使用 Google Web Toolkit 和 Apache Geronimo 构建启用 Ajax 的应用程序，第 1 部分: 在 Geronimo 上运行经过编译的 Google Web Toolkit 应用程序

Michael Galpin (mike.sr@gmail.com)，开发人员, Adomo, Inc.

2007 年 8 月 02 日

启用 Asynchronous Java™Script + XML (Ajax) 的 Web 应用程序已经在软件开发界掀起了一股风潮。Google 已经构建了一些最著名的启用 Ajax 的 Web 应用程序。本系列教程共分两部分，将向您展示 Google Web Toolkit (GWT) 和 Apache Geronimo 如何帮助您快速构建复杂的 Ajax Web 应用程序，而无需编写任何 JavaServer Page (JSP) 组件、servlet 或 JavaScript。

开始之前

本教程是 两部分系列教程 中的第一部分，它适用于需要创建启用 Ajax 的 Web 应用程序和需要了解关于 GWT 和 Apache Geronimo 的信息的 Java 开发人员。

关于本系列

启用 Ajax 的 Web 应用程序最近的流行程度令人难以置信。Ajax 使 Web 应用程序行为与桌面应用程序极为相似。启用 Ajax 的 Web 应用程序比以前生成的 Web 应用程序提供更优秀的交互性和功能性。并且 Geronimo 将为构建启用 Ajax 的 Web 应用程序提供优秀平台。

但是，构建启用 Ajax 的 Web 应用程序比构建传统的 Web 应用程序难得多。它将涉及大量 JavaScript 和 Dynamic HTML (DHTML) 开发。不同浏览器甚至同一浏览器的不同版本之间的变化使事情进一步复杂化。GWT 是众多 Ajax 框架之一，它可以使您更轻松地完成工作，可以使用新颖的方法用 Java 语言编写所有代码并为您生成全部 JavaScript。本系列中的两个教程将涵盖以下主题：

- **第 1 部分** 将主要介绍如何开始使用 GWT 并创建简单的启用 Ajax 的 Web 应用程序以检索股票报价。
- **第 2 部分** 将向您展示如何利用 GWT 和 Geronimo 使股票报价应用程序更高级以及如何把它改造成股票投资应用程序。

关于本教程

在本教程中 ,您将使用 GWT 构建启用 Ajax 的 Web 应用程序并了解使用 Geronimo 开发和部署 GWT 应用程序是多么轻松。

本教程包括以下主题：

- 安装 GWT。
- 使用 GWT 的命令行工具。
- 使用 GWT 的部件进行编程 UI 开发。
- 创建由 Web 应用程序调用的启用 Ajax 的服务。
- 使用 Geronimo 构建和部署应用程序。

查看 Ajax 资源中心，这是您的一站式 Ajax 编程模型信息中心，包括文章、教程、论坛、blog、wiki、事件和新闻。这里将介绍所有最新信息。

回页首

先决条件

本教程介绍的是如何构建 Java Web 应用程序，但您只需要了解 Java servlet 的基础知识。GWT 将利用许多常见于其他技术的原理，例如布局管理、事件系统和远程过程调用（RPC)，因此预先了解这些原理将使您可以轻松地掌握 GWT。您还将使用 Geronimo Console、HTML 和层叠样式表（CSS)，因此稍微熟悉这些工具会十分有帮助。

系统要求

开始之前，需要使用以下软件设置开发环境：

- 配有 Tomcat 的 Geronimo 2.0 —— 本教程中的样例 Web 应用程序是使用配有 Tomcat 的 Geronimo 构建的，但是它应当也可以与配有 Jetty 的 Geronimo 结合使用，因为所有内容都是标准的 Java Platform, Enterprise Edition (Java EE)。本教程将使用 Geronimo 2.0 (M3)，但是也应当可以使用早期版本的 Geronimo。
- Java 5 或 Java 6 —— 样例 Web 应用程序将使用泛型和注释。它是使用 Java 6 开发的 ,但是也针对 Java 5 进行过测试。
- JSTL 1.1 的 Apache Jakarta 实现 —— Geronimo 1.1 是经过认证的 J2EE 1.4 实现，因此您需要使用 JSTL 1.1。
- Google Web Toolkit —— 本教程的所有内容都与 GWT 有关；请下载本教程所需的 GWT 1.3.3。
- Eclipse —— 样例 Web 应用程序是使用 Eclipse 构建的，正如您将从屏幕捉图中看到的一样。GWT 附带了一个方便的命令行工具以创建骨架 Eclipse 项目。如果没有使用 Eclipse，构建 GWT 应用程序也不会很难，但是 Eclipse 将提供一个优秀的调试程序可以调试 GWT 应用程序。

安装 GWT

要开始使用 GWT，需要先安装它。当转到 GWT 下载页面时，它将为您提供特定于操作系统的 GWT 发行版。那是因为 GWT 附带了生成应用程序所需的工件的几个命令行可执行文件。它创建的一些工件包括用于运行 GWT Java-to-JavaScript 编译器和以托管模式运行应用程序的可执行文件。

下载 GWT 后，将其解压缩并转到它所在的解压缩目录。您看到的文件应当类似于图 1 中所示的那些文件。

图 1. GWT 安装目录

请注意可执行文件，例如 applicationCreator.cmd 和 projectCreator.cmd。您将使用这些可执行文件来生成应用程序并为应用程序生成 Eclipse 项目。您可能还需要创建一个环境变量，例如 GWT_HOME。本教程将引用此变量。

安装 GWT 所需的全部工作已经完成！不需要运行安装程序或类似安装程序的内容。只需解压缩归档文件，现在您已经准备好开始使用 GWT 构建 Web 应用程序。

构建股票报价应用程序

在本教程中，您将构建一个用于查找股票信息的 Web 应用程序。用户输入股票代码，应用程序就将检索关于该支股票的信息。您将使用 Yahoo 的报价服务来查找此信息，因为它是免费的并易于使用。

如果构建的是传统的 Web 应用程序，则需要创建和提交表单。提交表单将触发服务器调用业务逻辑（本例中是指从 Yahoo 查找股票信息）。但是在这里您将使用 Ajax，因此根本不使用表单。请求信息（本例中为股票代码）将被异步发送到调用业务逻辑的服务器上。您的服务器随后将返回股票信息。处理 Web 页面的文档对象模型（Document Object Model，DOM）的 JavaScript 回调方法将使用该信息来动态生成用于显示股票信息的 HTML 元素。听起来很复杂，是不是？使用 GWT 使一切变得简单！

GWT 的众多优秀特性之一就是它提供了易于使用的命令行工具可以快速投入到应用程序开发中。您将使用这些工具来设置应用程序并为应用程序创建 Eclipse 项目。

让我们开始吧！

applicationCreator

在 $GWT_HOME 中打开命令提示符，并输入命令 applicationCreator -out *<directory for code>* org.developerworks.stocks.client.StocksApplication，替换需要用于代码的目录路径。执行这条命令将为您创建所有目录及相关的骨架文件，如清单 1 所示。

清单 1. 创建应用程序

```
C:\code\gwt-windows-1.3.3>applicationCreator -out c:\code\stocks
org.developerworks.stocks.client.StocksApplication
Created directory c:\code\stocks\src
Created directory c:\code\stocks\src\org\developerworks\stocks
Created directory
c:\code\stocks\src\org\developerworks\stocks\client
Created directory
```

```
c:\code\stocks\src\org\developerworks\stocks\public
Created file
c:\code\stocks\src\org\developerworks\stocks\StocksApplication.gwt.
xml
Created file
c:\code\stocks\src\org\developerworks\stocks\public\StocksApplicati
on.html
Created file
c:\code\stocks\src\org\developerworks\stocks\client\StocksApplicati
on.java
Created file c:\code\stocks\StocksApplication-shell.cmd
Created file c:\code\stocks\StocksApplication-compile.cmd
```

在这里，有几个事项必须注意：

- 需要把应用程序 org.developerworks.stocks.client.StocksApplication 放入客户机软件包（GWT 的推荐实践），而服务应当放在服务器软件包中。客户机软件包中的所有内容都将被编译为 Java 代码和 JavaScript，然而服务器软件包中的所有内容将只被编译为 Java 字节码。
- 公共目录对于 HTML 文件十分有用。使用 GWT 只需 HTML 文件，而不需要 JSP 或 JavaServer Face (JSF)，虽然也可以使用这些文件。

- 创建了两个可执行文件：StocksApplication-shell 和 StocksApplication-compile。编译可执行文件将促成 GWT 编译需要编译成 JavaScript 的客户机软件包中的 Java 代码。shell 应用程序将启动 GWT 的嵌入式 Web 服务器（Tomcat），因此可以以托管模式运行应用程序。您稍后将看到此应用程序运行。

现在您已经设置了应用程序，您可以准备好将其与 Eclipse 结合使用。

回页首

projectCreator

现在是时候创建 Eclipse 项目了。输入命令 projectCreator -eclipse stocks -out *<path to source code>* ，如清单 2 所示：

清单 2. 创建 Eclipse 项目

```
C:\code\gwt-windows-1.3.3>projectCreator -eclipse stocks -out
c:\code\stocks
Created directory c:\code\stocks\test
Created file c:\code\stocks\.project
```

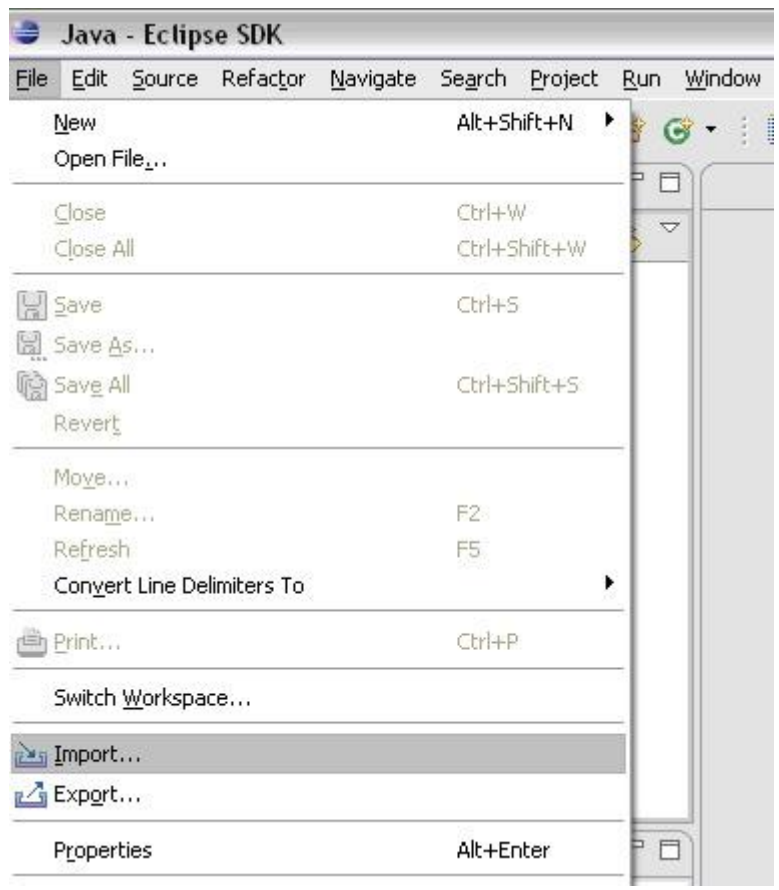
```
Created file c:\code\stocks\.classpath
```

projectCreator 为 Eclipse 项目创建了 .project 和 .classpath 文件，还甚至为单元测试创建了一个测试目录。现在您只需将新项目导入 Eclipse。

将项目导入 Eclipse

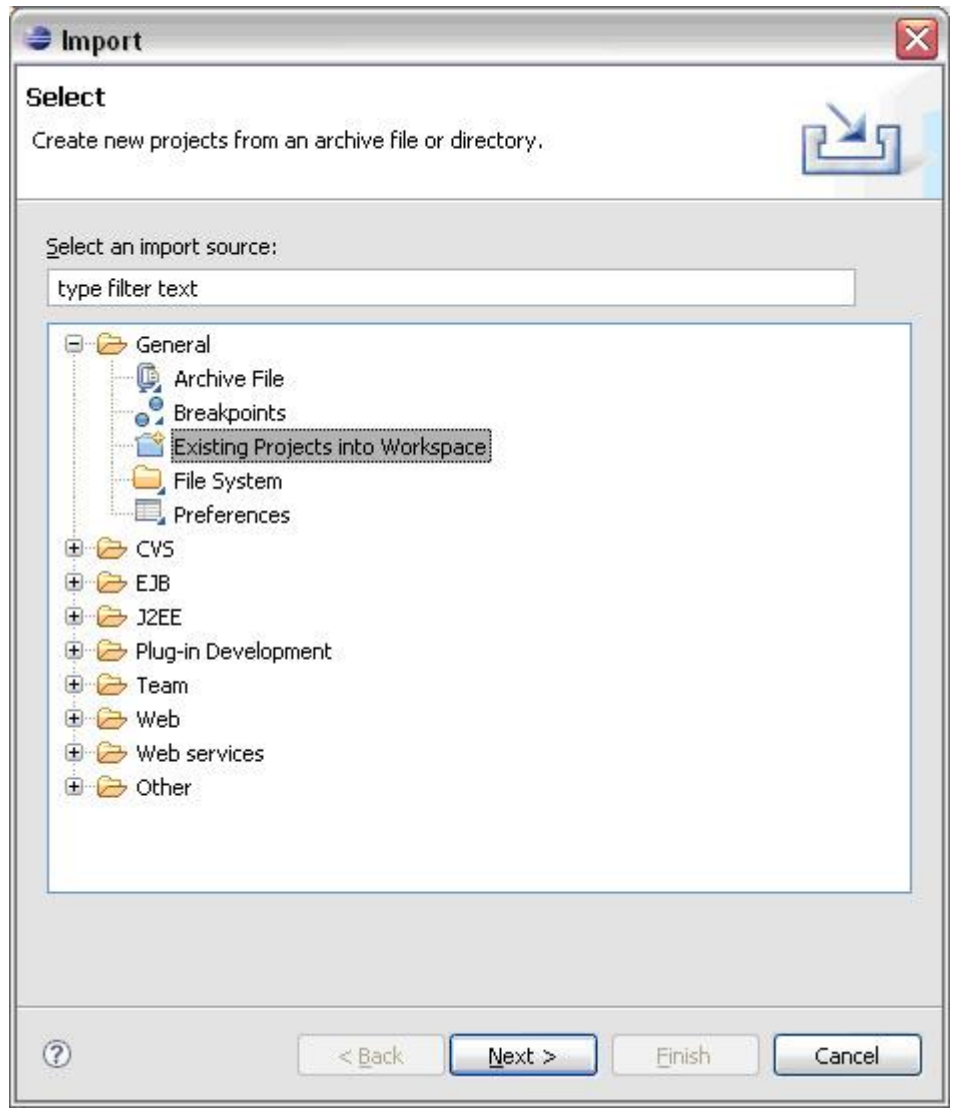
1. 打开 Eclipse，然后选择 **File > Import**，如图 2 所示：

