



Servlet 教程

Servlet 为创建基于 web 的应用程序提供了基于组件、独立于平台的方法，可以不受 CGI 程序的性能限制。Servlet 有权限访问所有的 Java API，包括访问企业级数据库的 JDBC API。

本教程将讲解如何使用 Java Servlet 来开发基于 web 的应用程序。

[现在开始学习 Servlet!](#)

谁适合阅读本教程？

本教程是专为 Java 程序员设计的。在阅读本教程之前，需要先了解 Java Servlet 框架和它的 API。学习完本教程后，您会发现自己已经达到使用 Java Servlet 的中等水平，后续您可以自行通过更深入的学习和实践完成进阶。

阅读本教程前，您需要了解的知识：

在您开始阅读本教程之前，最好对 Java 编程语言有一个很好的理解。如果您对 web 应用程序和互联网如何工作的有基本的认识，将有助于您理解本教程。

Servlet 相关教程

[Java 教程](#)

[Jsp 教程](#)

[点我分享笔记](#)

[反馈/建议](#)



Servlet 简介

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**。

下一步呢？

接下来，本教程会带你一步一步地设置您的 **Servlet** 环境，以便开始后续的 **Servlet** 使用。因此，请系紧您的安全带，随我们一起开始 **Servlet** 的学习之旅吧！相信您会很喜欢这个教程的。

☐ Servlet 教程

Servlet 环境设置 ☐



1 篇笔记
#1

☐ 写笔记



Servlet 创有三种方式。

1、实现 **Servlet** 接口

因为是实现 **Servlet** 接口，所以我们需要实现接口里的方法。

下面我们也说明了 **Servlet** 的执行过程，也就是 **Servlet** 的生命周期。

//Servlet的生命周期:从Servlet被创建到Servlet被销毁的过程

//一次创建，到处服务

//一个Servlet只会有一个对象，服务所有的请求

/*

* 1.实例化（使用构造方法创建对象）

```

* 2.初始化  执行init方法

* 3.服务      执行service方法

* 4.销毁      执行destroy方法

*/

public class ServletDemo1 implements Servlet {

    //public ServletDemo1(){

    //生命周期方法:当Servlet第一次被创建对象时执行该方法,该方法在整个生命周期中只执行一次

    public void init(ServletConfig arg0) throws ServletException {

        System.out.println("=====init=====");

    }

    //生命周期方法:对客户端响应的方法,该方法会被执行多次,每次请求该servlet都会执行该方法

    public void service(ServletRequest arg0, ServletResponse arg1)

        throws ServletException, IOException {

        System.out.println("hehe");

    }

    //生命周期方法:当Servlet被销毁时执行该方法

    public void destroy() {

        System.out.println("*****destroy*****");

    }

    //当停止tomcat时也就销毁的servlet。

    public ServletConfig getServletConfig() {

        return null;

    }

    public String getServletInfo() {

        return null;

```

```
}  
  
}
```

2、继承 **GenericServlet** 类

它实现了 **Servlet** 接口除了 **service** 的方法，不过这种方法我们极少用。

```
public class ServletDemo2 extends GenericServlet {  
  
    @Override  
  
    public void service(ServletRequest arg0, ServletResponse arg1)  
  
        throws ServletException, IOException {  
  
        System.out.println("heihei");  
  
    }  
  
}
```

3、继承 **HttpServlet** 方法

```
public class ServletDemo3 extends HttpServlet {  
  
    @Override  
  
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)  
  
        throws ServletException, IOException {  
  
        System.out.println("haha");  
  
    }  
  
    @Override  
  
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)  
  
        throws ServletException, IOException {  
  
        System.out.println("ee");  
  
        doGet(req,resp);  
  
    }  
  
}
```

创建 **Servlet** 的第三种方法，也是我们经常用的方法。

这里只简单讲 **Servlet** 的三种创建方式，关于更详细的应用我们后面再说。

南离4个月前 (05-29)

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)[□ Servlet 简介](#)[Servlet 生命周期 □](#)

Servlet 环境设置

开发环境是您可以开发、测试、运行 **Servlet** 的地方。

就像任何其他 **Java** 程序，您需要通过使用 **Java** 编译器 **javac** 编译 **Servlet**，在编译 **Servlet** 应用程序后，将它部署在配置的环境中以便测试和运行。

如果你使用的是 **Eclipse** 环境，可以直接参阅：[Eclipse JSP/Servlet 环境搭建](#)。

这个开发环境设置包括以下步骤：

设置 Java 开发工具包（Java Development Kit）

这一步涉及到下载 **Java** 软件开发工具包（SDK，即 **Software Development Kit**），并适当地设置 **PATH** 环境变量。

您可以从 **Oracle** 的 **Java** 网站下载 SDK：[Java SE Downloads](#)。

一旦您下载了 **SDK**，请按照给定的指令来安装和配置设置。最后，设置 **PATH** 和 **JAVA_HOME** 环境变量指向包含 **java** 和 **javac** 的目录，通常分别为 **java_install_dir/bin** 和 **java_install_dir**。

如果您运行的是 **Windows**，并把 **SDK** 安装在 **C:\jdk1.5.0_20** 中，则需要在您的 **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 NT/2000/XP** 中，您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** 的值，按下"确定"按钮。

在 **Unix**（**Solaris**、**Linux** 等）上，如果 **SDK** 安装在 **/usr/local/jdk1.5.0_20** 中，并且您使用的是 **C shell**，则需要在您的 **.cshrc** 文件中放入下列的行：

```
setenv PATH /usr/local/jdk1.5.0_20/bin:$PATH

setenv JAVA_HOME /usr/local/jdk1.5.0_20
```

另外，如果您使用集成开发环境（**IDE**，即 **Integrated Development Environment**），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** 知道您安装的 **Java** 路径。

更详细内容可参阅：[Java 开发环境配置](#)

设置 Web 服务器：Tomcat

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

Apache Tomcat 是一款 **Java Servlet** 和 **JavaServer Pages** 技术的开源软件实现，可以作为测试 **Servlet** 的独立服务器，而且可以集成到 **Apache Web** 服务器。下面是在电脑上安装 **Tomcat** 的步骤：

从 <http://tomcat.apache.org/> 上下载最新版本的 **Tomcat**。

一旦您下载了 **Tomcat**，解压缩到一个方便的位置。例如，如果您使用的是 **Windows**，则解压缩到 **C:\apache-tomcat-5.5.29** 中，如果您使用的是 **Linux/Unix**，则解压缩到 **/usr/local/apache-tomcat-5.5.29** 中，并创建 **CATALINA_HOME** 环境变量指向这些位置。

在 **Windows** 上，可以通过执行下面的命令来启动 **Tomcat**：

```
%CATALINA_HOME%\bin\startup.bat
```

或者

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

在 **Unix (Solaris、Linux 等)** 上，可以通过执行下面的命令来启动 **Tomcat**：

```
$CATALINA_HOME/bin/startup.sh
```

或者

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

Tomcat 启动后，可以通过在浏览器地址栏输入 **http://localhost:8080/** 访问 **Tomcat** 中的默认应用程序。如果一切顺利，那么会显示以下结果：



有关配置和运行 **Tomcat** 的进一步信息可以查阅应用程序安装的文档，或者可以访问 **Tomcat** 网站：<http://tomcat.apache.org>。

