Servlet教程

极客学院出版

前言

Servlet 为构建基于 Web 的应用程序提供了一个基于组件的、独立于平台的方法，没有 CGI 程序的性能限制。Servlet 访问 Java API 的整个家族，包括 JDBC API 来访问企业数据库。

本教程将用简单易学的方法教你使用 Java Servlet 开发基于 Web 的应用程序。

适用人群

本教程是为需要了解 Java Servlet 框架及其应用程序的 Java 程序员设计的。学完本教程后你会发现自己处于一个使用 Java Servelt 专业知识的中等水平，之后你可以达到更高的水平。

学习前提

我们假设你对 Java 编程语言非常了解。如果你对 Web 应用程序及网络是如何工作的有一个基本的了解，那就非常棒了。

更新日期	更新内容
2015-06-05	第一版发布

目录

1

概述

什么是 Servlets?

Java servlet 是运行在 Web 或应用服务器上的程序，作为在来自 Web 浏览器或其他 HTTP 客户机的请求和在 HTTP 服务器上的数据库或应用程序的中间层。

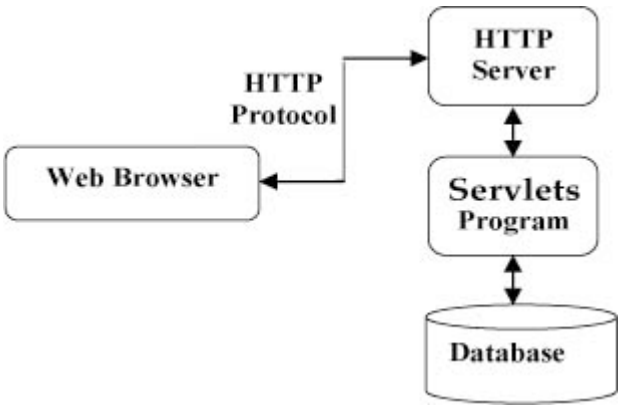
使用 Servlet，你可以通过 web 页面表单来收集用户的输入，显示从数据库或其他来源的记录，动态地创建 web 页面。

Java servlet 通常服务于使用 Common Gateway Interface (CGI) 实现的同样的目的程序。但与 CGI 相比，Servlet 具有几个优点。

- 性能更好。
- Servlet 在 Web 服务器的地址空间内执行。没有必要创建一个单独的进程来处理每个客户端请求。
- 由于 Servlet 是用 Java 编写的，所以它是跨平台的。
- 在服务器上的 Java 安全性管理器执行的一些限制来保护服务器上的资源。所以 servlet 是可信的。
- Java 类库的完整的功能是对 Servlet 来说是可用的。它可以与小应用程序、数据库或其他软件通过通信接口和你已经了解的RMI机制进行通信。

Servlets 架构

下图显示了在 Web 应用程序中 Servlets 的位置。



Servlets 任务

Servlet 执行以下主要任务：

- 读取由客户端（浏览器）发送的显式数据。这包括网页上的 HTML 表单，或者也可以是来自 applet 或自定义的 HTTP 客户端程序的表单。
- 读取由客户端（浏览器）发送的隐式 HTTP 请求数据。这包括 cookies、媒体类型和浏览器能理解的压缩格式等等。
- 处理数据并生成结果。这个过程可能需要访问数据库，执行 RMI 或 CORBA 调用，调用 Web 服务，或者直接计算响应。
- 发送显式数据（即文档）到客户端（浏览器）。该文档可以以多种多样的格式被发送，包括文本文件（HTML 或 XML）、二进制文件（GIF 图像）、Excel 等。
- 发送隐式的 HTTP 响应到客户端（浏览器）。这包括告诉浏览器或其他客户端被返回的文档类型（例如 HTML），设置 cookies 和缓存参数，以及其他类似的任务。

Servlets 包

Java Servlet 是运行在 Web 服务器上的 Java 类，在 Web 服务器上有一个支持 Java Servlet 规范的解释器。

Servlet 可以使用 javax.servlet 和 javax.servlet.http 包来创建。它们是 Java 企业版的一个标准部分，也是支持大型开发项目的 Java 类库的扩展版。

这些类实现了 Java Servlet 和 JSP 规范。在写这篇教程的时候，使用的版本分别是 Java Servlet 2.5 和 JSP 2.5。

就像任何其他 Java 类一样，Java Servlet 可以创建和编译。在安装 Servlet 包，并将它们添加到你的电脑的 Classpath 中之后，你可以使用 JDK 的 Java 编译器或其他任何当前编译器来编译 Servlet。

后续内容

接下来，本教程会带你一步一步地设置你的环境，以便开始后续的 Servlet 使用。因此，请系紧安全带，随我们一起开始 Servlet 的学习之旅吧！相信你会很喜欢这个教程的。

2

环境设置

开发环境是你开发你的 Servlet，测试它们并最终运行它们的地方。

和任何其他的 Java 程序一样，你需要通过使用 Java 编译器 javac 来编译 servlet，并且在编译 servlet 应用程序后，它将会被部署在配置的环境中来测试和运行。

这个开发环境设置包括以下步骤：

设置 Java 开发工具包

这一步涉及到下载 Java 软件开发工具包（SDK），并正确的设置 PATH 环境变量。

你可以从 Oracle 的 Java 网站 Java SE Downloads (http://www.oracle.com/technetwork/java/javase/downloads/index.html) 下载 SDK。

当你下载了你的 Java 实现后，按照给定的指令来安装和配置设置。最后，设置 PATH 和 JAVA _ HOME 环境变量来与包含 java 和 javac 的目录相关联，通常分别为 java _ install _ dir / bin 和 java _ install _ dir。

如果你运行的是 Windows 操作系统且 SDK 安装在 C:\jdk1.6.0_20 中，那么在你的 C:\autoexec.bat 文件中放入下列的行：

```
set PATH=C:\jdk1.5.0_20\bin;%PATH%
set JAVA_HOME=C:\jdk1.5.0_20
```

或者，在 Windows XP/7/8 操作系统中，你也可以用鼠标右键单击“我的电脑”，选择“属性”，再选择“高级”，“环境变量”。然后，更新 PATH 值，按下 “OK” 按钮。

在 Unix（Solaris、Linux 等）操作系统中，如果 SDK 安装在 /usr/local/jdk1.6.0_20 中，并且你使用的是 C shell，则在你的 .cshrc 文件中放入下列的行：

```
setenv PATH /usr/local/jdk1.6.0_20/bin:$PATH
setenv JAVA_HOME /usr/local/jdk1.6.0_20
```

另外，如果你使用一个集成的开发环境（IDE），比如 Borland JBuilder、Eclipse、IntelliJ IDEA 或 Sun ONE Studio，编译并运行一个简单的程序，以确认该 IDE 知道你把 Java 安装在哪里。

设置 Web 服务器：Tomcat

市场上有许多 Web 服务器支持 servlets。有些 web 服务器是免费下载的，Tomcat 就是其中的一个。

Apache Tomcat 是 Java Servlet 和 JavaServer Pages 技术的开源软件实现，可以作为测试 servlets 的独立服务器，而且可以用 Apache Web 服务器集成。下面是在你的电脑上安装 Tomcat 的步骤：

- 从 http://tomcat.apache.org/ 上下载最新版本的 Tomcat。
- 一旦你下载了 Tomcat，将该二进制发布包解压缩到一个方便的位置。例如，如果你使用的是 Windows操作系统，则解压缩到 C:\apache-tomcat-5.5.29 中，如果你使用的是 Linux/Unix 操作系统，则解压缩到 /usr/local/apache-tomcat-5.5.29 中，并创建指向这些位置的 CATALINA_HOME 环境变量。

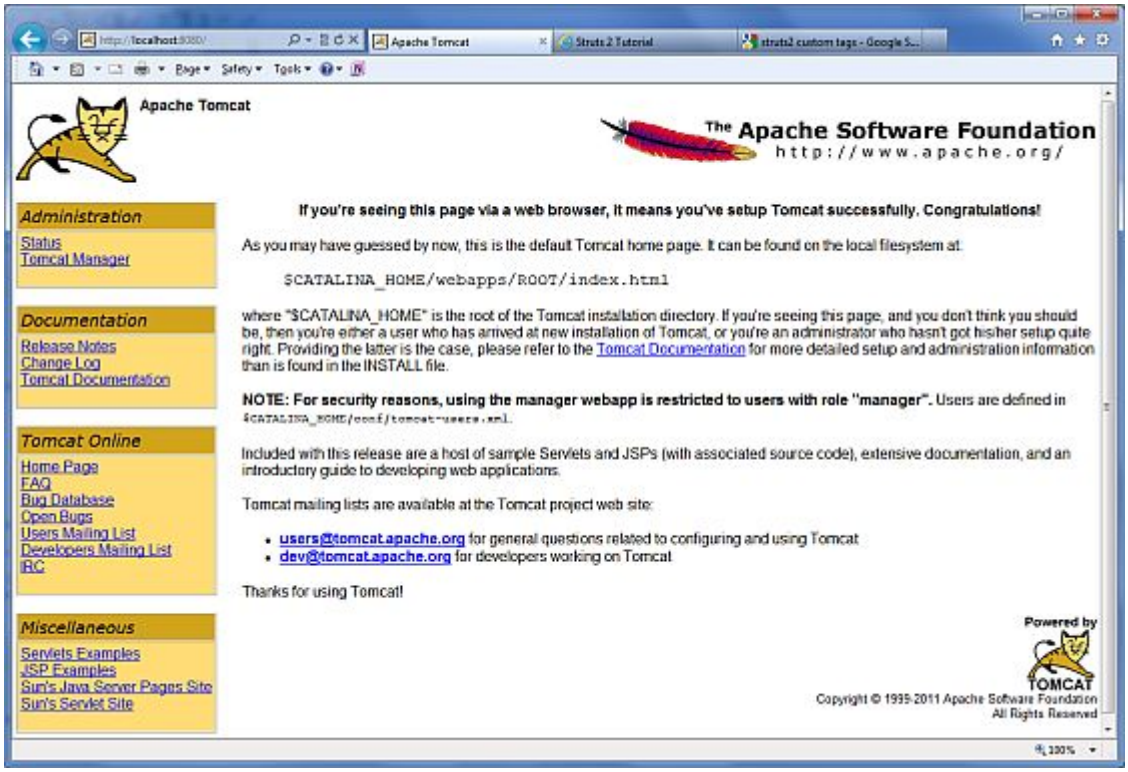
在 Windows 操作系统的计算机上，可以通过执行下述命令来启动 Tomcat：

```
%CATALINA_HOME%\bin\startup.bat
 or
 C:\apache-tomcat-5.5.29\bin\startup.bat
```

在 Unix（Solaris、Linux 等）操作系统的计算机上，可以通过执行下述命令来启动 Tomcat：

```
$CATALINA_HOME/bin/startup.sh
or
/usr/local/apache-tomcat-5.5.29/bin/startup.sh
```

Tomcat 启动后，通过访问 **http://localhost:8080/**，Tomcat 包含的默认 web 应用程序会变得可用。如果一切顺利，那么会显示如下所示结果：



有关配置和运行 Tomcat 的更多信息可以查阅这里包含的文档，也可以访问 Tomcat 网站：http://tomcat.apache.org。

在 Windows 操作系统的计算机上，可以通过执行下面的命令来停止 Tomcat：

```
C:\apache-tomcat-5.5.29\bin\shutdown
```

在 Unix（Solaris、Linux 等）操作系统的计算机上，可以通过执行下面的命令来停止 Tomcat：

```
/usr/local/apache-tomcat-5.5.29/bin/shutdown.sh
```

设置 CLASSPATH

由于 servlets 不是 Java 平台标准版的组成部分，所以你必须为编译器指定 servlet 类。

如果你运行的是 Windows 操作系统，则需要在你的 C:\autoexec.bat 文件中放入下列的行：

```
set CATALINA=C:\apache-tomcat-5.5.29
set CLASSPATH=%CATALINA%\common\lib\servlet-api.jar;%CLASSPATH%
```

或者，在 Windows XP/7/8 操作系统中，你也可以用鼠标右键单击“我的电脑”，选择“属性”，再选择“高级”，“环境变量”。然后，更新 CLASSPATH 的值，按下“OK”按钮。

在 Unix（Solaris、Linux 等）操作系统中，如果你使用的是 C shell，则需要在你的 .cshrc 文件中放入下列的行：

```
setenv CATALINA=/usr/local/apache-tomcat-5.5.29
setenv CLASSPATH $CATALINA/common/lib/servlet-api.jar:$CLASSPATH
```

注意：假设你的开发目录是 C:\ServletDevel（Windows 操作系统中）或 /user/ServletDevel（UNIX 操作系统中），那么你还需要在 CLASSPATH 中添加这些目录，添加方式与上面的添加方式类似。

3

生命周期

Servlet 生命周期可被定义为从它被创建直到被销毁的整个过程。以下是 servlet 遵循的过程：

- 通过调用 init () 方法 servlet 被初始化。
- Servlet 调用 service() 方法来处理客户端的请求。
- 通过调用 destroy() 方法 servlet 终止。
- 最后，servlet 是由 JVM 的垃圾回收器进行垃圾回收的。

现在让我们详细的讨论生命周期的方法。

init() 方法

init 方法被设计成只调用一次。它在第一次创建 servlet 时被调用，在后续每次用户请求时不再调用。因此，它用于一次性初始化，与 applets 的 init 方法一样。

通常情况下，当用户第一次调用对应于该 servlet 的 URL 时，servlet 被创建，但是当服务器第一次启动时，你也可以指定 servlet 被加载。

当用户调用 servlet 时，每个 servlet 的一个实例就会被创建，并且每一个用户请求都会产生一个新的线程，该线程在适当的时候移交给 doGet 或 doPost 方法。init() 方法简单地创建或加载一些数据，这些数据将被用于 servlet 的整个生命周期。

init 方法的定义如下：

```
public void init() throws ServletException {
  // Initialization code...
}
```

service() 方法

service() 方法是执行实际任务的主要方法。Servlet 容器（即 web 服务器）调用 service() 方法来处理来自客户端（浏览器）的请求，并将格式化的响应写回到客户端。

每次服务器接收到一个 servlet 请求时，服务器会产生一个新的线程并调用服务。service() 方法检查 HTTP 请求类型（GET、POST、PUT、DELETE 等），并在适当的时候调用 doGet、doPost、doPut、doDelete 等方法。

下面是该方法的特征：

```
public void service(ServletRequest request,
                    ServletResponse response)
    throws ServletException, IOException{
}
```

service() 方法由容器调用，且 service 方法在适当的时候调用 doGet、doPost、doPut、doDelete 等。所以对 service() 方法你什么都不需要做，只是根据你接收到的来自客户端的请求类型来重写 doGet() 或 doPost()。

doGet() 和 doPost() 方法在每次服务请求中是最常用的方法。下面是这两种方法的特征。

doGet() 方法

GET 请求来自于一个 URL 的正常请求，或者来自于一个没有 METHOD 指定的 HTML 表单，且它由 doGet() 方法处理。

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
  throws ServletException, IOException {
  // Servlet code
}
```

doPost() 方法

POST 请求来自于一个 HTML 表单，该表单特别的将 POST 列为 METHOD 且它由 doPost() 方法处理。

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
  throws ServletException, IOException {
  // Servlet code
}
```

destroy() 方法

destroy() 方法只在 servlet 生命周期结束时被调用一次。destroy() 方法可以让你的 servlet 关闭数据库连接、停止后台线程、将 cookie 列表或点击计数器写入磁盘，并执行其他类似的清理活动。