图 2. 将项目导入 Eclipse



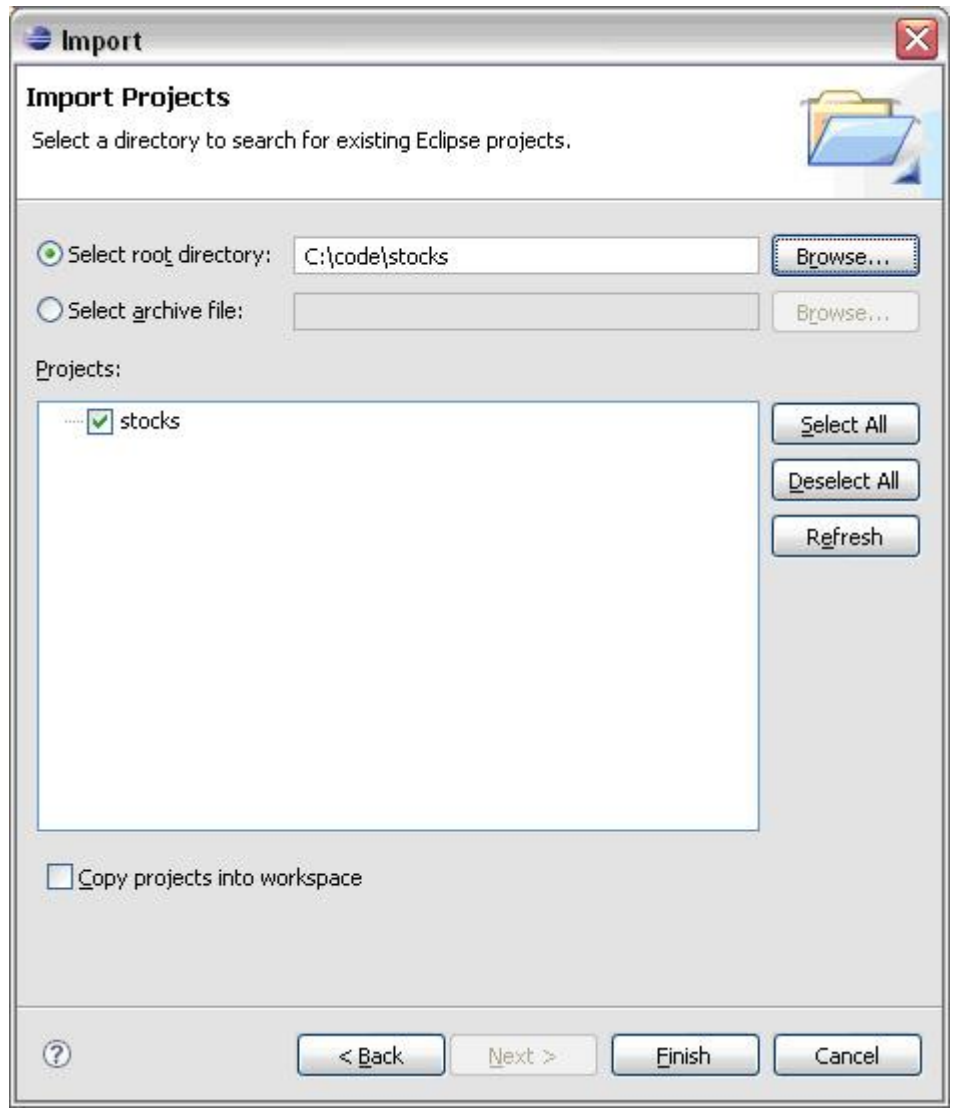
2. 选择 **Existing Projects into Workspace**，然后单击 **Next**，如图 3 所示：

图 3. 导入现有项目



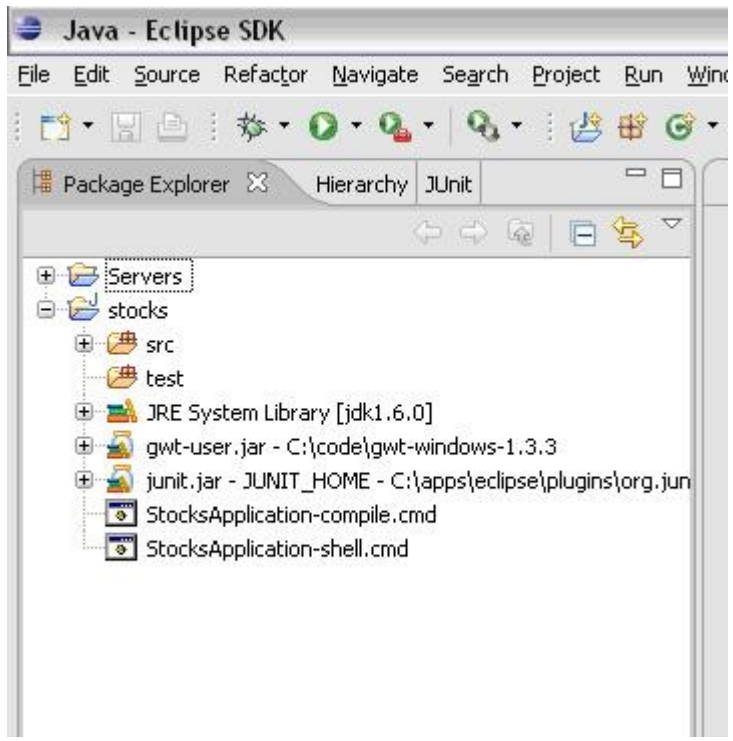
3. 输入指向先前创建的代码的路径。Eclipse 应当会检测到该项目，如图 4 所示：

图 4. 选择要导入的项目



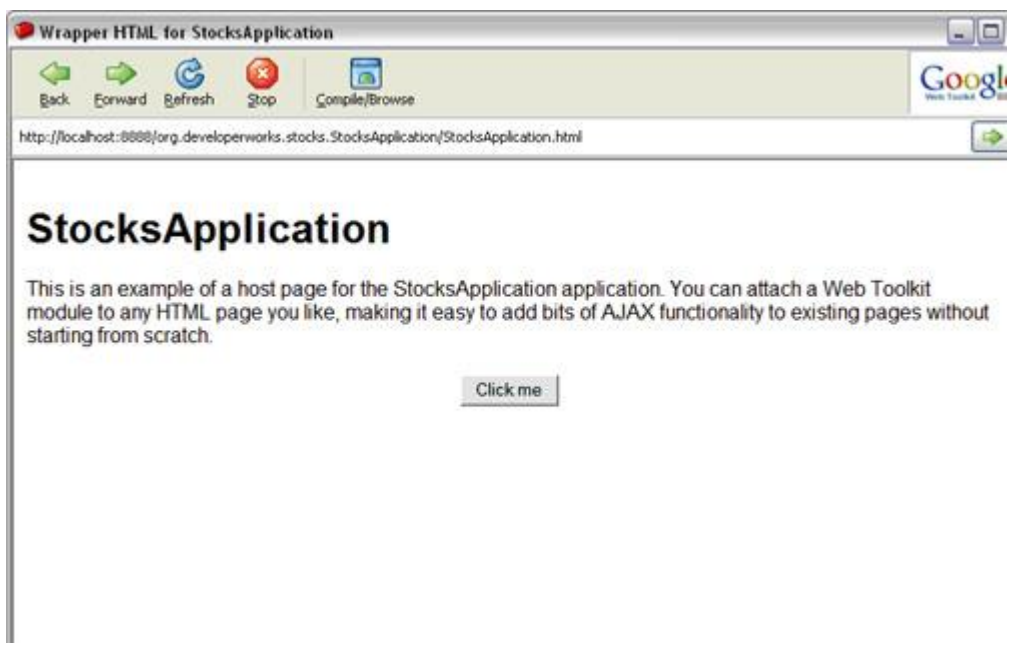
4. 单击 **Finish**，然后 Eclipse Package Explorer 应当显示该项目（参见图 5）。

图 5. 项目内容



现在您已经创建了基本项目。继续前进并动手尝试。双击 **StocksApplication-shell** 可执行文件，然后您应当会看到类似图 6 所示的内容。

图 6. 处于托管模式的默认应用程序



您尚未编写任何代码，但是已经有了一个运行应用程序。GWT 为您提供了大量样板代码。现在您已经准备好开始为应用程序编写代码。

开发应用程序

应用程序开发应当全都是编写用于实现应用程序业务逻辑的代码。不幸的是，开发人员经常需要多花时间创建应用程序的信息管道。对于该项任务，GWT 将为您执行大部分操作，因此您可以集中注意力去编写特定于应用程序的逻辑。您已经创建了 GWT 创建启用 Ajax 的应用程序所需的大量基础工件，并且已经为应用程序设置了 Eclipse 项目。在此部分中，您将集中处理业务逻辑。

股票服务

现在该创建股票服务了，该服务将检索给定代码的股票信息。需要为服务创建一个接口并且在客户机软件包（GWT 中的另一个重要约定）中声明它。参见清单 3 中的 StocksService 接口的代码。

清单 3. StocksService.java

```
package org.developerworks.stocks.client;

import com.google.gwt.user.client.rpc.RemoteService;

public interface StocksService extends RemoteService {
        public Stock getQuote(String symbol);
}
```

它是一个十分简单的接口，但是有一些重要的注意事项。首先，需要扩展 GWT RemoteService 接口。这只是一个让 GWT 知道此接口将用于 Ajax 调用的标记接口。接下来，需要注意您的服务将返回一个名为 Stock 的类。您接下来需要定义该类。

回页首

股票数据结构

Stock 类是股票数据的简单数据结构。参见清单 4。

清单 4. Stock.java

```
package org.developerworks.stocks.client;

import com.google.gwt.user.client.rpc.IsSerializable;

public class Stock implements IsSerializable {
        private String symbol;
```

```
    private String companyName;
    private double price;
    private double change;


    public String getCompanyName() {
        return companyName;
    }
    public void setCompanyName(String companyName) {
        this.companyName = companyName;
    }
    public double getPrice() {
        return price;
    }
    public void setPrice(double price) {
        this.price = price;
    }
    public String getSymbol() {
        return symbol;
    }
    public void setSymbol(String symbol) {
        this.symbol = symbol;
    }
```

```
    public double getChange() {
        return change;
    }
    public void setChange(double change) {
        this.change = change;
    }
    public String toString(){
        StringBuffer str = new StringBuffer();
        str.append(this.companyName);
        str.append(": ");
        str.append(this.symbol);
        str.append(" $");
        str.append(this.price);
        str.append(" ");
        str.append(this.change);
        return str.toString();
    }
}
```

这是一个十分典型的 JavaBeans 组件。值得注意的是它将实现 IsSerializable 接口，另一个 GWT 标记接口。它类似于 Java 的 java.io.Serilizable 接口，不同之处是 IsSerializable 表示可以被序列化为 JavaScript 的对象。因此它比 Java 的 Serializable 接口更严格一些。既然您已经定义了服务接口，那么就需要实现它。

回页首

实现股票服务

如前述，GWT 规定将服务器端代码放入服务器软件包中。这是放置服务接口的实现类的位置，如清单 5 所示：

清单 5. StocksServiceImpl.java

```
package org.developerworks.stocks.server;


import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.net.MalformedURLException;

import java.net.URL;

import java.net.URLConnection;
```

```
import org.developerworks.stocks.client.Stock;

import org.developerworks.stocks.client.StocksService;


import com.google.gwt.user.client.rpc.InvocationException;

import com.google.gwt.user.server.rpc.RemoteServiceServlet;


public class StocksServiceImpl extends RemoteServiceServlet implements

StocksService {


        private static final String YAHOO_QUOTE_URL =

"http://quote.yahoo.com/d/quotes.csv?s=:symbol&f=sl1d1t1c1ohgvj1pp2o

wern&e=.csv";


        public Stock getQuote(String symbol) {

                String urlStr = YAHOO_QUOTE_URL.replace(":symbol", symbol);

                try {

                        URL url = new URL(urlStr);

                        URLConnection conn = url.openConnection();

                        conn.connect();

                   InputStream is = conn.getInputStream();

                   InputStreamReader isr = new InputStreamReader(is);

                   BufferedReader br = new BufferedReader(isr);
```

```
                String line = null;
                Stock stock =new Stock();
                while ( (line = br.readLine()) != null){
                      String[] stockArray = line.split(",");
                      for (int i=0;i<stockArray.length;i++){
                            stock.setCompanyName(stockArray[16].replaceAll("\"",
"""));
                            stock.setSymbol(stockArray[0].replaceAll("\"", ""));
                            stock.setPrice(new Double(stockArray[1]));
                            stock.setChange(new Double(stockArray[4]));
                      }
                       }
                       return stock;
                } catch (MalformedURLException e) {
                      throw new InvocationException("Something was wrong with
the URL",e);
                } catch (IOException e) {
                      throw new InvocationException("Exception reading data
from Yahoo",e);
                }
        }
}
```

这里需要注意几个问题。最重要的是，您的类将扩展 GWT 类 RemoteServiceServlet。这将促成您的实现类成为用于处理远程请求的 servlet。此外，注意您捕捉到了所有异常并重新抛出它们作为 GWT InvocationExceptions。抛出某种异常以便 servlet 创建错误响应十分重要。

最后，注意虽然服务将作为 servlet 运行，但是您尚未编写任何基于 servlet 的代码。您是否看到清单 5 中提及任何 HttpRequest 或 HttpResponse？

回页首

GWT 模块声明

您的服务将被部署为 servlet。您一定会想：“不需要一个指向此 servlet 的 URL 映射吗？”完全正确。当部署到 Geronimo 上时，您当然需要把 URL 映射到股票服务 servlet 上。您还需要让 GWT 知道此映射的信息。为此，需要编辑 StocksApplication.gwt.xml 文件（先前由 applicationCreator 可执行文件创建）并参见清单 6。

清单 6. StocksApplication.gwt.xml

```
<module>
```

```
        <!-- Inherit the core Web Toolkit stuff.                        -->
        <inherits name='com.google.gwt.user.User'/>


        <!-- Specify the app entry point class.                         -->
        <entry-point
class='org.developerworks.stocks.client.StocksApplication'/>


          <servlet path="/stocksService" class=
"org.developerworks.stocks.server.StocksServiceImpl"/>
</module>
```

更改的惟一内容是添加 servlet 节点。inherits 和 entry-point 节点是 applicationCreator 所提供的样板代码。

回页首

异步版本的股票服务接口

完成服务还有最后一个与 GWT 相关的操作需要执行。您需要异步版本的服务。如果您熟悉 RPC 样式编程，则这只意味着接口中操作需要具有 void 返回类型。如果没有返回类型如何传递回股票信息？GWT 将为名为 AsyncCallback 的响应数据提供打包类。

这里要多使用一个 GWT 约定 您将通过把 Async 附加到接口名称后来命名异步接口(例如 ,StocksServiceAsync)。清单 7 将显示接口声明。

清单 7. StocksServiceAsync.java

```
package org.developerworks.stocks.client;

import com.google.gwt.user.client.rpc.AsyncCallback;

public interface StocksServiceAsync {
        public void getQuote(String symbol, AsyncCallback callback);
}
```

该接口类似于您的 StocksService 接口。您不会实际实现此接口。它被编译成 JavaScript 并用作 GWT 所生成的 Ajax 调用的一部分。

回页首

股票应用程序

您的服务已经准备好被调用，因此您需要做的全部操作就是创建应用程序。在这里，您将用 Java 语言编写应用程序。在许多方面，它都看起来像是一个 Swing 或 Standard Widget Toolkit (SWT) 应用程序。这就是 GWT 的原理。只需记得所有应用程序代码实际被编译成 JavaScript 并部署到 Web 浏览器中。因此，请查看清单 8 中的应用程序代码。

清单 8. StocksApplication.java

```
package org.developerworks.stocks.client;

import com.google.gwt.core.client.EntryPoint;

import com.google.gwt.core.client.GWT;

import com.google.gwt.user.client.rpc.AsyncCallback;

import com.google.gwt.user.client.rpc.ServiceDefTarget;

import com.google.gwt.user.client.ui.Button;

import com.google.gwt.user.client.ui.ClickListener;

import

com.google.gwt.user.client.ui.KeyboardListenerAdapter;

import com.google.gwt.user.client.ui.Label;
```

```
import com.google.gwt.user.client.ui.RootPanel;

import com.google.gwt.user.client.ui.TextBox;

import com.google.gwt.user.client.ui.Widget;

/**

* Entry point classes define <code>onModuleLoad()</code>.

*/

public class StocksApplication implements EntryPoint {

StocksService service;

/**

* This is the entry point method.

*/

public void onModuleLoad() {

// declare the UI elements used in the app

final Button button = new Button("Get Quote");

final Label stockLabel = new Label();

final TextBox textBox = new TextBox();



// get a handle on the async version of our service

final StocksServiceAsync stocksService =

(StocksServiceAsync)

GWT.create(StocksService.class);

ServiceDefTarget endpoint = (ServiceDefTarget)
```

```
stocksService;
String moduleRelativeURL = GWT.getModuleBaseURL() +
"stocksService";
endpoint.setServiceEntryPoint(moduleRelativeURL);


// create the response handler
final AsyncCallback callback = new AsyncCallback()
{
public void onFailure(Throwable caught) {
stockLabel.setText("Sorry there was an
error");
}
public void onSuccess(Object result) {
Stock stock = (Stock) result;
stockLabel.setText(stock.toString());
// show things as green or red depending on if the stock is up or
down
if (stock.getChange() > 0){
stockLabel.setStyleName("stockUp");
} else if (stock.getChange() < 0){
stockLabel.setStyleName("stockDown");
}
```

```
}
};

// create event listener for the button
button.addClickListener(new ClickListener() {
public void onClick(Widget sender) {
String symbol = textBox.getText();
stocksService.getQuote(symbol, callback);
}
});

// make hitting return the same as clicking the button
textBox.addKeyboardListener(new
KeyboardListenerAdapter() {
public void onKeyPress(Widget sender, char
keyCode, int modifiers){
if (keyCode == '\r'){
button.click();
}
}
}
);
```

```
// These are the elements that the widgets are being added to
RootPanel.get("quoteButton").add(button);
RootPanel.get("stockInfo").add(stockLabel);
RootPanel.get("quoteBox").add(textBox);
}
}
```