在 **Windows** 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)** 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 **Servlet** 不是 **Java** 平台标准版的组成部分，所以您必须为编译器指定 **Servlet** 类的路径。

如果您运行的是 **Windows**，则需要在您的 **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 NT/2000/XP** 中，您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"确定"按钮。

在 **Unix (Solaris、Linux 等)** 上，如果您使用的是 **C shell**，则需要在您的 **.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：假设您的开发目录是 `C:\ServletDevel`（在 `Windows` 上）或 `/user/ServletDevel`（在 `UNIX` 上），那么您还需要在 `CLASSPATH` 中添加这些目录，添加方式与上面的添加方式类似。

[Servlet 简介](#)

[Servlet 生命周期](#)



1 篇笔记
#1

[写笔记](#)



设置Classpath的目的，在于告诉Java执行环境，在哪些目录下可以找到您所执行的Java程序所需要的类或者包。事实上JDK 5.0默认就会到当前工作目录(上面的.设置)，以及JDK的lib目录(这里假设是C:\Program Files\Java\jdk1.5.0_06\lib)中寻找Java程序。所以如果Java程序是在这两个目录中，则不必设置Classpath变量也可以找得到，将来如果Java程序不是放置在这两个目录时，则可以按上述设置Classpath。

wk 我主沉浮1年前
(2017-07-06)

[反馈/建议](#)

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[Servlet 环境设置](#)

[Servlet 实例](#)

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写 **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 环境设置](#)

[Servlet 实例](#)

[点我分享笔记](#)

[反馈/建议](#)



[Servlet 生命周期](#)

[Servlet 表单数据](#)

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 `javax.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 S**

DK 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：

```
<web-app>

    <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>

</web-app>
```

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 生命周期

Servlet 表单数据 ☐



2 篇笔记
#2

☐ 写笔记



`destory` 方法被调用后，`servlet` 被销毁，但是并没有立即被回收，再次请求时，并没有重新初始化。
代码示例：

```
private String message;

@Override

public void init() throws ServletException {

    message = "Hello World , Nect To Meet You: " + System.currentTimeMillis();
```

```

        System.out.println("servlet初始化.....");

        super.init();

    }

    @Override

    public void doGet(HttpServletRequest req, HttpServletResponse response) throws ServletException, IOException {

        response.setContentType("text/html");

        PrintWriter writer = response.getWriter();

        writer.write("<h1>" + message + "</h1>");

        destroy();

    }

    @Override

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

        // TODO Auto-generated method stub

        super.doPost(req, resp);

    }

    @Override

    public void destroy() {

        System.out.println("servlet销毁! ");

        super.destroy();

    }

```

控制台打印:

```

servlet初始化.....

servlet销毁!

2017-7-6 19:48:52 org.apache.catalina.core.StandardContext reload

信息: Reloading Context with name [/myServlet] has started

servlet销毁!

2017-7-6 19:48:52 org.apache.catalina.core.StandardContext reload

信息: Reloading Context with name [/myServlet] is completed

servlet初始化.....

```

servlet销毁！

servlet销毁！

servlet销毁！

servlet销毁！

servlet销毁！

servlet销毁！

servlet销毁！

wk 我主沉浮1年前
(2017-07-06)

#1



servlet 浏览器访问路径配置有个小问题：

1、java 类里的注解 ——@WebServlet("/HelloServlet") 对应浏览器路径：

http://localhost:8080/TomcatTest/HelloServlet

2、配置文件（web.xml）里对应的浏览器访问路径：

http://localhost:8080/TomcatTest/TomcatTest/HelloServlet

这两种配一个就好了，不然路径重名的话反而会让tomcat启动不了。

例如这样就启动不了：

修改 web.xml：

<url-pattern>/HelloServlet</url-pattern>

修改后，web.xml 和 java 类的注解，对应路径都是：

http://localhost:8080/TomcatTest/HelloServlet

导致

命名的 servlet[HelloServlet]和 [com.runoob.test.HelloServlet] 都被映射到 URL 模式 [/ HelloServlet] 这是不允许的。

解决办法：

将注解去掉或者保留注解进入web.xml将映射删除既可以。

董大dj1年前 (2017-09-15)

反馈/建议



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/TomcatTest/HelloForm?name=菜鸟教程&url=www.runoob.com

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：

```
package com.runoob.test;

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";

        // 处理中文
        String name = new String(request.getParameter("name").getBytes("ISO8859-1"), "UTF-8");

        String docType = "<!DOCTYPE html> \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>站点名</b>: "

```

```

        + name + "\n" +

        "    <li><b>网址</b>: "

        + request.getParameter("url") + "\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);

    }

}

```

然后我们在 **web.xml** 文件中创建以下条目：

```

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

<web-app>

    <servlet>

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

        <servlet-class>com.runoob.test.HelloForm</servlet-class>

    </servlet>

    <servlet-mapping>

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

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

    </servlet-mapping>

</web-app>

```

现在在浏览器的地址栏中输入 **<http://localhost:8080/TomcatTest/HelloForm?name=菜鸟教程&url=www.runoob.com>**，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">

网址名: <input type="text" name="name">

<br />

网址: <input type="text" name="url" />

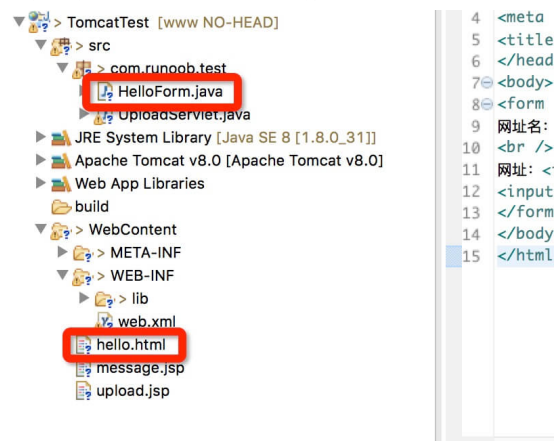
<input type="submit" value="提交" />

</form>

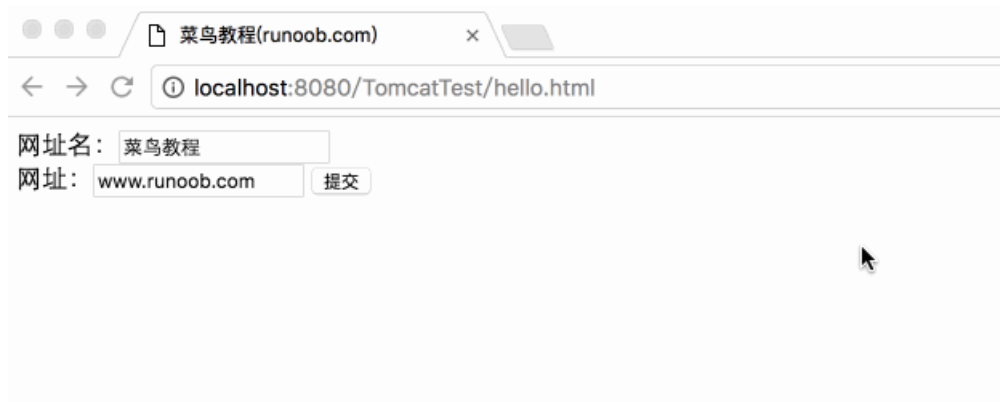
</body>

</html>
```

保存这个 HTML 到 hello.html 文件中，目录结构如下所示：



尝试输入网址名和网址，然后点击"提交"按钮，Gif 演示如下：



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。

注意：如果表单提交的数据中有中文数据则需要转码：

