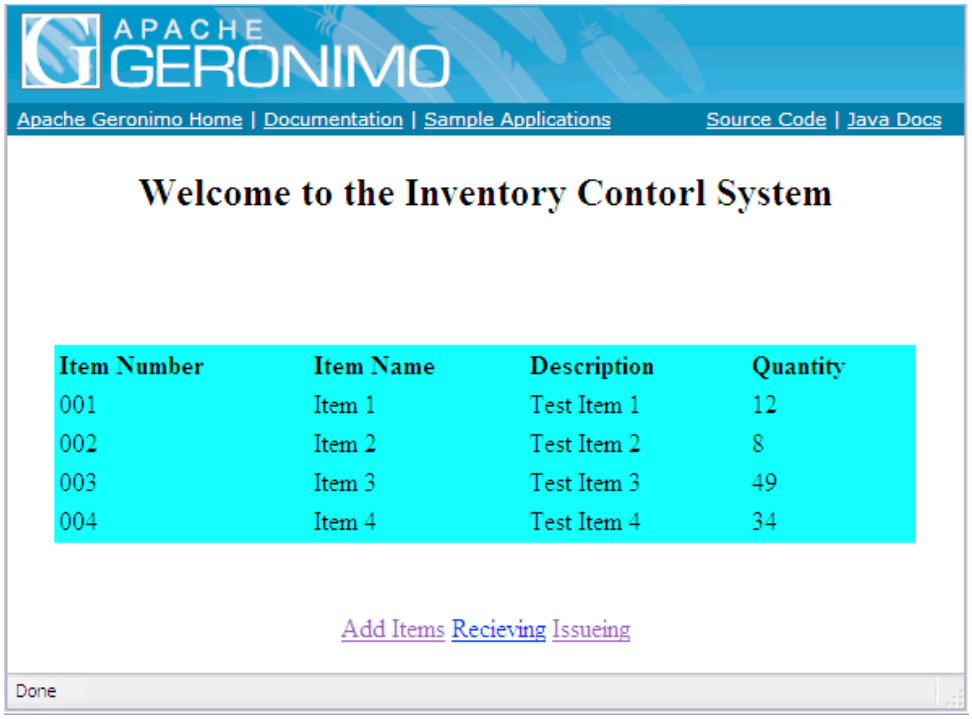
Geronimo Console の利用によって、サンプルアプリケーションのデプロイはかなり簡単です。

1. Console Navigation パネルから Deploy New を選択してください。
2. Archive 入力欄に inventory フォルダーの inventory-ear-2.0-SNAPSHOT.ear を読み込んでください。
3. Install ボタンを押してアプリケーションをサーバーへデプロイしてください。

Back to Top

サンプル・アプリケーションのテスト

サンプル・アプリケーションをテストするには、ブラウザーを開いて http://localhost:8080/inventory を入力してください。
Inventory アプリケーションの Welcome ページが表示され、そこには注目ボードがあります。



ユーザーは Welcome ページからアイテムの追加、物の入荷、物の出荷機能へ接続できます。

まとめ

この文章では Geronimo アプリケーション・サーバーでの JDBC 機能の利用方法を示しました。説明にしたがってサンプル・アプリケーションを順々にビルド、デプロイ、テストすればこれらの機能がよく分かります。

この文章の注目点は、以下の通りです。

- Apache Geronimo の JDBC 機能
- Geronimo に組み込まれた Derby データベースを利用してデータベースの作成、データの追加
- データベースへの接続のためのデータベース・プール計画のデプロイ
- Geronimo において、定義されたプールを通じてデータベースへ接続するウェブ・アプリケーションのデプロイ

This page last changed on 4 25, 2008 by JAGUG.

アプリケーションの概略

この文章で述べられているサンプル・アプリケーションは2つの整数の足し算を行う単純な計算機です。クライアント・アプリケーションは J2EE アプリケーションではありません。Web サービスをアプリケーションの機能として呼び出す通常の Java アプリケーションです。Web サービスは Geronimo アプリケーション・サーバー上にサーブレットとして公開されています。

サービスの実装

Calculator インターフェースは Web サービスのサービス・エンドポイント・インターフェース (SEI) を定義しています。

Calculator.java

```
package org.apache.geronimo.samples.jws;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

@WebService(name="CalculatorPortType",
targetNamespace = "http://jws.samples.geronimo.apache.org")
public interface Calculator {

@WebMethod
public int add(@WebParam(name = "value1") int value1,
@WebParam(name = "value2") int value2);

}
```

CalculatorService クラスは Web サービスのビジネス・ロジックを実装しています。SEI にあるすべてのメソッドを実装しています。このクラスでは Calculator インターフェースを実装する必要はありませんが、このインターフェースは @WebService.endpointInterface アノテーションを通じて参照されます。このクラスは javax.servlet.Servlet クラスを継承していませんが、web.xml ファイルを通じてサーブレットとして公開されます。
@Resource アノテーションが付けられた context 変数へ実行時に注入されます。WebServiceContext はメッセージ・コンテキストとセキュリティー情報を保持するために利用されます。

CalculatorService.java

```
package org.apache.geronimo.samples.jws;

import javax.annotation.Resource;
import javax.jws.WebService;
import javax.xml.ws.WebServiceContext;

@WebService(serviceName = "Calculator",
portName="CalculatorPort",
endpointInterface = "org.apache.geronimo.samples.jws.Calculator",
targetNamespace = "http://jws.samples.geronimo.apache.org",
wsdlLocation = "WEB-INF/wsdl/CalculatorService.wsdl")
public class CalculatorService implements Calculator {

@Resource
private WebServiceContext context;

public int add(int value1, int value2) {
System.out.println("User Principal: " + context.getUserPrincipal());
return value1 + value2;
}
}
```

web.xml は Web サービスのデプロイに利用されます。

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">
<servlet>
<display-name>CalculatorService</display-name>
<servlet-name>CalculatorService</servlet-name>
<servlet-class>
org.apache.geronimo.samples.jws.CalculatorService
</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>CalculatorService</servlet-name>
<url-pattern>/calculator</url-pattern>
</servlet-mapping>

.....
</web-app>
```

web.xml

web.xml は、簡単な JAX-WS Web サービスのデプロイには必要ありません。web.xml が与えられない場合、デプロイ時に自動的に生成されます。

geronimo-web.xml ファイルは任意ですが、このサンプルではモジュール名を記述しています。このプロジェクトの情報（例えばモジュールの一意名や依存関係など）は、<environment> タグ内に記述されています。今回は依存関係がありませんので列挙されていません。しかし、このモジュールに一連の一意名を付けておくほうがよいです。そうすれば、後々他のアプリケーションから参照することができます。

geronimo-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.1">
<dep:moduleId>
<dep:groupId>${pom.groupId}</dep:groupId>
<dep:artifactId>${pom.artifactId}</dep:artifactId>
<dep:version>${version}</dep:version>
<dep:type>war</dep:type>
</dep:moduleId>
</dep:environment>

<context-root>/jaxws-calculator</context-root>

<service-ref>
<service-ref-name>services/Calculator</service-ref-name>
<port>
<port-name>CalculatorPort</port-name>
<protocol>http</protocol>
<host>localhost</host>
<port>8080</port>
<uri>/jaxws-calculator/calculator</uri>
</port>
</service-ref>
</web-app>
```

以下のように、WSDL ファイルは Web サービスを記述します。

CalculatorService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="Calculator"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
targetNamespace="http://jws.samples.geronimo.apache.org"
xmlns:tns="http://jws.samples.geronimo.apache.org">

<wsdl:types>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://jws.samples.geronimo.apache.org"
targetNamespace="http://jws.samples.geronimo.apache.org"
attributeFormDefault="unqualified" elementFormDefault="qualified">

<xsd:element name="add">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="value1" type="xsd:int"/>
<xsd:element name="value2" type="xsd:int"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="addResponse">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="return" type="xsd:int"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
</wsdl:types>

<wsdl:message name="add">
<wsdl:part name="add" element="tns:add"/>
</wsdl:message>

<wsdl:message name="addResponse">
<wsdl:part name="addResponse" element="tns:addResponse"/>
</wsdl:message>

<wsdl:portType name="CalculatorPortType">
<wsdl:operation name="add">
<wsdl:input name="add" message="tns:add"/>
<wsdl:output name="addResponse" message="tns:addResponse"/>
</wsdl:operation>
</wsdl:portType>

<wsdl:binding name="CalculatorSoapBinding" type="tns:CalculatorPortType">
<soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>

<wsdl:operation name="add">
<soap:operation soapAction="add" style="document"/>
<wsdl:input name="add">
<soap:body use="literal"/>
</wsdl:input>
<wsdl:output name="addResponse">
<soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>

</wsdl:binding>

<wsdl:service name="Calculator">
<wsdl:port name="CalculatorPort" binding="tns:CalculatorSoapBinding">
<soap:address location="http://localhost:8080/jaxws-calculator-1.0/calculator"/>
</wsdl:port>
</wsdl:service>

</wsdl:definitions>
```

webservices.xml

J2EE バージョン 1.4 では、Web サービスの定義のために webservices.xml ファイルも必要です。Java EE 5 では、このファイルは任意です。今回の例では必要としていません。

JSP ベースの JAX-WS クライアント

add.jsp は CalculatorService Web サービスを利用する基本的なクライアントです。

add.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ page
import="javax.naming.InitialContext,javax.xml.ws.Service,org.apache.geronimo.samples.jws.Calculator"%>
<html  xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<title>Apache Geronimo Sample Application - JAX-WS Calculator</title>
<meta content="text/html; CHARSET=iso-8859-1" http-equiv="Content-Type">
</head>

<BODY>
<font face="Verdana, Helvetica, Arial">

<h3>This is a JAX-WS web service sample application.   Please type the value 1 and value 2 below to
see the add result.</h3>
<form action="add.jsp">
Value 1: <input type="text" name="value1">   Value 2: <input type="text" name="value2"> <input
type="submit" value="Add">
</form>
<br>
<%
String value1 = request.getParameter( "value1" );
String value2 = request.getParameter( "value2" );

if (value1 != null && value1.trim().length() > 0 &&
value2 != null && value2.trim().length() > 0) {

out.println("<h4>");

try {
int v1 = Integer.parseInt(value1);
int v2 = Integer.parseInt(value2);

InitialContext ctx = new InitialContext();
Service service = (Service)ctx.lookup("java:comp/env/services/Calculator");
Calculator calc = service.getPort(Calculator.class);
int sum = calc.add(v1, v2);
out.println("Result: " + v1 + " + " + v2 + " = " + sum);
} catch ( Exception e ) {
out.println("Error: " + e.getMessage());
}

out.println("</h4>");
}
%>

</FONT>
</body>
</html>
```

add.jsp は JNDI から Web サービスへの参照を探します。Web サービスへの参照は web.xml ファイルに追加されます。

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">
.....

<service-ref>
<service-ref-name>services/Calculator</service-ref-name>
<service-interface>javax.xml.ws.Service</service-interface>
<wsdl-file>WEB-INF/wsdl/CalculatorService.wsdl</wsdl-file>
</service-ref>