让我们浏览一下清单 8 中执行的操作。首先需要声明一些 UI 元素，包括文本框、按钮和标签。文本框是用户输入股票代码的位置。按钮将触发对服务的 Ajax 调用。并且标签用于显示股票信息。

股票应用程序 —— 部件

在 GWT（以及许多其他 UI 框架）中，诸如文本框、按钮和标签之类的 UI 元素通常都被引用为*部件*。GWT 附带了大量部件。您已经看到了一些简单部件，如 Button、TextBox 和 Label。有些较为复杂的部件，如 MenuBar、Table 和 Tree。在 GWT 中，还有一个称为*面板* 的 UI 组件，这是一个可以包含其他部件的部件。面板可以定义一个布局算法。例如，HorizontalPanel 将像其名称隐含的含义一样从左到右布置部件。

清单 8 中接下来的四行代码将通过使用实用程序库类 GWT 来创建一个指向异步版本的服务的句柄。最后两行将设置用于调用服务的 URL。记住，全部这段代码将被编译成 JavaScript！

清单 8 中的代码的下一部分将为来自服务的响应创建处理程序。记住响应与请求是异步的，因此您将创建一个用于处理响应的回调方法。将把异步类用于此回调方法。此类编程常用于诸如 Swing 和 JavaScript 之类的事件驱动的 UI 代码。此外，这段代码将被编译成 JavaScript，在这里，当 Ajax 请求被发送给服务时，回调对象将被作为*算符*（或函数对象）来传递。

股票应用程序 —— Ajax 调用

实际回调逻辑十分简单。有一些方法用于回调：onFailure 和 onSuccess。onFailure 表示它处理的是来自服务的异常（可抛出）。当然，这将成为任何异常的序列化版本（记住，那就是您为什么需要确保在实现服务时抛出所有异常）。它将简单地显示错误消息。

如果一切运行正常，将调用 onSuccess 方法。在那种情况下，需要返回一个 Stock 对象。只需显示关于股票的信息。在这里，有一个很奇特的逻辑部分。根据股票价格变动是涨还是跌来设定信息的样式。这将通过调用 stockLabel.setStyleName() 来执行。此调用将在股票上涨时用绿色显示信息，在股票下跌时用红色显示信息。

回页首

把应用程序与服务绑定在一起

接下来在 清单 8 中，将向按钮和文本框中添加事件侦听程序。对于任何事件驱动的 UI 框架，这是标准操作。您将再次使用异步类，并且使用 GWT 的一些方便的类。例如，ClickListener 类使您可以只需处理所关心的一个事件：单击按钮。同样地，您使用了 KeyboardListenerAdapter 来捕捉按键按下事件。您只寻找点击 **Return** 按钮的人员，然后把这个操作转换为单击按钮操作。

那么，单击按钮执行什么操作？您只需获得文本框中文本的值并调用服务。注意如何把回调作为参数传递给服务调用。如前述，当这一切被转换为 JavaScript 后，它将成为一个被传递给 Ajax 调用的算符。

清单 8 中的最后一部分将简单地把您已经创建的 UI 组件（文本框、按钮和标签）映射为 HTML 页面的 DOM 中的元素。您将简短地查看该 HTML 页面。

您已经编写了应用程序逻辑。它现在全都是 Java 代码，但是当运行应用程序时它将全都是 JavaScript。您需要创建视图（如清单 9 中所示），在本例中只是一个 HTML 页面。

清单 9. StocksApplication.html

```
<html>
	<head>
		<title>GWT Stocks Application</title>
		<style>
			#stockInfo{
				text-align: center;
				padding-top: 10px;
			}
			#quotes{
				text-align: center;
			}
			#quoteBox{
				padding-right: 5px;
			}
			.stockUp{
				color:green;
			}
```

```
        .stockDown{
            color:red;
        }
    </style>

    <!--                                                -->
    <!-- The module reference below is the link       -->
    <!-- between html and your Web Toolkit module    -->
    <!--                                                -->
    <meta name='gwt:module' content=
'org.developerworks.stocks.StocksApplication'>

    </head>
    <body>

        <!--                                                -->
        <!-- This script is required bootstrap stuff.     -->
        <!-- You can put it in the HEAD, but startup       -->
        <!-- is slightly faster if you include it here. -->
        <!--                                                -->
        <script language="javascript" src="gwt.js"></script>
```

```
        <!-- OPTIONAL: include this if you want history support
-->
        <iframe id="__gwt_historyFrame"
style="width:0;height:0;border:0"></iframe>

        <h1>Stocks Application</h1>

        <p>
            Enter in the symbol for a stock below.
        </p>


        <div id="quotes">
          <span id="quoteBox"></span>
          <span id="quoteButton"></span>
        </div>
        <div id="stockInfo"/>
    </body>
</html>
```

这十分简单。您将输入一些静态文本并创建用于放入 UI 组件中的应用程序代码的页面元素。您使用一些 CSS 代码来布置组件。还将创建两个用于使股票信息显示变成绿色或红色的样式。

回页首

关于浏览器历史记录

您可能会注意到的最后一个事项是 iframe 元素。这是一个可选元素，它将使 GWT 可以解决 Ajax 应用程序中常见的 Back 按钮问题。传统的 Web 应用程序拥有从一个页面指向另一个页面的链接。Ajax 应用程序通常可以在不使用实际页面链接（不指向任何位置的链接）的情况下完成同样的工作。一些 JavaScript 将简单地捕捉已经被单击的内容并向服务器发出某种异步请求。页面随后将根据来自服务器的响应更改。显示内容将被更改，就像在传统应用程序中一样。在传统应用程序中，您可以使用浏览器的 **Back** 按钮返回到上一页。这种机制在 Ajax 应用程序中不起作用，并且一些 Web 站点使用一些复杂的 JavaScript 来检测浏览器是否已被单击并随后恢复应用程序的先前状态。GWT 极大地简化了此操作。您需要做的全部操作就是通过实现 onHistoryChanged() 方法使应用程序代码实现 HistoryListener 接口。您随后将用 GWT 的 History 对象注册应用程序，以便知道浏览器历史记录。这将使 GWT 可以处理用户点击 Back 按钮。这是 GWT 的最常见特性之一，但是您将不会在本文的应用程序中使用它，因为没有任何链接。

运行和测试应用程序

现在已经开发了股票应用程序，您可能十分渴望运行并测试应用程序。如果您曾经从事过 Web 开发，则知道此阶段的任务通常涉及把应用程序部署到 Web 服务器上并在该 Web 服务器上运行该应用程序。使用 GWT，您可以把该步骤推迟到应用程序准备好被部署再执行。使用 GWT 的托管模式，您可以在不使用 Web 服务器的情况下运行和测试。托管模式将运行 GWTShell 应用程序，它是使用 SWT 编写的桌面应用程序，SWT 是用于构建 Eclipse 的同一项技术。托管模式将使用嵌入版本的 Tomcat。

在托管模式下编译和运行

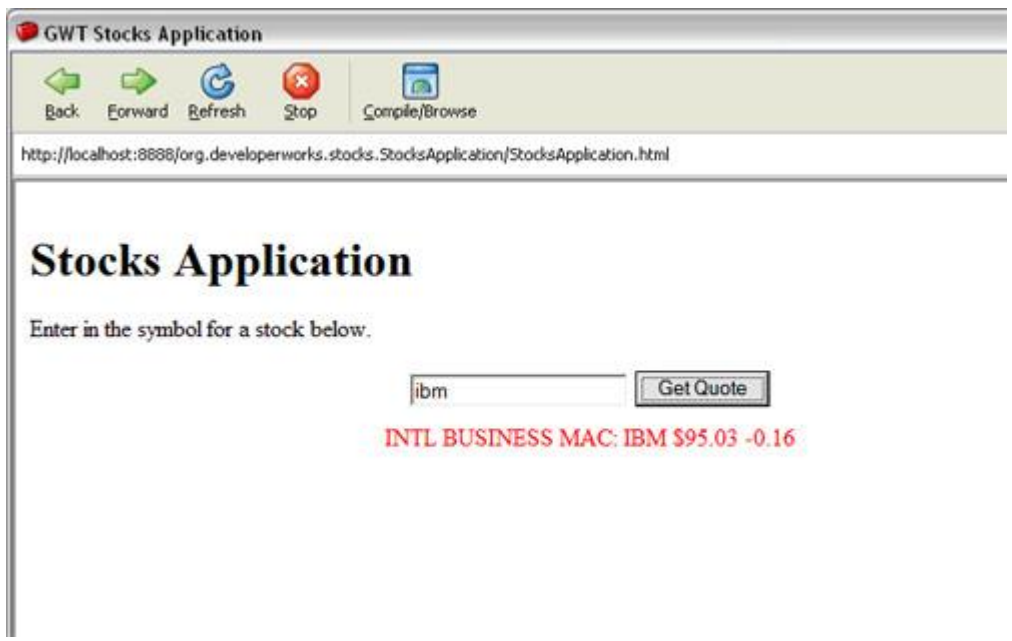
尝试应用程序。您可以简单地双击 **StocksApplication-shell** 可执行文件或从命令行调用它。您应当会看到类似图 7 中所示的内容。

图 7. 托管模式下的 StocksApplication

注意这里的 URL：

http://localhost:8888/org.developerworks.stocks.StocksApplication/StocksApplication.html。这是调试所需的 URL。尝试一下。输入股票代码。您可以按 **Enter** 或单击 **Get Quote** 按钮。通过任意一种方法您都应当会看到类似图 8 中所示的内容。

图 8. 显示信息的股票应用程序



如果输入无效的股票代码，您的服务将在尝试解析结果时抛出异常。GWT 将很好地处理这种情况，如图 9 所示：

图 9. 无效的股票代码将抛出异常

单击 **Compile/Browse** 按钮在 Web 浏览器中启动应用程序，如图 10 所示：

图 10. 运行 Web 浏览器的托管模式

您可以多在浏览器中测试该应用程序 ,甚至在其他浏览器中测试它以确保它运行正常。只需使用导航栏中看到的 URL。

构建 Web 应用程序

您已经看到使用 GWT 开发启用 Ajax 的应用程序是多么轻松。现在是时候把此应用程序转换为一个可以部署到 Geronimo 上的 Java Web 应用程序了。您需要做的全部操作就是添加 Geronimo 所需的基本 Web 应用程序元数据并把应用程序打包成一个 WAR 文件。

创建 web.xml

每个 Java Web 应用程序都需要有一个 web.xml 文件告诉容器如何部署应用程序。公共目录是放置 HTML 的位置，因此向该目录中添加一个 WEB-INF 文件夹。那将是放置 web.xml 文件的位置，如清单 10 中所示：

清单 10 中有一些必须注意的内容。首先 ,需要声明 stocksServiceServlet ,然后把 /stocksService URL 映射给它。接下来 ,把 StocksApplication.html 设为欢迎页面 ,因此在默认情况下 ,所有请求都会被发送给该页面。web.xml 文

件符合 Java Servlet 2.4 规范。但是 ,有一些 Servlet 2.4 规范不支持的内容需要添加 ,因此您需要有一个 Geronimo 部署计划。

清单 10. 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">
  <servlet>
                <servlet-name>stocksServiceServlet</servlet-name>
                <servlet-class>org.developerworks.
                        stocks.server.StocksServiceImpl</servlet-class>
    </servlet>
    <servlet-mapping>
                <servlet-name>stocksServiceServlet</servlet-name>
                 <url-pattern>/stocksService</url-pattern>
     </servlet-mapping>
    <welcome-file-list>
```

```
            <welcome-file>
                StocksApplication.html
            </welcome-file>
            </welcome-file-list>
</web-app>
```

回页首

创建 geronimo-web.xml 文件

您需要添加 geronimo-web.xml 文件，因此可以配置应用程序的根上下文。您将使用刚创建的 web.xml 文件把它放在 WEB-INF 目录中（参见清单 11）。

清单 11. geronimo-web.xml

```
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1">
    <environment>
```

```
        <moduleId>
            <artifactId>StocksApplication</artifactId>
        </moduleId>
    </environment>
    <context-root>/stocks</context-root>
</web-app>
```

这是十分简单的。把 /stocks 映射到应用程序，因此 http://*<domain>*/stocks 将访问应用程序。您无需把 /StocksApplication.html 放置在那里，因为将在 web.xml 中把该页面设为 welcome-file。

最后，将为应用程序提供一个 artifactId，以便您在使用 Geronimo Console 时可以轻松地识别它。您已经创建了 Web 应用程序所需的全部内容，现在只需将它打包。

回页首

创建 WAR 文件

您可以轻松地手动创建 WAR 文件，但是使用 Apache Ant 执行此操作将更简单且更具有可移植性。因此，您将为应用程序创建 Ant build.xml 文件（参见清单 12）。

清单 12. build.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<project name="StocksGWT" default="all" basedir=".">
  <description>
        The Stocks GWT build file
  </description>

        <property name="gwt.module" value="StocksApplication"/>
      <property name="GWT_HOME"
value="C:/code/gwt-windows-1.3.3"/>
       <property name="app_packge"
value="org.developerworks.stocks"/>

  <!-- set classpath -->
  <path id="project.class.path">
    <pathelement path="${java.class.path}/"/>
    <pathelement path="${GWT_HOME}/gwt-user.jar"/>
  </path>

  <target name="build" description="Compile src to bin">
```

```
    <mkdir dir="bin"/>
    <javac srcdir="src" destdir="bin" includes="**" debug="on"
debuglevel="lines,vars,source" source="1.5">
      <classpath refid="project.class.path"/>
    </javac>
  </target>

  <target name="jar" depends="build" description="Package up the
project as a jar">
    <mkdir dir="www/${gwt.module}/WEB-INF/lib"/>
    <jar destfile="www/${gwt.module}/WEB-INF/lib/stocks.jar">
      <fileset dir="bin">
        <include name="**/*.class"/>
      </fileset>
      <fileset dir="src">
        <include name="**"/>
      </fileset>
    </jar>
  </target>

   <target name="gwt" description="Compiles Java to JavaScript">
        <exec spawn="false"
```

```
executable="${gwt.module}-compile.cmd"/>
            <copy todir="www/${gwt.module}">
                <fileset dir="www/${app_packge}.${gwt.module}">
                    <include name="**"/>
                </fileset>
            </copy>
        </target>

    <target name="war" depends="jar,gwt" description="Builds the war
file">
        <mkdir dir="www/${gwt.module}/WEB-INF/lib"/>
        <copy file="${GWT_HOME}/gwt-servlet.jar"
                todir="www/${gwt.module}/WEB-INF/lib"/>
        <jar destfile="stocks.war" basedir="www/${gwt.module}"
duplicate="preserve"/>
    </target>

    <target name="clean">
        <!-- Delete the bin directory tree -->
        <delete dir="bin"/>
        <delete dir="www/${gwt.module}"/>
        <delete file="stocks.war"/>
```

```
  </target>

  <target name="all" depends="war"/>

</project>
```

build.xml 文件是十分典型的 Web 应用程序构建文件。您可能会注意到的一个与众不同的目标是 gwt 目标。这将调用 GWT 编译器，它将把客户机软件包中的 Java 代码编译成 JavaScript 并创建应用程序所需的其他工件。现在您已经准备好使用 Ant 构建 Web 应用程序。