```
String name =new String(request.getParameter("name").getBytes("ISO8859-1"),"UTF-8");
```

```
package com.runoob.test;

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 POST 方法读取表单数据";

    // 处理中文

    String name =new String(request.getParameter("name").getBytes("ISO8859-1"),"UTF-8");

    String docType = "<!DOCTYPE html> \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>站点名</b>: "

        + name + "\n" +

        "    <li><b>网址</b>: "

        + request.getParameter("url") + "\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.html** 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">

网址名: <input type="text" name="name">

<br />

网址: <input type="text" name="url" />

<input type="submit" value="提交" />

</form>

</body>

</html>
```

下面是上面表单的实际输出，尝试输入网址名和网址，然后点击"提交"按钮，**Gif** 演示如下：



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 `checkboxbox.html`，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="runoob" checked="checked" /> 菜鸟教程

<input type="checkbox" name="google" /> Google

<input type="checkbox" name="taobao" checked="checked" /> 淘宝

<input type="submit" value="选择站点" />

</form>

</body>

</html>
```

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

```
package com.runoob.test;

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 CheckBox
 */
@WebServlet("/CheckBox")

public class CheckBox extends HttpServlet {

    private static final long serialVersionUID = 1L;


    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 = "读取复选框数据";

        String docType = "<!DOCTYPE html> \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>菜鸟按教程标识: </b>: "

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

            "  <li><b>Google 标识: </b>: "

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

            "  <li><b>淘宝标识: </b>: "

```

```

        + request.getParameter("taobao") + "\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);
    }
}

```

设置对应的 web.xml:

```

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

<web-app>

    <servlet>

        <servlet-name>CheckBox</servlet-name>

        <servlet-class>com.runoob.test.CheckBox</servlet-class>

    </servlet>

    <servlet-mapping>

        <servlet-name>CheckBox</servlet-name>

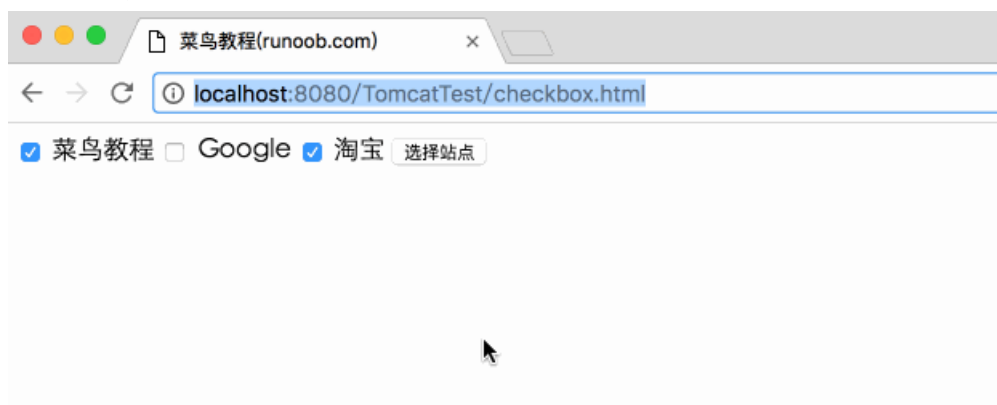
        <url-pattern>/TomcatTest/CheckBox</url-pattern>

    </servlet-mapping>

</web-app>

```

上面的实例将显示下面的结果:



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, IOExcepti
```



```

on {

    // 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>
```

设置相应的 **web.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app>

  <servlet>

    <servlet-name>ReadParams</servlet-name>

    <servlet-class>com.runoob.test.ReadParams</servlet-class>

  </servlet>

  <servlet-mapping>

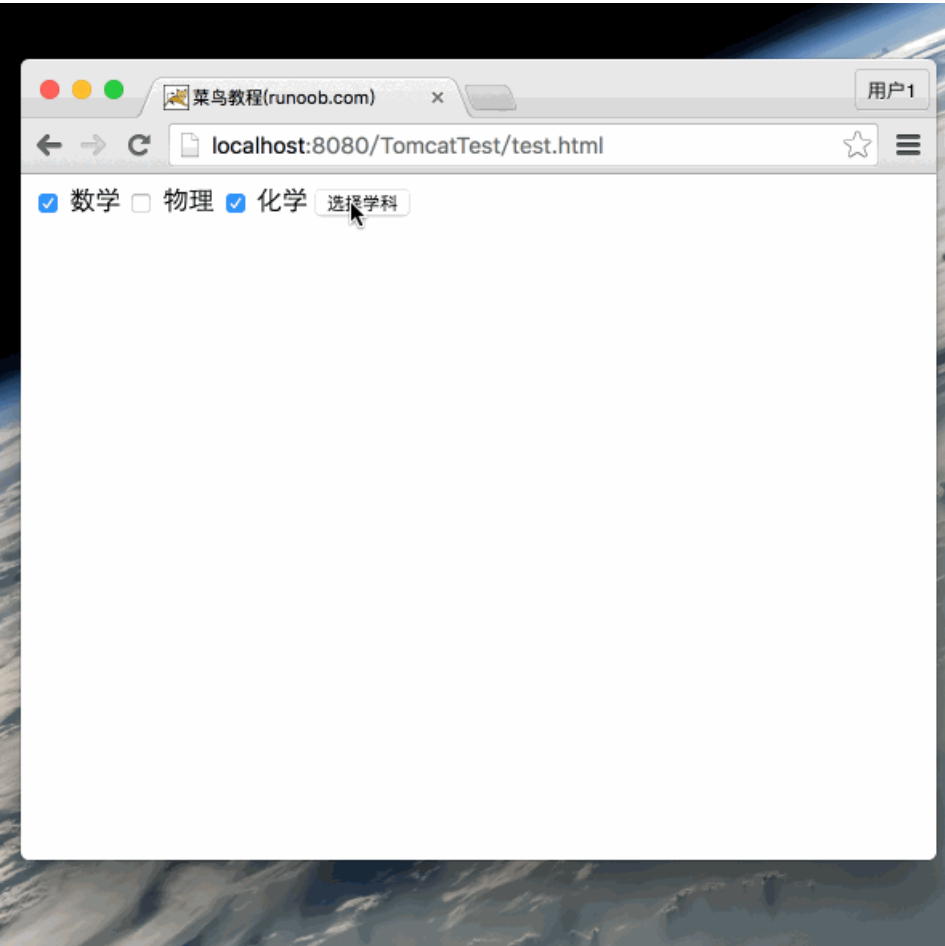
    <servlet-name>ReadParams</servlet-name>

    <url-pattern>/TomcatTest/ReadParams</url-pattern>

  </servlet-mapping>

</web-app>
```

现在使用上面的表调用 **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 实例

Servlet 客户端 HTTP 请求 ☐

☐

1 篇笔记
#1

☐

写笔记

如果用 `getParameter()` 获取中文只有问号：

```
String name =new String(request.getParameter("name").getBytes("ISO8859-1"),"UTF-8");
```

