## Servlet 是什么？

Java Servlet 是运行在 Web 服务器或应用服务器上的程序，它是作为来自 Web 浏览器或其他 HTTP 客户端的请求和 HTTP 服务器上的数据库或应用程序之间的中间层。

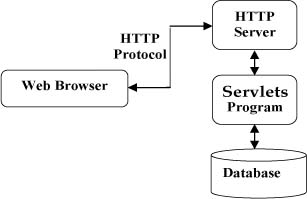
使用 Servlet，您可以收集来自网页表单的用户输入，呈现来自数据库或者其他源的记录，还可以动态创建网页。

Java Servlet 通常情况下与使用 CGI（Common Gateway Interface，公共网关接口）实现的程序可以达到异曲同工的效果。但是相比于 CGI，Servlet 有以下几点优势：

* 性能明显更好。
* Servlet 在 Web 服务器的地址空间内执行。这样它就没有必要再创建一个单独的进程来处理每个客户端请求。
* Servlet 是独立于平台的，因为它们是用 Java 编写的。
* 服务器上的 Java 安全管理器执行了一系列限制，以保护服务器计算机上的资源。因此，Servlet 是可信的。
* Java 类库的全部功能对 Servlet 来说都是可用的。它可以通过 sockets 和 RMI 机制与 applets、数据库或其他软件进行交互。

## Servlet 架构

下图显示了 Servlet 在 Web 应用程序中的位置。



## Servlet 任务

Servlet 执行以下主要任务：

* 读取客户端（浏览器）发送的显式的数据。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。
* 读取客户端（浏览器）发送的隐式的 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，或者直接计算得出对应的响应。
* 发送显式的数据（即文档）到客户端（浏览器）。该文档的格式可以是多种多样的，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。

## Servlet 包

Java Servlet 是运行在带有支持 Java Servlet 规范的解释器的 web 服务器上的 Java 类。

Servlet 可以使用 **javax.servlet** 和 **javax.servlet.http** 包创建，它是 Java 企业版的标准组成部分，Java 企业版是支持大型开发项目的 Java 类库的扩展版本。

这些类实现 Java Servlet 和 JSP 规范。在写本教程的时候，二者相应的版本分别是 Java Servlet 2.5 和 JSP 2.1。

Java Servlet 就像任何其他的 Java 类一样已经被创建和编译。在您安装 Servlet 包并把它们添加到您的计算机上的 Classpath 类路径中之后，您就可以通过 JDK 的 Java 编译器或任何其他编译器来编译 Servlet。

# Servlet 生命周期

Servlet 生命周期可被定义为从创建直到毁灭的整个过程。以下是 Servlet 遵循的过程：

* Servlet 通过调用 **init ()** 方法进行初始化。
* Servlet 调用 **service()** 方法来处理客户端的请求。
* Servlet 通过调用 **destroy()** 方法终止（结束）。
* 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，它是用于一次性初始化，就像 Applet 的 init 方法一样。

Servlet 创建于用户第一次调用对应于该 Servlet 的 URL 时，但是您也可以指定 Servlet 在服务器第一次启动时被加载。

当用户调用一个 Servlet 时，就会创建一个 Servlet 实例，每一个用户请求都会产生一个新的线程，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 {

// 初始化代码...}

## 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，并把格式化的响应写回给客户端。

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，service 方法在适当的时候调用 doGet、doPost、doPut、doDelete 等方法。所以，您不用对 service() 方法做任何动作，您只需要根据来自客户端的请求类型来重载 doGet() 或 doPost() 即可。

doGet() 和 doPost() 方法是每次服务请求中最常用的方法。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，或者来自于一个未指定 METHOD 的 HTML 表单，它由 doGet() 方法处理。

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

// Servlet 代码}

## doPost() 方法

POST 请求来自于一个特别指定了 METHOD 为 POST 的 HTML 表单，它由 doPost() 方法处理。

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

// Servlet 代码}

## destroy() 方法

destroy() 方法只会被调用一次，在 Servlet 生命周期结束时被调用。destroy() 方法可以让您的 Servlet 关闭数据库连接、停止后台线程、把 Cookie 列表或点击计数器写入到磁盘，并执行其他类似的清理活动。