.....
</web-app>
```

Resource injection in JSP

JSP ではリソースの注入をサポートしていないので、service-ref は明確に web.xml へ追加しなければいけません。

サンプル・アプリケーションのビルドとデプロイ

SVN からサンプルのソースコードをチェックアウトすることができます。

svn checkout http://svn.apache.org/repos/asf/geronimo/samples/trunk/samples/jaxws-calculator calculator

ソースコードは calculator/ ディレクトリーにチェックアウトされます。このサンプルコードは calculator/jaxws-calculator-war/ の中に入っています。

必要なツール

Calculator サンプル・アプリケーションのデプロイとビルドに必要なツールは以下のとおりです。

Apache Maven 2.0.x

Apache Maven は Calculator アプリケーションのビルドに利用されます。

ビルド

ソースコードのコンパイル

コマンド・プロンプトで、calculator/jaxws-calculator-war/ フォルダーに移動し以下のコマンドを実行してください。

mvn install

コンパイルが成功すると、jaxws-calculator-war-2.0-SNAPSHOT.war ファイルが target/ サブ・フォルダーに作られます。

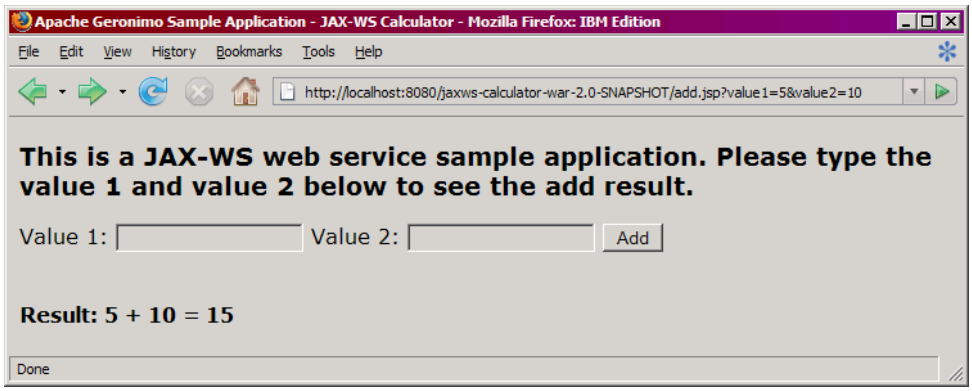
デプロイ

axws-calculator-war-2.0-SNAPSHOT.war を Geronimo コンソール (http://localhost:8080/console) を利用してデプロイします。

1. Console Navigation パネルから Deploy New を選択してください。
2. Archive 入力欄に calculator/jaxws-calculator-war/target/ フォルダーの jaxws-calculator-war-2.0-SNAPSHOT.war を読み込んでください。
3. Install ボタンを押してアプリケーションをサーバへデプロイしてください。

サンプルのテスト

テストすると、このサンプルのサービスは add.jsp (http://localhost:8080/jaxws-calculator/add.jsp) を使って Web サービスを呼び出します。JSP が読み込まれたら、2つの値を入力し、Add ボタンを押してください。足し算の結果が下に表示されます。例えば、以下のとおりです。



9.d. SPECjAppServer2004

This page last changed on 4 27, 2008 by JAGUG.

この文章は、SPECjAppServer2004 v1.05 と Geronimo v1.2 からコピーした バージョン です。まだアップデートが完了していなければ、まずは SPECjAppServer2004 v1.08 と Geronimo v2.0.2 にアップデートしてください。

Geromino での SPECjAppServer2004 ベンチマークの実行

まだ成功していません、貴方の助けが必要です!

この文章は標準規格である SPECjAppServer2004 のベンチマークを利用して Geronimo アプリケーション・サーバーのパフォーマンスを計測する方法を示します。

まだ問題があり、Geronimo で SPECjAppServer2004を実行できていません。
この文章は成功しているものではありませんが、どのようにしていけばよいかの資料集となるでしょう。
貴方に助けてもらいたい現在の問題は次のリンクにあります。Running the benchmark

免責事項: この文章は体験をもとにして書かれていて、再現性があります。いずれの製品についてもこの文章は広範なガイドをするものではありません。製品のドキュメントを置き換えることはなく、ただの順を追った手順のガイドであり、私の環境において単純な構成で動作させるためにものです。本文章の内容を実施する前に、少なくとも両方(訳注: SPECjAppServer2004 と Geronimo のこと)のドキュメントに目を通しておいてください。

SPECjAppServer は Standard Performance Evaluation Corp. (SPEC) の登録商標です。SPECjAppServer2004 の公式ウェブ・サイトは http://www.spec.org/jAppServer2004/です。
SPECjAppServer2004 version 1.08 は、オープン・ソースの研究開発プロジェクトと共有して見やすい結果表示にするレポート・ルールが利用できる EAStress2004 という縮小版ワークロードを利用しています。
SPECjAppServer2004 v1.08 の EAStress2004 ワークロードは、パフォーマンスの最適化のために開発中であり、公開フォーラムで結果を共有しています。SPECjAppServer2004 の結果と違って、EAStress2004 ワークロードのテスト結果は SPEC のように公開に先立ってレビューされる必要がありません。
EAStress2004 ワークロードの結果は、他の SPECjAppServer2004 での結果が許されていないのと同じように、商用目的で利用できません。
SPECjAppServer2004/EAStress2004 v1.08 に関するプレスリリース全文はこちらをご覧ください。http://www.spec.org/jAppServer2004/jAppServer2004v108.html.
この文章のすべてのログ、スタック、結果のファイルは EAStress2004 v1.08 実行結果から抽出したものです。

この文章は SPECjAppServer2004 v1.08 と Geronimo v2.0.2向けにかかれています。違うバージョンの場合は、いくつかの場面で記述が違っているかもしれません。この文章の古いバージョンは SPECjAppServer2004 v1.05 と以前の Geronimo のバージョンについて、次のリンク先にかかれています。v1.0, v1.1, v1.2.

記述されている構成はできるだけ Geronimo コンポーネントを利用しています。これには組み込みの Derby データベースや、Jetty や Tomcat サーブレット・コンテナーも含みます。実際、今回の構成は Java, Geronimo, 外部サーブレットコンテナー(例えば Tomcat) と SPECjAppServer2004のみを利用しています。 外部のコンポーネント(多くの例ではデーtqaベース)を組み込む場合は、それ相応に構成を変更しなければなりません。

また、この構成はすべてのコンポーネント(おそらく SPECjAppServer2004 ドライバーと SPECjAppServer2004サプライヤー・エミュレーターおよびそのサーブレット・コンテナーを除く)は同じ機械で実行されていることを想定しています。 ディストリビュート・ワークロードを利用したい場合は、違う構成になります。

この文章を書くための構成は Microsoft Windows XP Professional Service Pack 2 オペレーティング・システムと Cygwin シェル、Sun Java SE 5.0 Update 11 および Tomcat v5.0.30 としました。もし他の OS、Java またはサーブレット・コンテナーを使う場合は、いくつかの場面で違いがあるでしょう。

この文章ではコマンドラインでスラッシュ(/) を使っています。Windows コマンド・プロンプトを利用しているときは、バック・スラッシュ (¥) に適宜置き換えてください。

この文章は以下のような構成になっています。

一般的な情報

Geronimo について

Geronimo は Apache Software Foundation Java EE 5 認証済みアプリケーション・サーバーです。Apache Licenseのもと開発され、自由にダウンロードできます。

Apache サイト: http://apache.org

製品サイト: http://geronimo.apache.org

最新バージョン 2.0.2: http://geronimo.apache.org/apache-geronimo-v202-release.html

リリース・ノート: http://cwiki.apache.org/GMOxDOC20/release-notes-202txt.html

ドキュメントページ: http://geronimo.apache.org/documentation.html

参考ドキュメント "Apache Geronimo: J2EE Development and Deployment" 著者 Aaron Mulder: http://chariotsolutions.com/geronimo/

その他のリソース　FAQ: http://cwiki.apache.org/GMOxKB Wiki: http://cwiki.apache.org/geronimo

SPECjAppServer2004/EAStress2004 について

SPECjAppServer2004 Java EE アプリケーション・サーバーの性能計測のための商用ベンチマークです。

EAStress2004 は SPECjAppServer2004 v1.08の一部である縮小版ワークロードです。オープン・ソースの研究開発プロジェクトと共有して見やすい結果表示にするレポート・ルールが利用できる EAStress2004 という縮小版ワークロードを利用しています。

SPEC サイト: http://www.spec.org

製品サイト: http://www.spec.org/jAppServer2004/

Press release on v1.08 and EAStress2004: http://www.spec.org/jAppServer2004/jAppServer2004v108.html

v1.08 と EAStress2004 のプレスリリース: http://www.spec.org/jAppServer2004/jAppServer2004v108.html

FAQ: http://www.spec.org/jAppServer2004/docs/FAQ.html

ユーザー・ガイド: http://www.spec.org/jAppServer2004/docs/UserGuide.html

実行とレポートのルール: http://www.spec.org/jAppServer2004/docs/RunRules.html

Back to Top

製品の入手

Geronimo の入手

最新の Geronimo のバージョンは現在 2.0.2です。

ダウンロード・ページ: http://geronimo.apache.org/downloads.html

Geronimo には、Jetty か Tomcat のどちらかのサーブレット・コンテナーがデフォルトで利用できる2つのビルドがあります。いずれも http://geronimo.apache.org/downloads.htmlからダウンロードできます。おおよそ 55 MB のサイズです。この文章では Jetty バージョンでかかれていますが、Tomcat バージョンでも問題ありません。

SPECjAppServer2004 の入手

SPECjAppServer2004 は $2000 (非営利・教育目的なら $250) です。オンラインで注文できます。See FAQ for details.

最新バージョンは 1.08です。`SPECjAppServer2004-Kit-v1.08.jar` というファイルで、サイズは 12 MBです。

Back to Top

環境設定

このセクションには、ホストとディレクトリーについて重要な記述があります。

ホスト

この文章は以下のマシンについてかかれています。

- **`geronimo.host`** - Geronimo を実行し、SPECjAppServer2004 がデプロイされているマシン
- **`emulator.host`** - SPECjAppServer2004 サプライヤー・エミュレーターがデプロイされているマシン
- **`driver.host`** - SPECjAppServer2004ドライバーを実行しているマシン。複数のドライバー構成を利用している時は、すべてのホストで同じ操作を繰り返す必要があります
- **`master.host`** - 複数のドライバー構成の際の、メインとなる **`driver.host`**

`emulator.host` と **`driver.host`** は同じマシンでもよいです。

`geronimo.host` と **`emulator.host`** は同じマシンでもよいです。さらに SPECjAppServer2004 サプライヤー・エミュレーターは Geronimo 組み込みのサーブレット・コンテナー(Jetty または Tomcat) にデプロイしてもよいです。

geronimo.host と **driver.host** は同じマシンでもよいですが、その際は Geronimo と SPECjAppServer2004 がそれぞれ作成する RMI レジストリーのデフォルトの通信ポート (1099) が衝突するので、Geronimoの構成を 調整する必要があります。

> **geronimo.host** と、**emulator.host** か **driver.host** を共有することは SPECjAppServer2004ドキュメントに矛盾しますし、動作に深刻な影響をあたえ、ベンチマークの結果が不正となります。ただし、技術的には可能です。

ディレクトリー

このセクションではこの文章全体にわたって重要なディレクトリーを列挙します。これらのディレクトリーは任意に選択できますが、ダブってはいけません。

- **<GERONIMO>** - **geronimo.host** ### Geronimo ################
- **<SPEC>** - **geronimo.host** にあるSPECjAppServer2004 がインストールされたディレクトリー
- **<KIT>** - **geronimo.host** にあるこの文章の添付ファイルを含んでいるディレクトリー
- **<TOMCAT>** - **emulator.host** にある Tomcat がインストールされたディレクトリー
- **<DRIVER>** - **driver.host** にある **<SPEC>** ディレクトリーをコピーしたディレクトリー
- **<DRIVER_GERONIMO>** - **driver.host** にある **<GERONIMO>** ディレクトリーをコピーしたディレクトリー
- **<JAVA_HOME>** - **driver.host** にある JAVA_HOME の位置
- **<OUTPUT>** - **driver.host** にある SPECjAppServer2004 ドライバーの出力を保存したいディレクトリー
- **<DUMP>** - **driver.host** にある SPECjAppServer2004 ドライバーの一時ファイルを保存したいディレクトリー

> Windows ではディレクトリーのパス名が長すぎたりスペースを含んでいる場合、正常に動作しないコンポーネントがあるかもしれません。長いパス名をやめたり、パス名からスペースを取り除くことをお勧めします。

Back to Top

製品の導入

最初に、この文章の添付ファイルをローカル・ディレクトリーに保存してください。これが **<KIT>** ディレクトリーになります。

Geronimo の導入

Geronimo は `.zip` または `.tar.gz` アーカイブを利用すると簡単に導入できます。

ダウンロードしたアーカイブをローカル・ディレクトリーに展開してください。`geronimo-jetty6-jee5-2.0.2` or `geronimo-tomcat6-jee5-2.0.2` ディレクトリーが作られます。これが **<GERONIMO>** ディレクトリーになります。

SPECjAppServer2004 の導入

下記を実行してください。

```
java -jar SPECjAppServer2004-Kit-v1.08.jar
```

Next をクリックし、ライセンス契約を読み、同意し、SPECjAppServer2004 をインストールしたいディレクトリーを入力してください。これが **<SPEC>** ディレクトリーになります。

Install をクリックしてください。

導入処理が完了するまで待ったのち、Ready をクリックしてください。

Back to Top

Geronimo の構成

構成の調整

もし **geronimo.host** と **driver.host** が同じマシンでしたら、Geronimo RMI レジストリーの通信ポート番号を調整しなければなりません (例えば 1199 へ)。そうしないと、SPECjAppServer2004 ドライバーがデフォルトで利用する 1099 番ポートと衝突してしまいます。**<GERONIMO>**/var/config/config-substitutions.properties ファイルを編集し、NamingPort 変数値を変更してください。

Geronimo の始動

<GERONIMO> ディレクトリーへ移動してください。

以下のように入力して、Geronimo を始動してください。

```
java -Djava.endorsed.dirs=lib/endorsed -javaagent:bin/jpa.jar -
Dopenejb.jndiname.failoncollision=true -Dopenejb.jndiname.format={ejbName} -jar bin/
server.jar
```


2つの OpenEJB 設定に注目してください。これらは Geronimo に対して SPECjAppServer2004 という名前の EJB JNDI を利用するように設定しています。
詳しくは http://cwiki.apache.org/GMOxDEV/client-jndi-names.html と http://cwiki.apache.org/OPENEJB/service-locator.htmlをご覧ください。

########## **<GERONIMO>**/bin/geronimo.sh または **<GERONIMO>**/bin/geronimo.bat というスクリプトを利用することもできます。

Geronimo v2.0.2 は SPECjAppServer2004 アプリケーションがデプロイされる際にインターネットへ接続する必要があります。それは XML スキーマを http://java.sun.com サイトから取得するためです。
ですので、ファイアー・ウォールの内側にある場合、適切な -Dhttp.proxyHost= と -Dhttp.proxyPort= オプションを Geronimo 始動コマンドに含めてください。
この問題は OPENEJB-700 のバグにより引き起こされます。バグが修正された後は不要になります。

さて、実行すると以下のような表示になります。

```
Booting Geronimo Kernel (in Java 1.5.0_11)...
Starting Geronimo Application Server v2.0.2
[*******************************] 100%  92s Startup complete
Listening on Ports:
1050 127.0.0.1 CORBA Naming Service
1099 0.0.0.0   RMI Naming
1527 0.0.0.0   Derby Connector
2001 127.0.0.1 OpenEJB ORB Adapter
4201 0.0.0.0   OpenEJB Daemon
6882 127.0.0.1 OpenEJB ORB Adapter
8009 0.0.0.0   Jetty Connector AJP13
8080 0.0.0.0   Jetty SelectChannel Connector HTTP
8443 0.0.0.0   Jetty SelectChannel Connector HTTPS
9999 0.0.0.0   JMX Remoting Connector
61613 0.0.0.0   ActiveMQ Transport Connector
61616 0.0.0.0   ActiveMQ Transport Connector

Started Application Modules:
EAR: org.apache.geronimo.configs/webconsole-jetty6/2.0.2/car
```

```
JAR: org.apache.geronimo.configs/mejb/2.0.2/car
RAR: org.apache.geronimo.configs/activemq-ra/2.0.2/car
RAR: org.apache.geronimo.configs/system-database/2.0.2/car
WAR: org.apache.geronimo.configs/dojo-jetty6/2.0.2/car
WAR: org.apache.geronimo.configs/remote-deploy-jetty/2.0.2/car
WAR: org.apache.geronimo.configs/welcome-jetty/2.0.2/car

Web Applications:
/
/console
/console-standard
/dojo
/remote-deploy

Geronimo Application Server started
```

もし違う結果が表示されていたら、特にネットワークのエラーが見えているようでしたら、どこかの構成がうまくできていないと思われます。

ローカル・ネットワーク・アドレスへの接続ができないために始動に失敗することがあります。この場合、例えばVPN インターフェースを利用していて、現在利用できないのかもしれません。デフォルトでは Geronimoは最初のローカル・アドレスを利用して各コンポーネントへ接続します。このアドレスが失効していたりすると始動時のエラーになります。
このような問題解決のために、利用されていないネットワーク・インターフェースを一度無効にし、再度有効化してみてください。

コンソールへの接続

ウェブ・ブラウザーを開き、**`http://geronimo.host:8080/console/`** にある Geronimo コンソールへ接続してください。

ユーザー名とパスワード (system と manager がデフォルトです) を利用してログインしてください。

これでコンソールを自由に操作することができます。

データベースの作成

Console Navigation にある Embedded DB - DB Manager を開いてください。

ベンチマーク・データベースを作成しますので、Create DB 欄にデータベース名として (SPECDB) を入力し、Create をクリックしてください。

SQL ファイルの読み込み

データベース・テーブルを作成するために、**<SPEC>**`/schema/sql` ディレクトリーにあるデフォルトの SQL スクリプトを利用しましょう。しかし、このディレクトリーには5つのスクリプトが入って、DROP TABLE コマンドも含まれます。これを実行すると、テーブルが未作成の場合の実行時にはエラーが発生します。

ですから、最初にテーブルを作成するときは allTablesNoDrop.sql ファイルを利用し、作られているテーブルを削除し再作成したいときは allTables.sql ファイルを利用することをお勧めします。両方のファイルとも **<SPEC>**`/schema/sql` にあるファイルで、単に同じ内容がかかれていますが、allTablesNoDrop.sql では `DROP TABLE` コマンドが省かれています。

テーブルの作成

Use DB 欄で SPECDB が選択されていることを確認し、SQL Command/s フレームに SQL をコピー・ペーストしてください。その上にある Run SQL ボタンをクリックしてください。

少し経つと、フレームがクリアーされ、その下の Result 欄に SQL command/s successful と表示されます。もしこのように表示されなかったら何か間違ってしまっているので、再度実施してください。

複数の SQL スクリプトを利用する時は、上記処理を必要回数繰りかえしてください。

Back to Top

SPECjAppServer2004 の構成

基本構成

deploy directory ディレクトリーのデプロイ

<SPEC>/src/deploy ディレクトリーへ移動し、reference サブ・ディレクトリーを中身ごと geronimo という名前でコピーしてください。

> ⚠ <SPEC>/src/geronimo にあるデプロイメント・プランを編集します。mfg.xml、orders.xml、supplier.xml ファイルからすべての message-driven-destination タグを取り除いてください。これらのタグを Geronimo v2.0.2 は適切に扱えません。
> これらのタグを取り除くかわりに、これらのタグを <message-destination-type>javax.jms.Queue</message-destination-type> と置き換えることも可能です。
> この問題は OPENEJB-701のバグにより引き起こされます。バグが修正された後は不要になります。

geronimo.env ファイル

<SPEC>/config ディレクトリーへ移動してください。

そこに添付されている geronimo.env のテンプレート・ファイルを配置してください。これを編集し、以下の値を正しく設定してください。