由于 `tomcat8` 默认编码是 `utf-8`，而这个过滤器把他当成 `ISO8859-1` 解码时（即解码了两次），就会出现问问题，导致所有中文变成问号
所以我们只需要不使用这个过滤器就可以了。
也就是采用以下代码：

```
String name =new String(request.getParameter("name"));
```

这样就能正常显示中文。
hell6个月前 (04-16)

反馈/建议



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](#) 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 getSessionId() 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。
31	int getParameterMap() 将参数封装成 Map 类型。

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.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;

@WebServlet("/DisplayHeader")

//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;charset=UTF-8");

        PrintWriter out = response.getWriter();

        String title = "HTTP Header 请求实例 - 菜鸟教程实例";

        String docType =

            "<!DOCTYPE html> \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>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);
}

}
```

以上测试实例是位于 **TomcatTest** 项目下，对应的 **web.xml** 配置为：

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app>

    <servlet>

        <!-- 类名 -->

        <servlet-name>DisplayHeader</servlet-name>

        <!-- 所在的包 -->

        <servlet-class>com.runoob.test.DisplayHeader</servlet-class>

    </servlet>

    <servlet-mapping>

        <servlet-name>DisplayHeader</servlet-name>

        <!-- 访问的网址 -->

        <url-pattern>/TomcatTest/DisplayHeader</url-pattern>

    </servlet-mapping>

</web-app>
```

现在，调用上面的 **Servlet**，访问 **http://localhost:8080/TomcatTest/DisplayHeader** 会产生以下结果：

□

□ Servlet 表单数据

Servlet 服务器 HTTP 响应 □

□ 点我分享笔记

反馈/建议

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.IOException;

import java.io.PrintWriter;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.Date;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;
```

```

@WebServlet("/Refresh")

//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;charset=UTF-8");

        //使用默认时区和语言环境获得一个日历

        Calendar cale = Calendar.getInstance();

        //将Calendar类型转换成Date类型

        Date tasktime=cale.getTime();

        //设置日期输出的格式

        SimpleDateFormat df=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

        //格式化输出

        String nowTime = df.format(tasktime);

        PrintWriter out = response.getWriter();

        String title = "自动刷新 Header 设置 - 菜鸟教程实例";

        String docType =

            "<!DOCTYPE html>\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: " + nowTime + "</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);

    }

}

```

以上测试实例是位于 **TomcatTest** 项目下，对应的 **web.xml** 配置为：

```

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

<web-app>

    <servlet>

        <!-- 类名 -->

        <servlet-name>Refresh</servlet-name>

        <!-- 所在的包 -->

        <servlet-class>com.runoob.test.Refresh</servlet-class>

    </servlet>

    <servlet-mapping>

        <servlet-name>Refresh</servlet-name>

        <!-- 访问的网址 -->

        <url-pattern>/TomcatTest/Refresh</url-pattern>

    </servlet-mapping>

</web-app>

```

现在，调用上面的 **Servlet**，每隔 5 秒会显示当前系统时间。只要运行 **Servlet** 并稍等片刻，即可看到如下的结果：

自动刷新 Header 设置 - 菜鸟教程实例

当前时间是：2016-08-25 13:59:05

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	<i>Unused</i>	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	<i>您还不能使用该代码。</i>
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。 <code>setStatus</code> 方法接受一个 <code>int</code> （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 <code>PrintWriter</code> 实际返回任何内容之前调用 <code>setStatus</code> 。
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 <i>Location</i> 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.*;

@WebServlet("/showError")

// 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

[☐ Servlet 服务器 HTTP 响应](#)

[Servlet 编写过滤器](#) ☐

[☐ 点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[☐ Servlet HTTP 状态码](#)

[Servlet 异常处理](#) ☐

Servlet 编写过滤器

Servlet 过滤器可以动态地拦截请求和响应，以变换或使用包含在请求或响应中的信息。

可以将一个或多个 **Servlet** 过滤器附加到一个 **Servlet** 或一组 **Servlet**。**Servlet** 过滤器也可以附加到 **JavaServer Pages (JSP)** 文件和 **HTML** 页面。调用 **Servlet** 前调用所有附加的 **Servlet** 过滤器。

Servlet 过滤器是可用于 **Servlet** 编程的 **Java** 类，可以实现以下目的：

- 在客户端的请求访问后端资源之前，拦截这些请求。
- 在服务器的响应发送回客户端之前，处理这些响应。

根据规范建议的各种类型的过滤器：

- 身份验证过滤器（**Authentication Filters**）。
- 数据压缩过滤器（**Data compression Filters**）。
- 加密过滤器（**Encryption Filters**）。
- 触发资源访问事件过滤器。
- 图像转换过滤器（**Image Conversion Filters**）。
- 日志记录和审核过滤器（**Logging and Auditing Filters**）。
- MIME-TYPE** 链过滤器（**MIME-TYPE Chain Filters**）。
- 标记化过滤器（**Tokenizing Filters**）。
- XSL/T** 过滤器（**XSL/T Filters**），转换 **XML** 内容。

过滤器通过 **Web** 部署描述符（**web.xml**）中的 **XML** 标签来声明，然后映射到您的应用程序的部署描述符中的 **Servlet** 名称或 **URL** 模式。

当 **Web** 容器启动 **Web** 应用程序时，它会为您在部署描述符中声明的每一个过滤器创建一个实例。

Filter的执行顺序与在**web.xml**配置文件中的配置顺序一致，一般把**Filter**配置在所有的**Servlet**之前。

Servlet 过滤器方法

过滤器是一个实现了 **javax.servlet.Filter** 接口的 **Java** 类。**javax.servlet.Filter** 接口定义了三个方法：

序号	方法 & 描述
1	public void doFilter (ServletRequest, ServletResponse, FilterChain) 该方法完成实际的过滤操作，当客户端请求方法与过滤器设置匹配的URL时， Servlet 容器将先调用过滤器的 doFilter 方法。 FilterChain 用户访问后续过滤器。
2	public void init(FilterConfig filterConfig) web 应用程序启动时， web 服务器将创建 Filter 的实例对象，并调用其 init 方法，读取 web.xml 配置，完成对象的初始化功能，从而为后续的用户请求作好拦截的准备工作（ filter 对象只会创建一次， init 方法也只会执行一次）。开发人员通过 init 方法的参数，可获得代表当前 filter 配置信息的 FilterConfig 对象。
3	public void destroy() Servlet 容器在销毁过滤器实例前调用该方法，在该方法中释放 Servlet 过滤器占用的资源。

FilterConfig 使用

Filter 的 **init** 方法中提供了一个 **FilterConfig** 对象。

如 **web.xml** 文件配置如下：

```
<filter>

    <filter-name>LogFilter</filter-name>

    <filter-class>com.runoob.test.LogFilter</filter-class>

    <init-param>

        <param-name>Site</param-name>
```

```
<param-value>菜鸟教程</param-value>

</init-param>

</filter>
```

在 `init` 方法使用 `FilterConfig` 对象获取参数:

```
public void init(FilterConfig config) throws ServletException {

    // 获取初始化参数

    String site = config.getInitParameter("Site");

    // 输出初始化参数

    System.out.println("网站名称: " + site);