在调用 destroy() 方法之后，servlet 对象被标记为垃圾回收。destroy 方法定义如下所示：

public void destroy() {

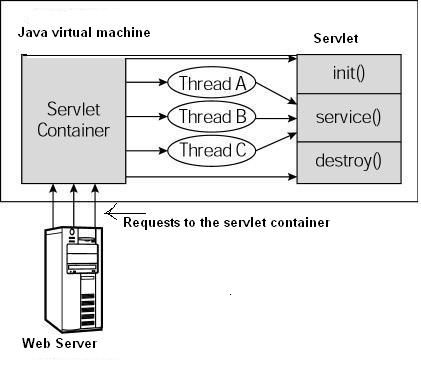
// 终止化代码...

}

## 架构图

下图显示了一个典型的 Servlet 生命周期方案。

* 第一个到达服务器的 HTTP 请求被委派到 Servlet 容器。
* Servlet 容器在调用 service() 方法之前加载 Servlet。
* 然后 Servlet 容器处理由多个线程产生的多个请求，每个线程执行一个单一的 Servlet 实例的 service() 方法。



# Servlet 实例

Servlet 是服务 HTTP 请求并实现 **javax.servlet.Servlet** 接口的 Java 类。Web 应用程序开发人员通常编写 Servlet 来扩展 javax.servlet.http.HttpServlet，并实现 Servlet 接口的抽象类专门用来处理 HTTP 请求。

## Hello World 示例代码

下面是 Servlet 输出 Hello World 的示例源代码：

// 导入必需的 java 库import java.io.\*;import javax.servlet.\*;import javax.servlet.http.\*;

// 扩展 HttpServlet 类public class HelloWorld extends HttpServlet {

private String message;

public void init() throws ServletException

{

// 执行必需的初始化

message = "Hello World";

}

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException

{

// 设置响应内容类型

response.setContentType("text/html");

// 实际的逻辑是在这里

PrintWriter out = response.getWriter();

out.println("<h1>" + message + "</h1>");

}

public void destroy()

{

// 什么也不做

}}

## 编译 Servlet

让我们把上面的代码写在 HelloWorld.java 文件中，把这个文件放在 C:\ServletDevel（在 Windows 上）或 /usr/ServletDevel（在 UNIX 上）中，您还需要把这些目录添加到 CLASSPATH 中。

假设您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，您必须在 CLASSPATH 中包含那些 JAR 文件。在这里，我只包含了 servlet-api.jar JAR 文件，因为我没有在 Hello World 程序中使用任何其他库。

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

如果一切顺利，上面编译会在同一目录下生成 HelloWorld.class 文件。下一节将讲解已编译的 Servlet 如何部署在生产中。

## 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。

如果您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。

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

<servlet>

<servlet-name>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，但不要在意。

到这里，您基本上已经完成了，现在让我们使用 <Tomcat-installation-directory>\bin\startup.bat（在 Windows 上）或 <Tomcat-installation-directory>/bin/startup.sh（在 Linux/Solaris 等上）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，您会看到下面的结果：

# Servlet 表单数据

很多情况下，需要传递一些信息，从浏览器到 Web 服务器，最终到后台程序。浏览器使用两种方法可将这些信息传递到 Web 服务器，分别为 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中间用 ? 字符分隔，如下所示：

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

GET 方法是默认的从浏览器向 Web 服务器传递信息的方法，它会产生一个很长的字符串，出现在浏览器的地址栏中。如果您要向服务器传递的是密码或其他的敏感信息，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，并可以通过 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

另一个向后台程序传递信息的比较可靠的方法是 POST 方法。POST 方法打包信息的方式与 GET 方法基本相同，但是 POST 方法不是把信息作为 URL 中 ? 字符后的文本字符串进行发送，而是把这些信息作为一个单独的消息。消息以标准输出的形式传到后台程序，您可以解析和使用这些标准输出。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 处理表单数据，这些数据会根据不同的情况使用不同的方法自动解析：

* **getParameter()：**您可以调用 request.getParameter() 方法来获取表单参数的值。
* **getParameterValues()：**如果参数出现一次以上，则调用该方法，并返回多个值，例如复选框。
* **getParameterNames()：**如果您想要得到当前请求中的所有参数的完整列表，则调用该方法。