回页首

构建 WAR 文件

您可以从 Eclipse 内运行 Ant，可以从命令行运行 Ant，因为默认情况下将调用 WAR 目标。当您运行 Ant 时，应当会看到类似清单 13 中所示的输出。

清单 13. Ant 输出

```
Buildfile: C:\code\stocks\build.xml

build:

jar:

      [mkdir] Created dir:
C:\code\stocks\www\StocksApplication\WEB-INF\lib

        [jar] Building jar:
C:\code\stocks\www\StocksApplication\WEB-INF\lib\stocks.jar

gwt:

       [exec] Output will be written into
C:\code\stocks\www\org.developerworks.stocks.
                StocksApplication

       [exec] Copying all files found on public path

       [exec] Compilation succeeded

       [copy] Copying 17 files to C:\code\stocks\www\StocksApplication

war:

       [copy] Copying 1 file to
C:\code\stocks\www\StocksApplication\WEB-INF\lib

        [jar] Building jar: C:\code\stocks\stocks.war

BUILD SUCCESSFUL

Total time: 4 seconds
```

Ant 输出显示它已经为 WAR 文件创建了结构并随后把所有代码和生成的代码打包到 WAR 文件中。注意生成的 WAR 文件的位置，因为现在您将要把它部署到 Geronimo 中。

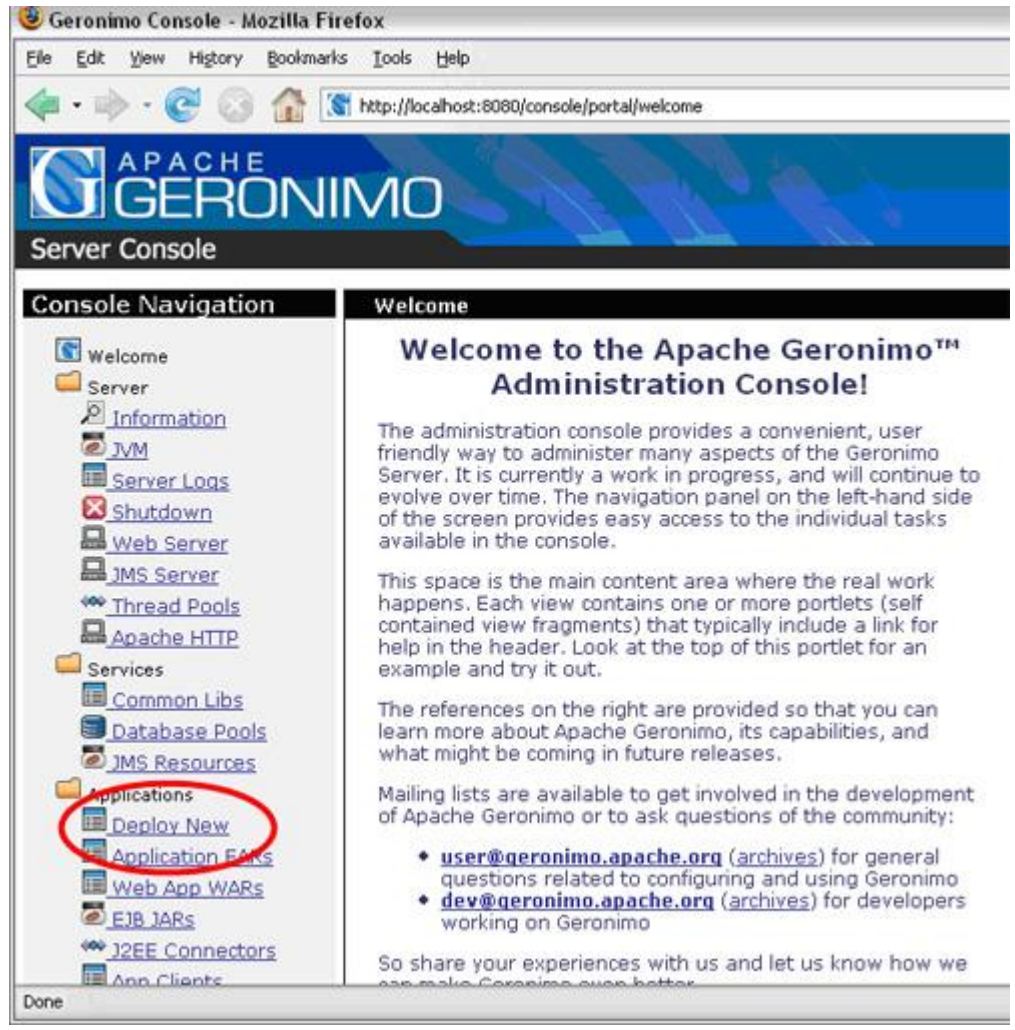
回页首

部署到 Geronimo 中

您已经构建了 Web 应用程序，因此现在只需把它部署到 Geronimo 中：

1. 启动 Geronimo，并打开其 Console，通常可以在 http://localhost:8080/console 找到它，如图 11 所示：

图 11. Geronimo Console



2. 选择 **Applications > Deploy New**，并浏览到先前为 Web 应用程序创建的 WAR 文件。

3. 单击 **Install**，如图 12 所示：

图 12. 选择要安装到 Geronimo 中的 Stocks WAR 文件

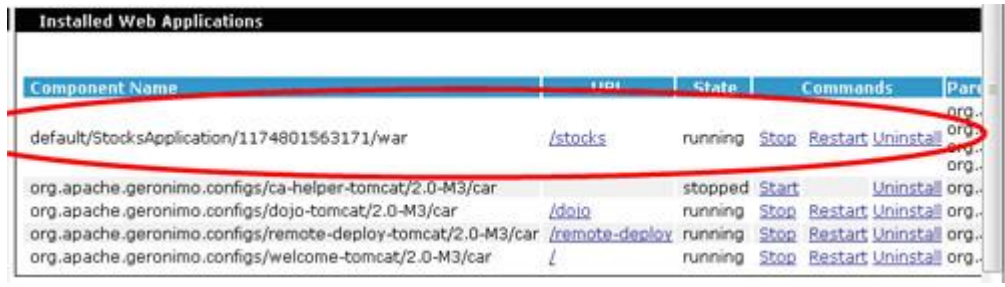
4. 您应当会看到指示 Web 应用程序已被成功安装和启动的消息。选择 **Applications > Web App WARs** 来检验此结果，如图 13 所示：

图 13. 股票应用程序已成功部署到 Geronimo 中

这样做将打开已经安装到 Geronimo 上的 Web 应用程序列表。您应当会在列表中看到股票应用程序。

5. 单击应用程序的 URL 来启动它，如图 14 所示：

图 14. 已经安装到 Geronimo 上的 Web 应用程序

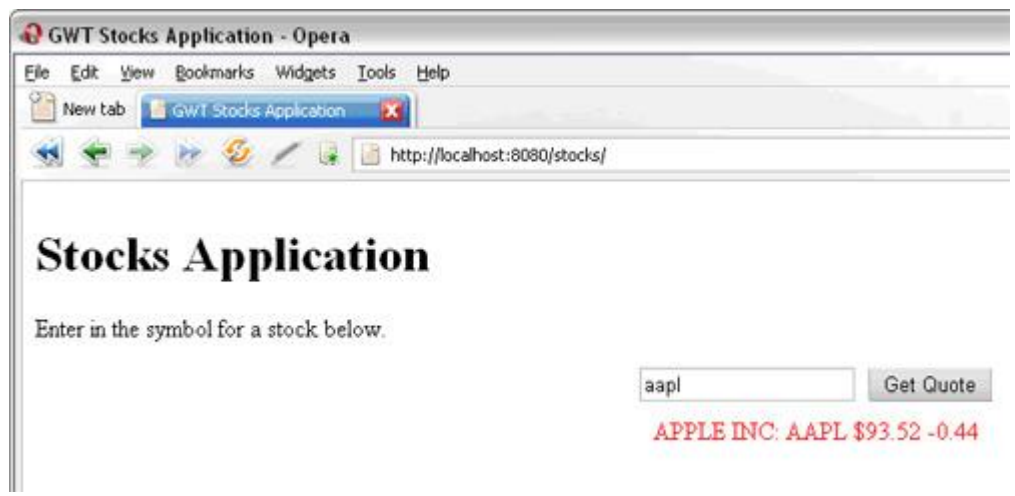


这样做应当在 Web 浏览器中打开应用程序，并且它应当像在托管模式下一样运行，如图 15 所示：

图 15. 运行在 Geronimo 上的股票应用程序

6. 您可以在各种浏览器中装入应用程序以确认它可在其中运行。例如，图 16 将展示它正在 Opera 浏览器中运行。

图 16. Opera 中的股票应用程序



您的应用程序现在已经在 Geronimo 上启动并运行。您已经编写了一个启用 Ajax 的 Web 应用程序，而没有编写任何 JSP、servlet 或 JavaScript，一切都要感谢 GWT！

级别： 中级

Michael Galpin (mike.sr@gmail.com), 开发人员, Adomo, Inc.

2007 年 8 月 02 日

在本系列教程的 第一部分 中，您已经了解了如何使用 Google Web Toolkit (GWT) 快速构建启用 Asynchronous JavaScript + XML (Ajax) 的 Web 应用程序并将其部署到 Apache Geronimo 中。此部分是两部分系列教程的第 2 部分，将把更多功能添加到在第 1 部分教程中已经构建的应用程序中。通过管理对后端数据库的访问来利用 Geronimo 把新功能添加到应用程序中。然后使用 GWT 添加更加动态的功能并与服务所提供的新功能轻松

地集成。此外，查看 GWT 的一些 Dynamic HTML (DHTML) 功能并使用 GWT 应用程序内的原生 JavaScript。

开始之前

关于本系列

启用 Ajax 的 Web 应用程序最近的流行程度令人难以置信。Ajax 使 Web 应用程序行为与桌面应用程序极为相似。启用 Ajax 的 Web 应用程序比以前生成的 Web 应用程序提供更优秀的交互性和功能性。并且 Geronimo 将为构建启用 Ajax 的 Web 应用程序提供优秀平台。

但是，构建启用 Ajax 的 Web 应用程序比构建传统的 Web 应用程序难得多。它将涉及大量 JavaScript 和 Dynamic HTML (DHTML) 开发。不同浏览器甚至同一浏览器的不同版本之间的变化使事情进一步复杂化。GWT 是众多 Ajax 框架之一，它可以使您更轻松地完成工作，可以使用新颖的方法用 Java 语言编写所有代码并为您生成全部 JavaScript。本系列中的两个教程将涵盖以下主题：

- **第 1 部分** 将主要介绍如何开始使用 GWT 并创建简单的启用 Ajax 的 Web 应用程序以检索股票报价。
- 此部分是 **第 2 部分**，将向您展示如何利用 GWT 和 Geronimo 使股票报价应用程序更高级以及如何把它改造成股票投资应用程序。

回页首

关于本教程

在本系列教程的 第一部分 中，您构建了启用 Ajax 的 Web 应用程序。现在将扩展此应用程序并使用更多企业特

性，如数据库访问。查看 GWT 是怎样使用 Ajax 让您可以轻松地向服务器发送消息，以及 Geronimo 怎样使服务器端应用程序可以轻松地异步更新数据库。

在本教程中，您将把股票应用程序改造成具有以下功能的投资管理应用程序：

> 查看 Ajax 资源中心，这是您的一站式 Ajax 编程模型信息中心，包括文章、教程、论坛、blog、wiki、事件和新闻。这里将介绍所有最新信息。

- 允许用户获得多支股票的报价，从而进行投资。
- 通过给用户提供用户名让用户保存投资。
- 每次询价时都把股票添加到用户的投资中。
- 使用 Geronimo 创建指向该数据库的连接池，然后使用 JDBC 读取数据库中的数据以及向数据库中写入数据。
- 在关系数据库中保存用户的名称和投资的股票。

回页首

先决条件

本教程介绍的是如何构建 Java Web 应用程序，但您只需要了解 Java servlet 的基础知识。GWT 将利用许多常见于其他技术的原理，例如布局管理、事件系统和远程过程调用（RPC），因此预先了解这些原理将使您可以轻松地掌握 GWT。本教程将使用 SQL 和 Java 数据库连接（Java Database Connectivity）来处理关系数据库，因此稍微熟悉这些工具会十分有帮助。

系统要求

开始之前，需要使用以下软件设置开发环境：

- 配有 Tomcat 的 Geronimo 2.0 —— 本教程中的样例 Web 应用程序是使用配有 Tomcat 的 Geronimo 构建的，但是它应当也可以与配有 Jetty 的 Geronimo 结合使用，因为所有内容都是标准的 Java Platform, Enterprise Edition (Java EE)。本教程将使用 Geronimo 2.0 (M3)，但是也应当可以使用早期版本的 Geronimo。
- Java 5 或 Java 6 —— 样例 Web 应用程序将使用泛型和注释。它是使用 Java 6 开发的 ,但是也针对 Java 5 进行过测试。
- JSTL 1.1 的 Apache Jakarta 实现 —— Geronimo 1.1 是经过认证的 J2EE 1.4 实现，因此您需要使用 JSTL 1.1。
- Google Web Toolkit —— 本教程的所有内容都与 GWT 有关；请下载本教程所需的 GWT 1.3.3。
- Eclipse —— 样例 Web 应用程序是使用 Eclipse 构建的，正如您将从屏幕捉图中看到的一样。GWT 附带了一个方便的命令行工具以创建骨架 Eclipse 项目。如果没有使用 Eclipse，构建 GWT 应用程序也不会很难，但是 Eclipse 将提供一个优秀的调试程序可以调试 GWT 应用程序。
- MySQL 5.0 —— 样例应用程序将使用 MySQL 作为数据库。Enterprise Java 和 Geronimo 的优点之一就是可以轻松地断开并使用其他数据库。另一种优秀的备选方法是使用嵌入到 Geronimo 中的 Apache Derby。使用嵌入式 Derby 进行 2.0 开发时 Geronimo 会有些 bug，因此本教程将使用外部数据库。

有关下载和安装 GWT 的说明，请参阅本系列的 第 1 部分。

概览

在此部分中，您将：

- 查看股票投资应用程序的扩展功能。

- 设置数据库。
- 创建数据库连接池。
- 建立可用于运行在 Geronimo 上的任何应用程序的连接池。
- 把连接池添加到投资应用程序中。

股票投资应用程序

您在 第 1 部分 中构建的启用 Ajax 的 Web 应用程序接受了股票代码，然后使用了 Ajax 请求从服务器获得关于该支股票的信息。使用了 GWT 生成向服务器发起请求的客户端代码。当然，服务器是 Apache Geronimo，它运行着 Java Web 应用程序。

现在您将把股票应用程序改造成使用户可以获得多支股票的报价从而构建投资的投资管理应用程序。然后，用户可以通过提供用户名来保存投资。登录后，用户每次询价时，都把股票添加到用户的投资中。用户名和投资中保存的股票存储在关系数据库中。使用 Geronimo 创建指向该数据库的连接池，然后使用 JDBC 读取数据库中的数据以及向数据库中写入数据。GWT 将生成对服务器的所有 Ajax 回调。

首先需要做的是设置数据库。

回页首

设置数据库

JDBC 提供了常见数据库功能的通用接口。这将使应用程序几乎可以使用所有配有 JDBC 驱动程序的数据库。对于本应用程序，将使用 MySQL（有关下载信息，请参阅 系统要求）。它是一个十分受欢迎的数据库，因为它是开源的并且易于使用。