}
```

Servlet 过滤器实例

以下是 `Servlet` 过滤器的实例，将输出网站名称和地址。本实例让您对 `Servlet` 过滤器有基本的了解，您可以使用相同的概念编写更复杂的过滤器应用程序:

```
package com.runoob.test;

//导入必需的 java 库

import javax.servlet.*;

import java.util.*;

//实现 Filter 类

public class LogFilter implements Filter {

    public void init(FilterConfig config) throws ServletException {

        // 获取初始化参数

        String site = config.getInitParameter("Site");

        // 输出初始化参数

        System.out.println("网站名称: " + site);

    }

    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws java.io.IOException, ServletException {
```

```

        // 输出站点名称

        System.out.println("站点网址: http://www.runoob.com");

        // 把请求传回过滤链

        chain.doFilter(request,response);

    }

    public void destroy( ){

        /* 在 Filter 实例被 Web 容器从服务移除之前调用 */

    }

}

```

这边使用前文提到的 `DisplayHeader.java` 为例子:

```

//导入必需的 java 库

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;


@WebServlet("/DisplayHeader")


//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;charset=UTF-8");

PrintWriter out = response.getWriter();

String title = "HTTP Header 请求实例 - 菜鸟教程实例";

String docType =

    "<!DOCTYPE html> \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>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);

}

}

```

Web.xml 中的 Servlet 过滤器映射（Servlet Filter Mapping）

定义过滤器，然后映射到一个 URL 或 Servlet，这与定义 Servlet，然后映射到一个 URL 模式方式大致相同。在部署描述符文件 **web.xml** 中为 filter 标签创建下面的条目：

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app>

<filter>

    <filter-name>LogFilter</filter-name>

    <filter-class>com.runoob.test.LogFilter</filter-class>

    <init-param>

        <param-name>Site</param-name>

        <param-value>菜鸟教程</param-value>

    </init-param>

</filter>

<filter-mapping>

    <filter-name>LogFilter</filter-name>

    <url-pattern>/*</url-pattern>

</filter-mapping>

<servlet>

    <!-- 类名 -->

    <servlet-name>DisplayHeader</servlet-name>

    <!-- 所在的包 -->

    <servlet-class>com.runoob.test.DisplayHeader</servlet-class>

</servlet>

<servlet-mapping>

    <servlet-name>DisplayHeader</servlet-name>

    <!-- 访问的网址 -->

    <url-pattern>/TomcatTest/DisplayHeader</url-pattern>

</servlet-mapping>

</web-app>
```

上述过滤器适用于所有的 Servlet，因为我们在配置中指定 **/***。如果您只想在少数的 Servlet 上应用过滤器，您可以指定一个特定的 Servlet 路径。现在试着以常用的方式调用任何 Servlet，您将会看到在 Web 服务器中生成的日志。您也可以使用 Log4J 记录器来把上面的日志记录到一个单独的文件中。

接下来我们访问这个实例地址 **http://localhost:8080/TomcatTest/DisplayHeader**，然后在控制台看下输出内容，如下所示：

```
9 public void init(FilterConfig config) throws ServletException {
10     // 获取初始化参数
11     String site = config.getInitParameter("Site");
12
13     // 输出初始化参数
14     System.out.println("网站名称: " + site);
15 }
16 public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws java
17
18     // 输出站点名称
19     System.out.println("站点网址: http://www.runoob.com");
20
21     // 把请求传回过滤链
```

输出一次

请求一次执行一次

Console

Tomcat v8.0 Server at localhost [Apache Tomcat] /Library/Java/JavaVirtualMachines/jdk1.8.0_31.jdk/Contents/Home/bin/java (2016年8月25日 下午2)

严重: Context [/testjsp] startup failed due to previous errors

网站名称: 菜鸟教程

八月 25, 2016 2:57:52 下午 org.apache.coyote.AbstractProtocol start

信息: Starting ProtocolHandler ["http-nio-8080"]

八月 25, 2016 2:57:52 下午 org.apache.coyote.AbstractProtocol start

信息: Starting ProtocolHandler ["ajp-nio-8009"]

八月 25, 2016 2:57:52 下午 org.apache.catalina.startup.Catalina start

信息: Server startup in 443 ms

站点网址: http://www.runoob.com

站点网址: http://www.runoob.com

站点网址: http://www.runoob.com

站点网址: http://www.runoob.com

使用多个过滤器

Web 应用程序可以根据特定的目的定义若干个不同的过滤器。假设您定义了两个过滤器 *AuthenFilter* 和 *LogFilter*。您需要创建一个如下所述的不同的映射，其余的处理与上述所讲解的大致相同：

```
<filter>

    <filter-name>LogFilter</filter-name>

    <filter-class>com.runoob.test.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>com.runoob.test.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 容器应用过滤器到 `Servlet` 的顺序。若要反转过滤器的顺序，您只需要在 `web.xml` 文件中反转 `filter-mapping` 元素即可。

例如，上面的实例将先应用 `LogFilter`，然后再应用 `AuthenFilter`，但是下面的实例将颠倒这个顺序：

```
<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>
```

web.xml配置各节点说明

`<filter>`指定一个过滤器。

`<filter-name>`用于为过滤器指定一个名字，该元素的内容不能为空。

`<filter-class>`元素用于指定过滤器的完整的限定类名。

`<init-param>`元素用于为过滤器指定初始化参数，它的子元素`<param-name>`指定参数的名字，`<param-value>`指定参数的值。

在过滤器中，可以使用`FilterConfig`接口对象来访问初始化参数。

`<filter-mapping>`元素用于设置一个 **Filter** 所负责拦截的资源。一个**Filter**拦截的资源可通过两种方式来指定：**Servlet** 名称和资源访问的请求路径

`<filter-name>`子元素用于设置**filter**的注册名称。该值必须是在`<filter>`元素中声明过的过滤器的名字

<url-pattern>设置 **filter** 所拦截的请求路径(过滤器关联的URL样式)

<servlet-name>指定过滤器所拦截的**Servlet**名称。

<dispatcher>指定过滤器所拦截的资源被 **Servlet** 容器调用的方式，可以是REQUEST,INCLUDE,FORWARD和ERROR之一，默认REQUEST。用户可以设置多个<dispatcher>子元素用来指定 **Filter** 对资源的多种调用方式进行拦截。

<dispatcher>子元素可以设置的值及其意义

REQUEST：当用户直接访问页面时，**Web**容器将会调用过滤器。如果目标资源是通过**RequestDispatcher**的**include()**或**forward()**方法访问时，那么该过滤器就不会被调用。

INCLUDE：如果目标资源是通过**RequestDispatcher**的**include()**方法访问时，那么该过滤器将被调用。除此之外，该过滤器不会被调用。

FORWARD：如果目标资源是通过**RequestDispatcher**的**forward()**方法访问时，那么该过滤器将被调用，除此之外，该过滤器不会被调用。

ERROR：如果目标资源是通过声明式异常处理机制调用时，那么该过滤器将被调用。除此之外，过滤器不会被调用。

[☐ Servlet HTTP 状态码](#)

[Servlet 异常处理 ☐](#)



1 篇笔记
#1

[☐ 写笔记](#)



过滤器中我们可以根据 **doFilter()** 方法中的 **request** 对象获取表单参数信息，例如我们可以获取到请求的用户名和密码进行逻辑处理，也可以通过 **response** 对用户做出回应。比如如果验证用户名不正确，禁止用户访问 **web** 资源，并且向浏览器输出提示，告诉用户用户名或者密码不正确等等；