```
JAS_HOME=<SPEC>
JAVA_HOME=<JAVA_HOME>
J2EE_HOME=<GERONIMO>
JAS_HOST=geronimo.host
EMULATOR_HOST=emulator.host
```

> ⚠ デレクトリーのセパレーターとしてスラッシュ (/) を利用します。

他の影響のない変数を消すこともできます。

appsserver ファイル

<SPEC>/config/appsserver ファイルを編集し、そこにある default という文字を geronimo に置き換えてください。

run.properties ファイル

<SPEC>/config/run.properties ファイルを編集し、**driver.host** 上で動作するように以下の変数が適切な値になるようにしてください。

```
Url = http://geronimo.host:8080/SPECjAppServer/app?
outDir = <OUTPUT>
dumpDir = <DUMP>
```

setenv.bat ファイル

<SPEC>/bin/setenv.bat ファイルを編集し、以下の変数値の設定を確認してください。

```
JAVA_HOME=<JAVA_HOME>
JAS_HOME=<DRIVER>
APPSSERVER=geronimo
```

アプリケーションのビルド

<SPEC> に移動してください。

以下のコマンドでインストール状態をクリーン・アップします。

```
ant/bin/ant clean
```

以下のコマンドでアプリケーションをビルドし、Geronimo向けに構成します。

```
ant/bin/ant -Dappserver=geronimo
```

BUILD SUCCESSFUL という結果になるでしょう。

<SPEC>`/jars` ディレクトリーに `SPECjAppServer.ear` と `emulator.war` というファイルが作られていることを確認してください。

`emulator.war` を `Emulator.war` へリネームしてください。

データベース構成の準備

この構成では、すべてのテーブルが同じデータベースを利用するようになっています。

<SPEC>`/config` ディレクトリーに移動してください。添付されている db.properties テンプレート・ファイルの内容を参考にして **`db.properties`** の内容を置き換えてください。`pipeDir` 変数の値がテンポラリー・ディレクトリーを指し示しているか確認し、必要な調整をしてください。

テーブルの読み込み

下記を実行してください。

```
ant/bin/ant -Dappserver=geronimo loaddb
```

しばらくすると、BUILD SUCCESSFUL と表示されるでしょう。

Back to Top

コンポーネントのデプロイ

この段階で、Geronimoに構成済みコンポーネントをデプロイする必要があります。

> geronimo.host と driver.host が同じマシンの場合、Geronimo RMI レジストリーの通信ポート番号を 変更 していますので、deployer へのコマンドには次のような通信ポート番号指定をすることがあることを注意してください。
>
> ```
> java -jar bin/deployer.jar -port 1199 ...
> ```

ログイン

deployer を呼び出す際に毎回ログイン認証を行うことをやめるには、最初に以下のようにログインします。

```
java -jar bin/deployer.jar -u system -p manager login
```

データベース・コネクターのデプロイ

先に作成した Derby の SPECDB データベース用コネクタをデプロイするには、**<GERONIMO>** ディレクトリーに移動し、以下を実行します。

```
java -jar bin/deployer.jar deploy repository/org/tranql/tranql-connector-derby-embed-xa/1.4/tranql-connector-derby-embed-xa-1.4.rar <KIT>/sjas-db.xml
```

SPECjAppServer2004/DB/1.08/rar という処理結果が表示されるでしょう。

JMS コネクターのデプロイ

SPECjAppServer2004用 ActiveMQJMS コネクターをデプロイするには、**<GERONIMO>** ディレクトリーに移動し、以下を実行します。

```
java -jar bin/deployer.jar deploy repository/org/apache/geronimo/modules/geronimo-activemq-ra/2.0.2/geronimo-activemq-ra-2.0.2.rar <KIT>/sjas-jms.xml
```

SPECjAppServer2004/JMS/1.08/rar という処理結果が表示されるでしょう。

メイン・アプリケーションのデプロイ

Geronimoに SPECjAppServer2004 をデプロイするために、今回の構成は Geronimo バージョン 2.0.2 用に更新された http://svn.apache.orgの Geronimo ソースの中にあるデプロイメント・プランを利用しています。

<GERONIMO> ディレクトリーに移動し、以下を実行します。

```
java -jar bin/deployer.jar deploy <SPEC>/jars/SPECjAppServer.ear <KIT>/sjas-app.xml
```

SPECjAppServer2004/Application/1.08/ear という処理結果が表示されるでしょう。

デプロイの確認

この段階で、デプロイが成功し、SPECjAppServer2004 が操作可能であることを確認します。

手動トランザクション

デプロイされた SPECjAppServer2004 の次のページを開きます。**http://geronimo.host:8080/SPECjAppServer/**

左側のメニューにある Go Trade Autos! または Go Build Cars! リンクをクリックします。

デフォルトの認証である (**1**) を利用して、Log in をクリックしてログインします。

プログラムのインターフェースが見えます。トランザクションが動作可能になります。

不可分性テスト

デプロイされた SPECjAppServer2004 の次のページを開きます。**http://geronimo.host:8080/SPECjAppServer/**

左側メニューにある Atomicity Tests リンクをクリックします。

3つの atomicity tests が実行され、結果が表示されます。すべての結果に PASSED と表示されたときはデプロイが正常ということです。

Back to Top

サプライヤー・エミュレーターのデプロイ

emulator.host に SPECjAppServer2004 サプライヤー・エミュレーターをデプロイするには、以下のいずれかひとつを利用してください。

Geronimo 組み込みのサーブレット・コンテナー (今回は **emulator.host** と **geronimo.host** は同じマシンです)、または **emulator.host** にあるスタンド・アロンのサーブレット・コンテナー

> SPECjAppServer2004 ドキュメント によると、サプライヤー・エミュレーターのサーブレット・コンテナーの keep-alive オプションがオフである必要があります。これを無視することもできますが、性能に重大な影響を与えてしまいます。

Geronimo サーブレット・コンテナーの利用

<GERONIMO> ディレクトリーに移動し、以下を実行してください。

```
java -jar bin/deployer.jar deploy <SPEC>/jars/Emulator.war <KIT>/sjas-emulator.xml
```

SPECjAppServer2004/Emulator/1.08/war @ /Emulator という処理結果が表示されるでしょう。

スタンド・アロン・サーブレット・コンテナーの利用

この構成では、**emulator.host** にあるスタンド・アロンのサーブレット・コンテナーが Tomcat がデフォルトの通信ポート (8080) で実行されているとします。

emulator.host の **<TOMCAT>** ディレクトリーに Tomcat をインストールしてください。

> わざわざ <SPEC>/config/tomcat.env ファイルを編集したり、ant/bin/ant -f tomcat.xml コマンドを実行したりしないでください。
> Building the applicationの時点で既に作成されている Emulator.war ファイルが生成するので、両ファイルとも不要です。

<SPEC>/jars/Emulator.war ファイルを **<TOMCAT>**/webapps ディレクトリーへコピーし、**<TOMCAT>**/webapps/Emulator ディレクトリーが存在していたら削除してください。

emulator.host 上の **<TOMCAT>** ディレクトリーへ移動し、以下のとおり Tomcatを始動してください。

```
bin/catalina run
```

デプロイの確認

http://emulator.host:8080/Emulator/ ページへ移動してください。通常どおり読み込まれると、中に2つのファイル delivery.dtd と po.dtd を含んでいるひとつのディレクトリー dtd が表示されます。

http://emulator.host:8080/Emulator/EmulatorServlet ページへ移動してください。以下のようなテキストのページが表示されます。

Emulator Servlet seems to work OK
JAS_HOST : emulator.host
JAS_PORT : 8080
Servlet URL : Supplier/DeliveryServlet

Number of Transactions : 0
Servlet invoked without command specified

Back to Top

ベンチマークの実行

driver.host へ **<GERONIMO>** ディレクトリーをコピーし、これを **<DRIVER_GERONIMO>** ディレクトリーとします (実際はいくつかの jar ファイルだけが必要です) 。

driver.host へ **<SPEC>** ディレクトリーをコピーし、これを **<DRIVER>** ディレクトリーとします。

<DRIVER>/config/geronimo.env ファイルの JAS_HOME 変数を **<DRIVER>** ディレクトリーに変更し、J2EE_HOME 変数を **<DRIVER_GERONIMO>** ディレクトリーに変更してください。

driver.host の **<DRIVER>** ディレクトリーに移動し、以下を実行してください。

```
bin/setenv.bat
```

これにより Driver を実行する環境が構成されます。

ドライバーを起動するときは、以下を実行します。

```
bin/driver.bat
```

もし複数の Driver を分散して読み込みたいなら、最初のホスト (**master.host** のこと)の始動後、他の Driver ホストで以下のように Driver を始動してください。

```
bin/driver.bat master.host
```

Driver が始動すると、以下のような出力が表示されるでしょう。

```
The following environment settings are in effect for SPECjAppServer2004
* ======================== *
JAVA_HOME=<JAVA_HOME>
JAS_HOME=<DRIVER>
CONFIG_DIR=<DRIVER>\config
APPSSERVER=geronimo
ENVFILE=<DRIVER>\config\geronimo.env
* ======================== *
Driver Host: <driver.host>
Binding controller to //<driver.host>/Controller
DriverDebug: DealerAgent <propsFile> <agentName> <masterMachine>
Controller: Registering M1 on machine <driver.host IP address>
Controller: Registering O1 on machine <driver.host IP address>
Controller: Registering L1 on machine <driver.host IP address>
Calling switchLog as master
RunID for this run is : 75
Output directory for this run is : <OUTPUT>\75
TTF1 = 93
```

```
ttf = 93
Configuring 1 DealerAgent(s)...
DealerAgent O1, Thread 0 started
DealerAgent O1, Thread 1 started
DealerAgent O1, Thread 2 started
DealerAgent O1, Thread 3 started
DealerAgent O1, Thread 4 started
DealerAgent O1, Thread 5 started
DealerAgent O1, Thread 6 started
DealerAgent O1, Thread 7 started
DealerAgent O1, Thread 8 started
DealerAgent O1, Thread 9 started
Configuring 1 MfgAgent(s)...
MfgAgent M1, Thread 0 started
MfgAgent M1, Thread 1 started
MfgAgent M1, Thread 2 started
Configuring 1 LargeOLAgent(s)...


MfgAgent L1, Thread 0 started
Rampup      = Fri May 12 20:49:51 MSD 2006
SteadyState = Fri May 12 20:59:51 MSD 2006
Rampdown    = Fri May 12 21:59:51 MSD 2006
Finish      = Fri May 12 22:04:51 MSD 2006

sleeptime is 28417 note this is time in excess needed for trigger
Starting Ramp Up...
```

この意味は、Driver が通常どおり始動した、ということを表します。

Rampup、SteadyState、Rampdown、Finish の時間を確認してください。ベンチマークが完了するまでに要した時間が表示されています。

> `Ctrl-C` を押せば、いつでも実行を中断できます。

Driver 始動時にエクセプションが発生したり、他の問題が起こることがあります。この場合は、`Ctrl-C` を押してテストを中断し、再度実行してください。うまくいくことがあります。

> 毎回実行前にデータベース・テーブルを 再読み込みすることをお勧めします。直前の実行が正常に終了しなかった場合などでは特に勧めます。そうしないと、以下のようなエラーが発生するかもしれません。
>
> ```
> java.rmi.RemoteException: Failure in calling validateInitialValues()
> java.rmi.RemoteException: Invalid initial Order DB State
> at org.spec.jappserver.driver.Auditor.validateInitialValues(Auditor.java:201)
> at org.spec.jappserver.driver.Driver.configure(Driver.java:330)
> at org.spec.jappserver.driver.Driver.<init>(Driver.java:160)
> at org.spec.jappserver.driver.Driver.main(Driver.java:1137)
> ```

実行中、Driver のウィンドウに次のような処理結果が表示されることがあります。

```
AbstractSJASLoad> Application error has already been cancelled
```

また、同じタイミングで様々な `TransactionRolledback` や他の同じようなエクセプションが Geronimoのシェルで発生しています。

> これらの処理結果は Geronimo v2.0.2 に含まれる TranQL バージョン 1.3 が十分なトランザクションの分離レベルを与えていないから発生しているかもしれません。TranQL バージョン 1.3.1 でこの問題が修正されることを願います。

実行が無事完了すると、以下のような出力が表示されるでしょう。

```
Gathering DealerStats ...
Gathering MfgStats ...
summary file is <OUTPUT>\75\SPECjAppServer.summary
SPECjAppServer2004 v1.08 Results
```

```
JOPS: ***
Dealer Response Times
Purchase...0.4
Manage.....1.5
Browse.....0.4
Manufacturing Response Times
Mfg........0.0
Calling getLog as master
```

JOPS の数は、ベンチマークの最終測定値です。

> Geronimoでは現在、利用したハードウェアによらずこれらの値がかなり低くなります。おそらくなんらかの構成上の問題がまだ残っているのか、あるいは前述のTranQL の問題かもしれません。

Back to Top

実行結果

Driver が実行されると、**<OUTPUT>** ディレクトリーの中に数字で名前がつけられたサブ・ディレクトリーが作られます。一番大きい数字のサブ・ディレクトリーが最後に実行した結果です。Driver の実行が完了したら、`result.props` ファイルがディレクトリー内に作られていて、これがベンチマークの未加工である結果データを持っています。

<DRIVER>`/reporter` ディレクトリーへ移動し、`Sample_Submission.txt` を任意の名前 (たとえば `Your_Submission.txt`) としてコピーし、そのコピーを SPECjAppServer2004 User's Guide :: Section 5 - Resultsの記述にしたがって編集し、`result.props` という未加工の結果データをこのディレクトリーに追加します。

以下のコマンドを実行してください。HTML 形式のベンチマーク・レポートを `Your_Submission.report.html` という名前で生成します。