## 使用 URL 的 GET 方法实例

下面是一个简单的 URL，将使用 GET 方法向 HelloForm 程序传递两个值。

**http://localhost:8080/HelloForm?first\_name=ZARA&last\_name=ALI**

下面是处理 Web 浏览器输入的 **HelloForm.java** Servlet 程序。我们将使用 **getParameter()** 方法，可以很容易地访问传递的信息：

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

\* Servlet implementation class HelloForm

\*/

@WebServlet("/HelloForm")

public class HelloForm extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

public HelloForm() {

super();

// TODO Auto-generated constructor stub

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// 设置响应内容类型

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

String title = "使用 GET 方法读取表单数据";

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>名字</b>："

+ request.getParameter("first\_name") + "\n" +

" <li><b>姓氏</b>："

+ request.getParameter("last\_name") + "\n" +

"</ul>\n" +

"</body></html>");

}

}

假设您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，您就必须把该类文件复制到 <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes 中，并在位于 <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/ 的**web.xml** 文件中创建以下条目：

<servlet>

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

<servlet-class>HelloForm</servlet-class>

</servlet>

<servlet-mapping>

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

<url-pattern>/HelloForm</url-pattern>

</servlet-mapping>

现在在浏览器的地址栏中输入 *http://localhost:8080/HelloForm?first\_name=ZARA&last\_name=ALI* ，并在触发上述命令之前确保已经启动 Tomcat 服务器。如果一切顺利，您会得到下面的结果：

|  |
| --- |
| 使用 GET 方法读取表单数据  * **名字**：ZARA * **姓氏**：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 来处理输入。

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<form action="HelloForm" method="GET">

名字：<input type="text" name="first\_name">

<br />

姓氏：<input type="text" name="last\_name" />

<input type="submit" value="提交" />

</form>

</body>

</html>

保存这个 HTML 到 hello.htm 文件中，并把它放在 <Tomcat-installation-directory>/webapps/ROOT 目录下。当您访问*http://localhost:8080/Hello.htm* 时，下面是上面表单的实际输出。

窗体顶端

名字：   
姓氏：

窗体底端

尝试输入名字和姓氏，然后点击"提交"按钮，在您本机上查看输出结果。基于所提供的输入，它会产生与上一个实例类似的结果。

## 使用表单的 POST 方法实例

让我们对上面的 Servlet 做小小的修改，以便它可以处理 GET 和 POST 方法。下面的 **HelloForm.java** Servlet 程序使用 GET 和 POST 方法处理由 Web 浏览器给出的输入。

// 导入必需的 java 库import java.io.\*;import javax.servlet.\*;import javax.servlet.http.\*;

// 扩展 HttpServlet 类public class HelloForm extends HttpServlet {

// 处理 GET 方法请求的方法

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException

{

// 设置响应内容类型

response.setContentType("text/html;charset=UTF-8");

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>名字</b>："

+ request.getParameter("first\_name") + "\n" +

" <li><b>姓氏</b>："

+ request.getParameter("last\_name") + "\n" +

"</ul>\n" +

"</body></html>");

}

// 处理 POST 方法请求的方法

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

doGet(request, response);

}}

现在，编译部署上述的 Servlet，并使用带有 POST 方法的 Hello.htm 进行测试，如下所示：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<form action="HelloForm" method="POST">

名字：<input type="text" name="first\_name">

<br />

姓氏：<input type="text" name="last\_name" />

<input type="submit" value="提交" />

</form>

</body>

</html>

下面是上面表单的实际输出，尝试输入名字和姓氏，然后点击"提交"按钮，在您本机上查看输出结果。

窗体顶端

名字：   
姓氏：

窗体底端

基于所提供的输入，它会产生与上一个实例类似的结果。

## 将复选框数据传递到 Servlet 程序

当需要选择一个以上的选项时，则使用复选框。

下面是一个 HTML 代码实例 CheckBox.htm，一个带有两个复选框的表单。

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<form action="CheckBox" method="POST" target="\_blank">