```
public void doFilter(ServletRequest req, ServletResponse resp,
    FilterChain chain) throws IOException, ServletException {

    // 获取请求信息(测试时可以通过get方式在URL中添加name)

    //http://localhost:8080/servlet_demo/helloworld?name=123

    String name = req.getParameter("name");

    // 过滤器核心代码逻辑

    System.out.println("过滤器获取请求参数:"+name);

    System.out.println("第二个过滤器执行--网站名称: www.runoob.com");

    if("123".equals(name)){

        // 把请求传回过滤链

        chain.doFilter(req, resp);

    }else{

        //设置返回内容类型

        resp.setContentType("text/html;charset=GBK");

        //在页面输出响应信息
```

```
PrintWriter out = resp.getWriter();

out.print("<b>name不正确，请求被拦截，不能访问web资源</b>");

System.out.println("name不正确，请求被拦截，不能访问web资源");

}

}
```

孙大圣1年前 (2017-08-30)

反馈/建议



Servlet 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 中搜索与抛出异常类型相匹配的配置。您必须在 web.xml 中使用 error-page 元素来指定对特定异常 或 HTTP 状态码 作出相应的 Servlet 调用。

web.xml 配置

假设，有一个 ErrorHandler 的 Servlet 在任何已定义的异常或错误出现时被调用。以下将是在 web.xml 中创建的项。

```
<!-- servlet 定义 -->

<servlet>

    <servlet-name>ErrorHandler</servlet-name>

    <servlet-class>ErrorHandler</servlet-class>

</servlet>

<!-- servlet 映射 -->

<servlet-mapping>

    <servlet-name>ErrorHandler</servlet-name>

    <url-pattern>/ErrorHandler</url-pattern>

</servlet-mapping>

<!-- error-code 相关的错误页面 -->
```

```

<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 相关的错误页面 -->

<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>

```

如果您想对所有的异常有一个通用的错误处理程序，那么应该定义下面的 **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>

```

以下是关于上面的 **web.xml** 异常处理要注意的点：

Servlet ErrorHandler 与其他的 **Servlet** 的定义方式一样，且在 **web.xml** 中进行配置。

如果有错误状态代码出现，不管为 **404**（**Not Found** 未找到）或 **403**（**Forbidden** 禁止），则会调用 **ErrorHandler** 的 **Servlet**。

如果 **Web** 应用程序抛出 ***ServletException*** 或 ***IOException***，那么 **Web** 容器会调用 **ErrorHandler** 的 **Servlet**。

您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。上面的实例是非常通用的，希望您能通过实例理解基本的概念。

请求属性 - 错误/异常

以下是错误处理的 `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	<code>javax.servlet.error.status_code</code> 该属性给出状态码，状态码可被存储，并在存储为 <code>java.lang.Integer</code> 数据类型后可被分析。
2	<code>javax.servlet.error.exception_type</code> 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 <code>java.lang.Class</code> 数据类型后可被分析。
3	<code>javax.servlet.error.message</code> 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 <code>java.lang.String</code> 数据类型后可被分析。
4	<code>javax.servlet.error.request_uri</code> 该属性给出有关 URL 调用 <code>Servlet</code> 的信息，信息可被存储，并在存储为 <code>java.lang.String</code> 数据类型后可被分析。
5	<code>javax.servlet.error.exception</code> 该属性给出异常产生的信息，信息可被存储，并在存储为 <code>java.lang.Throwable</code> 数据类型后可被分析。
6	<code>javax.servlet.error.servlet_name</code> 该属性给出 <code>Servlet</code> 的名称，名称可被存储，并在存储为 <code>java.lang.String</code> 数据类型后可被分析。

Servlet 错误处理程序实例

以下是 `Servlet` 实例，将应对任何您所定义的错误或异常发生时的错误处理程序。

本实例让您对 `Servlet` 中的异常处理有基本的了解，您可以使用相同的概念编写更复杂的异常处理应用程序：

```

//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 ErrorHandler extends HttpServlet {

    // 处理 GET 方法请求的方法

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

        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";

}

// 设置响应内容类型

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

PrintWriter out = response.getWriter();

String title = "菜鸟教程 Error/Exception 信息";

String docType = "<!DOCTYPE html>\n";

out.println(docType +

    "<html>\n" +

    "<head><title>" + title + "</title></head>\n" +

    "<body bgcolor=\"#f0f0f0\">\n");

out.println("<h1>菜鸟教程异常信息实例演示</h1>");

if (throwable == null && statusCode == null){

    out.println("<h2>错误信息丢失</h2>");

    out.println("请返回 <a href=\"\" +

response.encodeURL("http://localhost:8080/") +

    "\">主页</a>。");

}else if (statusCode != null) {

    out.println("错误代码 : " + statusCode);

}else{

    out.println("<h2>错误信息</h2>");

    out.println("Servlet Name : " + servletName +

        "</br></br>");

    out.println("异常类型 : " +
```

```

        throwable.getClass( ).getName( ) +

        "</br></br>");

    out.println("请求 URI: " + requestUri +

        "<br><br>");

    out.println("异常信息: " +

        throwable.getMessage( ));

}

out.println("</body>");

out.println("</html>");

}

// 处理 POST 方法请求的方法

public void doPost(HttpServletRequest request,

        HttpServletResponse response)

    throws ServletException, IOException {

    doGet(request, response);

}

}

```

以通常的方式编译 **ErrorHandler.java**，把您的类文件放入<Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes 中。
 让我们在 **web.xml** 文件中添加如下配置来处理异常：

```

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

<web-app>

<servlet>

    <servlet-name>ErrorHandler</servlet-name>

    <servlet-class>com.runoob.test.ErrorHandler</servlet-class>

</servlet>

<!-- servlet mappings -->

<servlet-mapping>

    <servlet-name>ErrorHandler</servlet-name>

    <url-pattern>/TomcatTest/ErrorHandler</url-pattern>

</servlet-mapping>

<error-page>

```

```
<error-code>404</error-code>

<location>/TomcatTest/ErrorHandler</location>

</error-page>

<error-page>

    <exception-type>java.lang.Throwable</exception-type >

    <location>/ErrorHandler</location>

</error-page>

</web-app>
```

现在，尝试使用一个会产生异常的 **Servlet**，或者输入一个错误的 URL，这将触发 Web 容器调用 **ErrorHandler** 的 **Servlet**，并显示适当的消息。例如，如果您输入了一个错误的 URL（如：<http://localhost:8080/TomcatTest/UnKonwPage>），那么它将显示下面的结果：

菜鸟教程异常信息实例演示

错误代码：404

[Servlet 编写过滤器](#)

[Servlet Cookie 处理](#)

[点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[Servlet 异常处理](#)

[Servlet Session 跟踪](#)