```
java -cp reporter.jar reporter Your_Submission.txt
```

以下のコマンドを実行してください。テキストのみのベンチマーク・レポートを `Your_Submission.report.txt` という生で生成します。

```
java -cp reporter.jar reporter -a Your_Submission.txt
```

実行結果を提出する詳しい方法は SPECjAppServer2004 User's Guide :: Section 5.3 - Submitting the Results をご覧ください。

Back to Top

9.e. Geronimo のデフォルト JavaMail セッションの利用

This page last changed on 4 27, 2008 by JAGUG.

> ⚠ Trunk（訳注: 開発中の状態のもの）でテストしました。Geronimo v1.2 から v2.0 への JavaMail のモジュール ID 参照定義のマイナーな変更をした上で v1.2 のサンプルと同じステップとなります。

Apache JAMES の導入

1. http://apache2.openmirrors.org/james/server/binaries/james-2.3.0.zip から Apache JAMES をダウンロードしてください。

2. ファイルを解凍し、¥james-2.3.0¥bin¥run.bat コマンドで JAMES を実行してください。以下のように表示されます。

```
Phoenix 4.2

James Mail Server 2.3.0
Remote Manager Service started plain:4555
POP3 Service started plain:110
SMTP Service started plain:25
NNTP Service started plain:119
FetchMail Disabled
```

3. 次の JavaMail モジュールが始動していることを確認してください。

- For trunk and v2.0-M1: org.apache.geronimo.configs/javamail/2.0-SNAPSHOT/car (by default this is started)
- For v1.2-beta: org.apache.geronimo.configs/javamail/1.2-beta/car (by default this is not started)
- To start the module you can:
- Trunk か v2.0-M1 の場合: org.apache.geronimo.configs/javamail/2.0-SNAPSHOT/car（デフォルトでは始動しています）
- v1.2-beta の場合: org.apache.geronimo.configs/javamail/1.2-beta/car（デフォルトでは始動していません）
- このモジュールを始動するには、次のいずれかの方法があります

a. コンソールを使って Application > System Modules > JavaMail モジュール ID の横にある 'Start' をクリックします
b. コマンドライン・デプロイヤーを利用して、以下のコマンドを入力します

```
cd <geronimo_home>\bin
java -jar deployer.jar --user system --password manager start org.apache.geronimo.configs/
javamail/2.0-SNAPSHOT/car
```

c. サーバーを停止し、<geronimo_home>¥var¥config¥config.xml を編集し、以下のとおり JavaMail モジュールの load 属性を 'true' にします

```
...
<module load="true" name="org.apache.geronimo.configs/javamail/2.0-SNAPSHOT/car">
<gbean name="SMTPTransport">
<attribute name="host">localhost</attribute>
<attribute name="port">25</attribute>
</gbean>
</module>
...
```

> ⚠ SMTP Transport GBean の値を上書きすることで、SMTP トランスポートのホストや通信ポートを構成することができます。このサンプルはデフォルトの設定を利用してテストしました。こうすると、localhost で 25 番の通信ポート（デフォルトの SMTP ポートです）を使って実行されている JAMES に接続することができます。

Web アプリケーションのテスト

以下のファイルを含むシンプルな webapp を作成してください。

```
sendmail.war
+ index.jsp
+ WEB-INF
+ web.xml
```

```
+ geronimo-web.xml
```

web.xml:

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd"
version="2.4">
<display-name>Send Mail Webapp</display-name>
<welcome-file-list>
<welcome-file>index.html</welcome-file>
</welcome-file-list>
<resource-ref>
<!-- Used in index.jsp -->
<res-ref-name>mail/testMailSession</res-ref-name>
<res-type>javax.mail.Session</res-type>
<res-auth>Container</res-auth>
<res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
</web-app>
```

geronimo-web.xml: このアプリケーションは Geronimo 2.0 の JavaMail コンポーネントを利用していますので、コンポーネントを見つける場所を指定する必要があります。これは <dependencies> タグの中にあります。Geronimo が変更されたので、これらのパスを変更して同じ物が参照できるようにする必要があります。<resource-ref> 要素の中に、JavaMail セッションのことがかかれています。これは開発者にサーバー内のリソースの参照を可能にします。<ref-name> 要素は web.xml から利用され、両者は密接に関連しています。<resource-link> エレメントはデフォルトの Geronimo メール・セッションの記述です。

geronimo-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1">
<environment>
<moduleId>
<groupId>${pom.groupId}</groupId>
<artifactId>${pom.artifactId}</artifactId>
<version>${version}</version>
<type>war</type>
</moduleId>
<dependencies>
<dependency>
<groupId>org.apache.geronimo.configs</groupId>
<artifactId>javamail</artifactId>
<type>car</type>
</dependency>
</dependencies>
</environment>
<context-root>/sendmail</context-root>
<resource-ref>
<!-- Used is web.xml -->
<ref-name>mail/testMailSession</ref-name>
<!-- Default Geronimo mail session -->
<resource-link>mail/MailSession</resource-link>
</resource-ref>
</web-app>
```

index.jsp:

index.jsp

```
<%@page import="java.util.Date,
javax.mail.Message,
javax.mail.Session,
javax.mail.Transport,
```

```
javax.mail.internet.InternetAddress,
javax.mail.internet.MimeMessage,
javax.naming.InitialContext" %>

<%
String resultMsg = "";
String action = request.getParameter("action");
if ("Send".equals(action)) {
String from = request.getParameter("from");
String to = request.getParameter("to");
String subject = request.getParameter("subject");
String content = request.getParameter("message");

// Get mail session and transport
InitialContext context = new InitialContext();
// Mail session from web.xml's resource reference
Session mailSession = (Session) context.lookup("java:comp/env/mail/testMailSession");
Transport transport = mailSession.getTransport("smtp");

// Setup message
MimeMessage message = new MimeMessage(mailSession);
// From address
message.setFrom(new InternetAddress(from));
// To address
message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
// Subject
message.setSubject(subject);
// Content
message.setText(content);

// Send message
transport.connect();
transport.send(message);

// Build result message
resultMsg = "<b>Result:</b>";
resultMsg += "<br>Message sent: " + new Date();
resultMsg += "<br>To: " + to;
resultMsg += "<br>From: " + from;
}
%>

<html>
<head>
<title>Send Mail</title>
</head>
<body>
<form>
<table>
<tr>
<td align="center" colspan="2"><b>Send Mail</b></td>
</tr>
<tr>
<td align="right">From:</td>
<td><input type="text" name="from"></td>
</tr>http://localhost:8080/sendmail
<tr>
<td align="right">To:</td>
<td><input type="text" name="to"></td>
</tr>
<tr>
<td align="right">Subject:</td>
<td><input type="text" name="subject"></td>
</tr>
<tr>
<td align="right">Message:</td>
```

```
<td><textarea rows="5" cols="20" name="message"></textarea></td>
</tr>
<tr>
<td align="right" colspan="2">
<input type="submit" name="action" value="Send"> <input type="reset"></td>
</td>
</tr>
</table>
</form>
<%= resultMsg %>
</body>
</html>
```

ウェブ・アプリケーションのビルドとデプロイ

次のリンクから sendmail アプリケーションをダウンロードしてください。
sendmail

ファイルを解凍すると、sendmail ディレクトリーが作られます。

ソース・コード

SVN からサンプルのソースコードをチェックアウトすることができます。

svn checkout http://svn.apache.org/repos/asf/geronimo/samples/trunk/samples/sendmail

ウェブ・アプリケーションのビルド

sendmail フォルダーにはデプロイできる ear ファイルを含んでいますが、ソースを利用してビルドすることもできます。

コマンドプロンプトを利用して sendmail ディレクトリーへ移動し、mvn install site コマンドを入力するとビルドされます。sendmail フォルダーの下に sendmail-ear-2.0-SNAPSHOT.ear が作られます。

ウェブ・アプリケーションのデプロイ

1. Console Navigation パネルから Deploy New を選択してください。
2. Archive 入力欄に sendmail フォルダーの sendmail-ear-2.0-SNAPSHOT.ear を読み込んでください。
3. Install ボタンを押してアプリケーションをサーバーへデプロイしてください。
4. send mail ウェブ・アプリケーションをテストする際は、次のリンクを開いてください。http://localhost:8080/sendmail

フォームの内容（From、To、Subject、Message フィールド）を埋めて 'Send' ボタンをクリックしてください。メールが無事送信されると、次のようなメッセージが表示されます。

```
Result:
Message sent: Fri Dec 15 01:10:05 PST 2006
To: manny@pacquiao.com
From: chris@cardona.com
```

9.f. Geronimo 2.0 での JNDI の利用

This page last changed on 4 27, 2008 by JAGUG.

紹介

Java Naming and Directory Interface (JNDI) は、Apache Geronimo アプリケーション・サーバーのコネクション・プールへのインターフェースです。このインターフェースを通して、開発者はエンタープライズ Java Beans (EJB) も含むすべての Java オブジェクトに接続できます。

データソース、Java Messaging Services (JMS)、メール・セッション、URL 接続用のコネクション・プールに接続するための概念的文章を紹介します。Apache Geronimo JNDI naming and Java resource connection pools, Part 1: Data source connections. URL: http://www-128.ibm.com/developerworks/opensource/library/os-ag-jndi1

ただし、上記文章は Geronimo 1.x 向けに書かれています。このサンプルのチュートリアルは同じような内容ですが、EJB 3.0 を利用しています。

アプリケーションの概略

CustomerService アプリケーションは EJB を使ってデータベースへ接続し、画面に結果を戻します。このサンプルのポイントは、JNDI を使って EJB へ接続する方法を示します。今回の EAR ファイルの中身の構造の概略は次のとおりです。

```
|-CustomerService-ear-2.0-SNAPSHOT.ear
|-CustomerService-ejb-2.0-SNAPSHOT.jar
|-META-INF
|-persistence.xml
|-openejb-jar.xml
|-CustomerService-war-2.0-SNAPSHOT.war
|-WEB-INF
|-web.xml
|-geronimo-web.xml
|-META-INF
|-application.xml
|-geronimo.application.xml
```

persistence.xml は、エンティティ Bean に利用するデータベース・プールを明示的に割り当てるために参照されます。今回は、Customer というエンティティ Bean が CustomerServicePool というデータベース・プールを利用します。

persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence    xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/
persistence_1_0.xsd">
<persistence-unit name="CustomerPU">
<description>Entity Beans for Customer</description>
<provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider>
<class>com.service.customer.ejb.Customer</class>
<properties>
<property name="openjpa.jdbc.SynchronizeMappings" value="false"/>
</properties>
<jta-data-source>CustomerServicePool</jta-data-source>
<non-jta-data-source>CustomerServicePool</non-jta-data-source>
</persistence-unit>
</persistence>
```

openejb-jar.xml は OpenEJB で利用しますが、次のとおりです。MDB (訳注: メッセージ駆動 Bean) を利用していないので、特別に定義することはありません。

openejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<openejb-jar xmlns="http://www.openejb.org/xml/ns/openejb-jar-2.1"
xmlns:naming="http://geronimo.apache.org/xml/ns/naming-1.1"
xmlns:security="http://geronimo.apache.org/xml/ns/security-1.1"
xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2">
<dep:environment>
<dep:moduleId>
<dep:groupId>${pom.groupId}</dep:groupId>
<dep:artifactId>${pom.artifactId}</dep:artifactId>
<dep:version>${version}</dep:version>
<dep:type>jar</dep:type>
</dep:moduleId>

<dep:dependencies>
</dep:dependencies>
<dep:hidden-classes/>
<dep:non-overridable-classes/>
</dep:environment>
</openejb-jar>
```

web.xml は ProcessCustomerSessionBean.java で作られた EJB を参照しています。こうすることで、WAR の内部からのこの EJB の利用を許可します。

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd"
version="2.4">

<welcome-file-list>
<welcome-file>index.html</welcome-file>
</welcome-file-list>

<ejb-local-ref>
<ejb-ref-name>ejb/ProcessCustomerSessionBean</ejb-ref-name>
<ejb-ref-type>Session</ejb-ref-type>
<local>com.service.customer.ejb.ProcessCustomerSessionLocal</local>
</ejb-local-ref>
</web-app>
```

geronimo-application.xml へは、モジュール情報とウェブ・アプリケーションのコンテキスト・ルートを記述します。さらに、データベース・プールのプランを記述します。Geronimo にこのプランをデプロイするために必要なコネクタにしたがってデプロイするようにプランを書きます。

geronimo-application.xml

```
<application xmlns="http://geronimo.apache.org/xml/ns/j2ee/application-1.1">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.1">
<dep:moduleId>
<dep:groupId>${pom.groupId}</dep:groupId>
<dep:artifactId>${pom.artifactId}</dep:artifactId>
<dep:version>${version}</dep:version>
<dep:type>ear</dep:type>
</dep:moduleId>

<dep:dependencies/>
<dep:hidden-classes/>
<dep:non-overridable-classes/>
</dep:environment>

<module>
<connector>tranql-connector-ra-1.3.rar</connector>
<alt-dd>CustomerServicePool.xml</alt-dd>
</module>
```

```
</application>
```

application.xml へは、EAR が組み込みデータベース・プール (CustomerServicePool.xml) をデプロイする際に利用されるコネクタを記述します。

application.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/
application_5.xsd" version="5">
<description>Geronimo Sample EAR for CustomerService</description>
<display-name>Geronimo Sample EAR for CustomerService</display-name>
<module>
<ejb>CustomerService-ejb-2.0-SNAPSHOT.jar</ejb>
</module>
<module>
<web>
<web-uri>CustomerService-war-2.0-SNAPSHOT.war</web-uri>
<context-root>/service</context-root>
</web>
</module>
<module>
<connector>tranql-connector-ra-1.3.rar</connector>
</module>
</application>
```

CustomerServiceJavaBean.java は JNDI をつかって ProcessCustomerSessionBean という EJB を探します。

CustomerServiceJavaBean.java