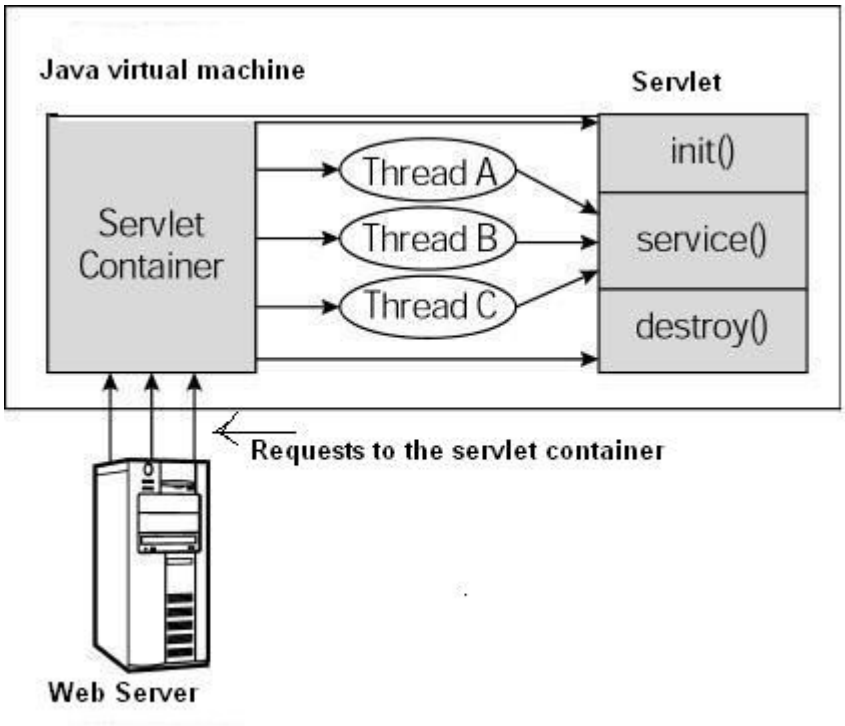
在调用 destroy() 方法之后，servlet 对象被标记用于垃圾回收。destroy 方法的定义如下所示：

```
public void destroy() {
  // Finalization code...
 }
```

结构框图

下图显示了一个典型的 servlet 生命周期场景。

- 最先到达服务器的 HTTP 请求被委派到 servlet 容器。
- 在调用 service() 方法之前 servlet 容器加载 servlet。
- 然后 servlet 容器通过产生多个线程来处理多个请求，每个线程执行 servlet 的单个实例的 service() 方法。



4 实例

Servlets 是 Java 类，服务于 HTTP 请求并实现了 javax.servlet.Servlet 接口。Web 应用程序开发人员通常编写扩展 javax.servlet.http.HttpServlet 的 servlets，它是一个实现了 Servlet 接口的抽象类并且是为处理 HTTP 请求专门设计的。

Hello World 的示例代码

下面是 servlet 编写 Hello World 的示例源代码：

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class HelloWorld extends HttpServlet {
  private String message;
  public void init() throws ServletException
  {
     // Do required initialization
     message = "Hello World";
  }
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     // Set response content type
     response.setContentType("text/html");
     // Actual logic goes here.
     PrintWriter out = response.getWriter();
     out.println("<h1>" + message + "</h1>");
  }
  public void destroy()
  {
     // do nothing.
  }
}
```

编译 Servlet

让我们把上述代码放入 HelloWorld.java 文件中，并把这个文件放在 C:\ServletDevel（Windows 操作系统中）或 /usr/ServletDevel（UNIX 操作系统中）中，然后你需要将这些目录添加到 CLASSPATH 中。

假设你的环境已经正确地设置，进入 ServletDevel 目录，并编译 HelloWorld.java，如下所示：

```
$ javac HelloWorld.java
```

如果 servlet 依赖于任何其他库，你必须还要在 CLASSPATH 中包含那些 JAR 文件。我只包含了 servlet-api.jar JAR 文件，因为我在 Hello World 程序中没有使用任何其他库。

该命令行使用了来自 Sun Microsystems Java 软件开发工具包（JDK）的内置的 javac 编译器。为使该命令正常工作，你必须包含在 PATH 环境变量中使用的 Java SDK 的位置。

如果一切顺利，上述编译会在相同的目录中产生 HelloWorld.class 文件。下一节将解释在生产中如何部署一个已编译的 servlet 。

Servlet 部署

默认情况下，servlet 应用程序是位于路径 /webapps/ROOT 中的，且类文件放在 /webapps/ROOT/WEB-INF/classes 中。

如果你有一个完全合格的 com.myorg.MyServlet 的类名称，那么这个 servlet 类必须被放置在 WEB-INF/classes/com/myorg/MyServlet.class 中。

现在，让我们把 HelloWorld.class 复制到 /webapps/ROOT/WEB-INF/classes 中，并在位于 /webapps/ROOT/WEB-INF/ 的 web.xml 文件中创建以下条目：

```
<servlet>
  <servlet-name>HelloWorld</servlet-name>
  <servlet-class>HelloWorld</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>HelloWorld</servlet-name>
  <url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

上面的条目要被创建在 web.xml 文件中可用的 `<web-app>...</web-app>` 标签内。在这个表中可能已经存在各种可用的条目，但没有关系。

你基本上已经完成了，现在让我们使用 `<Tomcat-installation-directory>\bin\startup.bat`（Windows 操作系统中）或 `<Tomcat-installation-directory>/bin/startup.sh`（Linux/Solaris 等操作系统中）启动 tomcat 服务器，最后在浏览器的地址栏中输入 `http://localhost:8080/HelloWorld` 。如果一切顺利，你会看到下面的结果：


http://localhost:8080/HelloWorld - Windows Internet Explorer
http://localhost:8080/HelloWorld
File Edit View Favorites Tools Help
http://localhost:8080/HelloWorld
Hello World

5

表单数据

当你需要从浏览器到 Web 服务器传递一些信息并最终传回到后台程序时，你一定遇到了许多情况。浏览器使用两种方法向 Web 服务器传递信息。这些方法是 GET 方法和 POST 方法。

GET 方法

GET 方法向页面请求发送已编码的用户信息。页面和已编码的信息用 ? 字符分隔，如下所示：

```
http://www.test.com/hello?key1=value1&key2=value2
```

GET 方法是从浏览器向 web 服务器传递信息的默认的方法，且它会在你的浏览器的地址栏中产生一个很长的字符串。如果你向服务器传递密码或其他敏感信息，请不要使用 GET 方法。GET 方法有大小限制：请求字符串中最多只能有 1024 个字符。

这些信息使用 QUERY_STRING 头传递，并通过 QUERY_STRING 环境变量访问，Servlet 使用 doGet() 方法处理这种类型的请求。

POST 方法

一般情况下，将信息传递给后台程序的一种更可靠的方法是 POST 方法。POST 方法打包信息的方式与 GET 方法相同，但是 POST 方法不是把信息作为 URL 中 ? 字符之后的文本字符串进行发送，而是把它作为一个单独的消息发送。消息以标准输出的形式传到后台程序，你可以在你的处理过程中解析并使用这些标准输出。Servlet 使用 doPost() 方法处理这种类型的请求。

使用 Servlet 读取表单数据

Servlet 以自动解析的方式处理表单数据，根据不同的情况使用如下不同的方法：

- getParameter()：你可以调用 request.getParameter() 方法来获取表单参数的值。
- getParameterValues()：如果参数出现不止一次，那么调用该方法并返回多个值，例如复选框。
- getParameterNames()：如果你想要得到一个当前请求的所有参数的完整列表，那么调用该方法。

使用 URL 的 GET 方法实例

这是一个简单的 URL，使用 GET 方法将两个值传递给 HelloForm 程序。

```
http://localhost:8080/HelloForm?first_name=ZARA&last_name=ALI
```

下面是 HelloForm.java servlet 程序，处理由 web 浏览器给定的输入。我们将使用 getParameter() 方法，使访问传递的信息变得非常容易：

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class HelloForm extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // Set response content type
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "Using GET Method to Read Form Data";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
                "<html>\n" +
                "<head><title>" + title + "</title></head>\n" +
                "<body bgcolor=\"#f0f0f0\">\n" +
                "<h1 align=\"center\">" + title + "</h1>\n" +
                "<ul>\n" +
                "  <li><b>First Name</b>: "
                + request.getParameter("first_name") + "\n" +
                "  <li><b>Last Name</b>: "
                + request.getParameter("last_name") + "\n" +
                "</ul>\n" +
                "</body></html>");
  }
}
```

假设你的环境已经正确地设置，编译 HelloForm.java，如下所示：

```
$ javac HelloForm.java
```

如果一切顺利，上述编译会产生 HelloForm.class 文件。接下来，你需要把这个类文件复制到 <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes 中，并在 <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/ 中的 web.xml 文件中创建以下条目：

```
  <servlet>
    <servlet-name>HelloForm</servlet-name>
```

```
    <servlet-class>HelloForm</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>HelloForm</servlet-name>
    <url-pattern>/HelloForm</url-pattern>
</servlet-mapping>
```

现在在你浏览器的地址栏中输入 http://localhost:8080/HelloForm?first_name=ZARA&last_name=ALI ，并在浏览器中触发上述命令之前，确保你已经启动 Tomcat 服务器。这将产生如下所示的结果：

Using GET Method to Read Form Data

- First Name: ZARA
- Last Name: ALI

使用表单的 GET 方法实例

下面是一个简单的实例，使用 HTML 表单和提交按钮传递两个值。我们将使用相同的 Servlet HelloForm 来处理这个输入。

```
<html>
<body>
<form action="HelloForm" method="GET">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

将这个 HTML 保存到 hello.htm 文件中，并把它放在 `<Tomcat-installation-directory>/webapps/ROOT` 目录下。当你访问 `http://localhost:8080/Hello.htm` 时，下面是上述表单的实际输出。

First Name: Last Name: Submit

尝试输入姓名，然后点击提交按钮来在 tomcat 运行的本地计算机上查看结果。基于提供的输入，它会产生与上述例子中相似的结果。

使用表单的 POST 方法实例

让我们对上述 servlet 做一点修改，以便它可以处理 GET 方法和 POST 方法。下面是 **HelloForm.java** servlet 程序，使用 GET 和 POST 方法处理由 web 浏览器给出的输入。

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class HelloForm extends HttpServlet {
  // Method to handle GET method request.
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
```

```
        throws ServletException, IOException
 {
    // Set response content type
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Using GET Method to Read Form Data";
    String docType =
    "<!doctype html public \"-//w3c//dtd html 4.0 " +
    "transitional//en\">\n";
    out.println(docType +
            "<html>\n" +
            "<head><title>" + title + "</title></head>\n" +
            "<body bgcolor=\"#f0f0f0\">\n" +
            "<h1 align=\"center\">" + title + "</h1>\n" +
            "<ul>\n" +
            "  <li><b>First Name</b>: "
            + request.getParameter("first_name") + "\n" +
            "  <li><b>Last Name</b>: "
            + request.getParameter("last_name") + "\n" +
            "</ul>\n" +
            "</body></html>");
 }
 // Method to handle POST method request.
 public void doPost(HttpServletRequest request,
                HttpServletResponse response)
    throws ServletException, IOException {
   doGet(request, response);
 }
}
```

现在编译，部署上述 Servlet，并使用带有 POST 方法的 Hello.htm 测试它，如下所示：

```
<html>
<body>
<form action="HelloForm" method="POST">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

这是上述表单的实际输出，尝试输入姓名，然后点击提交按钮，在 tomcat 运行的本地计算机上查看结果。

First Name: Last Name: Submit

基于提供的输入，它会产生与上述例子中相似的结果。

将复选框数据传递到 Servlet 程序

当要选择多个选项时，就要使用复选框。

这是一个 HTML 代码实例，CheckBox.htm，一个表单带有两个复选框。

```
<html>
<body>
<form action="CheckBox" method="POST" target="_blank">
<input type="checkbox" name="maths" checked="checked" /> Maths
<input type="checkbox" name="physics"  /> Physics
<input type="checkbox" name="chemistry" checked="checked" />
                                    Chemistry
<input type="submit" value="Select Subject" />
</form>
</body>
</html>
```

这段代码的结果是如下所示的表单：

? Maths ☐ Physics ? Chemistry Select Subject

下面是 CheckBox.java servlet 程序，来为复选框按钮处理 web 浏览器给定的输入。

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class CheckBox extends HttpServlet {
  // Method to handle GET method request.
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     // Set response content type
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     String title = "Reading Checkbox Data";
     String docType =
     "<!doctype html public \"-//w3c//dtd html 4.0 " +
     "transitional//en\">\n";
     out.println(docType +
               "<html>\n" +
               "<head><title>" + title + "</title></head>\n" +
               "<body bgcolor=\"#f0f0f0\">\n" +
```

```
            "<h1 align=\"center\">" + title + "</h1>\n" +
            "<ul>\n" +
            "  <li><b>Maths Flag : </b>: "
            + request.getParameter("maths") + "\n" +
            "  <li><b>Physics Flag: </b>: "
            + request.getParameter("physics") + "\n" +
            "  <li><b>Chemistry Flag: </b>: "
            + request.getParameter("chemistry") + "\n" +
            "</ul>\n" +
            "</body></html>");
  }
  // Method to handle POST method request.
  public void doPost(HttpServletRequest request,
                  HttpServletResponse response)
     throws ServletException, IOException {
    doGet(request, response);
  }
}
```

上面的实例将显示如下所示结果：

Reading Checkbox Data

- Maths Flag : : on
- Physics Flag: : null
- Chemistry Flag: : on

读取所有的表单参数

以下是使用 HttpServletRequest 的 getParameterNames() 方法的通用实例来读取所有可用的表单参数。该方法返回一个枚举，包含了未指定顺序的参数名称。

一旦我们得到一个枚举，我们可以以标准方式循环这个枚举，使用 *hasMoreElements()* 方法来确定何时停止循环，使用 *nextElement()* 方法来获取每个参数的名称。

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
// Extend HttpServlet class
public class ReadParams extends HttpServlet {
  // Method to handle GET method request.
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     // Set response content type
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     String title = "Reading All Form Parameters";
     String docType =
     "<!doctype html public \"-//w3c//dtd html 4.0 " +
     "transitional//en\">\n";
     out.println(docType +
       "<html>\n" +
       "<head><title>" + title + "</title></head>\n" +
       "<body bgcolor=\"#f0f0f0\">\n" +
       "<h1 align=\"center\">" + title + "</h1>\n" +
       "<table width=\"100%\" border=\"1\" align=\"center\">\n" +
       "<tr bgcolor=\"#949494\">\n" +
       "<th>Param Name</th><th>Param Value(s)</th>\n"+
       "</tr>\n");
```

```
    Enumeration paramNames = request.getParameterNames();
    while(paramNames.hasMoreElements()) {
      String paramName = (String)paramNames.nextElement();
      out.print("<tr><td>" + paramName + "</td>\n<td>");
      String[] paramValues =
            request.getParameterValues(paramName);
      // Read single valued data
      if (paramValues.length == 1) {
        String paramValue = paramValues[0];
        if (paramValue.length() == 0)
          out.println("<i>No Value</i>");
        else
          out.println(paramValue);
      } else {
          // Read multiple valued data
          out.println("<ul>");
          for(int i=0; i < paramValues.length; i++) {
            out.println("<li>" + paramValues[i]);
          }
          out.println("</ul>");
      }
    }
    out.println("</tr>\n</table>\n</body></html>");
  }
  // Method to handle POST method request.
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
    throws ServletException, IOException {
    doGet(request, response);
  }
}
```

现在，用下面的表单尝试上述 servlet：

```
<html>
<body>
<form action="ReadParams" method="POST" target="_blank">
<input type="checkbox" name="maths" checked="checked" /> Maths
<input type="checkbox" name="physics"  /> Physics
<input type="checkbox" name="chemistry" checked="checked" /> Chem
<input type="submit" value="Select Subject" />
</form>
</body>
</html>
```