当数据库已经安装并运行后，需要为数据库创建一个模式。然后需要创建两张表。清单 1 中的 SQL 脚本将创建模式和表。

清单 1. 创建 DB 的 SQL 脚本

```
CREATE DATABASE STOCKS;
USE STOCKS;
CREATE TABLE USER (
    USER_ID INT NOT NULL AUTO_INCREMENT,
    NAME VARCHAR(20) NOT NULL,
    PRIMARY KEY(USER_ID)
);

CREATE TABLE PORTFOLIO (
    USER_ID INT NOT NULL,
    SYMBOL VARCHAR(5) NOT NULL,
    FOREIGN KEY (USER_ID) REFERENCES USER (USER_ID)
);
```

用户表只存储用户 ID 和用户名。如果您要尝试构建更健壮的应用程序，则需要添加密码字段、捕捉更多用户信息等。但是对于投资应用程序，只有名称就足够了。

投资表将存储与用户相关的多支股票。此外，更健壮的应用程序可能把股票标准化，以便当两个用户的投资中都有 IBM® 时，IBM 只显示一次。然后可以使用关联表。清单 1 中的非规范化模式较为简单并且足够用于您的应用程序。

既然您已经有了存储投资数据的位置，那么就需要有一种访问和修改此数据的方法。那就是引入 Geronimo 的位置。

回页首

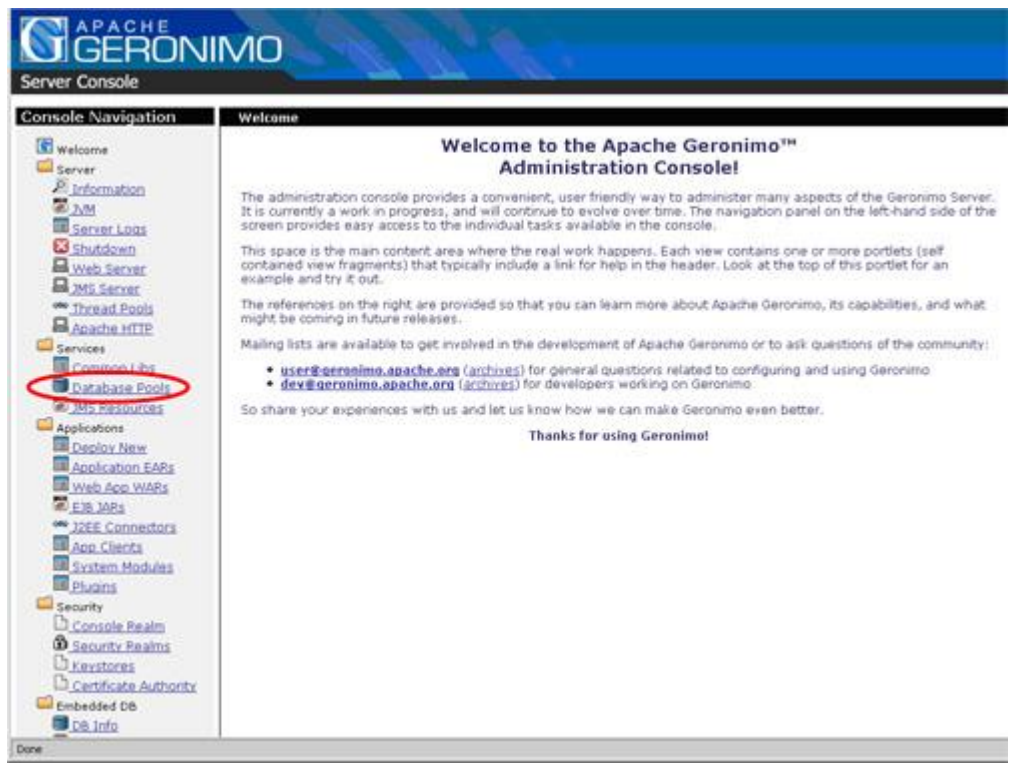
使用 Geronimo 数据库连接池

Geronimo 的优秀功能之一是您可以使用它创建数据库连接池。然后，该池可由运行在 Geronimo 上的任何应用程序使用。它将成为可以使用 Java 命名和目录接口（Java Naming and Directory Interface，JNDI）访问的 DataSource 对象。DataSource 随后可以给您提供 JDBC 连接，该连接将使您可以查询数据库并用新数据更新数据库。

要创建数据库连接池，请使用 Geronimo Console。通常，您可以在 http://localost:8080 访问 Geronimo Console。

1. 从菜单中选择 **Services > Database Pools**，如图 1 所示：

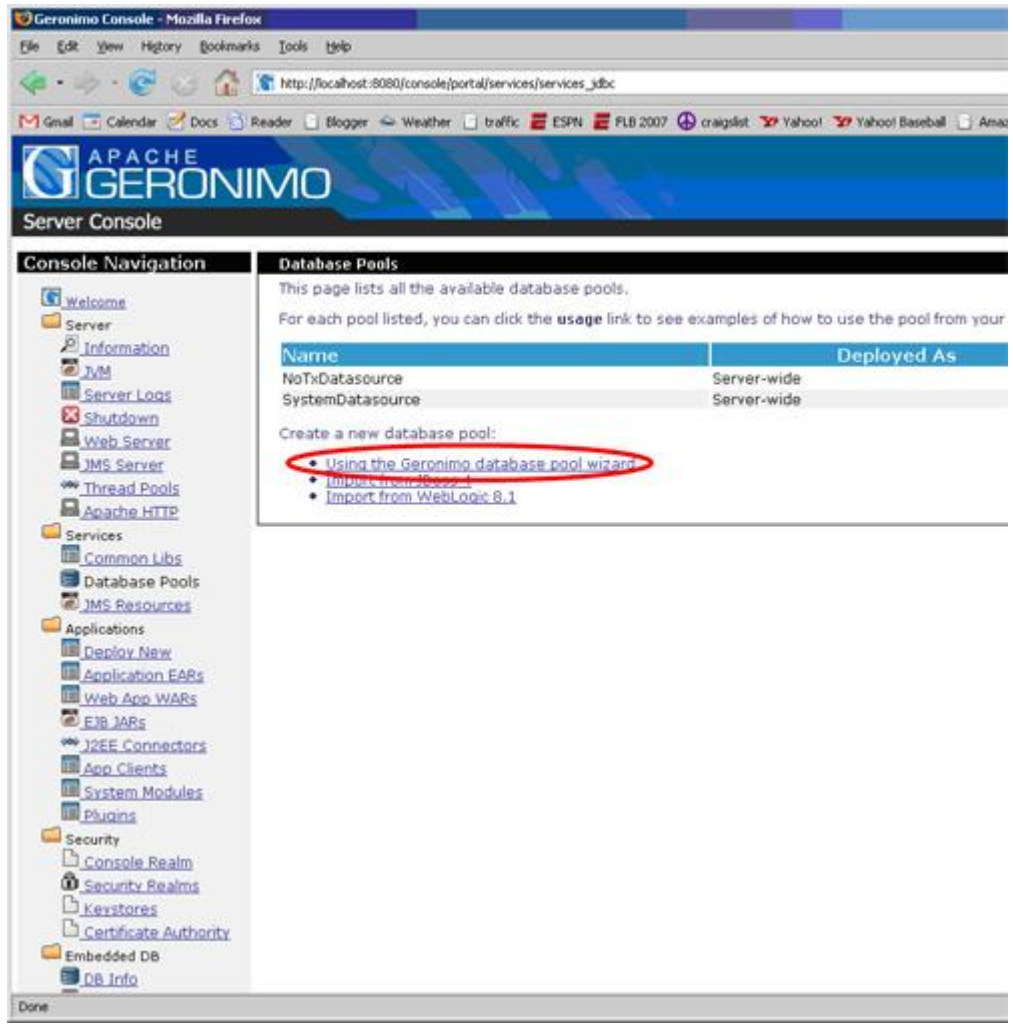
图 1. 从 Geronimo Console 选择数据库池



此操作应当打开当前数据库池的列表，并且您应当会看到 Geronimo 为内部服务使用的默认系统数据库池。

2. 通过选择 **Create a new database pool > Using the Geronimo database pool wizard** 创建一个新的数据库池，如图 2 所示：

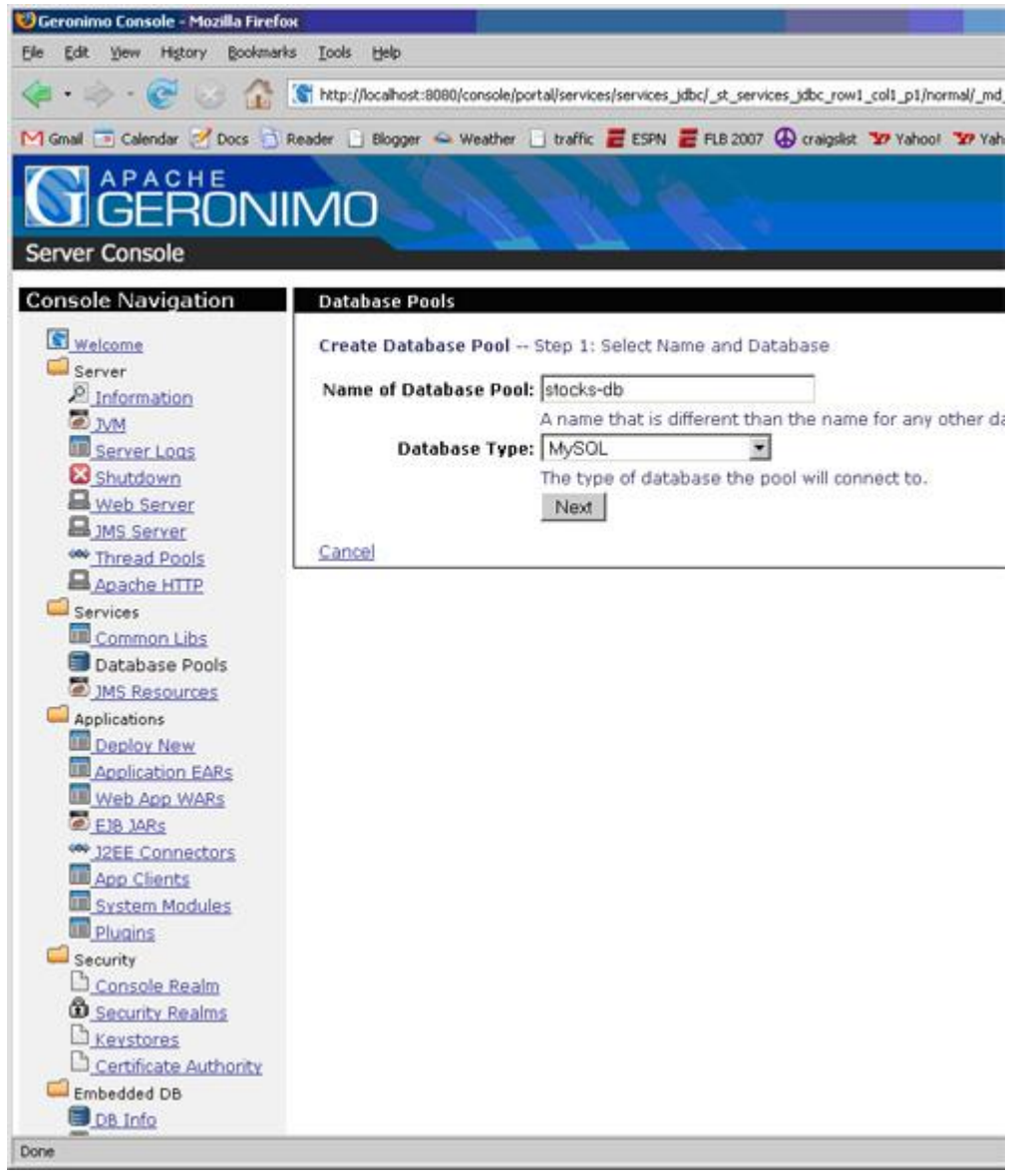
图 2. 选择数据库池向导



此操作应当打开 Create Database Pool 向导的 Step 1。

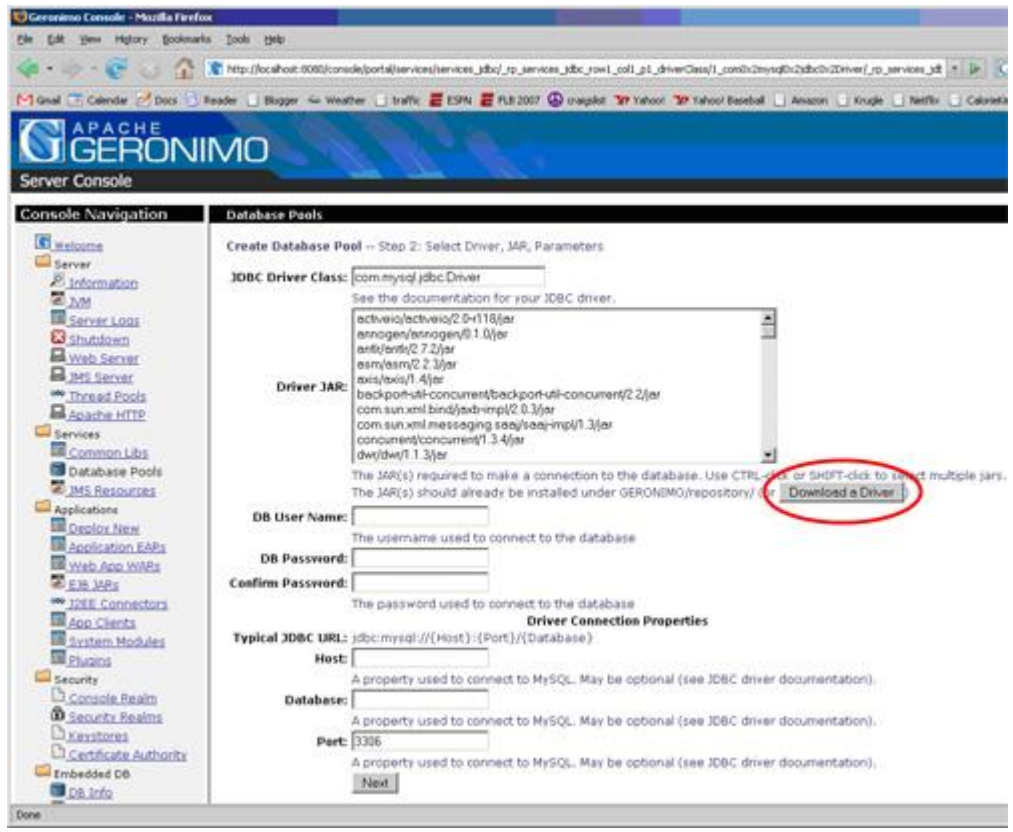
3. 调用数据库池 stocks-db，并从下拉式列表中选择数据库的类型，如图 3 所示：