```
package com.service.customer.web;

import com.service.customer.ejb.Customer;
import com.service.customer.ejb.ProcessCustomerSessionLocal;

import java.util.Locale;
import java.util.ResourceBundle;
import java.util.List;
import javax.naming.InitialContext;

public class CustomerServiceJavaBean
{
private ProcessCustomerSessionLocal process = null;
private ResourceBundle      bundle = null;

public CustomerServiceJavaBean()
{
InitialContext initial = null;

bundle = ResourceBundle.getBundle("customer", Locale.getDefault(),
CustomerServiceJavaBean.class.getClassLoader());
String jndiName = bundle.getString("jndi.process.ejb");

try
{
initial = new InitialContext();
process = (ProcessCustomerSessionLocal) initial.lookup(jndiName.trim());
System.out.println("Successful looking up: '" + jndiName.trim() + "'");
} // end try

catch (Exception e)
{
e.printStackTrace();
} // end catch
```

```
} // end CustomerServiceJavaBean


public List<Customer> getAllCustomers()
{
List<Customer> customerList = null;

try
{
customerList = process.findAllCustomers();
} // end try
catch (Exception e)
{
e.printStackTrace();
} // end catch

return customerList;
} // end getAllCustomerss
} // end CustomerServiceJavaBean
```

ProcessCustomerSessionBean.java は persistence.xml を利用して EntityManagerFactory が保持している ProcessCustomerSessionLocal を実装しています。保持には @PersistenceUnit アノテーションを利用しています。persistence.xml には定義するたった一つの永続化ユニットしかないので、特にこのアノテーションに加えるパラメーターはありません。

ProcessCustomerSessionBean.java

```
package com.service.customer.ejb;

import java.rmi.RemoteException;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;

import javax.persistence.PersistenceUnit;
import javax.ejb.EJBException;
import javax.ejb.Stateless;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;

@Stateless
public class ProcessCustomerSessionBean implements ProcessCustomerSessionLocal {
@PersistenceUnit
protected EntityManagerFactory emf;

public ProcessCustomerSessionBean() {

}

public List<Customer> findAllCustomers() {
EntityManager em = emf.createEntityManager();
String query = "SELECT * FROM customer";
List<Customer> customerList =
(List<Customer>)em.createNativeQuery(query, Customer.class).getResultList();
em.close();
return customerList;
}

public Customer findCustomer(String key) {
EntityManager em = emf.createEntityManager();
String query = "SELECT * FROM customer WHERE id='"+key+"'";
List<Customer> customerList =
(List<Customer>)em.createNativeQuery(query, Customer.class).getResultList();
```

```
if(customerList.size() == 1) {
return (Customer)customerList.get(0);
} else {
return null;
}
}
}
```

ProcessCustomerSessionLocal.java へは、ビジネス・メソッドとこの Bean との関連づけを記述します。

ProcessCustomerSessionLocal.java

```
package com.service.customer.ejb;

import com.service.customer.ejb.Customer;

public interface ProcessCustomerSessionLocal {
public java.util.List<Customer> findAllCustomers();

public Customer findCustomer(String key);
}
```

Customer.java はデータベースにある Customer テーブルを表すエンティティ Bean です。@Entity、@Table(name = "customer")、@Id を利用することで、OpenEJB にこのクラスがエンティティ Bean であり、"customer" テーブルを表し、"customerId" というプライマリ・キーを持っていることを通知します。これらのアノテーションを利用することによって、openejb-jar.xml (ejb-jar.xml もそうですが) に他の構成が必要ありません。

Customer.java

```
package com.service.customer.ejb;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "customer")
public class Customer implements Serializable {
private String customerId;
private String fullName;
private String emailAddress;
private String interests;

public Customer() {

}

public Customer(String customerId, String fullName, String emailAddress,
String interests) {
this.customerId = customerId;
this.fullName = fullName;
this.emailAddress = emailAddress;
this.interests = interests;
}

@Id
public String getCustomerId() {
return customerId;
}

public String getFullName() {
return fullName;
}

public String getEmailAddress() {
return emailAddress;
```

```
}

public String getInterests() {
return interests;
}

public void setCustomerId(String customerId) {
this.customerId = customerId;
}

public void setFullName(String fullName) {
this.fullName = fullName;
}

public void setEmailAddress(String emailAddress) {
this.emailAddress = emailAddress;
}

public void setInterests(String interests) {
this.interests = interests;
}
} // end Customer
```

Customer Service Database

このアプリケーションのデモンストレーションが使うデータベースは、組み込みの Derby データベースです。データベースの名前は CustomerDB で、一つのテーブルを持ちます。

Table Name	Fields
CUSTOMER	customerId (PRIMARY KEY) fullname emailaddress interests

CUSTOMER テーブルはひとつの顧客の情報を持ちます。

ツールの利用

Customer Service アプリケーションの開発、ビルドは次のツールを利用しています。

Apache Derby

Apache Derby は Apache DB サブ・プロジェクトであり、Java で実装されたリレーショナル・データベースです。サイズが小さく、Java に基づくソリューションに簡単に組み込めるものです。組み込みフレームワークに加え、Derby は Derby ネットワーク・サーバを利用することで、よくあるクライアント／サーバ・フレームワークもサポートします。
http://db.apache.org/derby/index.html

Apache Maven 2

Maven はエンタープライズ Java プロジェクト向けの有名なオープンソースのビルド・ツールです。ビルド作業の負荷を軽減できるように設計されました。Maven では Ant や他の伝統的な make ファイルなどで利用されるタスク・ベースの手法ではなく、宣言的手法を利用して、プロジェクトの構成や内容が定義されます。このことにより、企業全体で開発標準を適用することや、ビルド用スクリプトの記述やメンテナンスに必要な時間を削ることの助けとなります。宣言的で、ライフサイクルに基づく手法を使っていた Maven 1 は、伝統的なビルド方法よりも、多くの人にとって、根本的な発展となり、さらに Maven 2 はこの点を高めました。Maven 2 は次の URL からダウンロードできます。
http://maven.apache.org

サンプル・アプリケーションの構成、ビルド、デプロイ

以下のリンクから CustomerService アプリケーションをダウンロードしてください。
CustomerService

ファイルを解凍すると、CustomerService ディレクトリーが作られます。

ソース・コード

SVN からサンプルのソースコードをチェックアウトすることができます。

svn checkout http://svn.apache.org/repos/asf/geronimo/samples/trunk/samples/CustomerService

構成

アプリケーションの構成は、データベースの作成とデータベースへ接続するコネクション・プールの定義を含んでいます。

データベースの作成とデータ追加

Apache Geronimo サーバーを始動した後、Geronimo コンソールへログインし、以下の手順により CustomerDB を作成してください。

CustomerService.sql

```
create table customer (
customerid varchar(10) primary key,
fullname varchar(30),
emailaddress varchar(30),
interests varchar(100)
);

insert into customer values ('A100','John Doe10','Doe10@work.com','Java,Open Source, Computer
Graphics');
insert into customer values ('b100','Jane Doe20','Doe20@home.net','Budget Travel, New Zealand,
Martial Arts');
```

1. 左側の Console Navigation から DB Manager リンクを選択してください。
2. データベース名として CustomerDB を入力し、Create ボタンをクリックしてください。
3. Use DB 欄で CustomerDB を選択してください。
4. テキストエディタで CustomerService ディレクトリーから CustomerService.sql を開いてください。
5. SQL Commands のテキストエリアに CustomerService.sql の内容を貼りつけ、Run SQL ボタンを押してください。

ビルド

コマンド・プロンプトを利用して CustomerService ディレクトリーへ移動し、mvn clean install site コマンドを入力するとビルドされます。CustomerService フォルダーの下に CustomerService-ear-2.0-SNAPSHOT.ear が作られます。これで Geronimo アプリケーション・サーバーへ CustomerService アプリケーションをデプロイする準備ができました。

アプリケーションのデプロイ

Geronimo コンソールの利用によって、サンプルアプリケーションのデプロイはかなり簡単です。

1. Console Navigation パネルから Deploy New を選択してください。
2. Archive 入力欄に CustomerService フォルダーの CustomerService-ear-2.0-SNAPSHOT.ear を読み込んでください。
3. Install ボタンを押してアプリケーションをサーバーへデプロイしてください。

Customer Service ウェブ・アプリケーション

サンプル・ウェブ・アプリケーションをテストするには、ブラウザーを開いて http://localhost:8080/service を入力してください。アプリケーションのインデックス・ページが表示され、そこには顧客を閲覧するためのリンクがあります。

This page last changed on 4 25, 2008 by JAGUG.

EJB 3.0 ステートレス・セッション Bean のサンプル

このサンプルは以下に記述します EJB 3.0 からの新機能のデモンストレーションです。

1. セッション Bean 用に EJB コンポーネントのインターフェースが必要なくなりました。必要とされるセッション Bean 用のビジネス・インターフェースは、EJBObject、EJBLocalObject、java.rmi.Remote インターフェースではなく、プレーン Java のインターフェースとなりました。
2. セッション Bean 用ホーム・インターフェースの必要なくなりました。
3. 環境依存部分のカプセル化や JNDI へは、アノテーションによる依存性注入方式で簡易検索方式を通じて接続します。
4. Java メタ・データのアノテーションの導入は、デプロイメント・ディスクリプターの代わりとして使われることになります。

計算機

Calculator.java: EJBObject、EJBLocalObject、java.rmi.Remote のような EJB コンポーネントのインターフェースの代わりとして、ステートレス・セッション Bean は単純な Java インターフェースを実装します。このクラスに @Stateless というアノテーションを記述することによって、別に記述していたデプロイメント・ディスクリプターが不要になりました。このクラスは、CalculatorLocal と CalculatorRemote という名前のローカルとリモート両方のビジネスインターフェースを実装しました。

Calculator.java

```
package org.apache.geronimo.samples.slsb.calculator;

import javax.ejb.Stateless;

@Stateless
public class Calculator implements CalculatorRemote, CalculatorLocal {

public int sum(int add1, int add2) {
return add1+add2;
}

public int multiply(int mul1, int mul2) {
return mul1*mul2;
}

}
```

CalculatorLocal.java: これはローカルのビジネス・インターフェースですので、@Local アノテーションをこのクラスに任意に記述することができます。@Local や @Remote のアノテーションがないビジネス・インターフェースは、ローカル扱いになります。

CalculatorLocal.java

```
package org.apache.geronimo.samples.slsb.calculator;

public interface CalculatorLocal {

public int sum(int add1, int add2);

public int multiply(int mul1, int mul2);
}
```

CalculatorRemote.java: これはリモートのインターフェースですので、@Remote アノテーションが必要です。

CalculatorRemote.java

```
package org.apache.geronimo.samples.slsb.calculator;

import javax.ejb.Remote;

@Remote
```

```
public interface CalculatorRemote {

public int sum(int add1, int add2);

public int multiply(int mul1, int mul2);

}
```

CalculatorServlet.java: これは JSP ページのフォームを処理するためのサーブレットです。ステートレス・セッション Bean の Calculator を利用して、計算結果を返します。CalculatorLocal に記述のある @EJB アノテーションに注目してください。EJB コンテナはすべてのリクエストに対して別々の Bean のインスタンスをもちます。メモ: @EJB アノテーションは、ステートフル・セッションの場合は型階層に記述しますが、ステートレス・セッション Bean はどの階層でも宣言できます。

CalculatorServlet.java

```
package org.apache.geronimo.samples.calculator;

import java.io.IOException;
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.geronimo.samples.slsb.calculator.CalculatorLocal;

public class CalculatorServlet extends HttpServlet {

@EJB
private CalculatorLocal calc = null;

public void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {

try {
String firstNumber = req.getParameter("firstNumber");
String secondNumber = req.getParameter("secondNumber");
String operation = req.getParameter("operation");

int firstInt = (firstNumber == null) ? 0 : Integer.valueOf(firstNumber).intValue();
int secondInt = (secondNumber == null) ? 0 : Integer.valueOf(secondNumber).intValue();

if ( "multiply".equals(operation) ) {
req.setAttribute("result", calc.multiply(firstInt, secondInt));
}
else if ( "add".equals(operation) ) {
req.setAttribute("result", calc.sum(firstInt, secondInt));
}

System.out.println("Result is " + req.getAttribute("result"));

getServletContext().getRequestDispatcher("/sample-docu.jsp").forward(req, resp);

}
catch ( Exception e ) {
e.printStackTrace();
throw new ServletException(e);
}
}
}
```

デプロイメント・プラン

次のような構造でデプロイします。

```
|- calculator-stateless-ear-2.0-SNAPSHOT.ear
|- META-INF
|- application.xml
|- geronimo-application.xml
|- calculator-stateless-ejb-2.0-SNAPSHOT.jar
|- calculator-stateless-war-2.0-SNAPSHOT.war
```

application.xml: JAR ファイルが参照され、このアプリケーションの機能を与えられます。WAR ファイルは Web ベースのインターフェースを通してこのアプリケーションが使えるようにするために参照されます。コンテキスト・ルートが /calculator-stateless となり、このアプリケーションの URL は http://<hostname>:<port>/calculator-stateless となります。

application.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/
application_5.xsd"
version="5">
<description>Geronimo Sample EAR for Stateless Session</description>
<display-name>Geronimo Sample EAR for Stateless Session</display-name>
<module>
<web>
<web-uri>calculator-stateless-war-2.0-SNAPSHOT.war</web-uri>
<context-root>/calculator-stateless</context-root>
</web>
</module>
<module>
<ejb>calculator-stateless-ejb-2.0-SNAPSHOT.jar</ejb>
</module>
</application>
```

geronimo-application.xml: このプロジェクト（モジュールには他に依存するモジュールとは違うユニークな名前をつけます）の情報が <environment> タグの中に記述されています。今回の場合は、他のモジュールに依存していませんので、リストには記述がありません。しかしながら、全体を通してユニークな名前をつけておく方がよいです。そうすれば、後々他のアプリケーションから参照できるようになります。

geronimo-application.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://geronimo.apache.org/xml/ns/j2ee/application-1.2">

<environment xmlns="http://geronimo.apache.org/xml/ns/deployment-1.2">
<moduleId>
<groupId>${pom.groupId}</groupId>
<artifactId>${pom.artifactId}</artifactId>
<version>${version}</version>
<type>ear</type>
</moduleId>
</environment>

<module>
<web>calculator-stateless-war-${version}.war</web>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.2">
<context-root>/calculator-stateless</context-root>
</web-app>
</module>