现在使用上述表单调用 servlet 将产生如下所示结果：

Reading All Form Parameters

Param Name	Param Value(s)
maths	on
chemistry	on

你可以尝试使用上述 servlet 来读取有其他对象的其他表单数据，比如文本框、单选按钮或下拉框等。

6

客户端 HTTP 请求

当浏览器请求网页时，它会向 web 服务器发送大量信息，这些信息不能被直接读取，因为这些信息是作为 HTTP 请求头的一部分行进的。

以下是来自浏览器端的重要的头信息，你会在 web 编程中频繁的使用：

头信息	描述
Accept	这个头信息指定浏览器或其他客户端可以处理的 MIME 类型。值 image/png 或 image/jpeg 是最常见的两种可能值。
Accept-Charset	这个头信息指定浏览器可以用来显示信息的字符集。例如 ISO-8859-1。
Accept-Encoding	这个头信息指定浏览器知道如何处理的编码类型。值 gzip 或 compress 是最常见的两种可能值。
Accept-Language	这个头信息指定客户端的首选语言，在这种情况下，Servlet 会产生多种语言的结果。例如，en、en-us、ru 等。
Authorization	这个头信息用于客户端在访问受密码保护的网页时识别自己的身份。
Connection	这个头信息指示客户端是否可以处理持久 HTTP 连接。持久连接允许客户端或其他浏览器通过单个请求来检索多个文件。值 Keep-Alive 意味着使用了持续连接。
Content-Length	这个头信息只适用于 POST 请求，并给出 POST 数据的大小（以字节为单位）。
Cookie	这个头信息把之前发送到浏览器的 cookies 返回到服务器。
Host	这个头信息指定原始的 URL 中的主机和端口。
If-Modified-Since	这个头信息表示只有当页面在指定的日期后已更改时，客户端想要的页面。如果没有新的结果可以使用，服务器会发送一个 304 代码，表示 Not Modified 头信息。
If-Unmodified-Since	这个头信息是 If-Modified-Since 的对立面，它指定只有当文档早于指定日期时，操作才会成功。
Referer	这个头信息指示所指向的 Web 页的 URL。例如，如果您在网页 1，点击一个链接到网页 2，当浏览器请求网页 2 时，网页 1 的 URL 就会包含在 Referer 头信息中。
User-Agent	这个头信息识别发出请求的浏览器或其他客户端，并可以向不同类型的浏览器返回不同的内容。

读取 HTTP 头信息的方法

下述方法可以用于读取 servlet 程序中的 HTTP 头信息。通过 *HttpServletRequest* 对象这些方法是可用的。

序号	方法 & 描述
1	Cookie[] getCookies() 返回一个数组，包含客户端发送该请求的所有的 Cookie 对象。

2	Enumeration getAttributeNames() 返回一个枚举，包含提供给该请求可用的属性名称。
3	Enumeration getHeaderNames() 返回一个枚举，包含在该请求中包含的所有的头名。
4	Enumeration getParameterNames() 返回一个 String 对象的枚举，包含在该请求中包含的参数的名称。
5	HttpSession getSession() 返回与该请求关联的当前 session 会话，或者如果请求没有 session 会话，则创建一个。
6	HttpSession getSession(boolean create) 返回与该请求关联的当前 HttpSession，或者如果没有当前会话，且创建是真的，则返回一个新的 session 会话。
7	Locale getLocale() 基于 Accept-Language 头，返回客户端接受内容的首选的区域设置。
8	Object getAttribute(String name) 以对象形式返回已命名属性的值，如果没有给定名称的属性存在，则返回 null。
9	ServletInputStream getInputStream() 使用 ServletInputStream，以二进制数据形式检索请求的主体。
10	String getAuthType()

	返回用于保护 Servlet 的身份验证方案的名称，例如，“BASIC” 或 “SSL”，如果JSP没有受到保护则返回 null。
11	String getCharacterEncoding() 返回请求主体中使用的字符编码的名称。
12	String getContentType() 返回请求主体的 MIME 类型，如果不知道类型则返回 null。
13	String getContextPath() 返回指示请求上下文的请求 URI 部分。
14	String getHeader(String name) 以字符串形式返回指定的请求头的值。
15	String getMethod() 返回请求的 HTTP 方法的名称，例如，GET、POST 或 PUT。
16	String getParameter(String name) 以字符串形式返回请求参数的值，或者如果参数不存在则返回 null。
17	String getPathInfo() 当请求发出时，返回与客户端发送的 URL 相关的任何额外的路径信息。
18	String getProtocol() 返回请求协议的名称和版本。

19	String getQueryString() 返回包含在路径后的请求 URL 中的查询字符串。
20	String getRemoteAddr() 返回发送请求的客户端的互联网协议（IP）地址。
21	String getRemoteHost() 返回发送请求的客户端的完全限定名称。
22	String getRemoteUser() 如果用户已通过身份验证，则返回发出请求的登录用户，或者如果用户未通过身份验证，则返回 null。
23	String getRequestURI() 从协议名称直到 HTTP 请求的第一行的查询字符串中，返回该请求的 URL 的一部分。
24	String getRequestedSessionId() 返回由客户端指定的 session 会话 ID。
25	String getServletPath() 返回调用 JSP 的请求的 URL 的一部分。
26	String[] getParameterValues(String name) 返回一个字符串对象的数组，包含所有给定的请求参数的值，如果参数不存在则返回 null。
27	boolean isSecure()

	返回一个布尔值，指示请求是否使用安全通道，如 HTTPS。
28	int getContentLength() 以字节为单位返回请求主体的长度，并提供输入流，或者如果长度未知则返回 -1。
29	int getIntHeader(String name) 返回指定的请求头的值为一个 int 值。
30	int getServerPort() 返回接收到这个请求的端口号。

HTTP 头信息请求实例：

下述例子使用了 HttpServletRequest 的 **getHeaderNames()** 方法来读取 HTTP 头信息。该方法返回了一个枚举，包含与当前的 HTTP 请求相关的头信息。

一旦我们得到一个枚举，我们可以以标准方式循环这个枚举，使用 hasMoreElements() 方法来确定何时停止循环，使用 *nextElement()* 方法来获取每个参数的名称。

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
// Extend HttpServlet class
public class DisplayHeader extends HttpServlet {
  // Method to handle GET method request.
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     // Set response content type
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     String title = "HTTP Header Request Example";
     String docType =
```

```
    "<!doctype html public \"-//w3c//dtd html 4.0 " +
    "transitional//en\">\n";
    out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n"+
      "<body bgcolor=\"#f0f0f0\">\n" +
      "<h1 align=\"center\">" + title + "</h1>\n" +
      "<table width=\"100%\" border=\"1\" align=\"center\">\n" +
      "<tr bgcolor=\"#949494\">\n" +
      "<th>Header Name</th><th>Header Value(s)</th>\n"+
      "</tr>\n");
    Enumeration headerNames = request.getHeaderNames();
    while(headerNames.hasMoreElements()) {
       String paramName = (String)headerNames.nextElement();
       out.print("<tr><td>" + paramName + "</td>\n");
       String paramValue = request.getHeader(paramName);
       out.println("<td> " + paramValue + "</td></tr>\n");
    }
    out.println("</table>\n</body></html>");
  }
  // Method to handle POST method request.
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
     throws ServletException, IOException {
    doGet(request, response);
  }
}
```

现在，调用上述 servlet 会产生如下所示的结果：

HTTP Header Request Example

Header Name	Header Value(s)
accept	*/*
accept-language	en-us
user-agent	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; InfoPath.2; MS-RTC LM 8)
accept-encoding	gzip, deflate
host	localhost:8080
connection	Keep-Alive
cache-control	no-cache

7

服务器 HTTP 响应

正如在前面的章节中讨论的一样，当一个 Web 服务器对浏览器响应一个 HTTP 请求时，响应通常包括一个状态行、一些响应头信息、一个空行和文档。一个典型的响应如下所示：

```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
  (Blank Line)
<!doctype ...>
<html>
<head>...</head>
<body>
...
</body>
</html>
```

状态行包括 HTTP 版本（例子中的 HTTP/1.1）、一个状态码（例子中的 200）和一个对应于状态码的短消息（例子中的 OK）。

下面是从 web 服务器端返回到浏览器的最有用的 HTTP 1.1 响应头信息的总结，且在 web 编程中你会频繁地使用它们：

头信息	描述
Allow	这个头信息指定服务器支持的请求方法（GET、POST 等）。
Cache-Control	这个头信息指定响应文档在何种情况下可以安全地缓存。可能的值有：public, private 或 no-cache 等。Public 意味着文档是可缓存，Private 意味着文档是单个用户私用文档，且只能存储在私有（非共享）缓存中，no-cache 意味着文档不应被缓存。
Connection	这个头信息指示浏览器是否使用持久 HTTP 连接。值 close 指示浏览器不使用持久 HTTP 连接，值 keep-alive 意味着使用持久连接。
Content-Disposition	这个头信息可以让您请求浏览器要求用户以给定名称的文件把响应保存到磁盘。
Content-Encoding	在传输过程中，这个头信息指定页面的编码方式。
Content-Language	这个头信息表示文档编写所使用的语言。例如，en、en-us、ru 等。
Content-Length	这个头信息指示响应中的字节数。只有当浏览器使用持久（keep-alive）HTTP 连接时才需要这些信息。
Content-Type	这个头信息提供了响应文档的 MIME（Multipurpose Internet Mail Extension）类型。
Expires	这个头信息指定内容过期的时间，在这之后内容不再被缓存。
Last-Modified	这个头信息指示文档的最后修改时间。然后，客户端可以缓存文件，并在以后的请求中通过 If-Modified-Since 请求头信息提供一个日期。
Location	这个头信息应被包含在所有的带有状态码的响应中。在 300 s 内，这会通知浏览器文档的地址。浏览器会自动重新连接到这个位置，并获取新的文档。
Refresh	这个头信息指定浏览器应该如何尽快请求更新的页面。您可以指定页面刷新的秒数。

Retry-After	这个头信息可以与 503（服务不可用）响应配合使用，这会告诉客户端多久就可以重复它的请求。
Set-Cookie	这个头信息指定一个与页面关联的 cookie。

设置 HTTP 响应头信息的方法

下面的方法可用于在 servlet 程序中设置 HTTP 响应头信息。通过 *HttpServletResponse* 对象这些方法是可用的。

序号	方法 & 描述
1	String encodeRedirectURL(String url) 为 sendRedirect 方法中使用的指定的 URL 进行编码，或者如果编码不是必需的，则返回 URL 未改变。
2	String encodeURL(String url) 对包含 session 会话 ID 的指定 URL 进行编码，或者如果编码不是必需的，则返回 URL 未改变。
3	boolean containsHeader(String name) 返回一个布尔值，指示是否已经设置已命名的响应头信息。
4	boolean isCommitted() 返回一个布尔值，指示响应是否已经提交。
5	void addCookie(Cookie cookie) 把指定的 cookie 添加到响应。
6	void addDateHeader(String name, long date) 添加一个带有给定的名称和日期值的响应头信息。
7	void addHeader(String name, String value) 添加一个带有给定的名称和值的响应头信息。
8	void addIntHeader(String name, int value)

	添加一个带有给定的名称和整数值的响应头信息。
9	void flushBuffer() 强制任何在缓冲区中的内容被写入到客户端。
10	void reset() 清除缓冲区中存在的任何数据，包括状态码和头信息。
11	void resetBuffer() 清除响应中基础缓冲区的内容，不清除状态码和头信息。
12	void sendError(int sc) 使用指定的状态码发送错误响应到客户端，并清除缓冲区。
13	void sendError(int sc, String msg) 使用指定的状态发送错误响应到客户端。
14	void sendRedirect(String location) 使用指定的重定向位置 URL 发送临时重定向响应到客户端。
15	void setBufferSize(int size) 为响应主体设置首选的缓冲区大小。
16	void setCharacterEncoding(String charset) 设置被发送到客户端的响应的字符编码（MIME 字符集）例如，UTF-8。
17	void setContentLength(int len) 设置在 HTTP Servlet 响应中的内容主体的长度，该方法设置 HTTP Content-Length 头信息。
18	void setContentType(String type) 如果响应还未被提交，设置被发送到客户端的响应的内容类型。
19	void setDateHeader(String name, long date)

	设置一个带有给定的名称和日期值的响应头信息。
20	void setHeader(String name, String value) 设置一个带有给定的名称和值的响应头信息。
21	void setIntHeader(String name, int value) 设置一个带有给定的名称和整数值的响应头信息。
22	void setLocale(Locale loc) 如果响应还未被提交，设置响应的区域。
23	void setStatus(int sc) 为该响应设置状态码。

HTTP 头信息响应实例

在前面的实例中你已经了解了 setContentType() 方法的工作方式，下面的实例也会用到同样的方法，此外，我们会用 setIntHeader() 方法来设置 Refresh 头信息。

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
// Extend HttpServlet class
public class Refresh extends HttpServlet {
  // Method to handle GET method request.
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     // Set refresh, autoload time as 5 seconds
     response.setIntHeader("Refresh", 5);
     // Set response content type
     response.setContentType("text/html");
     // Get current time
     Calendar calendar = new GregorianCalendar();
     String am_pm;
```

```
    int hour = calendar.get(Calendar.HOUR);
    int minute = calendar.get(Calendar.MINUTE);
    int second = calendar.get(Calendar.SECOND);
    if(calendar.get(Calendar.AM_PM) == 0)
      am_pm = "AM";
    else
      am_pm = "PM";
    String CT = hour+":"+ minute +":"+ second +" "+ am_pm;
    PrintWriter out = response.getWriter();
    String title = "Auto Refresh Header Setting";
    String docType =
    "<!doctype html public \"-//w3c//dtd html 4.0 " +
    "transitional//en\">\n";
    out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n"+
      "<body bgcolor=\"#f0f0f0\">\n" +
      "<h1 align=\"center\">" + title + "</h1>\n" +
      "<p>Current Time is: " + CT + "</p>\n");
  }
  // Method to handle POST method request.
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
     throws ServletException, IOException {
    doGet(request, response);
  }
}
```

现在调用上述 servlet，每 5 秒后就会显示当前的系统时间，如下所示。运行 servlet 并等着看结果：

Auto Refresh Header Setting

Current Time is: 9:44:50 PM

8

HTTP 状态码

HTTP 请求的格式和 HTTP 响应消息的格式是相似的且都有如下所示结构：

- 一个初始状态行 + CRLF(回车 + 换行 即新行)
- 零个或多个标题行 + CRLF
- 一个空白行，即一个 CRLF
- 一个可选的消息主体，如文件、查询数据或查询输出

例如，服务器的响应头信息看起来如下所示：

```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
  (Blank Line)
<!doctype ...>
<html>
<head>...</head>
<body>
...
</body>
</html>
```

状态行包括 HTTP 版本（例子中的 HTTP/1.1）、一个状态码（例子中的 200）和一个对应于状态码的短消息（例子中的 OK）。

以下 HTTP 状态码以及可能从 Web 服务器返回的相关的消息的列表：

代码：	消息：	描述：
100	Continue	只有请求的一部分已经被服务器接收，但只要它没有被拒绝，客户端应继续该请求。
101	Switching Protocols	服务器切换协议。
200	OK	请求成功。
201	Created	该请求是完整的，并创建一个新的资源。
202	Accepted	该请求被接受处理，但是该处理是不完整的。
203	Non-authoritative Information	
204	No Content	
205	Reset Content	
206	Partial Content	
300	Multiple Choices	链接列表。用户可以选择一个链接，进入到该位置。最多五个地址
301	Moved Permanently	所请求的页面已经转移到一个新的 URL。
302	Found	所请求的页面已经临时转移到一个新的 URL。
303	See Other	所请求的页面可以在另一个不同的 URL 下被找到。

304	Not Modified	
305	Use Proxy	
306	*Unused*	在以前的版本中使用该代码。现在已不再使用它，但代码仍被保留。
307	Temporary Redirect	所请求的页面已经临时转移到一个新的 URL。
400	Bad Request	服务器不理解请求。
401	Unauthorized	所请求的页面需要用户名和密码。
402	Payment Required	*你还不能使用该代码。*
403	Forbidden	禁止访问所请求的页面。
404	Not Found	服务器无法找到所请求的页面。
405	Method Not Allowed	在请求中指定的方法是不允许的。
406	Not Acceptable	服务器只生成一个不被客户端接受的响应。
407	Proxy Authentication Required	在请求送达之前，您必须使用代理服务器的验证。
408	Request Timeout	请求需要的时间比服务器能够等待的时间长，超时。
409	Conflict	请求因为冲突无法完成。
410	Gone	所请求的页面不再可用。
411	Length Required	"Content-Length" 未定义。服务器无法处理客户端发送的不带 Content-Length 的请求信息。
412	Precondition Failed	请求中给出的先决条件被服务器评估为 false。
413	Request Entity Too Large	服务器不接受该请求，因为请求实体过大。
414	Request-url Too Long	服务器不接受该请求，因为 URL 太长。当你转换一个 “post” 请求为一个带有长的查询信息的 “get” 请求时发生。
415	Unsupported Media Type	服务器不接受该请求，因为媒体类型不被支持。
417	Expectation Failed	
500	Internal Server Error	未完成的请求。服务器遇到了一个意外的情况。
501	Not Implemented	未完成的请求。服务器不支持所需的功能。
502	Bad Gateway	未完成的请求。服务器从上游服务器收到无效响应。
503	Service Unavailable	未完成的请求。服务器暂时超载或死机。
504	Gateway Timeout	网关超时。
505	HTTP Version Not Supported	服务器不支持“HTTP协议”版本。