Servlet Cookie 处理

Cookie 是存储在客户端计算机上的文本文件，并保留了各种跟踪信息。Java Servlet 显然支持 HTTP Cookie。

识别返回用户包括三个步骤：

服务器脚本向浏览器发送一组 **Cookie**。例如：姓名、年龄或识别号码等。

浏览器将这些信息存储在本地计算机上，以备将来使用。

当下一次浏览器向 Web 服务器发送任何请求时，浏览器会把这些 **Cookie** 信息发送到服务器，服务器将使用这些信息来识别用户。

本章将向您讲解如何设置或重置 **Cookie**，如何访问它们，以及如何将它们删除。

Servlet Cookie 处理需要对中文进行编码与解码，方法如下：


```
String str = java.net.URLEncoder.encode("中文", "UTF-8");           //编码

String str = java.net.URLDecoder.decode("编码后的字符串", "UTF-8"); // 解码
```

Cookie 剖析

Cookie 通常设置在 HTTP 头信息中（虽然 JavaScript 也可以直接在浏览器上设置一个 Cookie）。设置 Cookie 的 Servlet 会发送如下的头信息：

```
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=runoob.com

Connection: close

Content-Type: text/html
```

正如您所看到的，Set-Cookie 头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。expires 字段是一个指令，告诉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 Cookie，它将会保留此信息直到到期日期。如果用户的浏览器指向任何匹配该 Cookie 的路径和域的页面，它会重新发送 Cookie 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
```

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 Cookie 方法

以下是在 Servlet 中操作 Cookie 时可使用的有用的方法列表。

序号	方法 & 描述
1	public void setDomain(String pattern) 该方法设置 cookie 适用的域，例如 runoob.com。

2	public String getDomain() 该方法获取 cookie 适用的域，例如 runoob.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) 设置cookie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() 获取 cookie 的注释，如果 cookie 没有注释则返回 null。

通过 Servlet 设置 Cookie

通过 Servlet 设置 Cookie 包括三个步骤：

(1) 创建一个 Cookie 对象：您可以调用带有 cookie 名称和 cookie 值的 Cookie 构造函数，cookie 名称和 cookie 值都是字符串。

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

请记住，无论是名字还是值，都不应该包含空格或以下任何字符：

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

(2) 设置最大生存周期：您可以使用 setMaxAge 方法来指定 cookie 能够保持有效的时间（以秒为单位）。下面将设置一个最长有效期为 24 小时的 cookie。

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

(3) 发送 Cookie 到 HTTP 响应头：您可以使用 response.addCookie 来添加 HTTP 响应头中的 Cookie，如下所示：

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

实例

让我们修改我们的 [表单数据实例](#)，为名字和姓氏设置 **Cookie**。

```
package com.runoob.test;

import java.io.IOException;

import java.io.PrintWriter;

import java.net.URLEncoder;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class HelloServlet
 */
@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)
 */

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{
    // 为名字和姓氏创建 Cookie

    Cookie name = new Cookie("name",

        URLEncoder.encode(request.getParameter("name"), "UTF-8")); // 中文转码

    Cookie url = new Cookie("url",

        request.getParameter("url"));

    // 为两个 Cookie 设置过期日期为 24 小时后

    name.setMaxAge(60*60*24);

    url.setMaxAge(60*60*24);

    // 在响应头中添加两个 Cookie

    response.addCookie( name );

    response.addCookie( url );

    // 设置响应内容类型

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

    PrintWriter out = response.getWriter();

    String title = "设置 Cookie 实例";

    String docType = "<!DOCTYPE html>\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>站点名: </b>:"

```

```

        + request.getParameter("name") + "\n</li>" +

        "    <li><b>站点 URL: </b>: "

        + request.getParameter("url") + "\n</li>" +

        "</ul>\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);

}

}

```

编译上面的 **Servlet HelloForm**，并在 **web.xml** 文件中创建适当的条目：

```

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

<web-app>

    <servlet>

        <!-- 类名 -->

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

        <!-- 所在的包 -->

        <servlet-class>com.runoob.test.HelloForm</servlet-class>

    </servlet>

    <servlet-mapping>

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

        <!-- 访问的网址 -->

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

    </servlet-mapping>

</web-app>

```

最后尝试下面的 **HTML** 页面来调用 **Servlet**。

```
<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

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

  站点名 : <input type="text" name="name">

  <br />

  站点 URL: <input type="text" name="url" /><br>

  <input type="submit" value="提交" />

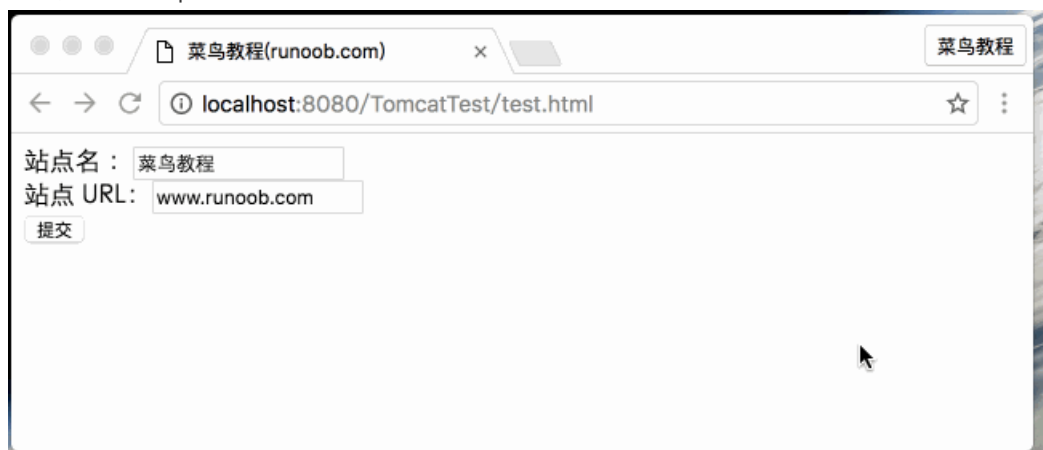
</form>

</body>

</html>
```

保存上面的 **HTML** 内容到文件 `/TomcatTest/test.html` 中。

接下来我们访问<http://localhost:8080/TomcatTest/test.html>，Gif 演示如下：



注意： 以上的一些路径需要根据你项目实际路径修改。

通过 **Servlet** 读取 **Cookie**

要读取 **Cookie**，您需要通过调用 `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** 和关联的值。

实例

让我们读取上面的实例中设置的 **Cookie**

```
package com.runoob.test;

import java.io.IOException;

import java.io.PrintWriter;

import java.net.URLDecoder;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class ReadCookies
 */
@WebServlet("/ReadCookies")

public class ReadCookies extends HttpServlet {

    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public ReadCookies() {

        super();

        // TODO Auto-generated constructor stub

    }

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

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
```

```

{

    Cookie cookie = null;

    Cookie[] cookies = null;

    // 获取与该域相关的 Cookie 的数组

    cookies = request.getCookies();


    // 设置响应内容类型

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


    PrintWriter out = response.getWriter();

    String title = "Delete Cookie Example";

    String docType = "<!DOCTYPE html>\n";

    out.println(docType +

        "<html>\n" +

        "<head><title>" + title + "</title></head>\n" +

        "<body bgcolor=\"\#f0f0f0\">\n" );

    if( cookies != null ){

        out.println("<h2>Cookie 名称和值</h2>");

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

            cookie = cookies[i];

            if((cookie.getName( )).compareTo("name") == 0 ){

                cookie.setMaxAge(0);

                response.addCookie(cookie);

                out.print("已删除的 cookie: " +

                    cookie.getName( ) + "<br/>");