</application>
```

導入方法

1. SVN からサンプルのソースコードをチェックアウトしてください。
   `svn checkout http://svn.apache.org/repos/asf/geronimo/samples/trunk/samples/calculator-stateless-pojo`

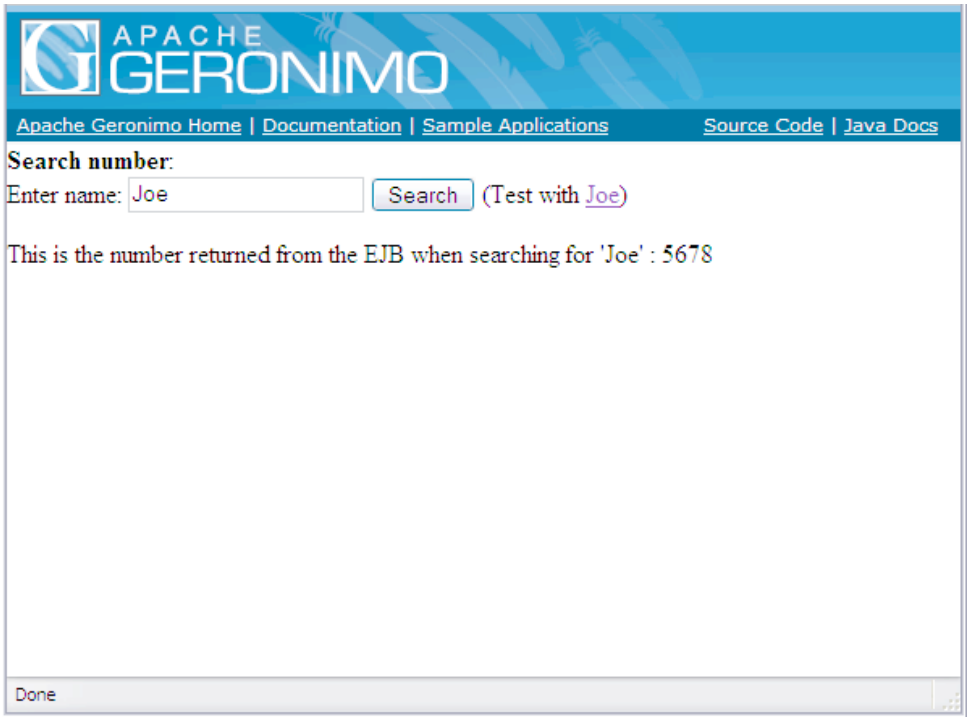
2. mvn install site コマンドを実行し、ソースコードをビルドしてください。現在のディレクトリーに calculator-stateless-ear-2.0-SNAPSHOT.ear ファイルが作られます。
3. $geronimo_home/bin ディレクトリーに移動し、Geronimo サーバを始動してください。
4. この リンクをクリックし、コンソールを利用して calculator-stateless-ear-2.0-SNAPSHOT.ear ファイルをデプロイしてください。コンソールのユーザー名/パスワードは system/manager です。または、以下のコマンドを利用してデプロイしてください。
   ```
   deploy --user system --password manager deploy <path to the ear file>
   ```
5. この URL http://localhost:8080/calculator-stateless をクリックしてください。表示されたドキュメントを読んでください。

サンプル・アプリケーションを開くと、右隅にある "source" タブをクリックすることでソース・コードを見ることができます。"javadoc" をクリックすると JavaDoc が読めます。

This page last changed on 4 25, 2008 by JAGUG.

Phone Book Bean の例

アノテーションを利用してエンティティ Bean を呼び出す JSP ページ の例です。実行結果として次のように表示されます。



アプリケーションの内容

まず、データベース内のテーブルを表す PhoneBook エンティティ Bean を見てみましょう。PhoneBook のどのインスタンスもテーブルの1レコードです。
PhoneBook.java は次の記述を利用しています。

1. @Entity アノテーションは、このクラスがエンティティ Bean という記述です
2. @Table アノテーションは、エンティティ Bean が表すテーブルの名前の記述です
3. @Id アノテーションは、テーブルのプライマリ・キーの記述です

なお通常は、エンティティ Bean には空のコンストラクターがあります。

PhoneBook.java

```
package org.apache.geronimo.samples.myphonebookpak;

import java.io.Serializable;

import javax.persistence.Id;
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity
@Table(name = "phonebook")
public class PhoneBook implements Serializable {

private String number;
private String name;
```

```
public PhoneBook() {

}

public PhoneBook(String name, String number) {
this.name = name;
this.number = number;
}

@Id
public String getName() {
return name;
}

public void setName(String name) {
this.name = name;
}

public String getNumber() {
return number;
}

public void setNumber(String number) {
this.number = number;
}
}
```

MyPhonebookLocal.java は上記で触れたエンティティ Bean への受け渡しをするビジネス・インターフェースです。

MyPhonebookLocal.java

```
package org.apache.geronimo.samples.myphonebookpak;

import org.apache.geronimo.samples.myphonebookpak.PhoneBook;

public interface MyPhonebookLocal {
public PhoneBook findByPrimaryKey(String name);
}
```

MyPhonebookBean.java は、ローカル・インターフェース（またはリモート・インターフェース）を実装したものです。このステートレス・セッション Bean にあるアノテーションの意味の説明は次のとおりです。

1. @Stateless - Geronimo にこのクラスがステートレス・セッション Bean であることを通知します
2. @PersistenceUnit - Geronimo に、persistence.xml に定義された永続化ユニット（persistence unit）を取得し、これを EntityManagerFactory とするように通知します

> ⚠ 直接 EntityManager を取得しようとするときは、PersistenceContext を利用します。EntityManagerFactory には PersistenceUnit を利用します。

MyPhonebookBean.java

```
package org.apache.geronimo.samples.myphonebookpak;

import javax.ejb.Stateless;
import javax.persistence.PersistenceUnit;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;

import org.apache.geronimo.samples.myphonebookpak.PhoneBook;

@Stateless
public class MyPhonebookBean implements MyPhonebookLocal {

@PersistenceUnit(unitName="PhonePU")
protected EntityManagerFactory emf;
```

```
    public MyPhonebookBean() {

    }

    public PhoneBook findByPrimaryKey(String name) {
    EntityManager em = emf.createEntityManager();

    PhoneBook phonebook = (PhoneBook)em.find(PhoneBook.class, name);

    em.close();

    return phonebook;
    }
    }
```

index.jsp は EJB を利用してデータベースへアクセスするための JSP ページです。

index.jsp

```
<%@ page contentType="text/html" import="org.apache.geronimo.samples.myphonebookpak.*,
javax.naming.* " %>

<%
String searchName = "";
if (request.getParameter("searchname") != null) {
searchName=request.getParameter("searchname");
}
%>

<html><head><title>Phonebook</title></head><body>
<form action="index.jsp">
<b>Search number</b>:<br>
Enter name: <input type="text" name="searchname" value="<%=searchName%>">
<input type="submit" value="Search">
(Test with <a href="index.jsp?searchname=Joe">Joe</a>)
</form>
<%
if (! searchName.equals("")) {
String number="";
try {
Context context = new InitialContext();
MyPhonebookLocal myPhonebookLocal = (MyPhonebookLocal)context.lookup("java:comp/env/ejb/
MyPhonebookBean");
PhoneBook phonebook = myPhonebookLocal.findByPrimaryKey(searchName);
if(phonebook != null) {
number =  phonebook.getNumber();
}
}
catch (Exception e) {
number=e.toString();
}
out.println("This is the number returned from the EJB when searching for '"+searchName+"' : " +
number);
}
%>
</body></html>
```

EJB のデプロイメント・プラン

openejb-jar.xml へはモジュール情報を記述します。

openejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<openejb-jar
```

```
xmlns="http://www.openejb.org/xml/ns/openejb-jar-2.1"
xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.1"
xmlns:pkgen="http://www.openejb.org/xml/ns/pkgen-2.0"
xmlns:sec="http://geronimo.apache.org/xml/ns/security-1.1"
xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2">
<sys:environment>
<sys:moduleId>
<sys:groupId>org.apache.geronimo.samples</sys:groupId>
<sys:artifactId>MyPhonebookBean</sys:artifactId>
<sys:version>1.0</sys:version>
<sys:type>car</sys:type>
</sys:moduleId>
</sys:environment>
</openejb-jar>
```

persistence.xml へは永続化ユニットの名前を記述します。この名前は EntityManagerFactory を参照する時に利用されます。今回は PhonePU という名前を与えました。どういうわけか、私は jta-data-source の参照ができませんでした。そこで代替手段として、ConnectionURL、ConnectionDriverName、ConnectionUserName を明記しました。また、データベースにあるデータが上書きされないように、追加属性として SynchronizeMappings を加えました。

以下は一つの方法です

persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/
persistence_1_0.xsd">
<persistence-unit name="PhonePU">
<description>Phone Book</description>
<provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider>
<class>org.apache.geronimo.samples.myphonebookpak.PhoneBook</class>
<properties>
<property name="openjpa.ConnectionURL" value="jdbc:derby:PhoneBookDB" />
<property name="openjpa.ConnectionDriverName" value="org.apache.derby.jdbc.EmbeddedDriver" />
<property name="ConnectionUserName" value="app" />
<property name="openjpa.jdbc.SynchronizeMappings" value="false" />
</properties>
</persistence-unit>
<!--
<jta-data-source>PhoneBookPool</jta-data-source>
<non-jta-data-source>PhoneBookPool</non-jta-data-source>
-->
</persistence>
```

Web-App のデプロイメント・プラン

web.xml には MyPhonebookLocal が所属するパッケージが記述されていて、ここから EJB が参照されます。

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd">
<display-name>MyPhonebookWeb</display-name>
<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<ejb-local-ref>
<ejb-ref-name>ejb/MyPhonebookBean</ejb-ref-name>
<ejb-ref-type>Entity</ejb-ref-type>
<local>org.apache.geronimo.samples.myphonebookpak.MyPhonebookLocal</local>
```

```
</ejb-local-ref>
</web-app>
```

geronimo-web.xml へはモジュール情報と web-app のコンテキスト・ルートを記述します。

geronimo-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1"
xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.1"
xmlns:sec="http://geronimo.apache.org/xml/ns/security-1.1"
xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.1">
<sys:environment>
<sys:moduleId>
<sys:groupId>${pom.groupId}</sys:groupId>
<sys:artifactId>${pom.artifactId}</sys:artifactId>
<sys:version>${version}</sys:version>
<sys:type>war</sys:type>
</sys:moduleId>
</sys:environment>
<context-root>/myphonebook</context-root>
</web-app>
```

アプリケーションのデプロイメント・プラン

geronimo-application.xml はデプロイする必要のあるデータベース・プールをアプリケーションに通知します。データベース・プールは PhoneBookPool.xml に定義されていて、デプロイのためのドライバは tranql-connector-ra-1.3.rar ファイルです。これら2つのファイルは生成された EAR ファイル内の最上位階層に置きます。

geronimo-application.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://geronimo.apache.org/xml/ns/j2ee/application-1.1"
xmlns:sec="http://geronimo.apache.org/xml/ns/security-1.1"
xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.1"
application-name="t6">
<sys:environment>
<sys:moduleId>
<sys:groupId>${pom.groupId}</sys:groupId>
<sys:artifactId>${pom.artifactId}</sys:artifactId>
<sys:version>${version}</sys:version>
<sys:type>ear</sys:type>
</sys:moduleId>
</sys:environment>
<module>
<connector>tranql-connector-ra-1.3.rar</connector>
<alt-dd>PhoneBookPool.xml</alt-dd>
</module>
</application>
```

アプリケーションの構成、ビルド、デプロイ

以下のリンクから MyPhoneBook アプリケーションをダウンロードしてください。
MyPhoneBook

ファイルを解凍すると、myphonebook ディレクトリーが作られます。

ソースコード

SVN からサンプルのソースコードをチェックアウトすることができます。

svn checkout http://svn.apache.org/repos/asf/geronimo/samples/trunk/samples/myphonebook

データベースの作成とデータ追加

Apache Geronimo サーバーを始動した後、Geronimo Console へログインし、以下の手順により PhoneBookDB を作成してください。

PhoneBook.sql

```
CREATE TABLE phonebook ( name VARCHAR(255) PRIMARY KEY, number VARCHAR(255) );
INSERT INTO phonebook VALUES ('John', '1234');
INSERT INTO phonebook VALUES ('Joe', '5678');
```

1. 左側の Console Navigation から DB Manager リンクを選択してください。
2. データベース名として PhoneBookDB を入力し、Create ボタンをクリックしてください。
3. Use DB 欄で PhoneBookDB を選択してください。
4. テキストエディタで myphonebook/myphonebook-ear/src/main/resources ディレクトリーから、PhoneBookDB.sql を開いてください。
5. SQL Commands のテキストエリアに PhoneBookDB.sql の内容を貼りつけ、Run SQL ボタンを押してください。

ビルド

コマンドプロンプトを利用して myphonebook ディレクトリーへ移動し、mvn install 、mvn site の順にコマンドを入力するとビルドされます。myphonebook フォルダーの下に myphonebook-ear-2.0-SNAPSHOT.ear が作られます。これで Geronimo アプリケーション・サーバーへ myphonebook アプリケーションをデプロイする準備ができました。

アプリケーションのデプロイ

Geronimo Console の利用によって、サンプルアプリケーションのデプロイはかなり簡単です。

1. Console Navigation パネルから Deploy New を選択してください。
2. Archive 入力欄に myphonebook フォルダーの myphone-ear-2.0-SNAPSHOT.ear を読み込んでください。
3. Install ボタンを押してアプリケーションをサーバへデプロイしてください。

MyPhoneBook ウェブ・アプリケーション

サンプル・ウェブ・アプリケーションをテストするには、ブラウザーを開いて http://localhost:8080/myphonebook を入力してください。

persistence.xml 内の jta-datasource を利用する方法の未テストの説明

このアプリケーションは openjpa のシーケンスを利用しないので、どうやら jta-datasource のみで利用できるようです。私の体験では、プライマリ・キーのために openjpa のシーケンスを利用するアプリケーションは、jta-datasource ではないものが必要です。このような jta-datasource ではないものをデプロイする時は、プランを確認し、<local-transaction/> や <xa-transaction/> ではなく <no-transaction/> を使うことを確認してください。

このアプリケーションのために、コンソールを利用して貴方が選択したデータベースが利用するデータソースをデプロイしてください。データソースに "MyDS" のような名前を与える必要があります。最終的に、console.dbpool/MyDS/1.0/rar のようなデータソースもジュールの名前を持つべきです。Geronimo が貴方のデータソースを使うために、2つのことが必要です。

1. Geronimo に新しいデータソース用のモジュールの探し方を知らせるために、アプリケーションの dependencies にモジュール名を含めてください。openejb-jar.xml は例えば以下のようになります。

openejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<openejb-jar
xmlns="http://www.openejb.org/xml/ns/openejb-jar-2.1"
xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.1"
xmlns:pkgen="http://www.openejb.org/xml/ns/pkgen-2.0"
xmlns:sec="http://geronimo.apache.org/xml/ns/security-1.1"
xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2">
<sys:environment>
```

```
<sys:moduleId>
<sys:groupId>org.apache.geronimo.samples</sys:groupId>
<sys:artifactId>MyPhonebookBean</sys:artifactId>
<sys:version>1.0</sys:version>
<sys:type>car</sys:type>
</sys:moduleId>
<sys:dependencies>
<sys:dependency>
<sys:groupId>console.dbpool</sys:groupId>
<sys:artifactId>MyDS</sys:artifactId>
<sys:version>1.0</sys:version>
<sys:type>rar</sys:type>
</sys:dependency>
</sys:dependencies>
</sys:environment>
</openejb-jar>
```

2. persistence.xml に、例えば以下のようにデータソース名を記述してください。

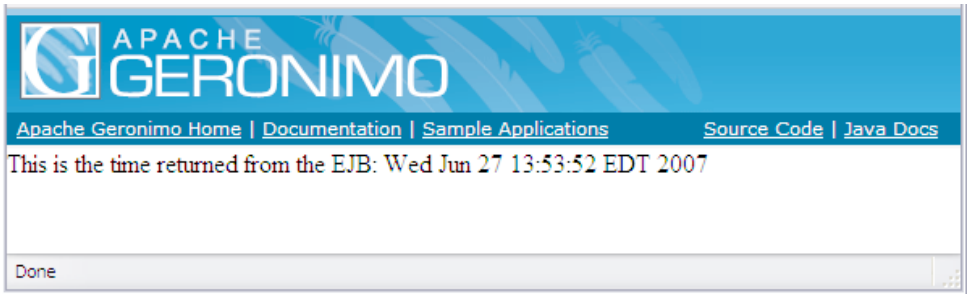
persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/
persistence_1_0.xsd">
<persistence-unit name="PhonePU">
<description>Phone Book</description>
<provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider>
<jta-datasource>MyDS</jta-datasource>
<class>org.apache.geronimo.samples.myphonebookpak.PhoneBook</class>
<properties>
<property name="openjpa.jdbc.SynchronizeMappings" value="false" />
</properties>
</persistence-unit>
<!--
<jta-data-source>PhoneBookPool</jta-data-source>
<non-jta-data-source>NoTXPhoneBookPool</non-jta-data-source>
-->
</persistence>
```

This page last changed on 4 25, 2008 by JAGUG.