设置 HTTP 状态码的方法：

下面是在 servlet 程序中可以用于设置 HTTP 状态码的方法。通过 *HttpServletResponse* 对象这些方法是可用的。

序号	方法&描述

1	public void setStatus (int statusCode) 该方法设置一个任意的状态码。setStatus 方法接受一个 int（状态码）作为参数。如果您的反应包含了一个特殊的状态码和文档，请确保在使用 *PrintWriter* 实际返回任何内容之前调用 setStatus。
2	public void sendRedirect(String url) 该方法生成一个 302 响应，连同一个带有新文档 URL 的 *Location* 头。
3	public void sendError(int code, String message) 该方法发送一个状态码（通常为 404），连同一个在 HTML 文档内部自动格式化并发送到客户端的短消息。

HTTP 状态码实例：

下述例子将发送 407 错误代码到客户端浏览器，且浏览器会向你显示 “需要身份验证！！！”的消息。

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
// Extend HttpServlet class
public class showError extends HttpServlet {
  // Method to handle GET method request.
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     // Set error code and reason.
     response.sendError(407, "Need authentication!!!" );
  }
  // Method to handle POST method request.
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
     throws ServletException, IOException {
    doGet(request, response);
```

```
 }
}
```

现在调用上述 servlet 会显示如下所示结果：

HTTP Status 407 - Need authentication!!!

type Status report

message Need authentication!!!

description The client must first authenticate itself with the proxy (Need authentication!!!).

Apache Tomcat/5.5.29

9

编写过滤器

Servlet 过滤器是Java 类，可用于 Servlet 编程中的下述目的：

- 在它们访问后端资源之前，拦截这些来自客户端的请求。
- 在它们发送回客户端之前，处理这些来自服务器端的响应。

这是规范建议的各种类型的过滤器：

- 身份验证过滤器。
- 数据压缩过滤器。
- 加密过滤器。
- 触发访问事件资源的过滤器。
- 图像转换过滤器。
- 日志记录和审核过滤器。
- MIME-类型链过滤器。
- Tokenizing 过滤器。
- 转换 XML 内容的 XSL/T 过滤器。

过滤器在部署描述符文件 web.xml 中被部署，然后被映射到 servlet 名称或你的应用程序的部署描述符中的 URL 模式。

当 web 容器启动你的 web 应用程序时，它会为每个在部署描述符中已声明的过滤器创建一个实例。过滤器按照它们在部署描述符中声明的顺序执行。

Servlet 过滤器方法

过滤器仅仅是一个实现了 javax.servlet.Filter 接口的 Java 类。javax.servlet.Filter 接口定义了三种方法：

序号	方法 & 描述
1	public void doFilter (ServletRequest, ServletResponse, FilterChain) 该方法在每次一个请求/响应对因客户端在链的末端请求资源而通过链传递时由容器调用。

2	public void init(FilterConfig filterConfig) 该方法由 Web 容器调用，指示一个过滤器被放入服务。
3	public void destroy() 该方法由 Web 容器调用，指示一个过滤器从服务被去除。

Servlet 过滤器实例

以下是 Servlet 过滤器的实例，将输出客户端的 IP 地址和当前的日期时间。这个例子使你对 Servlet 过滤器有了基本的了解，但是你可以使用相同的概念编写更复杂的过滤器应用程序：

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
// Implements Filter class
public class LogFilter implements Filter  {
  public void  init(FilterConfig config)
                    throws ServletException{
    // Get init parameter
    String testParam = config.getInitParameter("test-param");
    //Print the init parameter
    System.out.println("Test Param: " + testParam);
  }
  public void  doFilter(ServletRequest request,
              ServletResponse response,
              FilterChain chain)
              throws java.io.IOException, ServletException {
    // Get the IP address of client machine.
    String ipAddress = request.getRemoteAddr();
    // Log the IP address and current timestamp.
    System.out.println("IP "+ ipAddress + ", Time "
                              + new Date().toString());
    // Pass request back down the filter chain
    chain.doFilter(request,response);
  }
  public void destroy( ){
    /* Called before the Filter instance is removed
```

```
    from service by the web container*/
  }
}
```

用常用的方式编译 LogFilter.java 并把你的类文件放入 /webapps/ROOT/WEB-INF/classes 中。

Web.xml 中的 Servlet 过滤器映射

过滤器被定义然后被映射到一个 URL 或 Servlet中，这与Servlet被定义然后映射到一个 URL 模式中的方法是相同的。为在部署描述符文件 web.xml 中过滤器标签创建如下所示条目：

```
<filter>
  <filter-name>LogFilter</filter-name>
  <filter-class>LogFilter</filter-class>
  <init-param>
    <param-name>test-param</param-name>
    <param-value>Initialization Paramter</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>LogFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

上述过滤器可以应用到所有的 servlet 中，因为在配置中我们指定了 /* 。如果你只想在少数的 servlet 中应用过滤器，那么你可以指定一个特定的 servlet 路径。

现在尝试用常用的方式调用任何 servlet，然后你将会在 web 服务器日志中看到生成的日志。你也可以使用 Log 4J 记录器来在一个单独的文件中记录上述日志。

使用多个过滤器

Web 应用程序可以定义多个带有不同目的的过滤器。考虑这种情况，你定义了两个过滤器 *AuthenFilter* 和 *Log Filter*。除了你需要创建一个如下所述的不同的映射之外，其余的处理与上述解释的一样：

```
<filter>
  <filter-name>LogFilter</filter-name>
  <filter-class>LogFilter</filter-class>
  <init-param>
    <param-name>test-param</param-name>
    <param-value>Initialization Paramter</param-value>
  </init-param>
```

```
</filter>
<filter>
  <filter-name>AuthenFilter</filter-name>
  <filter-class>AuthenFilter</filter-class>
  <init-param>
    <param-name>test-param</param-name>
    <param-value>Initialization Paramter</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>LogFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>AuthenFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

过滤器的应用顺序

web.xml 中的 filter-mapping 元素的顺序决定了 web 容器把过滤器应用到 servlet 的顺序。若要反转过滤器的顺序，你只需要在 web.xml 文件中反转 filter-mapping 元素即可。

例如，上述实例首先应用 LogFilter然后再应用 AuthenFilter 到任何 servlet 中，但是下述实例将反转这个顺序：

```
<filter-mapping>
  <filter-name>AuthenFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>LogFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

异常处理

当一个 servlet 抛出一个异常时，web 容器在使用了 exception-type 元素的 web.xml 中搜索与抛出的异常类型相匹配的配置。

你必须在 web.xml 中使用 error-page 元素来指定 servlet 调用，作为对特定的**异常**或 HTTP **状态码**作出的响应。

web.xml 配置

考虑这种情况，你有一个 *ErrorHandler* servelt，当任何已定义的异常或错误出现时就会被调用。以下是要在 web.xml 中创建的条目。

```
<!-- servlet definition -->
<servlet>
      <servlet-name>ErrorHandler</servlet-name>
      <servlet-class>ErrorHandler</servlet-class>
</servlet>
<!-- servlet mappings -->
<servlet-mapping>
      <servlet-name>ErrorHandler</servlet-name>
      <url-pattern>/ErrorHandler</url-pattern>
</servlet-mapping>

<!-- error-code related error pages -->
<error-page>
   <error-code>404</error-code>
   <location>/ErrorHandler</location>
</error-page>
<error-page>
   <error-code>403</error-code>
   <location>/ErrorHandler</location>
</error-page>

<!-- exception-type related error pages -->
<error-page>
   <exception-type>
        javax.servlet.ServletException
   </exception-type >
   <location>/ErrorHandler</location>
</error-page>

<error-page>
   <exception-type>java.io.IOException</exception-type >
   <location>/ErrorHandler</location>
</error-page>
```

如果你想对所有的异常有一个通用的错误处理程序，那么你应该定义如下所示的 error-page，而不是为每个异常定义单独的 error-page 元素：

```
<error-page>
   <exception-type>java.lang.Throwable</exception-type >
   <location>/ErrorHandler</location>
</error-page>
```

以下是异常处理中有关上述 web.xml 需要注意的点：

- Servelt ErrorHandler 与其他的 servelt 的定义方式一样，且在 web.xml 中配置。
- 如果状态码有任何错误出现，不管是 404（未找到）还是 403（禁止），那么 ErrorHandler servlet 会被调用。
- 如果 web 应用程序抛出 *ServletException* 或 *IOException*，那么 web 容器就会调用 /ErrorHandler servlet。
- 你可以定义不同的错误处理程序来处理不同类型的错误或异常。上述例子非常通用，希望它达到了让你理解基本概念的目的。

请求属性 – 错误/异常

以下是错误处理 servlet 可以访问的请求属性列表，用来分析错误/异常的性质。

序号	属性 & 描述
1	javax.servlet.error.status_code 该属性给出状态码，状态码可被存储，并在存储为 java.lang.Integer 数据类型后可被分析。
2	javax.servlet.error.exception_type 该属性给出异常类型的信息，异常类型可被存储，并在存储为 java.lang.Class 数据类型后可被分析。
3	javax.servlet.error.message 该属性给出确切错误消息的信息，信息可被存储，并在存储为 java.lang.String 数据类型后可被分析。
4	javax.servlet.error.request_uri 该属性给出有关 URL 调用 Servlet 的信息，信息可被存储，并在存储为 java.lang.String 数据类型后可被分析。

5	javax.servlet.error.exception 该属性给出异常产生的信息，信息可被存储，并在存储为 java.lang.Throwable 数据类型后可被分析。
6	javax.servlet.error.servlet_name 该属性给出 servlet 的名称，名称可被存储，并在存储为 java.lang.String 数据类型后可被分析。

Servlet 错误处理程序实例

以下是 Servlet 实例，用于任何你定义的 servlet 出现任何错误或异常时的情况。

这个例子让你对 Servlet 中的异常处理有了基本的了解，但是你可以使用相同的概念编写更复杂的异常处理应用程序：

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
// Extend HttpServlet class
public class ErrorHandler extends HttpServlet {
  // Method to handle GET method request.
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     // Analyze the servlet exception
     Throwable throwable = (Throwable)
     request.getAttribute("javax.servlet.error.exception");
     Integer statusCode = (Integer)
     request.getAttribute("javax.servlet.error.status_code");
     String servletName = (String)
     request.getAttribute("javax.servlet.error.servlet_name");
     if (servletName == null){
        servletName = "Unknown";
     }
     String requestUri = (String)
     request.getAttribute("javax.servlet.error.request_uri");
```

```
    if (requestUri == null){
      requestUri = "Unknown";
    }
    // Set response content type
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Error/Exception Information";
    String docType =
    "<!doctype html public \"-//w3c//dtd html 4.0 " +
    "transitional//en\">\n";
    out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n" +
      "<body bgcolor=\"#f0f0f0\">\n");
    if (throwable == null && statusCode == null){
      out.println("<h2>Error information is missing</h2>");
      out.println("Please return to the <a href=\"" +
        response.encodeURL("http://localhost:8080/") +
        "\">Home Page</a>.");
    }else if (statusCode != null){
      out.println("The status code : " + statusCode);
    }else{
      out.println("<h2>Error information</h2>");
      out.println("Servlet Name : " + servletName +
                  "</br></br>");
      out.println("Exception Type : " +
                  throwable.getClass( ).getName( ) +
                  "</br></br>");
      out.println("The request URI: " + requestUri +
                  "<br><br>");
      out.println("The exception message: " +
                    throwable.getMessage( ));
    }
    out.println("</body>");
    out.println("</html>");
  }
  // Method to handle POST method request.
  public void doPost(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    doGet(request, response);
  }
}
```

以常用的方式编译 ErrorHandler.java 并把你的类文件放入/webapps/ROOT/WEB-INF/classes 中。

让我们将下述配置添加到 web.xml 文件中来处理异常：

```
<servlet>
      <servlet-name>ErrorHandler</servlet-name>
      <servlet-class>ErrorHandler</servlet-class>
</servlet>
<!-- servlet mappings -->
<servlet-mapping>
      <servlet-name>ErrorHandler</servlet-name>
      <url-pattern>/ErrorHandler</url-pattern>
</servlet-mapping>
<error-page>
   <error-code>404</error-code>
   <location>/ErrorHandler</location>
</error-page>
<error-page>
   <exception-type>java.lang.Throwable</exception-type >
   <location>/ErrorHandler</location>
</error-page>
```

现在，尝试使用一个会产生任何异常的 servlet 或者输入一个错误的 URL，这将触发 Web 容器调用 ErrorHandler servlet 并显示适当的消息。例如，如果你输入了一个错误的 URL，那么它将显示如下所示的结果：

```
The status code : 404
```

上述代码在一些 web 浏览器中可能无法工作。因此请尝试使用 Mozilla 和 Safari 浏览器，这样上述代码应该能正常工作。

11

Cookies 处理

Cookies 是存储在客户端计算机上的文本文件，用于各种信息的跟踪目的。Java Servlet 透明的支持 HTTP Cookies。

识别返回用户包括以下三个步骤：

- 服务器脚本向浏览器发送一组 cookies。例如姓名、年龄或身份证号码等。
- 浏览器将这些信息存储在本地计算机中以备将来使用。
- 当下次浏览器向 web 服务器发送任何请求时，它会把这些 cookies 信息发送到服务器，服务器使用这些信息来识别用户。

本章教你如何设置或重置 cookies，如何访问它们，以及如何删除它们。

Cookie 剖析

通常情况下，Cookies 设置在 HTTP 头信息中（尽管 JavaScript 也可以直接在浏览器上设置 cookie）。设置 cookie 的 servlet 可能会发送如下所示的头信息：

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name=xyz; expires=Friday, 04-Feb-07 22:03:38 GMT;
            path=/; domain=tutorialspoint.com
Connection: close
Content-Type: text/html
```

正如你所看到的，Set-Cookie 头信息包含了一个名称值对、一个 GMT 日期、一个路径和一个域。名称和值会被 URL 编码。有效期字段指示浏览器在给定的时间和日期之后“忘记”该 cookie。

如果浏览器被配置为存储 cookies，它将会把这个信息保留到截止日期。如果用户在任何与该 cookie 的路径和域匹配的页面点击浏览器，它就会将这个 cookie 重新发送到服务器。浏览器的头信息可能如下所示：

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name=xyz
```

之后 servlet 就能够通过请求方法 request.getCookies() 访问 cookie，该方法将返回一个 *Cookie* 对象的数组。

Servlet Cookies 方法

以下是在 servlet 中操作 cookies 时你可能会用到的有用的方法列表。

序号	方法 & 描述
1	public void setDomain(String pattern) 该方法设置 cookie 适用的域，例如 tutorialspoint.com。
2	public String getDomain() 该方法获取 cookie 适用的域，例如 tutorialspoint.com。
3	public void setMaxAge(int expiry) 该方法设置 cookie 过期的时间（以秒为单位）。如果不这样设置，cookie 只会在当前 session 会话中持续有效。
4	public int getMaxAge() 该方法返回 cookie 的最大生存周期（以秒为单位），默认情况下，-1 表示 cookie 将持续下去，直到浏览器关闭。
5	public String getName() 该方法返回 cookie 的名称。名称在创建后不能改变。
6	public void setValue(String newValue) 该方法设置与 cookie 关联的值。

7	public String getValue() 该方法获取与 cookie 关联的值。
8	public void setPath(String uri) 该方法设置 cookie 适用的路径。如果您不指定路径，与当前页面相同目录下的（包括子目录下的）所有 URL 都会返回 cookie。
9	public String getPath() 该方法获取 cookie 适用的路径。
10	public void setSecure(boolean flag) 该方法设置布尔值，表示 cookie 是否应该只在加密的（即 SSL）连接上发送。
11	public void setComment(String purpose) 该方法规定了描述 cookie 目的的注释。该注释在浏览器向用户呈现 cookie 时非常有用。
12	public String getComment() 该方法返回了描述 cookie 目的的注释，如果 cookie 没有注释则返回 null。