<input type="checkbox" name="maths" checked="checked" /> 数学

<input type="checkbox" name="physics" /> 物理

<input type="checkbox" name="chemistry" checked="checked" />

化学

<input type="submit" value="选择学科" />

</form>

</body>

</html>

这段代码的结果是下面的表单：

窗体顶端

 数学  物理  化学

窗体底端

下面是 CheckBox.java Servlet 程序，处理 Web 浏览器给出的复选框输入。

// 导入必需的 java 库import java.io.\*;import javax.servlet.\*;import javax.servlet.http.\*;

// 扩展 HttpServlet 类public class CheckBox extends HttpServlet {

// 处理 GET 方法请求的方法

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException

{

// 设置响应内容类型

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

String title = "读取复选框数据";

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>数学标识：</b>: "

+ request.getParameter("maths") + "\n" +

" <li><b>物理标识：</b>: "

+ request.getParameter("physics") + "\n" +

" <li><b>化学标识：</b>: "

+ request.getParameter("chemistry") + "\n" +

"</ul>\n" +

"</body></html>");

}

// 处理 POST 方法请求的方法

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

doGet(request, response);

}}

上面的实例将显示下面的结果：

|  |
| --- |
| 读取复选框数据  * **数学标识：**on * **物理标识：**null * **化学标识：**on |

## 读取所有的表单参数

以下是通用的实例，使用 HttpServletRequest 的 **getParameterNames()** 方法读取所有可用的表单参数。该方法返回一个枚举，其中包含未指定顺序的参数名。

一旦我们有一个枚举，我们可以以标准方式循环枚举，使用 *hasMoreElements()* 方法来确定何时停止，使用 *nextElement()* 方法来获取每个参数的名称。

import java.io.IOException;

import java.io.PrintWriter;

import java.util.Enumeration;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/\*\*

\* Servlet implementation class ReadParams

\*/

@WebServlet("/ReadParams")

public class ReadParams extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

public ReadParams() {

super();

// TODO Auto-generated constructor stub

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// 设置响应内容类型

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

String title = "读取所有的表单数据";

String docType =

"<!doctype html public \"-//w3c//dtd html 4.0 " +

"transitional//en\">\n";

out.println(docType +

"<html>\n" +

"<head><meta charset=\"utf-8\"><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>参数名称</th><th>参数值</th>\n"+

"</tr>\n");

Enumeration paramNames = request.getParameterNames();

while(paramNames.hasMoreElements()) {

String paramName = (String)paramNames.nextElement();

out.print("<tr><td>" + paramName + "</td>\n");

String[] paramValues =

request.getParameterValues(paramName);

// 读取单个值的数据

if (paramValues.length == 1) {

String paramValue = paramValues[0];

if (paramValue.length() == 0)

out.println("<td><i>没有值</i></td>");

else

out.println("<td>" + paramValue + "</td>");

} else {

// 读取多个值的数据

out.println("<td><ul>");

for(int i=0; i < paramValues.length; i++) {

out.println("<li>" + paramValues[i]);

}

out.println("</ul></td>");

}

out.print("</tr>");

}

out.println("\n</table>\n</body></html>");

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

doGet(request, response);

}

}

现在，通过下面的表单尝试上面的 Servlet：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<form action="ReadParams" method="POST" target="\_blank">