Time Bean の例

セッション Bean を呼び出す JSP ページ の例です。実行結果として次のように表示されます。



簡単に理解できるように、この例ではすべてを削ぎ落として作りました。Geronimo 2.0、Java 1.5、EJB 3.0 を利用しています。

Application Contents アプリケーションの内容

MyTimeBean.java は時刻を返答する EJB です。org.apache.geronimo.samples.mytimepak というパッケージに EJB を入れてあります。@Stateless アノテーションによって、Geronimo はこのクラスがステートレス・セッション Bean であると認識します。ejb-jar.xml は必要ありません。

MyTimeBean.java

```
package org.apache.geronimo.samples.mytimepak;

import javax.ejb.Stateless;

@Stateless
public class MyTimeBean implements MyTimeLocal {

public String getTime() {
String s = new java.util.Date().toString();
return s;
}
}
```

MyTimeLocal.java はローカルのインターフェースです。この EJB は同一のサーバー（同一の JVM）上で実行されている JSP のページからしか呼び出されないので、ネットワークからは利用しないローカルのインターフェースとして作ります。

MyTimeLocal.java

```
package org.apache.geronimo.samples.mytimepak;
public interface MyTimeLocal {
public java.lang.String getTime() ;
}
```

openejb-jar.xml does nothing but specifies the module's information.

openejb-jar.xml へはモジュール情報を記述するだけです。

openejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<openejb-jar
xmlns="http://www.openejb.org/xml/ns/openejb-jar-2.1"
xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.1"
xmlns:pkgen="http://www.openejb.org/xml/ns/pkgen-2.0"
xmlns:sec="http://geronimo.apache.org/xml/ns/security-1.1"
```

```
xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.2">
<sys:environment>
<sys:moduleId>
<sys:groupId>${pom.groupId}</sys:groupId>
<sys:artifactId>${pom.artifactId}</sys:artifactId>
<sys:version>${version}</sys:version>
<sys:type>jar</sys:type>
</sys:moduleId>
</sys:environment>
</openejb-jar>
```

index.jsp は MyTimeBean を利用して時刻を応答します。

index.jsp

```
<%@ page contentType="text/html" import="org.apache.geronimo.samples.mytimepak.*, javax.naming.* "
%>
<html<head><title>Time</title></head><body>
<%
String s="-"; // Just declare a string
try {
// This creates a context, it can be used to lookup EJBs. Using normal RMI you would
// have to know port number and stuff. The InitialContext holds info like
// server names, ports and stuff I guess.
Context context = new InitialContext();
// MyTimeLocalHome is a reference to the EJB
MyTimeLocal myTimeLocal = (MyTimeLocal)context.lookup("java:comp/env/ejb/MyTimeBean");
// So, just go ahead and call a method (in this case the only method).
s =  myTimeLocal.getTime();
}
catch (Exception e) {
s=e.toString();
}
%>
This is the time returned from the EJB: <%=s%>
</body></html>
```

geronimo-web.xml へはモジュール情報と web-app の url を記述します。

geronimo-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1"
xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.1"
xmlns:sec="http://geronimo.apache.org/xml/ns/security-1.1"
xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.1">
<sys:environment>
<sys:moduleId>
<sys:groupId>${pom.groupId}</sys:groupId>
<sys:artifactId>${pom.artifactId}</sys:artifactId>
<sys:version>${version}</sys:version>
<sys:type>war</sys:type>
</sys:moduleId>
</sys:environment>
<context-root>/mytime</context-root>
</web-app>
```

web.xml は EJB の場所が WEB-INF/classes/org/apache/geronimo/samples/mytimepak ディレクトリーであることを表します。

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd">
<display-name>MyTimeWeb</display-name>
<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<!-- To refer local EJB's  -->
<ejb-local-ref>
<ejb-ref-name>ejb/MyTimeBean</ejb-ref-name>
<ejb-ref-type>Session</ejb-ref-type>
<local>org.apache.geronimo.samples.mytimepak.MyTimeLocal</local>
</ejb-local-ref>
</web-app>
```

ツールの利用

Apache Maven 2

Maven はエンタープライズ Java プロジェクト向けの有名なオープンソースのビルドツールです。ビルド作業の負荷を軽減できるように設計されました。Maven では Ant や他の伝統的な make ファイルなどで利用されるタスク・ベースの手法ではなく、宣言的手法を利用して、プロジェクトの構成や内容が定義されます。このことにより、企業全体で開発標準を適用することや、ビルド用スクリプトの記述やメンテナンスに必要な時間を削ることの助けとなります。宣言的で、ライフサイクルに基づく手法を使っていた Maven 1 は、伝統的なビルド方法よりも、多くの人にとって、根本的な発展となり、さらに Maven 2 はこの点を高めました。Maven 2 は次の URL からダウンロードできます。
http://maven.apache.org

アプリケーションのビルド、デプロイ

次のリンクから mytime アプリケーションをダウンロードしてください。
MyTime (注意: このリンク先は最新ではないかもしれません。最新版は下記のソースコードをチェックアウトしてください)

ファイルを解凍すると、mytime ディレクトリーが作られます。

ソースコード

SVN からサンプルのソースコードをチェックアウトすることができます。

svn checkout http://svn.apache.org/repos/asf/geronimo/samples/trunk/samples/mytime

(ヒント: Windows 利用者は、svn を http://subversion.tigris.org/ から入手できます)

ビルド

コマンドプロンプトを利用して mytime ディレクトリーへ移動し、mvn clean install site コマンドを入力するとビルドされます。mytime フォルダーの下に mytime-ear-2.0-SNAPSHOT.ear が作られます。これで Geronimo アプリケーション・サーバーへ mytime アプリケーションをデプロイする準備ができました。

アプリケーションのデプロイ

Geronimo Console の利用によって、サンプルアプリケーションのデプロイはかなり簡単です。

1. Console Navigation パネルから Deploy New を選択してください。
2. Archive 入力欄に mytime フォルダーの mytime-ear-2.0-SNAPSHOT.ear を読み込んでください。
3. Install ボタンを押してアプリケーションをサーバーへデプロイしてください。

MyTime ウェブ・アプリケーション

サンプル・ウェブ・アプリケーションをテストするには、ブラウザーを開いて http://localhost:8080/mytime を入力してください。

This page last changed on 4 27, 2008 by JAGUG.

この文章では、Apache Geronimo サーバーの機能に関連したウェブ・アプリケーションのセキュリティーに注目します。この文章のサンプル・アプリケーションは、サーブレット、JSP や J2EE の宣言的セキュリティーを利用する基本的な Time Reporting システムについて書かれています。上記セキュリティーの機能に加えて、Geronimo に組み込まれている Derby データベースをこのシステムのユーザー情報保存のために利用しています。このアプリケーションはユーザー情報保持のためにデータベースを使いますが、これは単に構成の目的で利用しています。Geronimo での JDBC の利用方法の詳細情報は、9.b. データベース接続の簡単なサンプル・アプリケーション を参照してください。

この文章を読めば、宣言的セキュリティー機能を利用したウェブ・アプリケーションのための Geronimo アプリケーションサーバーの構成が可能になるでしょう。

この文章は、次のセクションで構成されています。

Geronimo のウェブ・アプリケーション

Apache Geronimo は J2EE ウェブ・アプリケーションをサポートするウェブ・アプリケーションコンテナを持っています。ウェブ・コンテナ自身は、通信ポートや SSL オプションなどの基本構成をサポートしています。ウェブ・アプリケーションは Geronimo 仕様の構成情報も含むことができます。ウェブ・アプリケーションを Geronimo のセキュリティー機構に参加させることができます。ウェブ・アプリケーションを認証することで、セキュアな EJB やコンテナにアクセスすることができます。

Apache Geronimo は現在、次の2つのウェブ・コンテナをサポートしています: Jetty, Tomcat

Jetty

Jetty は 100% Java の HTTP サーバー、サーブレット・コンテナ です。したがって、サーブレットや JSP を利用して動的にコンテンツを生成するために、別のウェブサーバーを設定したり実行する必要はありません。Jetty は静的、動的コンテンツを扱うすべての機能をもったウェブサーバーです。

サーバとコンテナが分離された方法とは違い、Jetty のウェブサーバーとウェブ・アプリケーションは同じプロセス上で実行されるので、内部通信のオーバーヘッドや複雑さがありません。さらに、ピュア Java コンポーネントなので Jetty は簡単に貴方のアプリケーションのデモンストレーションや配布、配備に利用できます。Jetty は Java をサポートするすべてのプラットフォームで利用できます。
http://jetty.mortbay.org/jetty/index.html

Tomcat

Apache Tomcat は Java サーブレット、JavaServer Pages 技術の公式リファレンス実装として利用されているサーブレット・コンテナです。
http://tomcat.apache.org/

アプリケーションの概略

Time Reporting アプリケーションは、各プロジェクトの従事時間の報告をするものです。大きなアプリケーションではありませんが、Apache Geronimo の持つ表示とセキュリティとの関連機能を含んでいます。

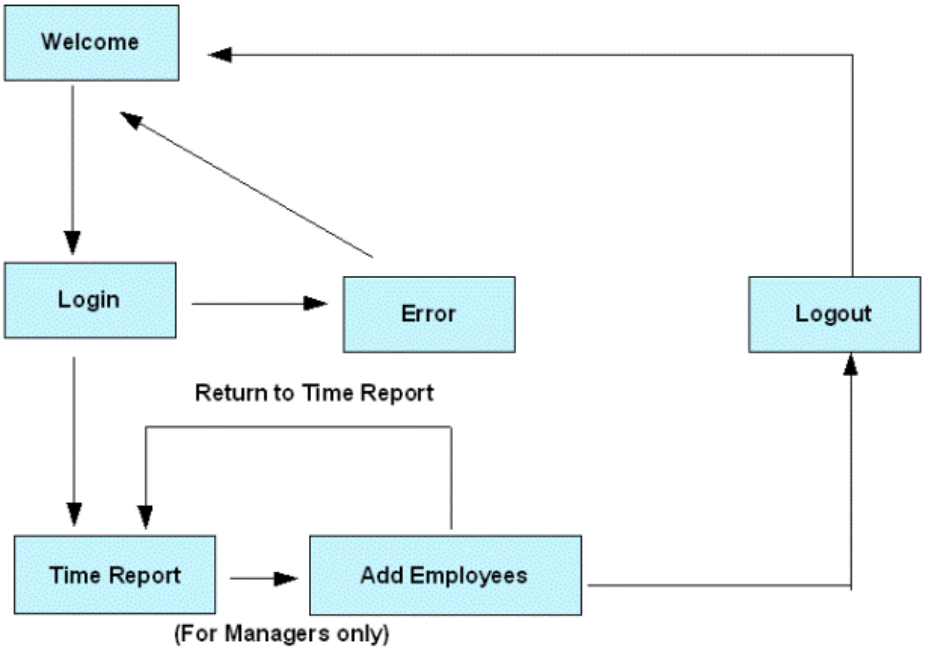
このサンプル・アプリケーションでの時間報告作業には「マネージャー(managers)」「従業員(employees)」の2種類のユーザグループがあります。両方のユーザとも時間報告作業前に自身の証明をする必要があります。マネージャーはこのシステムの特権ユーザですので、従業員を追加することができます。

Time Reporting アプリケーションには以下のページがあります。

- Welcome
- Login

- Time Report
- Add Employees
- Logout

次の図はアプリケーションの流れの概略です。



デフォルトのサンプル・アプリケーションは Time Reporting の機能へのリンクを持つ Welcome ページを表示します。利用者は Login ページで有効なユーザー名とパスワードを入力することで Time Report ページを表示できます。このユーザーの認証により マネージャーの役割が認められると、Time Report ページには Add Employees 機能へのリンクが追加されて表示されます。

アプリケーションの内容

以下に Time Reporting アプリケーションの主要なフォルダーの階層構成を示します。アプリケーションで利用する JSP と構成ファイルがあります。