用 Servlet 设置 Cookies

用 servlet 设置 cookies 包括三个步骤：

(1) 创建一个 Cookie 对象：用 cookie 名和 cookie 值调用 Cookie 构造函数，cookie 名和 cookie 值都是字符串。

```
Cookie cookie = new Cookie("key","value");
```

记住，无论是名字还是值，都不应该包含空格和以下任何字符：

```
[ ] ( ) = , " / ? @ : ;
```

(2) 设置最长有效期：你可以使用 setMaxAge 方法来指定 cookie 有效的时间（以秒为单位）。下面是设置了一个最长有效期为 24 小时的 cookie。

```
cookie.setMaxAge(60*60*24);
```

(3) 发送 Cookie 到 HTTP 响应头：你可以使用 response.addCookie 来在 HTTP 响应头中添加 cookies，如下所示：

```
response.addCookie(cookie);
```

实例：

让我们修改我们的 **表单实例** 来为姓名设置 cookies。

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class HelloForm extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     // Create cookies for first and last names.
     Cookie firstName = new Cookie("first_name",
                   request.getParameter("first_name"));
     Cookie lastName = new Cookie("last_name",
                   request.getParameter("last_name"));
     // Set expiry date after 24 Hrs for both the cookies.
     firstName.setMaxAge(60*60*24);
     lastName.setMaxAge(60*60*24);
     // Add both the cookies in the response header.
     response.addCookie( firstName );
     response.addCookie( lastName );
     // Set response content type
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     String title = "Setting Cookies Example";
     String docType =
     "<!doctype html public \"-//w3c//dtd html 4.0 " +
     "transitional//en\">\n";
```

```
    out.println(docType +
            "<html>\n" +
            "<head><title>" + title + "</title></head>\n" +
            "<body bgcolor=\"#f0f0f0\">\n" +
            "<h1 align=\"center\">" + title + "</h1>\n" +
            "<ul>\n" +
            "  <li><b>First Name</b>: "
            + request.getParameter("first_name") + "\n" +
            "  <li><b>Last Name</b>: "
            + request.getParameter("last_name") + "\n" +
            "</ul>\n" +
            "</body></html>");
  }
}
```

编译上述 servlet **HelloForm** 并在 web.xml 文件中创建适当的条目，最后尝试使用下述 HTML 页面来调用 servlet。

```
<html>
<body>
<form action="HelloForm" method="GET">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

将上述 HTML 内容保存到文件 hello.htm 中并把它放在 `<Tomcat-installation-directory>/webapps/ROOT` 目录中。当你访问 `http://localhost:8080/Hello.htm` 时，上述表单的实际输出如下所示：

First Name: [ ]

Last Name: [ ] Submit

尝试输入姓名，然后点击提交按钮。这将在你的屏幕上显示姓名，同时会设置 firstName 和 lastName 这两个 cookies，当下次你点击提交按钮时，会将这两个 cookies 传回到服务器。

下一节会解释如何在你的 web 应用程序中访问这些 cookies。

用 Servlet 读取 Cookies

要读取 cookies，你需要通过调用 *HttpServletRequest* 的 **getCookies( )** 方法创建一个 *javax.servlet.http.Cookie* 对象的数组。然后循环遍历数组，并使用 getName() 和 getValue() 方法来访问每个 cookie 及其相关的值。

实例

让我们读取上述例子中已经设置的 cookies：

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class ReadCookies extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     Cookie cookie = null;
     Cookie[] cookies = null;
     // Get an array of Cookies associated with this domain
     cookies = request.getCookies();
     // Set response content type
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     String title = "Reading Cookies Example";
     String docType =
     "<!doctype html public \"-//w3c//dtd html 4.0 " +
     "transitional//en\">\n";
     out.println(docType +
                "<html>\n" +
                "<head><title>" + title + "</title></head>\n" +
                "<body bgcolor=\"#f0f0f0\">\n" );
     if( cookies != null ){
       out.println("<h2> Found Cookies Name and Value</h2>");
       for (int i = 0; i < cookies.length; i++){
         cookie = cookies[i];
         out.print("Name : " + cookie.getName( ) + ",  ");
         out.print("Value: " + cookie.getValue( )+" <br/>");
       }
     }else{
        out.println(
          "<h2>No cookies founds</h2>");
     }
     out.println("</body>");
     out.println("</html>");
```

```
  }
}
```

编译上述 servlet ReadCookies 并在 web.xml 文件中创建适当的条目。如果你已经设置了 first _ name cookie 为 “John”，last _ name cookie 为 “Player”，那么尝试运行 `http://localhost:8080/ReadCookies`，将显示如下所示结果：

Found Cookies Name and Value

Name : first_name, Value: John

Name : last_name, Value: Player

用 Servlet 删除 Cookies

删除 cookies 非常简单。如果你想删除一个 cookie，那么只需要按照如下所示的三个步骤进行：

- 读取一个现存的 cookie 并把它存储在 Cookie 对象中。
- 使用 setMaxAge() 方法设置 cookie 的年龄为零来删除一个现存的 cookie。
- 将这个 cookie 添加到响应z中。

实例

下述例子将删除一个现存的命名为 “first _ name” 的 cookie，且当你下次运行 ReadCookies servlet 时，它会为 first _ name 返回空值。

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class DeleteCookies extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     Cookie cookie = null;
     Cookie[] cookies = null;
     // Get an array of Cookies associated with this domain
     cookies = request.getCookies();
     // Set response content type
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     String title = "Delete Cookies Example";
     String docType =
     "<!doctype html public \"-//w3c//dtd html 4.0 " +
```

```
    "transitional//en\">\n";
    out.println(docType +
            "<html>\n" +
            "<head><title>" + title + "</title></head>\n" +
            "<body bgcolor=\"#f0f0f0\">\n" );
     if( cookies != null ){
       out.println("<h2> Cookies Name and Value</h2>");
       for (int i = 0; i < cookies.length; i++){
          cookie = cookies[i];
          if((cookie.getName( )).compareTo("first_name") == 0 ){
              cookie.setMaxAge(0);
              response.addCookie(cookie);
              out.print("Deleted cookie : " +
                        cookie.getName( ) + "<br/>");
          }
          out.print("Name : " + cookie.getName( ) + ",  ");
          out.print("Value: " + cookie.getValue( )+" <br/>");
       }
     }else{
         out.println(
          "<h2>No cookies founds</h2>");
     }
     out.println("</body>");
     out.println("</html>");
   }
}
```

编译上述 servlet DeleteCookies 并在 web.xml 文件中创建适当的条目。现在运行 http://localhost:8080/DeleteCookies ，将显示如下所示的结果：

Cookies Name and Value

Deleted cookie : first_name

Name : first_name, Value: John

Name : last_name, Value: Player

现在尝试运行 `http://localhost:8080/ReadCookies`，它将只显示一个 cookie，如下所示：

Found Cookies Name and Value

Name : last_name, Value: Player

你可以在 IE 浏览器中手动删除 Cookies。在“工具”菜单，选择“Internet 选项”。如果要删除所有的 cookies，请按删除 Cookies。

会话跟踪

HTTP 是一种“无状态”协议，这意味着每次客户端检索 Web 页面时，客户端打开一个单独的连接到 Web 服务器，服务器不会自动保存之前客户端请求的任何记录。

仍然有以下三种方式来维持 web 客户端和 web 服务器之间的会话：

Cookies

一个 web 服务器可以分配一个唯一的会话 ID 作为每个 web 客户端的 cookie，并且对于来自客户端的后续请求，它们可以使用已接收的 cookie 来识别。

这可能不是一个有效的方法，因为很多时候浏览器不支持 cookie，所以我不建议使用这种方式来维持会话。

隐藏的表单字段

一个 web 服务器可以发送一个隐藏的 HTML 表单字段以及一个唯一的会话 ID，如下所示：

```
<input type="hidden" name="sessionid" value="12345">
```

该条目意味着，当表单被提交时，指定的名称和值会被自动包含在 GET 或 POST 数据中。每次当 web 浏览器发送回请求时，session_id 的值可以用于跟踪不同的 web 浏览器。

这可能是保持会话跟踪的一种有效的方式，但是点击常规的（<A HREF...>）超文本链接不会导致表单提交，因此隐藏的表单字段也不支持常规的会话跟踪。

URL 重写

你可以在每个标识会话的 URL 末尾追加一些额外的数据，且服务器会把该会话标识符与它已存储的有关会话的数据关联起来。

例如，`http://jikexueyuan.com/file.htm;sessionid=12345`，会话标识符被附加为 sessionid=12345，可能会在 web 服务器端被访问来识别客户端。

URL 重写是维持会话的一种更好的方式，当浏览器不支持 cookie 时为浏览器工作，但是它的缺点是会动态的生成每个 URL 来分配会话 ID，即使页面是简单的静态的 HTML 页面。

HttpSession 对象

除了上述提到的三种方式，servlet 还提供了 HttpSession 接口，该接口提供了一种对网站的跨多个页面请求或访问的方法来识别用户并存储有关用户的信息。

Servlet 容器使用这个接口来创建在 HTTP 客户端和 HTTP 服务器之间的会话。会话在一个指定的时间段内持续，跨多个连接或来自用户的请求。

你可以通过调用 HttpServletRequest 的公共方法 getSession() 来获取 HttpSession 对象，如下所示：

```
HttpSession session = request.getSession();
```

在向客户端发送任何文档内容之前，你需要调用 *request.getSession()*。这里是一些重要方法的总结，这些方法通过 HttpSession 对象是可用的：

序号	方法 & 描述
1	public Object getAttribute(String name) 该方法返回在该 session 会话中具有指定名称的对象，如果没有指定名称的对象，则返回 null。
2	public Enumeration getAttributeNames() 该方法返回 String 对象的枚举，String 对象包含所有绑定到该 session 会话的对象的名称。
3	public long getCreationTime() 该方法返回该 session 会话被创建的时间，自格林尼治标准时间 1970 年 1 月 1 日凌晨零点算起，以毫秒为单位。
4	public String getId() 该方法返回一个包含分配给该 session 会话的唯一标识符的字符串。
5	public long getLastAccessedTime()

	该方法返回客户端最后一次发送与该 session 会话相关的请求的时间自格林尼治标准时间 1970 年 1 月 1 日凌晨零点算起，以毫秒为单位。
6	public int getMaxInactiveInterval() 该方法返回 Servlet 容器在客户端访问时保持 session 会话打开的最大时间间隔，以秒为单位。
7	public void invalidate() 该方法指示该 session 会话无效，并解除绑定到它上面的任何对象。
8	public boolean isNew(如果客户端还不知道该 session 会话，或者如果客户选择不参入该 session 会话，则该方法返回 true。
9	public void removeAttribute(String name) 该方法将从该 session 会话移除指定名称的对象。
10	public void setAttribute(String name, Object value) 该方法使用指定的名称绑定一个对象到该 session 会话。
11	public void setMaxInactiveInterval(int interval) 该方法在 Servlet 容器指示该 session 会话无效之前，指定客户端请求之间的时间，以秒为单位。

会话跟踪实例

这个例子描述了如何使用 HttpSession 对象获取会话创建时间和上次访问的时间。如果不存在会话，我们将一个新的会话与请求联系起来。

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
// Extend HttpServlet class
public class SessionTrack extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     // Create a session object if it is already not  created.
     HttpSession session = request.getSession(true);
     // Get session creation time.
     Date createTime = new Date(session.getCreationTime());
     // Get last access time of this web page.
     Date lastAccessTime =
                       new Date(session.getLastAccessedTime());
     String title = "Welcome Back to my website";
     Integer visitCount = new Integer(0);
     String visitCountKey = new String("visitCount");
     String userIDKey = new String("userID");
     String userID = new String("ABCD");
     // Check if this is new comer on your web page.
     if (session.isNew()){
       title = "Welcome to my website";
       session.setAttribute(userIDKey, userID);
     } else {
       visitCount = (Integer)session.getAttribute(visitCountKey);
       visitCount = visitCount + 1;
       userID = (String)session.getAttribute(userIDKey);
     }
     session.setAttribute(visitCountKey,  visitCount);
     // Set response content type
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     String docType =
     "<!doctype html public \"-//w3c//dtd html 4.0 " +
     "transitional//en\">\n";
     out.println(docType +
               "<html>\n" +
               "<head><title>" + title + "</title></head>\n" +
               "<body bgcolor=\"#f0f0f0\">\n" +
               "<h1 align=\"center\">" + title + "</h1>\n" +
                "<h2 align=\"center\">Session Infomation</h2>\n" +
```

```
            "<table border=\"1\" align=\"center\">\n" +
            "<tr bgcolor=\"#949494\">\n" +
            "  <th>Session info</th><th>value</th></tr>\n" +
            "<tr>\n" +
            "  <td>id</td>\n" +
            "  <td>" + session.getId() + "</td></tr>\n" +
            "<tr>\n" +
            "  <td>Creation Time</td>\n" +
            "  <td>" + createTime +
            "  </td></tr>\n" +
            "<tr>\n" +
            "  <td>Time of Last Access</td>\n" +
            "  <td>" + lastAccessTime +
            "  </td></tr>\n" +
            "<tr>\n" +
            "  <td>User ID</td>\n" +
            "  <td>" + userID +
            "  </td></tr>\n" +
            "<tr>\n" +
            "  <td>Number of visits</td>\n" +
            "  <td>" + visitCount + "</td></tr>\n" +
            "</table>\n" +
            "</body></html>");
  }
}
```

编译上述 servlet SessionTrack 并在 web.xml 文件中创建适当的条目。在浏览器地址栏输入 http://localhost:8080/SessionTrack ，当你第一次运行时将显示如下所示的结果：

Welcome to my website

Session Infomation

Session info	value
id	0AE3EC93FF44E3C525B4351B77ABB2D5
Creation Time	Tue Jun 08 17:26:40 GMT+04:00 2010
Time of Last Access	Tue Jun 08 17:26:40 GMT+04:00 2010
User ID	ABCD
Number of visits	0

现在尝试再次运行相同的 servlet，它将显示如下所示的结果：

Welcome Back to my website

Session Infomation

info type	value
id	0AE3EC93FF44E3C525B4351B77ABB2D5
Creation Time	Tue Jun 08 17:26:40 GMT+04:00 2010
Time of Last Access	Tue Jun 08 17:26:40 GMT+04:00 2010
User ID	ABCD
Number of visits	1

删除会话数据

当你完成了一个用户的会话数据，你有以下几种选择：

- **移除一个特定的属性**：你可以调用 *public void removeAttribute(String name)* 方法来删除与特定的键相关联的值。
- **删除整个会话**：你可以调用 *public void invalidate()* 方法来删除整个会话。
- **设置会话超时**：你可以调用 *public void setMaxInactiveInterval(int interval)* 方法来单独设置会话超时。
- **注销用户**：支持 servlet 2.4 的服务器，你可以调用 logout 来注销 Web 服务器的客户端，并把属于所有用户的所有会话设置为无效。
- **web.xml 配置**：如果你使用的是 Tomcat，除了上述方法，你还可以在 web.xml 文件中配置会话超时，如下所示：

```
<session-config>
   <session-timeout>15</session-timeout>
 </session-config>
```

超时时间是以分钟为单位的，并覆盖了 Tomcat 中默认的 30 分钟的超时时间。

Servlet 中的 getMaxInactiveInterval() 方法为会话返回的超时时间是以秒为单位的。所以如果在 web.xml 中配置会话超时时间为 15 分钟，那么 getMaxInactiveInterval() 会返回 900。

13

数据库访问

本教程假定你已经了解了 JDBC 应用程序的工作方式。在你开始学习 servlet 数据库访问之前，确保你已经有适当的 JDBC 环境设置和数据库。

关于如何使用 JDBC 访问数据库及其环境配置的更多细节，你可以查看我们的 JDBC 教程 (http://wiki.jikexueyuan.com/project/jdbc) 。

从基本概念开始，让我们创建一个简单的表，并在表中创建几条记录，如下所示：

创建表

在 TEST 数据库中创建 Employees 表，请按以下步骤进行：

步骤 1

打开 Command Prompt，并改为安装目录，如下所示：