图 3. Create Database Pool 向导的 Step 1



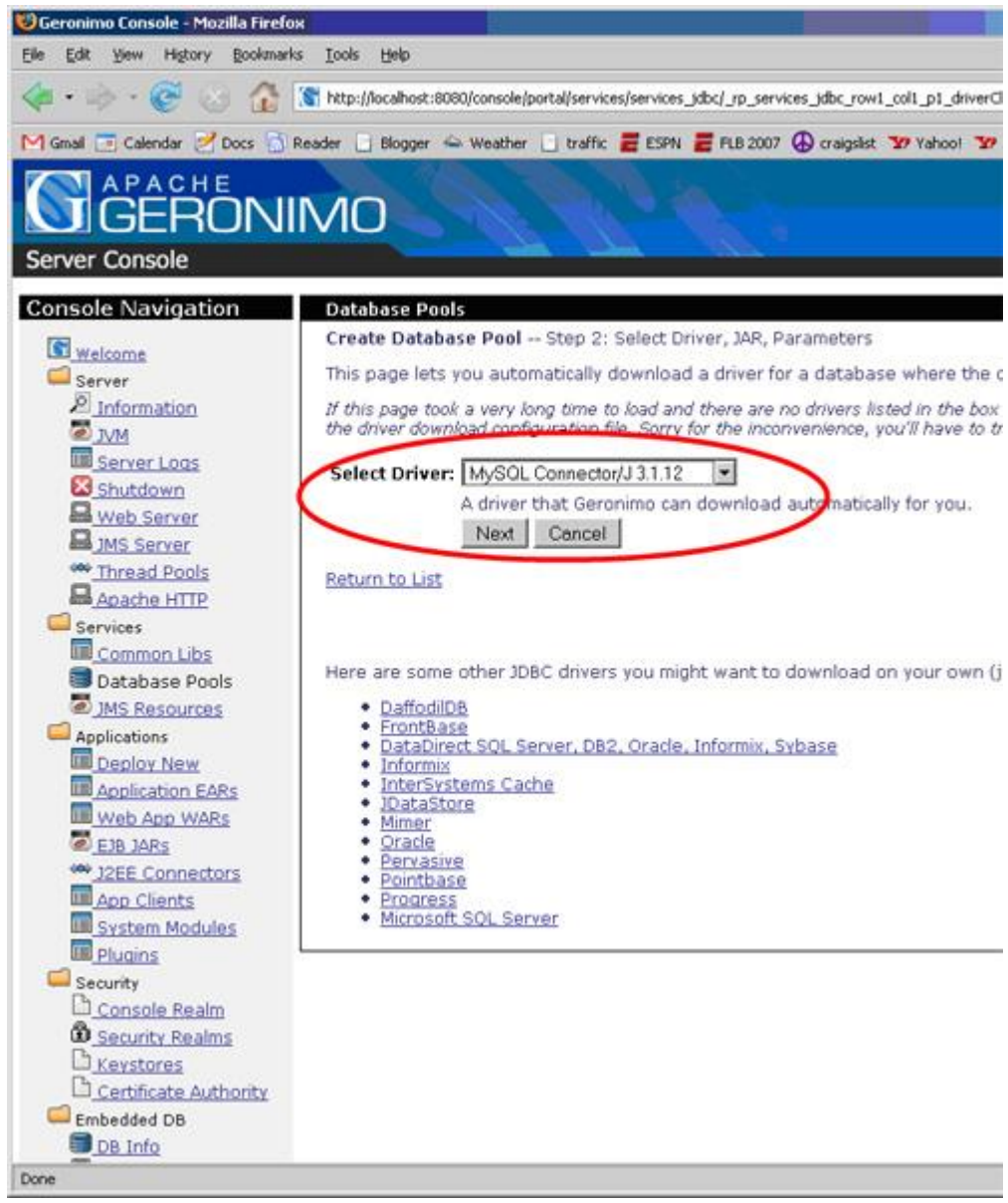
4. 单击 **Next** 按钮将打开向导中的 Step 2：Select Driver, JAR, Parameters 面板。JDBC Driver Class 应当已被填好。
5. 您需要有 JAR 文件与驱动程序结合使用。可以从 MySQL 下载，但是 Geronimo 实际上可以为您完成此操作。选择 **Download a Driver** 按钮，如图 4 所示，这将打开 Select Driver 界面。

图 4. 向导的 Step 2: Select Driver, JAR, Parameters



6. 从列表中选择最新的 MySQL JDBC 驱动程序，然后单击 **Next**，如图 5 所示：

图 5. 选择 MySQL JDBC 驱动程序

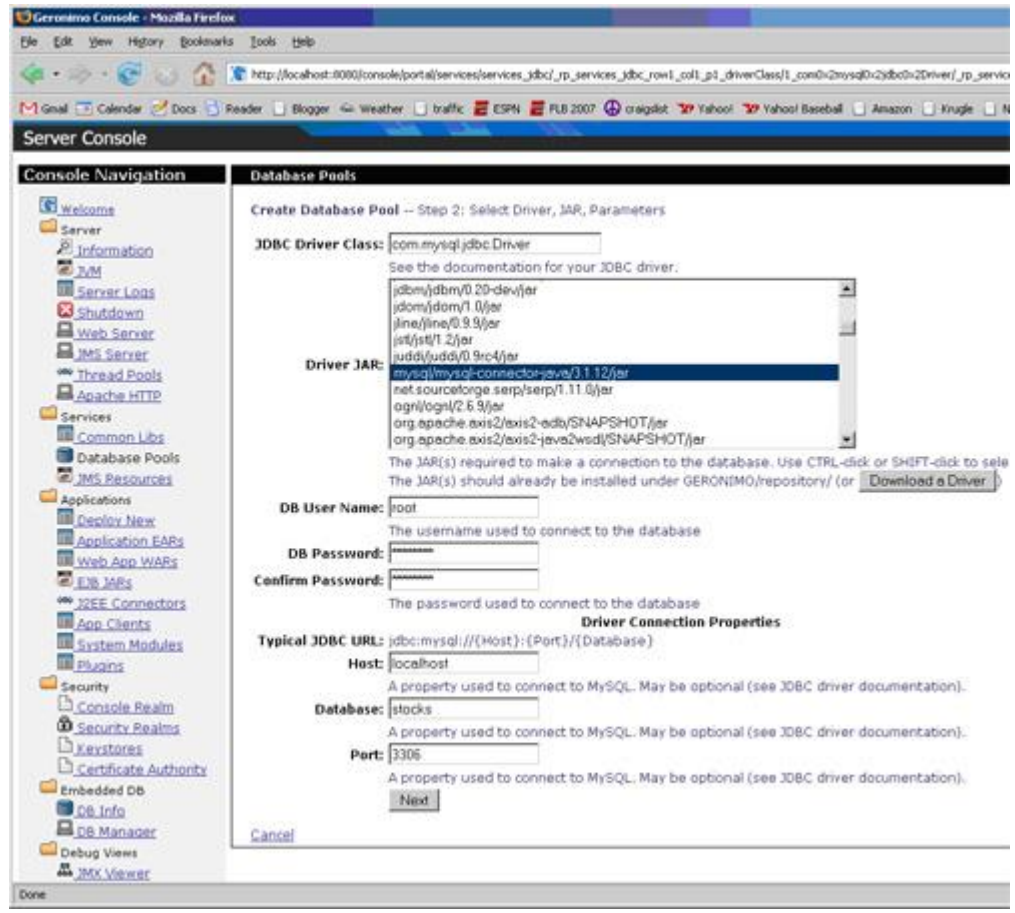


7. 此操作将打开 Driver Download 屏幕，该屏幕将向您展示下载 MySQL JDBC 驱动程序的进度，如图 6 所示：

图 6. 下载 MySQL JDBC 驱动程序

8. 下载完成后，您将被带回到 Select Driver, JAR, Parameters 界面。在 Driver JAR 字段中，您刚下载的 JAR 文件应当已被选中。现在填写关于数据库的其余信息，如图 7 所示：

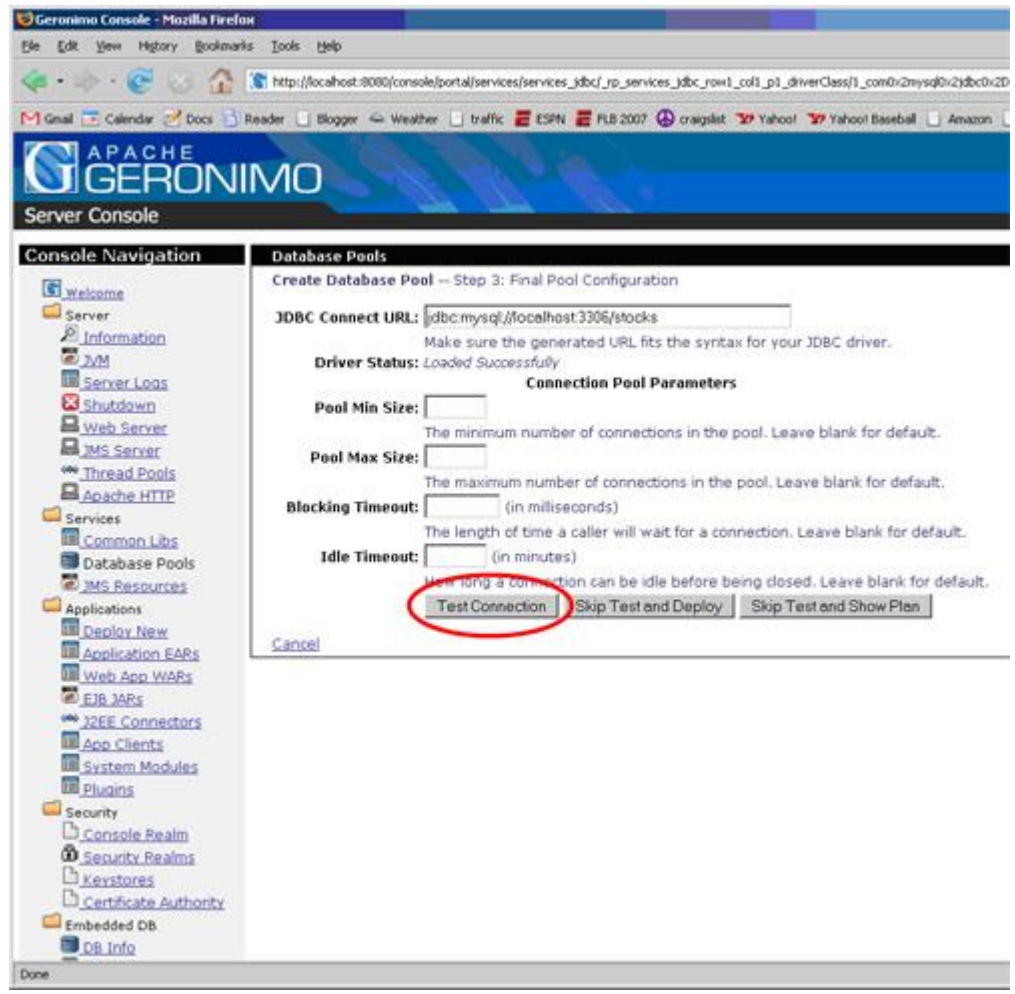
图 7. 填写驱动程序、JAR 和参数



9. 单击 **Next** 将打开向导中的 Step 3。

10.单击 **Test Connection** 按钮，如图 8 所示：

图 8. Create Database Pool 向导中的 Step 3

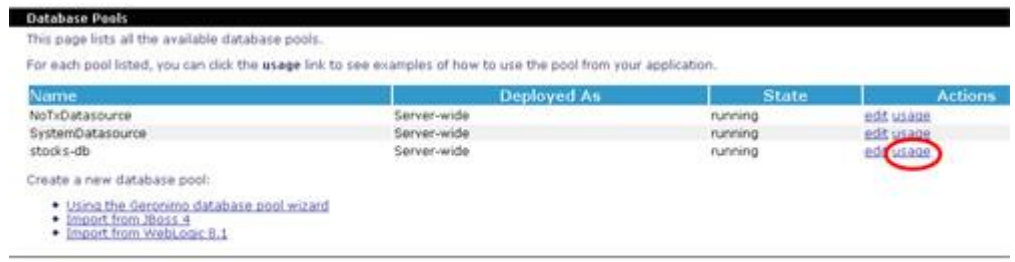


11.希望测试成功。如果成功，则单击 **Deploy** 按钮，如图 9 所示：

图 9. 测试连接

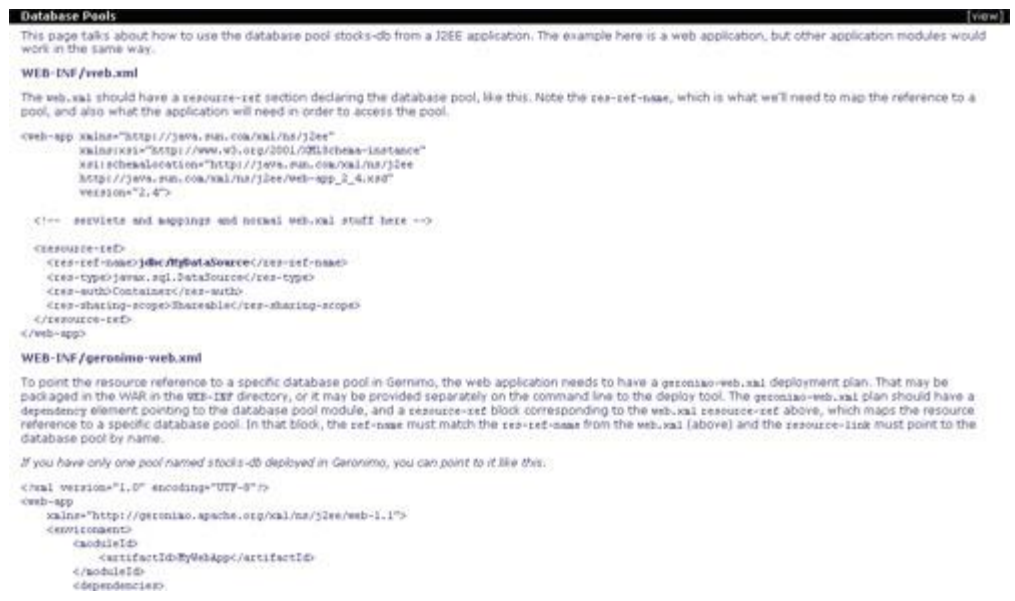
12.在部署了数据库池之后，您将会被带回到数据库池的列表。您应当会看到新的 stocks-db 池。单击新数据库池的 **usage** 链接，如图 10 所示：

图 10. 新数据库池



此操作应当会打开 usage 屏幕（参见图 11），该屏幕中包含将新数据库池添加到任何应用程序（包括新的投资应用程序）中所需的所有信息。

图 11. 数据库池的使用



正如您可以从 usage 屏幕中看到的 ,将像 web.xml 文件中的任何其他 DataSource 对象一样引用数据库池。不过，您可以比 Geronimo 的典型 JNDI DataSource 对象更有效地对数据库池进行管理。Geronimo 将把数据库池部署为连接器，并且将把连接器映射到 geronimo-web.xml 文件中的 DataSource 对象。这两个文件将在下一个部分中详细介绍，在该部分中您将把 DataSource 对象添加到应用程序中。

回页首

把 DataSource 对象添加到应用程序中

Geronimo 使您可以轻松地为数据库创建数据库连接池。现在数据库连接池可用于运行在 Geronimo 上的任何应用程序。您需要做的全部操作就是将其添加到投资应用程序中。首先，将把它添加到 geronimo-web.xml 文件中，如清单 2 所示：

清单 2. 把 DataSource 添加到 geronimo-web.xml 中

```
<web-app
xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1">
    <environment>
        <moduleId>
            <artifactId>StocksApplication</artifactId>
        </moduleId>
        <dependencies>
            <dependency>
```

```
                    <groupId>console.dbpool</groupId>
                    <artifactId>stocks-db</artifactId>
                </dependency>
            </dependencies>
        </environment>
        <context-root>/stocks</context-root>
        <resource-ref>
            <ref-name>jdbc/MyDataSource</ref-name>
            <resource-link>stocks-db</resource-link>
        </resource-ref>
</web-app>
```

如前述，数据库池已被部署为连接器。连接器就像部署到 Geronimo 上的任何模块一样。您的应用程序也是 Geronimo 中的模块，因此将使用 geronimo-web.xml 文件来建立应用程序与为数据库连接池创建的连接器模块之间的依赖性。然后，将为该依赖性创建一个别名，以便可以在 web.xml 文件中使用此依赖性，如清单 3 所示：

清单 3. 在 web.xml 中引用 DataSource

```
<?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">

      <servlet>


<servlet-name>stocksServiceServlet</servlet-name>




<servlet-class>org.developerworks.stocks.server.StocksServiceI

mpl

               </servlet-class>

        </servlet>

       <servlet-mapping>


<servlet-name>stocksServiceServlet</servlet-name>

              <url-pattern>/stocksService</url-pattern>

        </servlet-mapping>

       <resource-ref>


<res-ref-name>jdbc/MyDataSource</res-ref-name>
```

```
                    <res-type>javax.sql.DataSource</res-type>
                    <res-auth>Container</res-auth>


<res-sharing-scope>Shareable</res-sharing-scope>
        </resource-ref>
        <welcome-file-list>
          <welcome-file>
            StocksApplication.html
         </welcome-file>
        </welcome-file-list>
</web-app>
```