```
|- employee
|- index.jsp
|- login
|- login.jsp
|- login_error.jsp
|- logout.jsp
|- manager
|- index.jsp
|- WEB_INF
|- geronimo-web.xml
|- web.xml
|- index.jsp
```

上記の JSP と構成ファイルに加えて、このアプリケーション用ビジネス・ロジックをもつ2つのサーブレットが必要です。

- AddTimeRecordServlet - Read the input data from the Time Report page
- AddEmployeeServlet - Capture input information from Add Employee page
- AddTimeRecordServlet - Time Report ページで入力されたデータを読み込む
- AddEmployeeServlet - Add Employee ページで入力された情報を取得する

Time Reporting アプリケーションのセキュリティ構成は geronimo-web.xml と web.xml ファイルによって扱われています。geronimo-web.xml は TimeReportRealm としてユーザーの役割を定義しています。

geronimo-web.xml の最初の部分はわかりやすいのですが、セキュリティ構成は手が込んでいます。<security-realm-name> は <security> 要素内の一連の <realms> 要素の定義に利用されています。

web.xml のセキュリティ・ロール（security roles）の記述によって、geronimo-web.xml はユーザーやグループがどの Geronimo セキュリティー・レルムに所属しているかを割り当てます。ログインしていないユーザーには、<default-principal> 要素によって定義された realm が付与されます。

本プロジェクトには マネージャー（manager） と 従業員（employee） という2つの役割があります。マネージャーは会社の従業員でもありますので、employee としても登録され、さらに manager にも登録されています。

geronimo-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1">

<environment>
<moduleId>
<groupId>${pom.groupId}</groupId>
<artifactId>${pom.artifactId}</artifactId>
<version>${version}</version>
<type>war</type>
</moduleId>
</environment>

<context-root>/timereport</context-root>

<security-realm-name>TimeReportRealm</security-realm-name>

<security>
<default-principal realm-name="TimeReportRealm">
<principal name="anonymous"
class="org.apache.geronimo.security.realm.providers.GeronimoUserPrincipal"
/>
</default-principal>
<role-mappings>
<role role-name="employee">
<realm realm-name="TimeReportRealm">
<principal name="EmployeeGroup"
class="org.apache.geronimo.security.realm.providers.GeronimoGroupPrincipal"
/>
</realm>
<realm realm-name="TimeReportRealm">
<principal name="ManagerGroup"
class="org.apache.geronimo.security.realm.providers.GeronimoGroupPrincipal"
/>
</realm>
</role>
<role role-name="manager">
<realm realm-name="TimeReportRealm">
<principal name="ManagerGroup"
class="org.apache.geronimo.security.realm.providers.GeronimoGroupPrincipal"
/>
</realm>
</role>
</role-mappings>
</security>

</web-app>
```

web.xml はユーザーの役割をウェブ・アプリケーションのリソースに割り当てます。また、アプリケーションのログイン構成も定義しています。

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
```

```
version="2.4">

<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>

<security-constraint>
<web-resource-collection>
<web-resource-name>employee</web-resource-name>
<url-pattern>/employee/*</url-pattern>
</web-resource-collection>
<auth-constraint>
<role-name>employee</role-name>
</auth-constraint>
</security-constraint>

<security-constraint>
<web-resource-collection>
<web-resource-name>manager</web-resource-name>
<url-pattern>/manager/*</url-pattern>
</web-resource-collection>
<auth-constraint>
<role-name>manager</role-name>
</auth-constraint>
</security-constraint>

<login-config>
<auth-method>FORM</auth-method>
<realm-name>TimeReportRealm</realm-name>
<form-login-config>
<form-login-page>/login/login.jsp</form-login-page>
<form-error-page>/login/login_error.jsp</form-error-page>
</form-login-config>
</login-config>

<security-role>
<role-name>employee</role-name>
</security-role>
<security-role>
<role-name>manager</role-name>
</security-role>

<servlet>
<display-name>AddTimeRecordServlet</display-name>
<servlet-name>AddTimeRecordServlet</servlet-name>
<servlet-class>org.apache.geronimo.samples.timereport.web.AddTimeRecordServlet</servlet-class>
</servlet>
<servlet>
<display-name>AddEmployeeServlet</display-name>
<servlet-name>AddEmployeeServlet</servlet-name>
<servlet-class>org.apache.geronimo.samples.timereport.web.AddEmployeeServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>AddTimeRecordServlet</servlet-name>
<url-pattern>/employee/add_timerecord</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>AddEmployeeServlet</servlet-name>
<url-pattern>/manager/add_employee</url-pattern>
</servlet-mapping>

</web-app>
```

Time Report ページから Add Employee 機能への遷移を規制するために、以下のようなプログラムによる認証を利用しています。

employee/index.jsp

```
...
<BR>
<%if(request.isUserInRole("manager")){%>
<A href="../manager/">Add Employees</A>
<BR>
...
```

geronimo-application.xml はデプロイする必要のあるデータベース・プールをアプリケーションに通知します。セキュリティー・レルム構成はデータベース・プールと一緒に含まれています。データベース・プールは TimeReportPool.xml に定義されています。また、ドライバーとして tranql-connector-ra-1.3.rar ファイルをデプロイする必要があることが記述されています。これら2つのファイルは生成された EAR ファイル内の最上位階層に置きます。

geronimo-application.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://geronimo.apache.org/xml/ns/j2ee/application-1.2">

<environment xmlns="http://geronimo.apache.org/xml/ns/deployment-1.2">
<moduleId>
<groupId>${pom.groupId}</groupId>
<artifactId>${pom.artifactId}</artifactId>
<version>${version}</version>
<type>ear</type>
</moduleId>
</environment>

<module>
<connector>tranql-connector-ra-1.3.rar</connector>
<alt-dd>TimeReportPool.xml</alt-dd>
</module>
</application>
```

TimeReportPool.xml には2つの内容が定義されています。 データベース・プールとセキュリティー・レルムです。以下に示すように、最初の部分は他のデータベース・プールと似ています。次の部分には、必須のセキュリティー・レルム計画があります。この2つの内容をひとつの独立したファイルに入れることによって、アプリケーションにデータベース・プールとセキュリティー・レルムを入れて出荷でき、インストール時の作業を軽減するでしょう。

TimeReportPool.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<connector xmlns="http://geronimo.apache.org/xml/ns/j2ee/connector-1.2">
<dep:environment xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2">
<dep:moduleId>
<dep:groupId>console.dbpool</dep:groupId>
<dep:artifactId>TimeReportPool</dep:artifactId>
<dep:version>1.0</dep:version>
<dep:type>rar</dep:type>
</dep:moduleId>
<dep:dependencies>
<dep:dependency>
<dep:groupId>org.apache.geronimo.configs</dep:groupId>
<dep:artifactId>j2ee-security</dep:artifactId>
<dep:type>car</dep:type>
</dep:dependency>
<dep:dependency>
<dep:groupId>org.apache.geronimo.configs</dep:groupId>
<dep:artifactId>system-database</dep:artifactId>
<dep:type>car</dep:type>
</dep:dependency>
</dep:dependencies>
</dep:environment>
<!--db pool fragment-->
<resourceadapter>
<outbound-resourceadapter>
<connection-definition>
```

```
<connectionfactory-interface>javax.sql.DataSource</connectionfactory-interface>
<connectiondefinition-instance>
<name>TimeReportPool</name>
<config-property-setting name="Driver">org.apache.derby.jdbc.EmbeddedDriver</config-property-
setting>
<config-property-setting name="UserName">app</config-property-setting>
<config-property-setting name="ConnectionURL">jdbc:derby:TimeReportDB</config-property-setting>
<connectionmanager>
<local-transaction/>
<single-pool>
<max-size>10</max-size>
<min-size>0</min-size>
<match-one/>
</single-pool>
</connectionmanager>
</connectiondefinition-instance>
</connection-definition>
</outbound-resourceadapter>
</resourceadapter>
<!--security realm fragment-->
<gbean name="TimeReportRealm" class="org.apache.geronimo.security.realm.GenericSecurityRealm">
<attribute name="realmName">TimeReportRealm</attribute>
<reference name="ServerInfo">
<name>ServerInfo</name>
</reference>
<xml-reference name="LoginModuleConfiguration">
<log:login-config xmlns:log="http://geronimo.apache.org/xml/ns/loginconfig-1.1">
<log:login-module control-flag="REQUIRED" wrap-principals="false">
<log:login-domain-name>TimeReportRealm</log:login-domain-name>
<log:login-module-class>org.apache.geronimo.security.realm.providers.SQLLoginModule</log:login-
module-class>
<log:option name="jdbcDriver">org.apache.derby.jdbc.EmbeddedDriver</log:option>
<log:option name="jdbcUser">app</log:option>
<log:option name="userSelect">select userid, password from users where userid=?</log:option>
<log:option name="groupSelect">select userid, groupname from usergroups where userid=?</log:option>
<log:option name="jdbcURL">jdbc:derby:TimeReportDB</log:option>
</log:login-module>
</log:login-config>
</xml-reference>
</gbean>
</connector>
```

ツールの利用

Time Reporting サンプル・アプリケーションの開発、構築に利用したツールは以下のとおりです。

Apache Maven 2

Maven はエンタープライズ Java プロジェクト向けの有名なオープンソースのビルドツールです。ビルド作業の負荷を軽減できるように設計されました。Maven では Ant や他の伝統的な make ファイルなどで利用されるタスク・ベースの手法ではなく、宣言的手法を利用して、プロジェクトの構成や内容が定義されます。このことにより、企業全体で開発標準を適用することや、ビルド用スクリプトの記述やメンテナンスに必要な時間を削ることの助けとなります。宣言的で、ライフサイクルに基づく手法を使っていた Maven 1 は、伝統的なビルド方法よりも、多くの人にとって、根本的な発展となり、さらに Maven 2 はこの点を高めました。Maven 2 は次の URL からダウンロードできます。
http://maven.apache.org

サンプル・アプリケーションの構成、ビルド、デプロイ

以下のリンクから Time Reporting アプリケーションをダウンロードしてください。
Time Report

zip ファイルを展開すると、time_report ディレクトリーが作られます。

ソースコード

SVN からサンプル・アプリケーションのソースコードをチェックアウトすることができます。

svn checkout http://svn.apache.org/repos/asf/geronimo/samples/trunk/samples/timereport

構成

Time Reporting アプリケーションは J2EE の宣言的セキュリティを利用して実行するので、情報を保存するためにデータベースを作り、セキュリティー・レルムをデプロイする必要があります。

ユーザ情報保存用データベースの作成

Apache Geronimo サーバーを始動した後、Geronimo コンソール へログインし、以下の手順によりユーザー情報保存用の TimeReportDB を作成してください。

TimeReportDB.sql

```
CREATE TABLE users(
userid VARCHAR(15) PRIMARY KEY,
password VARCHAR(15),
name VARCHAR(40)
);

CREATE TABLE usergroups(
userid VARCHAR(15),
groupname VARCHAR(20),
PRIMARY KEY (userid, groupname)
);

INSERT INTO users VALUES('emp1', 'pass1', 'Employee 1');
INSERT INTO users VALUES('emp2', 'pass2', 'Employee 2');
INSERT INTO users VALUES('mgm1', 'pass3', 'Manager 1');
INSERT INTO users VALUES('mgm2', 'pass4', 'Manager 2');

INSERT INTO usergroups VALUES('emp1', 'EmployeeGroup');
INSERT INTO usergroups VALUES('emp2', 'EmployeeGroup');
INSERT INTO usergroups VALUES('mgm1', 'ManagerGroup');
INSERT INTO usergroups VALUES('mgm2', 'ManagerGroup');
```

1. 左側の Console Navigation から DB Manager リンクを選択してください。
2. Create DB 欄にデータベース名として TimeReportDB を入力し、Create ボタンをクリックしてください。
3. Use DB 欄で TimeReportDB を選択してください。
4. time_report/config ディレクトリーから、TimeReportDB.sql を開いてください。
5. SQL Commands のテキストエリアに TimeReportDB.sql の内容を貼りつけ、Run SQL ボタンを押してください。

ビルド

Time Report アプリケーションは pom.xml によってソースコードからのビルドされます。コマンド・プロンプト・ウィンドウを開き、timereport ディレクトリーへ移動し、mvn install site とコマンドを入力するとビルドされ、timereport フォルダーの下に timereport-ear-2.0-SNAPSHOT.ear が作られます。これで Geronimo アプリケーション・サーバーに Time Report アプリケーションをデプロイする準備ができました。

デプロイ

Geronimo コンソール の利用によって、サンプルアプリケーションのデプロイはかなり簡単です。

1. Console Navigation パネルから Deploy New を選択してください。
2. Archive 入力欄に time_report フォルダーの timereport-ear-2.0-SNAPSHOT.ear を読み込んでください。
3. Install ボタンを押してアプリケーションをサーバへデプロイしてください。

Back to Top

サンプル・アプリケーションのテスト

サンプル・アプリケーションのテストは、ブラウザを開き、http://localhost:8080/timereport を入力してください。アプリケーションの Welcome ページへ遷移します。

Time Report ページへは、ユーザー名を emp1 、パスワードを pass1 で接続できます。マネージャーとしてアプリケーションにログインするには、mgm1 と pass3 を利用してください。

まとめ

この文章では、J2EE の宣言的セキュリティー機能を持つ Geronimo アプリケーション・サーバーにウェブ・アプリケーションをデプロイする方法を示しました。順に説明にしたがって、サンプル・アプリケーションをビルド、デプロイ、テストをしました。

この文章のハイライトは、次のとおりです。

- Apache Geronimo は Jetty と Tomcat という2つのウェブ・コンテナを持っています
- 組み込みの Derby を利用してセキュリティーのデータを持つデータベースを作れます
- Geronimo ウェブ・アプリケーション内にセキュリティー・ロールを定義できます
- Geronimo コンソール を利用してデプロイメント・プランとウェブ・アーカイブをデプロイします

A. トラブルシューティング

This page last changed on 4 25, 2008 by JAGUG.

起動時の問題

よくある起動時の問題は、通信ポートの衝突です。Geronimo がデフォルトで利用する以下の通信ポートが他のアプリケーションによって使われていたり、塞がれていないことを確認してください。

- 1099
- 4242
- 9999
- 4201
- 1050
- 1527
- 61616
- 8080
- 8443
- 8009
- 25

パーソナル・ファイアウォールやウイルス・スパイウェア対策製品によって、これらの通信ポートが塞がれているかもしれません。仮にこれらのソフトを停止しても、これらのソフトで利用されているルールがまだ影響を与えていることがあります。もし Geronimo がデフォルト設定の状態で動かなかったら、このことに注意してください。

B.謝辞

謝辞

当翻訳は2007年秋から2008年春にかけ、Japan Apache Geronimo Users Groupの翻訳グループ有志により行われました。
下記の皆様の貢献に感謝します。(2008/5月)

- 井上さん
- 花田さん
- 橋本さん
- 坂本さん
- 小川さん
- 石田さん
- 大貫さん
- 樋口さん
- 武田さん
- 保坂さん
- 野宗さん
- 矢那浦さん
  （漢字ソート順）

Index

This page last changed on 11 09, 2007 by hcunico.

ここは、Apache Geronimo v2.0 ドキュメントの日本語翻訳のホームページです。

Apahce Geronimo v2.0 ドキュメントへようこそ。この Geronimo のバージョンと同じように、このドキュメントもまだ作成途中です。

Apache Geronimo v2.0 は最初の JEE 5 認証を得たリリースです。現時点では、ウェブ・サービスには CXF を、永続化には OpenJPA を組み込んだ Geronimo-Tomcat において、SUN の Java Enterprise Edition 5.0 Certification Test Suite を通過しています。私たちは他の組み込みソフトウェアでも認証を得ようと努力し続けています。8. RELEASE-NOTES-2.0.2.TXT を参照し、サポートされている機能や既知の問題点や制限事項を確認してください。

以下の文書では、サンプル・アプリケーションで機能をテストしながら、Apahce Geronimo v2.0 を構築し、利用する方法を説明しています。皆様のコメントや投稿は歓迎しています。