```
C:\>
C:\>cd Program Files\MySQL\bin
C:\Program Files\MySQL\bin>
```

步骤 2

登录数据库，如下所示：

```
C:\Program Files\MySQL\bin>mysql -u root -p
Enter password: ********
mysql>
```

步骤 3

在 TEST 数据库中创建 Employee 表，如下所示：

```
mysql> use TEST;
mysql> create table Employees
   (
    id int not null,
    age int not null,
    first varchar (255),
    last varchar (255)
   );
```

```
Query OK, 0 rows affected (0.08 sec)
mysql>
```

创建数据记录

最后在 Employee 表中创建几条记录，如下所示：

```
mysql> INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');
Query OK, 1 row affected (0.05 sec)
mysql> INSERT INTO Employees VALUES (101, 25, 'Mahnaz', 'Fatma');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO Employees VALUES (102, 30, 'Zaid', 'Khan');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO Employees VALUES (103, 28, 'Sumit', 'Mittal');
Query OK, 1 row affected (0.00 sec)
mysql>
```

访问数据库

这里的例子演示了如何使用 Servlet 访问 TEST 数据库。

```
// Loading required libraries
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class DatabaseAccess extends HttpServlet{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     // JDBC driver name and database URL
     static final String JDBC_DRIVER="com.mysql.jdbc.Driver";
     static final String DB_URL="jdbc:mysql://localhost/TEST";
     //  Database credentials
     static final String USER = "root";
     static final String PASS = "password";
     // Set response content type
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     String title = "Database Result";
     String docType =
```

```
  "<!doctype html public \"-//w3c//dtd html 4.0 " +
   "transitional//en\">\n";
   out.println(docType +
   "<html>\n" +
   "<head><title>" + title + "</title></head>\n" +
   "<body bgcolor=\"#f0f0f0\">\n" +
   "<h1 align=\"center\">" + title + "</h1>\n");
try{
   // Register JDBC driver
   Class.forName("com.mysql.jdbc.Driver");
   // Open a connection
   conn = DriverManager.getConnection(DB_URL,USER,PASS);
   // Execute SQL query
   stmt = conn.createStatement();
   String sql;
   sql = "SELECT id, first, last, age FROM Employees";
   ResultSet rs = stmt.executeQuery(sql);
   // Extract data from result set
   while(rs.next()){
      //Retrieve by column name
      int id  = rs.getInt("id");
      int age = rs.getInt("age");
      String first = rs.getString("first");
      String last = rs.getString("last");
      //Display values
      out.println("ID: " + id + "<br>");
      out.println(", Age: " + age + "<br>");
      out.println(", First: " + first + "<br>");
      out.println(", Last: " + last + "<br>");
   }
   out.println("</body></html>");
   // Clean-up environment
   rs.close();
   stmt.close();
   conn.close();
}catch(SQLException se){
   //Handle errors for JDBC
   se.printStackTrace();
}catch(Exception e){
   //Handle errors for Class.forName
   e.printStackTrace();
}finally{
   //finally block used to close resources
   try{
      if(stmt!=null)
```

```
            stmt.close();
        }catch(SQLException se2){
        }// nothing we can do
        try{
          if(conn!=null)
          conn.close();
        }catch(SQLException se){
          se.printStackTrace();
        }//end finally try
      } //end try
    }
}
```

现在让我们来编译述 servlet 并在 web.xml 文件中创建以下条目：

```
....
 <servlet>
    <servlet-name>DatabaseAccess</servlet-name>
    <servlet-class>DatabaseAccess</servlet-class>
 </servlet>
 <servlet-mapping>
    <servlet-name>DatabaseAccess</servlet-name>
    <url-pattern>/DatabaseAccess</url-pattern>
 </servlet-mapping>
....
```

现在使用 URL `http://localhost:8080/DatabaseAccess` 调用这个 servlet，将显示如下所示响应：

Database Result

```
ID: 100, Age: 18, First: Zara, Last: Ali
ID: 101, Age: 25, First: Mahnaz, Last: Fatma
ID: 102, Age: 30, First: Zaid, Last: Khan
ID: 103, Age: 28, First: Sumit, Last: Mittal
```

14

文件上传

Servlet 可以与 HTML form 标签一起使用允许用户将文件上传到服务器。上传的文件可以是文本文件或图像文件或任何文档。

创建一个文件上传表单

下述 HTML 代码创建了一个文件上传表单。以下是需要注意的几点：

- 表单 method 属性应该设置为 POST 方法且不能使用 GET 方法。
- 表单 enctype 属性应该设置为 multipart/form-data.
- 表单 action 属性应该设置为 servlet 文件，能够在后端服务器处理文件上传。下面的例子是使用 UploadServlet servlet 来上传文件的。
- 要上传单个文件，你应该使用单个带有属性 type=“file” 的 标签。为了允许多个文件上传，要包含多个带有 name 属性不同值的输入标签。浏览器将把一个浏览按钮和每个输入标签关联起来。

```
<html>
<head>
<title>File Uploading Form</title>
</head>
<body>
<h3>File Upload:</h3>
Select a file to upload: <br />
<form action="UploadServlet" method="post"
                    enctype="multipart/form-data">
<input type="file" name="file" size="50" />
<br />
<input type="submit" value="Upload File" />
</form>
</body>
</html>
```

这将显示如下所示的结果，允许从本地计算机中选择一个文件，当用户点击“上传文件”时，表单会和选择的文件一起提交：

File Upload:
Select a file to upload:

Browse...

Upload File

NOTE: This is just dummy form and would not work.

编写后台 Servlet

以下是 servlet UploadServlet，会接受上传的文件并把它储存在目录 <Tomcat-installation-directory>/webapps/data 中。使用外部配置，如 web.xml 中的 context-param 元素，这个目录名也可以被添加，如下所示：

```
<web-app>
....
<context-param>
   <description>Location to store uploaded file</description>
   <param-name>file-upload</param-name>
   <param-value>
       c:\apache-tomcat-5.5.29\webapps\data\
    </param-value>
</context-param>
....
</web-app>
```

以下是 UploadServlet 的源代码，可以一次处理多个文件的上传。在继续操作之前，请确认下列各项：

- 下述例子依赖于 FileUpload，所以一定要确保在你的 classpath 中有最新版本的 commons-fileupload.x.x.jar 文件。你可以从 http://commons.apache.org/fileupload/ 中下载。
- FileUpload 依赖于 Commons IO，所以一定要确保在你的 classpath 中有最新版本的 commons-io-x.x.jar 文件。可以从 http://commons.apache.org/io/ 中下载。
- 在测试下面实例时，你上传的文件大小不能大于 maxFileSize，否则文件将无法上传。
- 请确保已经提前创建好目录 c:\temp and c:\apache-tomcat-5.5.29\webapps\data。

```
// Import required java libraries
import java.io.*;
import java.util.*;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;
import org.apache.commons.io.output.*;
public class UploadServlet extends HttpServlet {
```

```
private boolean isMultipart;
private String filePath;
private int maxFileSize = 50 * 1024;
private int maxMemSize = 4 * 1024;
private File file ;
public void init( ){
  // Get the file location where it would be stored.
  filePath =
       getServletContext().getInitParameter("file-upload");
}
public void doPost(HttpServletRequest request,
           HttpServletResponse response)
          throws ServletException, java.io.IOException {
  // Check that we have a file upload request
  isMultipart = ServletFileUpload.isMultipartContent(request);
  response.setContentType("text/html");
  java.io.PrintWriter out = response.getWriter( );
  if( !isMultipart ){
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet upload</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<p>No file uploaded</p>");
    out.println("</body>");
    out.println("</html>");
    return;
  }
  DiskFileItemFactory factory = new DiskFileItemFactory();
  // maximum size that will be stored in memory
  factory.setSizeThreshold(maxMemSize);
  // Location to save data that is larger than maxMemSize.
  factory.setRepository(new File("c:\\temp"));
  // Create a new file upload handler
  ServletFileUpload upload = new ServletFileUpload(factory);
  // maximum file size to be uploaded.
  upload.setSizeMax( maxFileSize );
  try{
  // Parse the request to get file items.
  List fileItems = upload.parseRequest(request);
  // Process the uploaded file items
  Iterator i = fileItems.iterator();
  out.println("<html>");
  out.println("<head>");
  out.println("<title>Servlet upload</title>");
```

```
    out.println("</head>");
    out.println("<body>");
    while ( i.hasNext () )
    {
      FileItem fi = (FileItem)i.next();
      if ( !fi.isFormField () )
      {
        // Get the uploaded file parameters
        String fieldName = fi.getFieldName();
        String fileName = fi.getName();
        String contentType = fi.getContentType();
        boolean isInMemory = fi.isInMemory();
        long sizeInBytes = fi.getSize();
        // Write the file
        if( fileName.lastIndexOf("\\") >= 0 ){
          file = new File( filePath +
          fileName.substring( fileName.lastIndexOf("\\"))) ;
        }else{
          file = new File( filePath +
          fileName.substring(fileName.lastIndexOf("\\")+1)) ;
        }
        fi.write( file ) ;
        out.println("Uploaded Filename: " + fileName + "<br>");
      }
    }
    out.println("</body>");
    out.println("</html>");
  }catch(Exception ex) {
     System.out.println(ex);
  }
  }
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, java.io.IOException {
      throw new ServletException("GET method used with " +
              getClass( ).getName( )+": POST method required.");
  }
}
```

编译和运行 Servlet

编译上述 servlet UploadServlet 并在 web.xml 文件中创建所需的条目，如下所示：

```
<servlet>
  <servlet-name>UploadServlet</servlet-name>
  <servlet-class>UploadServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>UploadServlet</servlet-name>
  <url-pattern>/UploadServlet</url-pattern>
</servlet-mapping>
```

现在尝试使用上面创建的 HTML 表单来上传文件。当你访问 `http://localhost:8080/UploadFile.htm` 时，它会显示如下所示的结果，这将有助于你从本地计算机中上传任何文件。

File Upload:

Select a file to upload:

Browse...

Upload File

如果你的 servelt 脚本能正常工作，那么你的文件会被上传到 `c:\apache-tomcat-5.5.29\webapps\data\` 目录中。

处理日期

使用 Servlet 的最重要的优势之一是你可以使用核心 Java 中的大多数可用的方法。本教程将讲解 Java 提供的 Date 类，该类在 java.util 包中是可用的，这个类封装了当前的日期和时间。

Date 类支持两个构造函数。第一个构造函数用当前日期和时间初始化对象。

```
Date( )
```

下面的构造函数接受一个参数，该参数等于自 1970 年 1 月 1 日凌晨零点以来经过的毫秒数。

```
Date(long millisec)
```

一旦你得到一个可用的 Date 对象，你可以调用下列任意支持的方法来使用日期：

序号	方法和描述
1	boolean after(Date date) 如果调用的 Date 对象中包含的日期在 date 指定的日期之后，则返回 true，否则返回 false。
2	boolean before(Date date) 如果调用的 Date 对象中包含的日期在 date 指定的日期之前，则返回 true，否则返回 false。
3	Object clone( ) 重复调用 Date 对象。
4	int compareTo(Date date) 把调用对象的值与 date 的值进行比较。如果两个值是相等的，则返回 0。如果调用对象在 date 之前，则返回一个负值。如果调用对象在 date 之后，则返回一个正值。
5	int compareTo(Object obj) 如果 obj 是 Date 类，则操作等同于 compareTo(Date)。否则，它会抛出一个 ClassCastException。

6	boolean equals(Object date) 如果调用的 Date 对象中包含的时间和日期与 date 指定的相同，则返回 true，否则返回 false。
7	long getTime( ) 返回 1970 年 1 月 1 日以来经过的毫秒数。
8	int hashCode( ) 为调用对象返回哈希代码。
9	void setTime(long time) 设置 time 指定的时间和日期，这表示从 1970 年 1 月 1 日凌晨零点以来经过的时间（以毫秒为单位）。
10	String toString( ) 转换调用的 Date 对象为一个字符串，并返回结果。

获取当前的日期和时间

在 Java Servlet 中获取当前的日期和时间是非常容易的。你可以使用一个带有 *toString()* 方法的简单的 Date 对象来输出当前的日期和时间，如下所示：

```
// Import required java libraries
import java.io.*;
import java.util.Date;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class CurrentDate extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
```

```
    // Set response content type
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Display Current Date & Time";
    Date date = new Date();
    String docType =
    "<!doctype html public \"-//w3c//dtd html 4.0 " +
    "transitional//en\">\n";
    out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n" +
      "<body bgcolor=\"#f0f0f0\">\n" +
      "<h1 align=\"center\">" + title + "</h1>\n" +
      "<h2 align=\"center\">" + date.toString() + "</h2>\n" +
      "</body></html>");
  }
}
```

现在，让我们来编译上述 servlet 并在 web.xml 文件中创建适当的条目，然后使用 URL `http://localhost:8080/CurrentDate` 来调用该 servlet。这将会产生如下所示的结果：

Display Current Date & Time

```
Mon Jun 21 21:46:49 GMT+04:00 2010
```

尝试刷新 URL `http://localhost:8080/CurrentDate`，每隔几秒刷新一次你都会发现显示时间的差异。

日期比较

正如我上面所提到的一样，你可以在 Servlet 中使用所有可用的 Java 方法。如果你需要比较两个日期，以下是方法：

- 你可以使用 getTime() 来获取两个对象自 1970 年 1 月 1 日凌晨零点以来经过的毫秒数，然后比较这两个值。
- 你可以使用方法 before( )、after( ) 和 equals( )。由于一个月里 12 号在 18 号之前，例如，new Date(99, 2, 12).before(new Date (99, 2, 18)) 返回 true。
- 你可以使用 compareTo( ) 方法，该方法由 Comparable 接口定义并由 Date 实现。

使用 SimpleDateFormat 格式化日期

SimpleDateFormat 是一个以语言环境敏感的方式来格式化和解析日期的具体类。 SimpleDateFormat 允许你通过为日期时间格式化选择任何用户定义的模式开始。

让我们修改上述例子，如下所示：

```
// Import required java libraries
import java.io.*;
import java.text.*;
import java.util.Date;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class CurrentDate extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     // Set response content type
     response.setContentType("text/html");
```

```
    PrintWriter out = response.getWriter();
    String title = "Display Current Date & Time";
    Date dNow = new Date( );
    SimpleDateFormat ft =
    new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
    String docType =
    "<!doctype html public \"-//w3c//dtd html 4.0 " +
    "transitional//en\">\n";
    out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n" +
      "<body bgcolor=\"#f0f0f0\">\n" +
      "<h1 align=\"center\">" + title + "</h1>\n" +
      "<h2 align=\"center\">" + ft.format(dNow) + "</h2>\n" +
      "</body></html>");
  }
}
```

再次编译上述 servlet，然后使用 URL `http://localhost:8080/CurrentDate` 来调用该 servlet。这将会产生如下所示的结果：

Display Current Date & Time

Mon 2010.06.21 at 10:06:44 PM GMT+04:00

简单的日期格式的格式代码

要指定时间格式，那么使用时间模式的字符串。在这种模式下，所有的 ASCII 字母被保留为模式字母，这些字母定义如下：

字符	描述	实例
G	时代指示器	AD
y	四位数的年	2001
M	一年中的月	July 或 07
d	一月中的第几天	10
h	带有 A.M./P.M. 的小时（1~12）	12
H	一天中的第几小时（0~23）	22
m	一小时中的第几分	30
s	一分中的第几秒	55
S	毫秒	234
E	一周中的星期几	Tuesday
D	一年中的第几天	360
F	一个月中的某一周的某一天	2 (second Wed. in July)
w	一年中的第几周	40
W	一月中的第几周	1
a	A.M./P.M. 标记	PM
k	一天中的第几小时（1~24）	24
K	带有 A.M./P.M. 的小时（0~11）	10
z	时区	Eastern Standard Time
'	Escape for text	分隔符
"	单引号	`

处理日期方法的可用方法的完整列表，你可以参考标准的 Java 文档。

16

页面重定向

当文档移动到一个新的位置时，通常会使用页面重定向，我们需要将客户端发送到这个新位置或者也可能是由于负载均衡，或者只是为了简单的随机。

重定向请求到另一个页面的最简单的方式是使用 response 对象的 sendRedirect() 方法。下面是该方法的特征：