<input type="checkbox" name="maths" checked="checked" /> 数学

<input type="checkbox" name="physics" /> 物理

<input type="checkbox" name="chemistry" checked="checked" /> 化学

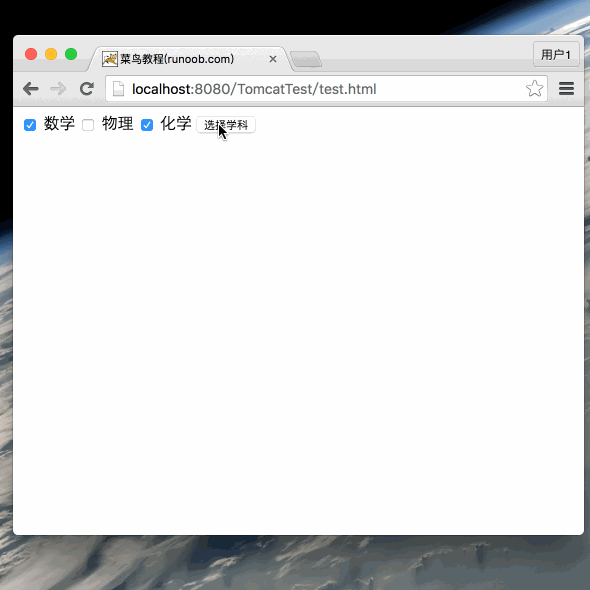
<input type="submit" value="选择学科" />

</form>

</body>

</html>

现在使用上面的表单调用 Servlet，将产生以下结果：



您可以尝试使用上面的 Servlet 来读取其他的表单数据，比如文本框、单选按钮或下拉框等。

# Servlet 客户端 HTTP 请求

当浏览器请求网页时，它会向 Web 服务器发送特定信息，这些信息不能被直接读取，因为这些信息是作为 HTTP 请求的头的一部分进行传输的。您可以查看 [HTTP 协议](http://www.runoob.com/http/http-tutorial.html" \t "http://www.runoob.com/servlet/_blank) 了解更多相关信息。

以下是来自于浏览器端的重要头信息，您可以在 Web 编程中频繁使用：

|  |  |
| --- | --- |
| **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 头的方法

下面的方法可用在 Servlet 程序中读取 HTTP 头。这些方法通过 *HttpServletRequest* 对象可用。

|  |  |
| --- | --- |
| **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 Header 请求实例

下面的实例使用 HttpServletRequest 的 **getHeaderNames()** 方法读取 HTTP 头信息。该方法返回一个枚举，包含与当前的 HTTP 请求相关的头信息。

一旦我们有一个枚举，我们可以以标准方式循环枚举，使用 *hasMoreElements()* 方法来确定何时停止，使用 *nextElement()* 方法来获取每个参数的名称。

// 导入必需的 java 库import java.io.\*;import javax.servlet.\*;import javax.servlet.http.\*;import java.util.\*;

// 扩展 HttpServlet 类public class DisplayHeader extends HttpServlet {

// 处理 GET 方法请求的方法

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException

{

// 设置响应内容类型

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String title = "HTTP Header 请求实例";

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 名称</th><th>Header 值</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>");

}

// 处理 POST 方法请求的方法

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

doGet(request, response);

}}

现在，调用上面的 Servlet 会产生以下结果：

# HTTP Header 请求实例

|  |  |
| --- | --- |
| **Header 名称** | **Header 值** |
| 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 |

← [Servlet 表单数据](http://www.runoob.com/servlet/servlet-form-data.html" \o "Servlet 表单数据)

[Servlet 服务器 HTTP 响应](http://www.runoob.com/servlet/servlet-server-response.html" \o "Servlet 服务器 HTTP 响应) →

# Servlet 服务器 HTTP 响应

正如前面的章节中讨论的那样，当一个 Web 服务器响应一个 HTTP 请求时，响应通常包括一个状态行、一些响应报头、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（在本例中为 HTTP/1.1）、一个状态码（在本例中为 200）和一个对应于状态码的短消息（在本例中为 OK）。

下表总结了从 Web 服务器端返回到浏览器的最有用的 HTTP 1.1 响应报头，您会在 Web 编程中频繁地使用它们：

|  |  |
| --- | --- |
| **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 300s 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（Service Unavailable 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 响应报头的方法

下面的方法可用于在 Servlet 程序中设置 HTTP 响应报头。这些方法通过 *HttpServletResponse* 对象可用。

|  |  |
| --- | --- |
| **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报头。 |
| 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报头。 |
| 7 | **void addHeader(String name, String value)** 添加一个带有给定的名称和值的响应报头。 |
| 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报头。 |
| 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。 |
| 11 | **void resetBuffer()** 清除响应中基础缓冲区的内容，不清除状态码和头。 |
| 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。 |
| 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报头。 |
| 20 | **void setHeader(String name, String value)** 设置一个带有给定的名称和值的响应报头。 |
| 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报头。 |
| 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 Header 响应实例

您已经在前面的实例中看到 setContentType() 方法，下面的实例也使用了同样的方法，此外，我们会用 **setIntHeader()** 方法来设置**Refresh** 头。

// 导入必需的 java 库

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.util.\*;