现在 Web 应用程序中有对 DataSource 对象的引用。您可以使用 res-ref-name 在 web.xml 文件中的 resource-ref 中查找该引用。现在，已经准备好在服务中使用 DataSource。

修改 StocksService 接口

要修改服务，需要先修改客户机所使用的接口。这些接口都被编译成 JavaScript，但是这些接口还为服务的服务器端实现定义了约定。在此部分中，首先将修改服务，然后再修改应用程序以利用服务中的新功能。

StocksService 接口

您已经准备好把投资添加到应用程序中。首先将重新定义服务接口。向服务接口中添加两种新方法，如清单 4 所示：

清单 4. StocksService 接口

```
package org.developerworks.stocks.client;

import com.google.gwt.user.client.rpc.RemoteService;

public interface StocksService extends RemoteService {
        public Stock getQuote(String symbol);
        public Stock[] getPortfolio(String username);
        public Stock[] addStocks(String username, String[] symbols);
}
```

您已经添加了两种新方法：

- getPortfolio 用于用户登录。
- addStocks 用于把股票保存到投资中。

两种方法都将返回股票数组。您必须对返回的内容十分小心。记住接口是在 JavaScript 中使用的，因此只允许使用 Java 类的子集。您可以使用 java.util.List、java.util.ArrayList 或 java.util.Vector。但是，在尝试使用泛型时不能够使用它 —— 例如，您不能使用 List<Stock> 作为返回类型。JavaScript 编译器只支持 JDK 1.4 和最新的功能。由于修改了接口，因此还需要修改该接口的异步版本，如清单 5 所示。

清单 5. 异步 StocksService 接口

```
package org.developerworks.stocks.client;

import com.google.gwt.user.client.rpc.AsyncCallback;

public interface StocksServiceAsync {
        public void getQuote(String symbol, AsyncCallback callback);
        public void getPortfolio(String username, AsyncCallback callback);
        public void addStocks(String username, String[] symbols,
AsyncCallback callback);
}
```

您已经遵循了在 第 1 部分 中使用的同一个模式。对于在接口中定义的每种方法，您已经把同名的方法添加到异步接口中。异步版本将全部返回 void 并且所有版本都将获取一个额外的参数 AsyncCallback 对象。此对象用于存储来自服务调用的返回值。既然已经更新了接口，就需要更新实现。

StocksService 实现

把 StocksService 接口中定义的两种新方法添加到实现类中，如清单 6 所示：

清单 6. StocksService 实现

```
package org.developerworks.stocks.server;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.net.MalformedURLException;

import java.net.URL;

import java.net.URLConnection;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;
```

```
import java.util.LinkedList;

import java.util.List;


import javax.naming.InitialContext;

import javax.naming.NamingException;

import javax.sql.DataSource;


import org.developerworks.stocks.client.Stock;

import org.developerworks.stocks.client.StocksService;


import com.google.gwt.user.client.rpc.InvocationException;

import com.google.gwt.user.server.rpc.RemoteServiceServlet;


public class StocksServiceImpl extends RemoteServiceServlet implements

StocksService {


        private static final String YAHOO_QUOTE_URL =

                "http://quote.yahoo.com/d/quotes.

                  csv?s=:symbol&f=sl1d1t1c1ohgvj1pp2owern&e=.csv";

        private static final String LOOKUP = "select user_id from user where

name=?";

        private static final String QUERY =
```

```
            "select p.symbol from portfolio p, user u where p.user_id =
                u.user_id and u.name=?";
    private static final String INSERT_STOCKS = "insert into portfolio
values (?,?)";
    private static final String INSERT_USER = "insert into user (name)
values (?)";

    public Stock getQuote(String symbol) {
        String urlStr = YAHOO_QUOTE_URL.replace(":symbol", symbol);
        try {
            URL url = new URL(urlStr);
            URLConnection conn = url.openConnection();
            conn.connect();
          InputStream is = conn.getInputStream();
          InputStreamReader isr = new InputStreamReader(is);
          BufferedReader br = new BufferedReader(isr);
          String line = null;
          Stock stock =new Stock();
          while ( (line = br.readLine()) != null){
              String[] stockArray = line.split(",");
              for (int i=0;i<stockArray.length;i++){
```

```
stock.setCompanyName(stockArray[16].replaceAll("\"", ""));
                        stock.setSymbol(stockArray[0].replaceAll("\"", ""));
                        stock.setPrice(new Double(stockArray[1]));
                        stock.setChange(new Double(stockArray[4]));
                    }
                }
                return stock;
            } catch (MalformedURLException e) {
                throw new InvocationException("Something was wrong
with the URL",e);
            } catch (IOException e) {
                throw new InvocationException("Exception reading data
from Yahoo",e);
            }
        }

        public Stock[] addStocks(String username, String[] symbols) {
            DataSource ds = getDataSource();
            try {
                Connection conn = ds.getConnection();
                PreparedStatement ps =
conn.prepareStatement(LOOKUP);
```

```
                ps.setString(1, username);
                ResultSet rs = ps.executeQuery();
                Integer userId = null;
                if (rs.next()){
                        userId = rs.getInt(1);
                } else {
                        ps = conn.prepareStatement(INSERT_USER);
                        ps.setString(1, username);
                        int num = ps.executeUpdate();
                        if (num > 0){
                                rs = ps.getGeneratedKeys();
                                if (rs.next()){
                                        userId = rs.getInt(1);
                                }
                        }
                }
                if (userId != null){
                        ps = conn.prepareStatement(INSERT_STOCKS);
                        for (String symbol : symbols){
                                ps.setInt(1, userId);
                                ps.setString(2, symbol);
                                ps.addBatch();
```

```
                        }
                        ps.executeBatch();
                        conn.commit();
                    }
                    conn.close();
                } catch (SQLException e) {
                    throw new InvocationException("Exception saving
stocks",e);
                }
                return this.getPortfolio(username);
        }

    public Stock[] getPortfolio(String username) {
        DataSource ds = getDataSource();
        try {
            Connection conn = ds.getConnection();
            PreparedStatement ps = conn.prepareStatement(QUERY);
            ps.setString(1, username);
            ResultSet results = ps.executeQuery();
            List<String> symbols = new LinkedList<String>();
            while (results.next()){
                symbols.add(results.getString(1));
```

```
                }
                // If this was "production" code, this would be in a finally
block, etc.
                conn.close();
                Stock[] stocks = new Stock[symbols.size()];
                int cnt = 0;
                for (String symbol : symbols){
                        stocks[cnt++] = this.getQuote(symbol);
                }
                return stocks;
            } catch (SQLException e){
                    throw new InvocationException("Exeption querying
database",e);
            }
        }

        private static DataSource getDataSource(){
            InitialContext ctx;
            try {
                    ctx = new InitialContext();
                    return (DataSource)
ctx.lookup("java:comp/env/jdbc/MyDataSource");
```

```
            } catch (NamingException e) {
                  throw new InvocationException("Exception getting
datasoure",e);
            }
      }

}
```

两种新方法全都充分利用了 JDBC。为了获得 JDBC 连接，两种方法将调用静态的 getDataSource() 方法。此方法看上去应当十分熟悉。只需在 JNDI 中查找您的 DataSource。将使用从 图 11 中所示的数据库连接池用法中获得的查找字符串。

getPortfolio 方法将通过执行针对数据库的联合查询来查找用户的投资中的所有股票。查询将检索股票代码，然后您可以重用 getQuote 方法以获得关于给定股票的元数据。

addStocks 方法将处理几个场景。首先它将处理用户向投资中添加股票的场景。它还将处理用户通过输入用户名添加并随后保存多支股票的场景。最后，它将处理用户在向新投资中添加一些股票之后登录的场景。持久保存数据后，它将调用 getPortfolio() 重新查询股票。这将使您可以处理所述的最后一个场景，在该场景中用户有投资，但是在登录前添加了更多股票。

既然已经修改了服务，就需要修改应用程序以在服务中使用新功能。

修改股票应用程序

您的后端服务已经利用了 Geronimo 向您提供新投资功能。现在需要修改应用程序代码。新代码如清单 7 所示。

清单 7. StocksApplication

```
package org.developerworks.stocks.client;

import java.util.ArrayList;

import java.util.List;


import com.google.gwt.core.client.EntryPoint;

import com.google.gwt.core.client.GWT;

import com.google.gwt.user.client.rpc.AsyncCallback;

import com.google.gwt.user.client.rpc.ServiceDefTarget;

import com.google.gwt.user.client.ui.Button;

import com.google.gwt.user.client.ui.ClickListener;

import com.google.gwt.user.client.ui.FlexTable;

import com.google.gwt.user.client.ui.KeyboardListenerAdapter;
```

```
import com.google.gwt.user.client.ui.Label;

import com.google.gwt.user.client.ui.RootPanel;

import com.google.gwt.user.client.ui.TextBox;

import com.google.gwt.user.client.ui.Widget;


/**

 * Entry point classes define onModuleLoad().

 */

public class StocksApplication implements EntryPoint {


            StocksService service;

/**

 * This is the entry point method.

 */

public void onModuleLoad() {

   final Button button = new Button("Get Quote");

   final TextBox textBox = new TextBox();

   final FlexTable stockTable = new FlexTable();

   final Button userButton = new Button("Login");

   final TextBox userBox = new TextBox();

   final List symbolsList = new ArrayList();

   final Label welcomeLabel = new Label();
```

```
        final StocksServiceAsync stocksService = (StocksServiceAsync)
 GWT.create(StocksService.class);
        ServiceDefTarget endpoint = (ServiceDefTarget) stocksService;
        String moduleRelativeURL = GWT.getModuleBaseURL() +
"stocksService";
        endpoint.setServiceEntryPoint(moduleRelativeURL);

        final AsyncCallback callback = new AsyncCallback() {

            public void onFailure(Throwable caught) {
                alert("Sorry there was an error");
            }

            public void onSuccess(Object result) {
                Stock stock = (Stock) result;
                int cnt = stockTable.getRowCount();
                stockTable.setText(cnt, 0, stock.toString());
                // show things as green or red depending on if the stock is
up or down
                if (stock.getChange() > 0){
                    stockTable.getCellFormatter().addStyleName(cnt, 0,
```

```
"stockUp");
                    } else if (stock.getChange() < 0){
                        stockTable.getCellFormatter().addStyleName(cnt, 0,
"stockDown");
                    }
                    symbolsList.add(stock.getSymbol());
                    textBox.setText("");
                }

            };

        final AsyncCallback userCallback = new AsyncCallback(){

            public void onFailure(Throwable caught) {
                alert("Sorry there was an error: " + caught.getMessage());
            }

            public void onSuccess(Object result) {
                Stock[] stocks = (Stock[]) result;
                int cnt = stockTable.getRowCount();
                // don't be tempted to use the new for loop --
                //     GWT does not support Java 5 in client code!
```

```
                    for (int i=0;i<stocks.length;i++){
                        if (!symbolsList.contains(stocks[i].getSymbol())){
                            Stock stock = stocks[i];
                            stockTable.setText(cnt, 0, stock.toString());
                            if (stock.getChange() > 0){
                                stockTable.getCellFormatter().
                                        addStyleName(cnt, 0,
"stockUp");
                            } else if (stock.getChange() < 0){
                                stockTable.getCellFormatter().
                                        addStyleName(cnt, 0,
"stockDown");
                            }
                            cnt++;
                            symbolsList.add(stocks[i].getSymbol());
                        }
                    }
                    userButton.setVisible(false);
                    userBox.setVisible(false);
                    welcomeLabel.setText("Welcome "+userBox.getText());
                }
```

```
        };


    button.addClickListener(new ClickListener() {
      public void onClick(Widget sender) {
            String symbol = textBox.getText();
            stocksService.getQuote(symbol, callback);
            // check if logged in already
            if (userBox.getText() != null && userBox.getText().trim().


length() > 0){
                  stocksService.addStocks(userBox.getText(), new String[]


{symbol}, callback);
                }
        }
    });


    // make hitting Return the same as clicking the button
    textBox.addKeyboardListener(new KeyboardListenerAdapter() {
          public void onKeyPress(Widget sender, char keyCode, int


modifiers){
                if (keyCode == '\r'){
```

```
                    button.click();
                }
            }
        }

);

userButton.addClickListener(new ClickListener(){
        public void onClick(Widget sender) {
                String username = userBox.getText();
                int cnt = stockTable.getRowCount();
                if (cnt == 0){
                        // empty so login
                        stocksService.getPortfolio(username, userCallback);
                } else {
                        // not empty add user and save stocks
                        String[] symbols = new String[cnt];
                        for (int i=0;i<cnt;i++){
                                symbols[i] = (String) symbolsList.get(i);
                        }
                        stocksService.addStocks(username, symbols,
userCallback);
```

```
                }


            }
        });

        // make hitting Return the same as clicking the button
        userBox.addKeyboardListener(new KeyboardListenerAdapter() {
                public void onKeyPress(Widget sender, char keyCode, int
modifiers){
                    if (keyCode == '\r'){
                        userButton.click();
                    }
            }
        }


        );
        // These are the elements that the widgets are being added to
        RootPanel.get("quoteButton").add(button);
        RootPanel.get("quoteBox").add(textBox);
        RootPanel.get("stockTable").add(stockTable);
        RootPanel.get("userBox").add(userBox);
        RootPanel.get("userButton").add(userButton);
```

```
    RootPanel.get("welcome").add(welcomeLabel);
    stockTable.setStyleName("center");
  }

  public static native void alert(String msg) /*-{
        $wnd.alert(msg);
  }-*/;
}
```

已经添加的新内容有很多，包括：

- 使用 FlexTable 而不是 Label，允许多支股票的信息显示在页面上。FlexTable 成为 HTML 表，但给您提供了一种可以轻松地向表中添加行和列的方法。
- 用于登录的文本框和按钮。
- 用于向用户显示欢迎信息的标签。
- 用于跟踪已添加的股票的临时数据结构（列表）。这将允许您随时保存股票的列表。注意，使用了 java.util.List 和 java.util.ArrayList，因为两者可以被 GWT 转换为 JavaScript。

处理事件

既然已经添加了所需的对象，就可以通过处理事件在服务上调用新功能。初始的按钮已经修改了它的处理程序。如果用户已登录，则将发出另一个异步调用把股票信息保存到该用户的投资中。由于异步本性，因此不必等待此异步调用完成，就可以显示用户刚输入的股票信息。将持久存储数据，而用户无需等待。这真的是使用 GWT 可以轻松完成的优秀功能。

添加了单击 Login 按钮的处理程序。此处理程序将查看用户是否已经添加了股票。它将根据是否需要把股票保存到用户的投资中来执行不同的调用。还把在用户框中点击 Return 按钮的操作映射为单击按钮，就像为 quote 按钮所做的处理一样。

接下来修改了现有的回调以把股票添加到数据结构中。新回调将处理已被检索到的投资的显示。如果已经有显示的股票，则回调只把从服务器检索到的新股票添加到表中。如果登录成功，则登录框和按钮将被隐藏，并显示欢迎信息。

最后，需要注意的是错误处理过程。您将使用静态方法 alert()。查看一下此方法的实现 —— 使用 GWT 的 JavaScript Native Interface (JSNI)。声明此方法为本机方法（Java 技术中的保留字，通常表示对 C/C++ 代码的调用），然后把 JavaScript 放到注释中。如果您曾经进行过任何 JavaScript 编程，则会发现这就像在弹出框中显示信息的简单代码一样。您可能想知道关于 $wnd 的信息。这将引用来自 GWT 文档的窗口对象（有关链接，请参阅 参考资料），引文如下：

在访问浏览器的窗口和 JSNI 的文档对象时，必须将它们分别引用为 **$wnd** 和 **$doc**。经过编译的脚本将在嵌套的框架中运行，并且 **$wnd** 和 **$doc** 都是被自动初始化以正确引用主机页面的窗口和文档。