```
public void HttpServletResponse.sendRedirect(String location)
throws IOException
```

该方法将响应和状态码及新的页面位置发送回浏览器。你也可以通过一起使用 setStatus() 和 setHeader() 方法来达到同样的效果：

```
....
String site = "http://www.newpage.com" ;
response.setStatus(response.SC_MOVED_TEMPORARILY);
response.setHeader("Location", site);
....
```

实例

这个例子显示了 servlet 如何将页面重定向到另一个位置：

```
import java.io.*;
import java.sql.Date;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class PageRedirect extends HttpServlet{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     // Set response content type
     response.setContentType("text/html");
     // New location to be redirected
     String site = new String("http://www.photofuntoos.com");
     response.setStatus(response.SC_MOVED_TEMPORARILY);
     response.setHeader("Location", site);
    }
}
```

现在让我们来编译述 servlet 并在 web.xml 文件中创建以下条目：

```
....
 <servlet>
    <servlet-name>PageRedirect</servlet-name>
    <servlet-class>PageRedirect</servlet-class>
 </servlet>
 <servlet-mapping>
    <servlet-name>PageRedirect</servlet-name>
    <url-pattern>/PageRedirect</url-pattern>
 </servlet-mapping>
....
```

现在使用 URL `http://localhost:8080/PageRedirect` 来调用这个 servlet。这将使你跳转到给定的 URL `http://www.photofuntoos.com` 中。

17

点击计数器

Web 页面的点击计数器

很多时候，你可能有兴趣知道你网站的某个特定页面上的总点击量。使用 servlet 来计算这些点击量是非常简单的，因为一个 servlet 的生命周期是由它运行的容器控制的。

以下是基于 Servlet 生命周期实现一个简单的页面点击计数器需要的步骤：

- 在 init() 方法中初始化一个全局变量。
- 每次调用 doGet() 或 doPost() 方法时，增加全局变量。
- 如果需要，你可以使用一个数据库表来存储 destroy() 方法中的全局变量。在下次初始化 servlet 时，这个值可以在 init() 方法内被读取。这一步是可选的。
- 如果你想计算一个会话内一个页面的点量击，那么你可以使用 isNew() 方法来查看该会话内是否已点击过相同的页面。这一步是可选的。
- 你可以显示全局计数器的值来显示网站中的总点击量。这一步是可选的。

在这里我假设 web 容器不会被重新启动。如果 web 容器被重新启动或 servlet 被销毁，计数器将被重置。

实例

这个例子演示了如何实现一个简单的页面点击计数器：

```
import java.io.*;
import java.sql.Date;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class PageHitCounter extends HttpServlet{
  private int hitCount;
  public void init()
  {
    // Reset hit counter.
    hitCount = 0;
  }
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
```

```
    // Set response content type
    response.setContentType("text/html");
    // This method executes whenever the servlet is hit
    // increment hitCount
    hitCount++;
    PrintWriter out = response.getWriter();
    String title = "Total Number of Hits";
    String docType =
    "<!doctype html public \"-//w3c//dtd html 4.0 " +
    "transitional//en\">\n";
    out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n" +
      "<body bgcolor=\"#f0f0f0\">\n" +
      "<h1 align=\"center\">" + title + "</h1>\n" +
      "<h2 align=\"center\">" + hitCount + "</h2>\n" +
      "</body></html>");
  }
  public void destroy()
  {
    // This is optional step but if you like you
    // can write hitCount value in your database.
  }
}
```

现在让我们来编译上述 servlet 并在 web.xml 文件中创建以下条目：

```
....
 <servlet>
    <servlet-name>PageHitCounter</servlet-name>
    <servlet-class>PageHitCounter</servlet-class>
 </servlet>
 <servlet-mapping>
    <servlet-name>PageHitCounter</servlet-name>
    <url-pattern>/PageHitCounter</url-pattern>
 </servlet-mapping>
....
```

现在使用 URL `http://localhost:8080/PageHitCounter` 来调用这个 servlet。每次页面刷新时，计数器的值都会加 1，这将产生如下所示的结果：

Total Number of Hits

6

网站点击计数器

很多时候，你可能有兴趣知道你整个网站的总点击量。在 Servlet 中，这也是非常简单的，我们可以使用过滤器实现这一点。

以下是实现一个基于过滤器生命周期的简单的网站点击计数器需要的步骤：

- 在过滤器的 init() 方法中初始化一个全局变量。
- 每次调用 doFilter 方法时，增加全局变量。
- 如果需要，你可以使用一个数据库表来存储过滤器的 destroy() 方法中的全局变量的值。在下次初始化过滤器时，该值可以在 init() 方法内被读取。这一步是可选的。

在这里我假设 web 容器不会被重新启动。如果 web 容器被重新启动或 servlet 被销毁，点击计数器将被重置。

实例

这个例子演示了如何实现一个简单的网站点击计数器：

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
public class SiteHitCounter implements Filter{
  private int hitCount;
  public void  init(FilterConfig config)
                  throws ServletException{
    // Reset hit counter.
    hitCount = 0;
  }
  public void  doFilter(ServletRequest request,
            ServletResponse response,
            FilterChain chain)
            throws java.io.IOException, ServletException {
     // increase counter by one
```

```
    hitCount++;
    // Print the counter.
    System.out.println("Site visits count :"+ hitCount );
    // Pass request back down the filter chain
    chain.doFilter(request,response);
  }
  public void destroy()
  {
    // This is optional step but if you like you
    // can write hitCount value in your database.
  }
}
```

现在让我们来编译上述 servlet 并在 web.xml 文件中创建以下条目：

```
....
<filter>
  <filter-name>SiteHitCounter</filter-name>
  <filter-class>SiteHitCounter</filter-class>
</filter>
<filter-mapping>
  <filter-name>SiteHitCounter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
....
```

现在调用任意 URL 如 URL `http://localhost:8080/` 。每次任意页面被点击时，计数器的值都会加 1 并且会在日志中显示如下所示的消息：

```
Site visits count : 1
Site visits count : 2
Site visits count : 3
Site visits count : 4
Site visits count : 5
.....................
```

18

自动刷新页面

假设一个 web 页面，显示了现场比赛得分或股票市场状况或货币兑换率。对于所有这些类型的页面，你都需要使用你浏览器中的 refresh 或 reload 按钮来定期刷新 web 页面。

Java Servlet 提供给你一个机制使这项工作变得简单，可以使得 web 页面在给定的时间间隔自动刷新。

刷新一个 web 页面最简单的方式是使用响应对象的方法 setIntHeader()。以下是这种方法的特征：

```
public void setIntHeader(String header, int headerValue)
```

此方法将头信息 “Refresh” 和一个表示时间间隔的整数值（以秒为单位）发送回浏览器。

自动刷新页面实例

这个例子演示了 servlet 如何使用 setIntHeader() 方法设置 Refresh 头信息，实现自动刷新页面。

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
// Extend HttpServlet class
public class Refresh extends HttpServlet {
  // Method to handle GET method request.
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     // Set refresh, autoload time as 5 seconds
     response.setIntHeader("Refresh", 5);
     // Set response content type
     response.setContentType("text/html");
     // Get current time
     Calendar calendar = new GregorianCalendar();
     String am_pm;
     int hour = calendar.get(Calendar.HOUR);
     int minute = calendar.get(Calendar.MINUTE);
     int second = calendar.get(Calendar.SECOND);
     if(calendar.get(Calendar.AM_PM) == 0)
       am_pm = "AM";
     else
       am_pm = "PM";
     String CT = hour+":"+ minute +":"+ second +" "+ am_pm;
     PrintWriter out = response.getWriter();
     String title = "Auto Page Refresh using Servlet";
```

```
    String docType =
    "<!doctype html public \"-//w3c//dtd html 4.0 " +
    "transitional//en\">\n";
    out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n"+
      "<body bgcolor=\"#f0f0f0\">\n" +
      "<h1 align=\"center\">" + title + "</h1>\n" +
      "<p>Current Time is: " + CT + "</p>\n");
  }
  // Method to handle POST method request.
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
    throws ServletException, IOException {
    doGet(request, response);
  }
}
```

现在让我们来编译上述 servlet 并在 web.xml 文件中创建以下条目：

```
....
 <servlet>
    <servlet-name>Refresh</servlet-name>
    <servlet-class>Refresh</servlet-class>
 </servlet>
 <servlet-mapping>
    <servlet-name>Refresh</servlet-name>
    <url-pattern>/Refresh</url-pattern>
 </servlet-mapping>
....
```

现在使用 URL `http://localhost:8080/Refresh` 来调用这个 servlet。这将会每隔 5 秒钟显示一次当前系统时间，如下所示。运行该 servlet 并等着看结果：

Auto Page Refresh using Servlet

Current Time is: 9:44:50 PM

19

发送电子邮件

使用 Servlet 发送一封电子邮件是非常简单的，但是开始之前你必须在你的计算机上安装 JavaMail API 和 Java Activation Framework(JAF)。

- 你可以从 Java 标准网站下载最新版本的 JavaMail版本 1.2 (http://java.sun.com/products/javamail/)
- 你可以从 Java 标准网站下载最新版本的 JAF版本 1.1.1 (http://java.sun.com/products/javabeans/glasgow/jaf.html)

下载并解压缩这些文件，在新创建的顶级目录中你会发现这两个应用程序的一些 jar 文件。你需要把 mail.jar 和 activation.jar 文件添加到你的 CLASSPATH 中。

发送一封简单的电子邮件

这是从你的计算机上发送一封简单的电子邮件的实例。这里假设你的**本地主机**已连接到互联网并可以发送电子邮件。同时确保来自 Java Email API 包和 JAF 包的所有的 jar 文件在 CLASSPATH 中是可用的。

```
// File Name SendEmail.java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;
public class SendEmail extends HttpServlet{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     // Recipient's email ID needs to be mentioned.
     String to = "abcd@gmail.com";
     // Sender's email ID needs to be mentioned
     String from = "web@gmail.com";
     // Assuming you are sending email from localhost
     String host = "localhost";
     // Get system properties
     Properties properties = System.getProperties();
     // Setup mail server
     properties.setProperty("mail.smtp.host", host);
     // Get the default Session object.
     Session session = Session.getDefaultInstance(properties);
     // Set response content type
     response.setContentType("text/html");
```

```
    PrintWriter out = response.getWriter();
    try{
      // Create a default MimeMessage object.
      MimeMessage message = new MimeMessage(session);
      // Set From: header field of the header.
      message.setFrom(new InternetAddress(from));
      // Set To: header field of the header.
      message.addRecipient(Message.RecipientType.TO,
                           new InternetAddress(to));
      // Set Subject: header field
      message.setSubject("This is the Subject Line!");
      // Now set the actual message
      message.setText("This is actual message");
      // Send message
      Transport.send(message);
      String title = "Send Email";
      String res = "Sent message successfully....";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n" +
      "<body bgcolor=\"#f0f0f0\">\n" +
      "<h1 align=\"center\">" + title + "</h1>\n" +
      "<p align=\"center\">" + res + "</p>\n" +
      "</body></html>");
    }catch (MessagingException mex) {
      mex.printStackTrace();
    }
  }
}
```

现在让我们来编译上述 servlet 并在 web.xml 文件中创建以下条目：

```
....
 <servlet>
    <servlet-name>SendEmail</servlet-name>
    <servlet-class>SendEmail</servlet-class>
 </servlet>
 <servlet-mapping>
    <servlet-name>SendEmail</servlet-name>
    <url-pattern>/SendEmail</url-pattern>
 </servlet-mapping>
....
```

现在使用 URL `http://localhost:8080/SendEmail` 来调用这个 servlet。这将会给给定的电子邮件 ID `abcd@gmail.com` 发送一封电子邮件并将显示如下所示的响应：

Send Email

```
Sent message successfully....
```

如果你想把一封电子邮件发送给多个收件人，那么使用下面的方法来指定多个电子邮件 ID：

```
void addRecipients(Message.RecipientType type,
                   Address[] addresses)
throws MessagingException
```

这里是对参数的描述：

- **type**：这将被设置为 TO、CC 或 BCC。这里 CC 代表抄送且 BCC 代表密件抄送。例如 *Message.RecipientType.TO*。
- **addresses**：这是电子邮件 ID 的数组。当指定电子邮件 ID 时，你需要使用 InternetAddress() 方法。

发送一封 HTML 电子邮件

这个例子将从你的计算机上发送一封 HTML 电子邮件。这里假设你的**本地主机**已连接到互联网且可以发送电子邮件。同时确保来自 Java Email API 包和 JAF 包的所有的 jar 文件在 CLASSPATH 中都是可用的。

这个例子与上一个例子非常相似，除了在这里我们使用的是 setContent() 方法来设置内容，它的第二个参数为“text/html” 用来指定 HTML 内容是包含在消息中的。

使用这个实例，你可以发送内容大小不限的 HTML 内容。

```
// File Name SendEmail.java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;
public class SendEmail extends HttpServlet{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     // Recipient's email ID needs to be mentioned.
     String to = "abcd@gmail.com";
     // Sender's email ID needs to be mentioned
     String from = "web@gmail.com";
     // Assuming you are sending email from localhost
```

```
    String host = "localhost";
    // Get system properties
    Properties properties = System.getProperties();
    // Setup mail server
    properties.setProperty("mail.smtp.host", host);
    // Get the default Session object.
    Session session = Session.getDefaultInstance(properties);
    // Set response content type
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    try{
      // Create a default MimeMessage object.
      MimeMessage message = new MimeMessage(session);
      // Set From: header field of the header.
      message.setFrom(new InternetAddress(from));
      // Set To: header field of the header.
      message.addRecipient(Message.RecipientType.TO,
                           new InternetAddress(to));
      // Set Subject: header field
      message.setSubject("This is the Subject Line!");
      // Send the actual HTML message, as big as you like
      message.setContent("<h1>This is actual message</h1>",
                         "text/html" );
      // Send message
      Transport.send(message);
      String title = "Send Email";
      String res = "Sent message successfully....";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n" +
      "<body bgcolor=\"#f0f0f0\">\n" +
      "<h1 align=\"center\">" + title + "</h1>\n" +
      "<p align=\"center\">" + res + "</p>\n" +
      "</body></html>");
    }catch (MessagingException mex) {
      mex.printStackTrace();
    }
  }
}
```

编译并运行上述 servlet 来在给定的电子邮件 ID 上发送 HTML 消息。

在电子邮件中发送附件

这里的例子将从你的计算机上发送一封带有附件的电子邮件。这里假设你的**本地主机**已连接到互联网并能够发送电子邮件。同时确保来自 Java Email API 包和 JAF 包的所有的 jar 文件在 CLASSPATH 中都是可用的。

```
// File Name SendEmail.java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;
public class SendEmail extends HttpServlet{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     // Recipient's email ID needs to be mentioned.
     String to = "abcd@gmail.com";
     // Sender's email ID needs to be mentioned
     String from = "web@gmail.com";
     // Assuming you are sending email from localhost
     String host = "localhost";
     // Get system properties
     Properties properties = System.getProperties();
     // Setup mail server
     properties.setProperty("mail.smtp.host", host);
     // Get the default Session object.
     Session session = Session.getDefaultInstance(properties);
     // Set response content type
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
      try{
        // Create a default MimeMessage object.
        MimeMessage message = new MimeMessage(session);
        // Set From: header field of the header.
        message.setFrom(new InternetAddress(from));
        // Set To: header field of the header.
        message.addRecipient(Message.RecipientType.TO,
                             new InternetAddress(to));
        // Set Subject: header field
        message.setSubject("This is the Subject Line!");
```