// 扩展 HttpServlet 类

public class Refresh extends HttpServlet {

// 处理 GET 方法请求的方法

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException

{

// 设置刷新自动加载时间为 5 秒

response.setIntHeader("Refresh", 5);

// 设置响应内容类型

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 = "自动刷新 Header 设置";

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>当前时间是：" + CT + "</p>\n");

}

// 处理 POST 方法请求的方法

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

doGet(request, response);

}

}

# Servlet 服务器 HTTP 响应

正如前面的章节中讨论的那样，当一个 Web 服务器响应一个 HTTP 请求时，响应通常包括一个状态行、一些响应报头、一个空行和文档。一个典型的响应如下所示：

HTTP/1.1 200 OKContent-Type: text/htmlHeader2: ......HeaderN: ...

(Blank Line)<!doctype ...><html><head>...</head><body>...</body></html>

状态行包括 HTTP 版本（在本例中为 HTTP/1.1）、一个状态码（在本例中为 200）和一个对应于状态码的短消息（在本例中为 OK）。

下表总结了从 Web 服务器端返回到浏览器的最有用的 HTTP 1.1 响应报头，您会在 Web 编程中频繁地使用它们：

|  |  |
| --- | --- |
| **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 300s 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（Service Unavailable 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 响应报头的方法

下面的方法可用于在 Servlet 程序中设置 HTTP 响应报头。这些方法通过 *HttpServletResponse* 对象可用。

|  |  |
| --- | --- |
| **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报头。 |
| 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报头。 |
| 7 | **void addHeader(String name, String value)** 添加一个带有给定的名称和值的响应报头。 |
| 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报头。 |
| 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。 |
| 11 | **void resetBuffer()** 清除响应中基础缓冲区的内容，不清除状态码和头。 |
| 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。 |
| 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报头。 |
| 20 | **void setHeader(String name, String value)** 设置一个带有给定的名称和值的响应报头。 |
| 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报头。 |
| 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 Header 响应实例

您已经在前面的实例中看到 setContentType() 方法，下面的实例也使用了同样的方法，此外，我们会用 **setIntHeader()** 方法来设置**Refresh** 头。

// 导入必需的 java 库import java.io.\*;import javax.servlet.\*;import javax.servlet.http.\*;import java.util.\*;

// 扩展 HttpServlet 类public class Refresh extends HttpServlet {

// 处理 GET 方法请求的方法

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException

{

// 设置刷新自动加载时间为 5 秒

response.setIntHeader("Refresh", 5);

// 设置响应内容类型

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 = "自动刷新 Header 设置";

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>当前时间是：" + CT + "</p>\n");

}

// 处理 POST 方法请求的方法

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

doGet(request, response);

}}

# Servlet HTTP 状态码

HTTP 请求和 HTTP 响应消息的格式是类似的，结构如下：

* 初始状态行 + 回车换行符（回车+换行）
* 零个或多个标题行+回车换行符
* 一个空白行，即回车换行符
* 一个可选的消息主体，比如文件、查询数据或查询输出

例如，服务器的响应头如下所示：

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（在本例中为 HTTP/1.1）、一个状态码（在本例中为 200）和一个对应于状态码的短消息（在本例中为 OK）。

以下是可能从 Web 服务器返回的 HTTP 状态码和相关的信息列表：

|  |  |  |
| --- | --- | --- |
| **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 | *您还不能使用该代码。* |
| 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。当您转换一个 "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 状态代码的方法

下面的方法可用于在 Servlet 程序中设置 HTTP 状态码。这些方法通过 *HttpServletResponse* 对象可用。

|  |  |
| --- | --- |
| **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

下面的例子把 407 错误代码发送到客户端浏览器，浏览器会显示 "Need authentication!!!" 消息。

// 导入必需的 java 库

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.util.\*;

// 扩展 HttpServlet 类

public class showError extends HttpServlet {

// 处理 GET 方法请求的方法

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException

{

// 设置错误代码和原因

response.sendError(407, "Need authentication!!!" );

}

// 处理 POST 方法请求的方法

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

doGet(request, response);

}

}

现在，调用上面的 Servlet 将显示以下结果：

|  |
| --- |
| 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** |