这段代码都被编译成 Web 页面所使用的 JavaScript。您现在只需修改 Web 页面，如清单 8 所示：

清单 8. StocksApplication.html

```
<html>
	<head>
		<title>GWT Stocks Application</title>
		<style>
			#stockInfo{
				text-align: center;
				padding-top: 10px;
			}
			#quotes{
				text-align: center;
			}
			#quoteBox{
				padding-right: 5px;
			}
			.stockUp{
				color:green;
			}
			.stockDown{
				color:red;
```

```
        }
        table.center {
                margin-left:auto;
                margin-right:auto;
        }
        body {
                text-align:center;
        }
        h1,p {
                text-align:left;
        }
                #welcome {
                        text-align:left;
                        font-size:large;
                }
        </style>


        <!--                                                    -->
        <!-- The module reference below is the link       -->
        <!-- between html and your Web Toolkit module    -->
        <!--                                                    -->
        <meta name='gwt:module'
```

```
content='org.developerworks.stocks.


StocksApplication'>


        </head>
        <body>


                <!--                                                    -->
                <!-- This script is required bootstrap stuff.     -->
                <!-- You can put it in the HEAD, but startup       -->
                <!-- is slightly faster if you include it here. -->
                <!--                                                    -->
                <script language="javascript" src="gwt.js"></script>


                <!-- OPTIONAL: include this if you want history support -->
                <iframe id="__gwt_historyFrame"
                                style="width:0;height:0;border:0"></iframe>


                <h1>Stocks Application</h1>
                <p>
                        <span id="welcome"/>
                        Enter in the symbol for a stock below.
```

```
        </p>


        <div id="quotes">
           <span id="quoteBox"></span>
           <span id="quoteButton"></span>
        </div>
        <div id="stockTable"/>
        <div id="userInfo">
              <span id="userBox"></span>
              <span id="userButton"></span>
        </div>
     </body>
</html>
```

这里没有太多更改。您已经添加了一些 div 和 span 以包含新 UI 组件。您还添加了这些对象的某个 CSS。现在已经准备好把应用程序部署到 Geronimo 中。

部署到 Geronimo 中

在此部分中，首先使用 Ant 构建新的 WAR 文件，然后将其部署到 Geronimo 中再测试它。

构建 WAR 文件

要部署应用程序，就需要编译新的 WAR 文件。您为编译 WAR 文件创建了 Ant 文件，该 WAR 文件仍可以为修改后的应用程序很好地工作。在编译文件上调用 WAR 目标，然后应当会看到类似清单 9 的输出。

您可能会注意到编译器警告。这是由在 StocksApplication 中使用非通用版本的 java.util.List 所致。当然，不能使用通用版本的 List，因为这是已经编译成 JavaScript 的代码，并且 GWT 不支持在客户机代码中使用泛型（或任何其他 Java 5 功能）。

清单 9. 编译 WAR 文件

```
Buildfile: C:\code\projects\stocks\build.xml
build:
    [mkdir] Created dir: C:\code\projects\stocks\bin
    [javac] Compiling 5 source files to C:\code\projects\stocks\bin
    [javac] Note: C:\code\projects\stocks\src\org\developerworks\stocks
\client\StocksApplication.java uses unchecked or unsafe operations.
    [javac] Note: Recompile with -Xlint:unchecked for details.
jar:
    [mkdir] Created dir: C:\code\projects\stocks\www\StocksApplication
\WEB-INF\lib
      [jar] Building jar: C:\code\projects\stocks\www\StocksApplication
\WEB-INF\lib\stocks.jar
gwt:
     [exec] Output will be written into
```

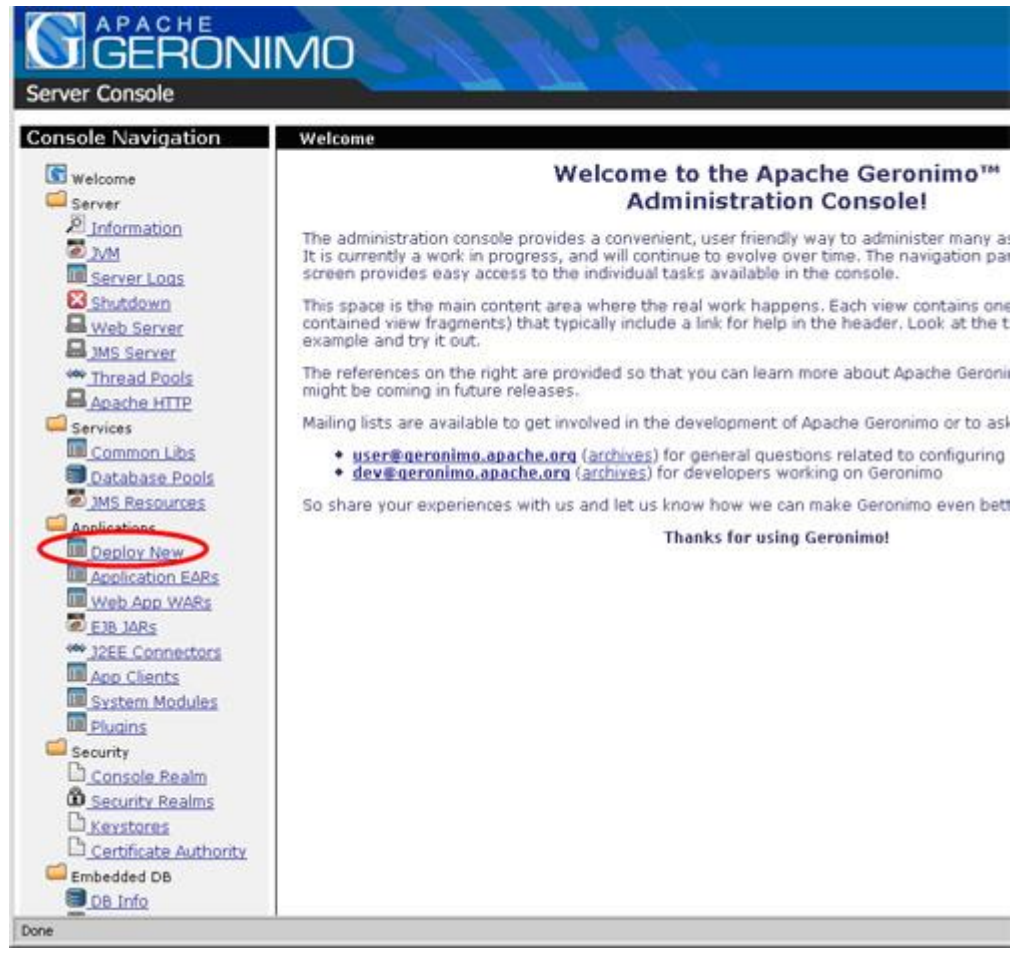
```
 C:\code\projects\stocks\www\org.developerworks.stocks.StocksApplication
      [exec] Copying all files found on public path
      [exec] Compilation succeeded
      [copy] Copying 17 files to C:\code\projects\stocks\www\StocksApplication
war:
      [copy] Copying 1 file to C:\code\projects\stocks\www\StocksApplication\WEB-INF\lib
       [jar] Building jar: C:\code\projects\stocks\stocks.war
BUILD SUCCESSFUL
Total time: 5 seconds
```

WAR 文件现已编译完成，因此只需将其部署到 Geronimo 中。

让我们返回到 Geronimo 控制台：

1. 这次选择 **Applications > Deploy New**，如图 12 所示：

图 12. 部署新应用程序



此操作将打开安装新应用程序界面。

2. 浏览到 WAR 文件并选中它。如果旧版本的应用程序已被部署，则选中 **Redeploy application** 复选框，如图 13 所示：

图 13. 要部署的新应用程序

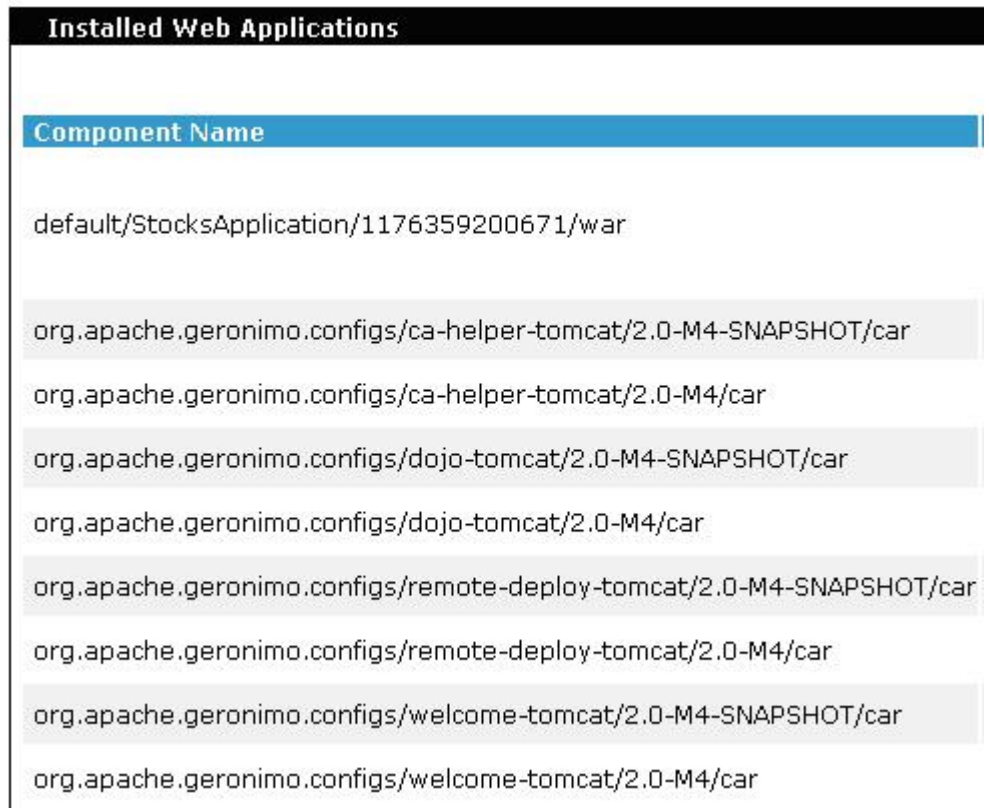
3. 单击 **Install** 之后，WAR 文件将被部署，并且应当会获得已成功的信息。

4. 单击 **Applications > Wab App WARs** 检验应用程序是否已部署，如图 14 所示：

图 14. 应用程序已成功部署

您应当会在 Web 应用程序的列表中看到您的应用程序，如图 15 所示：

图 15. Web 应用程序



现在您的应用程序已安装！可以开始测试它了。

回页首

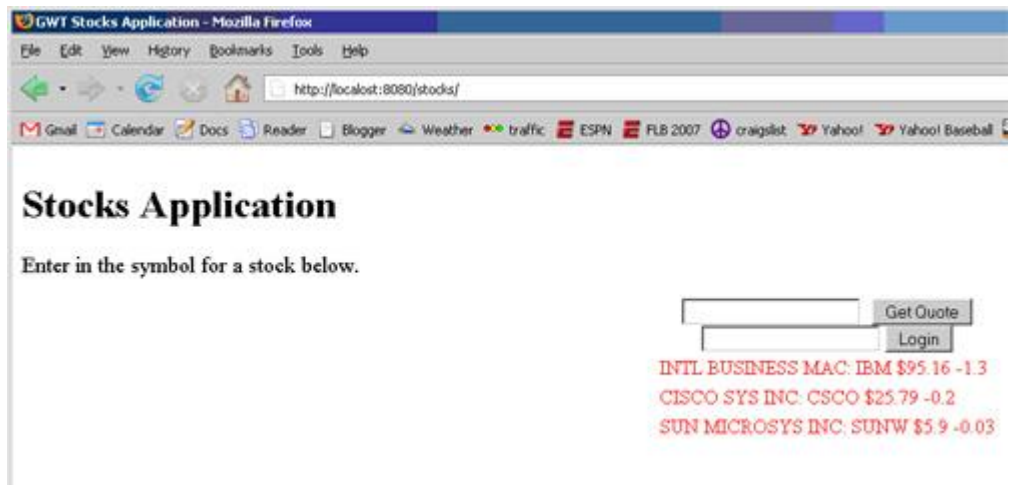
测试应用程序

1. 单击 Web 应用程序的 URL 链接以打开该应用程序，如图 16 所示：

图 16. 投资应用程序

2. 添加一些股票，如 IBM、CSCO 和 SUNW，如图 17 所示：

图 17. 添加的股票



3. 现在保存这些股票。在 Login 字段中键入 developerworks，如图 18 所示：

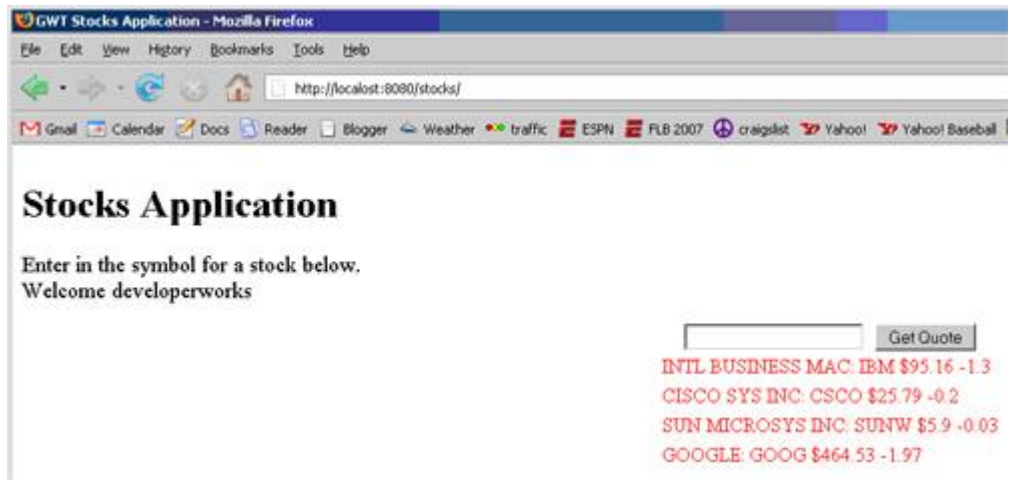
图 18. 登录

4. 单击 **Login** 按钮，这样做应当会得到类似图 19 所示的信息。

图 19. 已登录

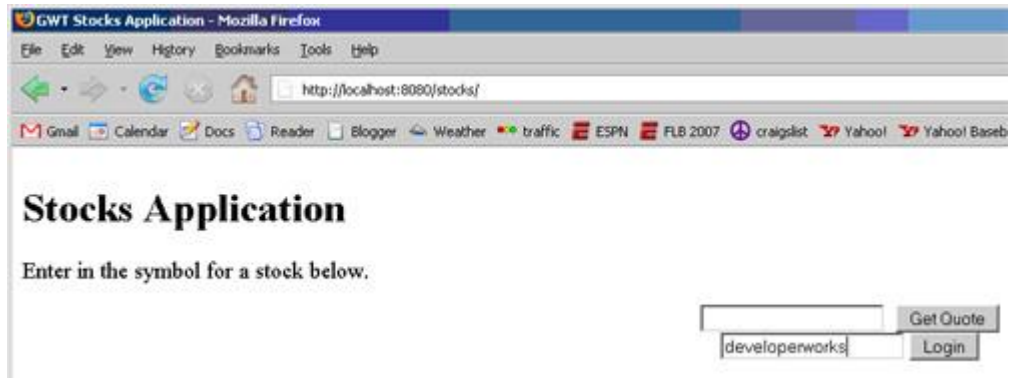
5. 现在可以再添加一些股票，这些股票将被异步保存。添加 GOOG，如图 20 所示：

图 20. 异步保存的股票



6. 要检验所有内容是否已保存，只需重新装入页面。您从未使用任何会话，并且跟踪了客户机上的所有状态，因此重新装入将实际上使您登出。键入 developerworks，并重新登录，如图 21 所示：

图 21. 重新登录



您应当会看到与图 20 相同的屏幕。请随便再添加一些股票、用户等等。