```
      // Create the message part
      BodyPart messageBodyPart = new MimeBodyPart();
      // Fill the message
      messageBodyPart.setText("This is message body");
      // Create a multipar message
      Multipart multipart = new MimeMultipart();
      // Set text message part
      multipart.addBodyPart(messageBodyPart);
      // Part two is attachment
      messageBodyPart = new MimeBodyPart();
      String filename = "file.txt";
      DataSource source = new FileDataSource(filename);
      messageBodyPart.setDataHandler(new DataHandler(source));
      messageBodyPart.setFileName(filename);
      multipart.addBodyPart(messageBodyPart);
      // Send the complete message parts
      message.setContent(multipart );
      // Send message
      Transport.send(message);
      String title = "Send Email";
      String res = "Sent message successfully....";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n" +
      "<body bgcolor=\"#f0f0f0\">\n" +
      "<h1 align=\"center\">" + title + "</h1>\n" +
      "<p align=\"center\">" + res + "</p>\n" +
      "</body></html>");
    }catch (MessagingException mex) {
      mex.printStackTrace();
    }
  }
}
```

编译并运行上面的 servlet 来在给定的电子邮件 ID 上发送附件文件和消息。

用户身份认证部分

如果需要向电子邮件服务器提供用户 ID 和密码来进行身份认证，那么你可以设置如下所示的属性：

```
props.setProperty("mail.user", "myuser");
 props.setProperty("mail.password", "mypwd");
```

电子邮件发送机制的其余部分与上面解释的一致。

20

包

涉及到 WEB-INF 子目录的 web 应用程序结构是所有的 Java web 应用程序的标准，并且是由 servlet API 规范指定的。给定一个 myapp 的顶级目录名，这里是目录结构，如下所示：

```
/myapp
   /images
   /WEB-INF
      /classes
      /lib
```

WEB-INF 子目录包含了应用程序的部署描述符，命名为 web.xml。所有的 HTML 文件都位于顶级目录 *myapp* 下。对于管理员用户，你会发现 ROOT 目录是和 myApp 一样的父目录。

创建包中的 Servlets

WEB-INF/classes 目录在与它们的包名称匹配的结构中包含了所有的 servlet 类和其他的类文件。例如，如果你有一个完全合格的类名称 com.myorg.MyServlet，那么这个 servlet 类必须被放置在如下所示的目录中：

```
/myapp/WEB-INF/classes/com/myorg/MyServlet.class
```

下面是创建包名为 *com.myorg* 的 MyServlet 类的例子

```
// Name your package
package com.myorg;
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MyServlet extends HttpServlet {
  private String message;
  public void init() throws ServletException
  {
     // Do required initialization
     message = "Hello World";
  }
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     // Set response content type
     response.setContentType("text/html");
     // Actual logic goes here.
     PrintWriter out = response.getWriter();
     out.println("<h1>" + message + "</h1>");
  }
  public void destroy()
```

```
 {
    // do nothing.
 }
}
```

编译包中的 Servlets

编译包中可用的类没有什么大的不同。最简单的方法是将你的 java 文件保存在完全限定路径中，正如上面所提到的一样，类将被保存在 com.myorg 中。你还需要将该目录添加到 CLASSPATH 中。

假设你的环境已正确设置，进入 `<Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes` 目录并编译 MyServlet.java，如下所示：

```
$ javac MyServlet.java
```

如果 servlet 依赖于任何其他的库，那么你必须在你的 CLASSPATH 中包含那些 JAR 文件。我只包含了 servlet-api.jar JAR 文件，因为我在 Hello World 程序中没有使用任何其他的库。

该命令行使用来自 Sun Microsystems Java 软件开发工具包（JDK）内置的 javac 编译器。为了让该命令正常工作，必须包括你在 PATH 环境变量中所使用的 Java SDK 的位置。

如果一切顺利，上述编译会在相同的目录下生成 MyServlet.class 文件。下一节将解释如何在生产中部署一个已编译的 servlet。

打包的 Servlet 部署

默认情况下，servlet 应用程序位于路径 `<Tomcat-installation-directory>/webapps/ROOT` 下且类文件放在 `<Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes` 中。

如果你有一个完全合格的类名称 com.myorg.MyServlet，那么这个 servlet 类必须位于 `WEB-INF/classes/com/myorg/MyServlet.class` 中，你需要在位于 `<Tomcat-installation-directory>/webapps/ROOT/WEB-INF/` 的 web.xml 文件中创建以下条目：

```
<servlet>
  <servlet-name>MyServlet</servlet-name>
  <servlet-class>com.myorg.MyServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>MyServlet</servlet-name>
  <url-pattern>/MyServlet</url-pattern>
</servlet-mapping>
```

上述条目要被创建在 web.xml 文件中的 `<web-app>...</web-app>` 标签内。在该文件中可能已经有各种可用的条目，但没有关系。

你基本上已经完成了，现在让我们使用 `<Tomcat-installation-directory>\bin\startup.bat`（Windows 操作系统中）或 `<Tomcat-installation-directory>/bin/startup.sh`（Linux/Solaris 等操作系统中）启动 tomcat 服务器，最后在浏览器的地址栏中输入 `http://localhost:8080/MyServlet`。如果一切顺利，你会得到如下所示的结果：

Hello World

21 调试

测试/调试 servlet 始终是困难的。Servlets 往往涉及大量的客户端/服务器交互，可能会出现错误但是又难以重现。

这里有一些提示和建议，可以帮助你调试。

System.out.println()

System.out.println() 作为一个标记用来测试某一代码片段是否被执行，使用方法非常简单。我们也可以输出变量值。另外：

- 由于 System 对象是核心 Java 对象的一部分，它可以用于任何不需要安装任何额外类的地方。这包括 Servlets、JSP、RMI、EJB's、普通的 Beans 和类，以及独立的应用程序。
- 与在断点处停止相比，写入 System.out 不会对应用程序的正常执行流程有太多干扰，这使得它在时序重要的时候显得非常有价值。

以下使用 System.out.println() 的语法：

```
System.out.println("Debugging message");
```

通过上述语法生成的所有消息将被记录在 web 服务器的日志文件中。

消息记录

利用标准日志记录方法，使用适当的日志记录方法来记录所有调试、警告和错误消息是非常好的想法，我使用的是 log4J (http://wiki.jikexueyuan.com/project/log4j) 来记录所有的消息。

Servlet API 还提供了一个简单的输出信息的方式，使用 log() 方法，如下所示：

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ContextLog extends HttpServlet {
  public void doGet(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
       java.io.IOException {
     String par = request.getParameter("par1");
     //Call the two ServletContext.log methods
     ServletContext context = getServletContext( );
     if (par == null || par.equals(""))
```

```
    //log version with Throwable parameter
    context.log("No message received:",
       new IllegalStateException("Missing parameter"));
    else
       context.log("Here is the visitor's message: " + par);
    response.setContentType("text/html");
    java.io.PrintWriter out = response.getWriter( );
    String title = "Context Log";
    String docType =
    "<!doctype html public \"-//w3c//dtd html 4.0 " +
    "transitional//en\">\n";
    out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n" +
      "<body bgcolor=\"#f0f0f0\">\n" +
      "<h1 align=\"center\">" + title + "</h1>\n" +
      "<h2 align=\"center\">Messages sent</h2>\n" +
      "</body></html>");
   } //doGet
}
```

ServletContext 把它的文本消息记录到 servlet 容器的日志文件中。使用 Tomcat，这些日志可以在 `<Tomcat-installation-directory>/logs` 目录中找到。

这些日志文件确实为新出现的错误或问题的频率给出了指示。正因为如此，在通常不会出现的异常 catch 子句中使用 log() 函数是很好的。

使用 JDB 调试器

你可以使用调试 applet 或应用程序的相同的 jdb 命令来调试 servlet。

为了调试一个 servlet，我们可以调试 sun.servlet.http.HttpServer，然后把它看成是 HttpServer 执行 servlet 来响应来自浏览器端的 HTTP 请求。这与调试 applet 小程序的方式非常相似。与调试 applet 不同的是，被调试的实际程序是 sun.applet.AppletViewer。

大多数调试器会自动隐藏了解如何调试 applet 的细节。直到他们为 servlet 做同样的事情，你必须做以下操作来帮助你的调试器：

- 设置你的调试器的类路径，以便它可以找到 sun.servlet.http.Http-Server 和相关的类。
- 设置你的调试器的类路径，以便它可以找到你的 servlet 和支持的类，通常是在 server_root/servlets 和 server_root/classes 中。

你通常不会希望 server_root/servlets 在你的 classpath 中，因为它会禁用 servlet 的重载。然而这种包含对于调试是有用的。在 HttpServer 中的自定义的 servlet 加载器加载 servlet 之前，它允许你的调试器在 servlet 中设置断点。

一旦你设置了正确的类路径，就可以开始调试 sun.servlet.http.HttpServer。你可以在任何你想要调试的 servlet 中设置断点，然后使用 web 浏览器为给定的 servlet（ `http://localhost:8080/servlet/ServletToDebug` ）向 HttpServer 发出请求。你会看到程序执行到你设置的断点处停止。

使用注释

代码中的注释有助于以各种方式调试程序。注释可用于调试过程中的许多其他方式中。

Servlet 使用 Java 注释，单行注释（ //... ）和多行注释（ /* ...*/ ）可用于暂时移除部分 Java 代码。如果 bug 消失，仔细看看你之前注释的代码并找出问题所在。

客户端和服务器端头信息

有时，当一个 servlet 并没有像预期那样工作时，查看原始的 HTTP 请求和响应是非常有用的。如果你对 HTTP 结构很熟悉，你可以阅读请求和响应，看看这些头信息中究竟是什么。

重要的调试技巧

这里是 servlet 调试中的一些调试技巧列表：

- 请注意 server _ root/classes 不会重载，而 server_root/servlets 可能会。
- 要求浏览器显示它所显示的页面的原始内容。这有助于识别格式的问题。它通常是视图菜单下的一个选项。
- 通过强制执行完全重载页面，来确保浏览器还没有缓存前一个请求的输出。在 Netscape Navigator 中，使用 Shift-Reload；在 IE 浏览器中，请使用 Shift-Refresh。
- 确认 servlet 的 init() 方法接受一个 ServletConfig 参数并立即调用 super.init(config)。

22 国际化

在我们继续之前，请让我解释三个重要术语：

- **国际化（i18n）：** 这意味着网站能够提供翻译成访问者的语言或国籍的不同版本的内容。
- **本地化（l10n）：** 这意味着向网站添加资源，使其适应特定的地理或文化区域，例如网站翻译成印地语。
- **区域设置：** 这是一个特殊的文化或地理区域。它通常指语言符号后跟一个由下划线分隔的国家符号。例如 "en_US" 表示 US 的英语区域设置。

当建立一个全球性的网站时有一些注意事项。本教程不会给出完整的细节，但它会通过一个很好的例子向你演示如何通过差异化定位（即区域设置）来让网页以不同的语言呈现。

Servlet 可以根据请求者的区域设置读出相应版本的网站，并根据当地的语言、文化和需求提供相应的网站版本。以下是 request 对象中的方法，它返回了 Locale 对象。

```
java.util.Locale request.getLocale()
```

检测区域设置

下面列出了重要的区域设置方法，你可以使用它们来检测请求者的地理位置、语言和区域设置。下面所有的方法都显示了请求者浏览器中设置的国家名称和语言名称。

序号	方法 & 描述
1	String getCountry() 该方法以 2 个大写字母形式的 ISO 3166 格式返回该区域设置的国家/地区代码。
2	String getDisplayCountry() 该方法返回适合向用户显示的区域设置的国家的名称。
3	String getLanguage() 该方法以小写字母形式的 ISO 639 格式返回该区域设置的语言代码。
4	String getDisplayLanguage() 该方法返回适合向用户显示的区域设置的语言的名称。

5	String getISO3Country() 该方法返回该区域设置的国家的三个字母缩写。
6	String getISO3Language() 该方法返回该区域设置的语言的三个字母的缩写。

实例

这个例子向你演示了如何为一个请求显示语言和相关的国家：

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Locale;
public class GetLocale extends HttpServlet{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
     //Get the client's Locale
     Locale locale = request.getLocale();
     String language = locale.getLanguage();
     String country = locale.getCountry();
     // Set response content type
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     String title = "Detecting Locale";
     String docType =
     "<!doctype html public \"-//w3c//dtd html 4.0 " +
     "transitional//en\">\n";
     out.println(docType +
       "<html>\n" +
       "<head><title>" + title + "</title></head>\n" +
       "<body bgcolor=\"#f0f0f0\">\n" +
       "<h1 align=\"center\">" + language + "</h1>\n" +
       "<h2 align=\"center\">" + country + "</h2>\n" +
       "</body></html>");
  }
}
```

语言设置

Servlet 可以输出以西欧语言编写的页面，如英语、西班牙语、德语、法语、意大利语、荷兰语等。在这里，设置 Content-Language 头信息来正确的显示所有字符是非常重要的。

第二点是使用 HTML 实体显示所有的特殊字符，例如，"ñ" 表示 "ñ"，"¡" 表示 "¡"，如下所示：

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Locale;
public class DisplaySpanish extends HttpServlet{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
    // Set response content type
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    // Set spanish language code.
    response.setHeader("Content-Language", "es");
    String title = "En Espa&ntilde;ol";
    String docType =
     "<!doctype html public \"-//w3c//dtd html 4.0 " +
     "transitional//en\">\n";
     out.println(docType +
     "<html>\n" +
     "<head><title>" + title + "</title></head>\n" +
     "<body bgcolor=\"#f0f0f0\">\n" +
     "<h1>" + "En Espa&ntilde;ol:" + "</h1>\n" +
     "<h1>" + "&iexcl;Hola Mundo!" + "</h1>\n" +
     "</body></html>");
  }
}
```

特定于区域设置的日期

你可以使用 java.text.DateFormat 类及其静态的 getDateTimeInstance() 方法来格式化特定于区域设置的日期和时间。下面的例子向你演示了如何格式化特定于一个给定的区域设置的日期：

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Locale;
import java.text.DateFormat;
import java.util.Date;
public class DateLocale extends HttpServlet{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
    // Set response content type
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    //Get the client's Locale
    Locale locale = request.getLocale( );
    String date = DateFormat.getDateTimeInstance(
                              DateFormat.FULL,
                              DateFormat.SHORT,
                              locale).format(new Date( ));
    String title = "Locale Specific Dates";
    String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n" +
      "<body bgcolor=\"#f0f0f0\">\n" +
      "<h1 align=\"center\">" + date + "</h1>\n" +
      "</body></html>");
  }
}
```

特定于区域设置的货币

你可以使用 java.text.NumberFormat 类及其静态的 getCurrencyInstance() 方法来在特定于区域设置的货币中格式化数字，比如 long 类型或 double 类型。下面的例子向你演示了如何格式化特定于一个给定的区域设置的货币：

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Locale;
import java.text.NumberFormat;
```

```
import java.util.Date;
public class CurrencyLocale extends HttpServlet{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
    // Set response content type
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    //Get the client's Locale
    Locale locale = request.getLocale( );
    NumberFormat nft = NumberFormat.getCurrencyInstance(locale);
    String formattedCurr = nft.format(1000000);
    String title = "Locale Specific Currency";
    String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n" +
      "<body bgcolor=\"#f0f0f0\">\n" +
      "<h1 align=\"center\">" + formattedCurr + "</h1>\n" +
      "</body></html>");
  }
}
```

特定于区域设置的百分比

你可以使用 java.text.NumberFormat 类及其静态的 getPercentInstance() 方法来格式化特定于区域设置的百分比。下面的例子向你演示了如何格式化特定于一个给定的区域设置的百分比：

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Locale;
import java.text.NumberFormat;
import java.util.Date;
public class PercentageLocale extends HttpServlet{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
    // Set response content type
    response.setContentType("text/html");
```

```
    PrintWriter out = response.getWriter();
    //Get the client's Locale
    Locale locale = request.getLocale( );
    NumberFormat nft = NumberFormat.getPercentInstance(locale);
    String formattedPerc = nft.format(0.51);
    String title = "Locale Specific Percentage";
    String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
      "<html>\n" +
      "<head><title>" + title + "</title></head>\n" +
      "<body bgcolor=\"#f0f0f0\">\n" +
      "<h1 align=\"center\">" + formattedPerc + "</h1>\n" +
      "</body></html>");
  }
}
```

极客学院

jikexueyuan.com

中国最大的IT职业在线教育平台

更多信息请访问

http://wiki.jikexueyuan.com/project/servlet/