            }

            out.print("名称: " + cookie.getName( ) + ", ");

            out.print("值: " + URLDecoder.decode(cookie.getValue(), "utf-8") + "<br/>");

        }

    }else{

        out.println(

            "<h2 class=\"\#tutheader\">No Cookie founds</h2>");

    }

}

```



```

        out.println("</body>");

        out.println("</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);

    }

}

```

编译上面的 **Servlet ReadCookies**，并在 `web.xml` 文件中创建适当的条目。尝试运行 `http://localhost:8080/TomcatTest/ReadCookies`，将显示如下结果：

Cookie 名称和值

名称: JSESSIONID, 值: 60A8DBEC591EC628B0C572AA5EBF5DD9
 名称: name, 值: 菜鸟教程
 名称: url, 值: www.runoob.com
 名称: CNZZDATA1254569789, 值: 1732093401-1466493487-|1468220728
 名称: pgv_pvid, 值: 1015625625

通过 Servlet 删除 Cookie

删除 Cookie 是非常简单的。如果您想删除一个 cookie，那么您只需要按照以下三个步骤进行：

1. 读取一个现有的 cookie，并把它存储在 Cookie 对象中。
2. 使用 `setMaxAge()` 方法设置 cookie 的年龄为零，来删除现有的 cookie。
3. 把这个 cookie 添加到响应头。

实例

下面的例子将删除现有的名为 "url" 的 cookie，当您下次运行 `ReadCookies` 的 Servlet 时，它会返回 url 为 null。

```

package com.runoob.test;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

```

```
import javax.servlet.annotation.WebServlet;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;


/**
 * Servlet implementation class DeleteCookies
 */
@WebServlet("/DeleteCookies")

public class DeleteCookies extends HttpServlet {

    private static final long serialVersionUID = 1L;


    /**
     * @see HttpServlet#HttpServlet()
     */
    public DeleteCookies() {

        super();

        // TODO Auto-generated constructor stub
    }


    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
    {

        Cookie cookie = null;

        Cookie[] cookies = null;

        // 获取与该域相关的 Cookie 的数组

        cookies = request.getCookies();


        // 设置响应内容类型
```

```

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

PrintWriter out = response.getWriter();

String title = "删除 Cookie 实例";

String docType = "<!DOCTYPE html>\n";

out.println(docType +

    "<html>\n" +

    "<head><title>" + title + "</title></head>\n" +

    "<body bgcolor=\"#f0f0f0\">\n" );

if( cookies != null ){

    out.println("<h2>Cookie 名称和值</h2>");

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

        cookie = cookies[i];

        if((cookie.getName( )).compareTo("url") == 0 ){

            cookie.setMaxAge(0);

            response.addCookie(cookie);

            out.print("已删除的 cookie: " +

                cookie.getName( ) + "<br/>");

        }

        out.print("名称: " + cookie.getName( ) + ", ");

        out.print("值: " + cookie.getValue( )+" <br/>");

    }

}else{

    out.println(

        "<h2 class=\"tutheader\">No Cookie founds</h2>");

    }

out.println("</body>");

out.println("</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);  
  
}  
  
}
```

编译上面的 **Servlet DeleteCookies**, 并在 **web.xml** 文件中创建适当的条目。现在运行 **http://localhost:8080/TomcatTest/DeleteCookies**, 将显示如下结果:

Cookie 名称和值
名称: JSESSIONID, 值: 60A8DBEC591EC628B0C572AA5EBF5DD9
已删除的 cookie: url
名称: url, 值: www.runoob.com
名称: CNZZDATA1254569789, 值: 1732093401-1466493487-%7C1468220728
名称: pgv_pvid, 值: 1015625625

☐ Servlet 异常处理

Servlet Session 跟踪 ☐

☐ 点我分享笔记

反馈/建议



☐ Servlet Cookie 处理

Servlet 数据库访问 ☐

Servlet Session 跟踪

HTTP 是一种"无状态"协议, 这意味着每次客户端检索网页时, 客户端打开一个单独的连接到 Web 服务器, 服务器会自动不保留之前客户端请求的任何记录。

但是仍然有以下三种方式来维持 Web 客户端和 Web 服务器之间的 session 会话:

Cookies

一个 Web 服务器可以分配一个唯一的 session 会话 ID 作为每个 Web 客户端的 cookie, 对于客户端的后续请求可以使用接收到的 cookie 来识别。这可能不是一个有效的方法, 因为很多浏览器不支持 cookie, 所以我们建议不要使用这种方式来维持 session 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 session 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` 值可以用于保持不同的 Web 浏览器的跟踪。

这可能是一种保持 session 会话跟踪的有效方式，但是点击常规的超文本链接（<A HREF...>）不会导致表单提交，因此隐藏的表单字段也不支持常规的 session 会话跟踪。

URL 重写

您可以在每个 URL 末尾追加一些额外的数据来标识 session 会话，服务器会把该 session 会话标识符与已存储的有关 session 会话的数据相关联。

例如，`http://w3cschool.cc/file.htm;sessionId=12345`，session 会话标识符被附加为 `sessionId=12345`，标识符可被 Web 服务器访问以识别客户端。

URL 重写是一种更好的维持 session 会话的方式，它在浏览器不支持 cookie 时能够很好地工作，但是它的缺点是会动态生成每个 URL 来为页面分配一个 session 会话 ID，即使是在很简单的静态 HTML 页面中也会如此。

HttpSession 对象

除了上述的三种方式，Servlet 还提供了 HttpSession 接口，该接口提供了一种跨多个页面请求或访问网站时识别用户以及存储有关用户信息的方式。Servlet 容器使用这个接口来创建一个 HTTP 客户端和 HTTP 服务器之间的 session 会话。会话持续一个指定的时间段，跨多个连接或页面请求。

您会通过调用 `HttpServletRequest` 的公共方法 `getSession()` 来获取 HttpSession 对象，如下所示：

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

你需要在向客户端发送任何文档内容之前调用 `request.getSession()`。下面总结了 HttpSession 对象中可用的几个重要的方法：

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 日午夜算起，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 日午夜算起，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与该 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。

Session 跟踪实例

本实例说明了如何使用 `HttpSession` 对象获取 `session` 会话创建时间和最后访问时间。如果不存在 `session` 会话，我们将通过请求创建一个新的 `session` 会话。

```
package com.runoob.test;

import java.io.IOException;

import java.io.PrintWriter;

import java.text.SimpleDateFormat;

import java.util.Date;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

/**
 * Servlet implementation class SessionTrack
 */
@WebServlet("/SessionTrack")

public class SessionTrack extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
    {
        // 如果不存在 session 会话，则创建一个 session 对象

        HttpSession session = request.getSession(true);

        // 获取 session 创建时间

        Date createTime = new Date(session.getCreationTime());

        // 获取该网页的最后一次访问时间

        Date lastAccessTime = new Date(session.getLastAccessedTime());
```

```
//设置日期输出的格式

SimpleDateFormat df=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");


String title = "Servlet Session 实例 - 菜鸟教程";

Integer visitCount = new Integer(0);

String visitCountKey = new String("visitCount");

String userIDKey = new String("userID");

String userID = new String("Runoob");

if(session.getAttribute(visitCountKey) == null) {

    session.setAttribute(visitCountKey, new Integer(0));

}


// 检查网页上是否有新的访问者

if (session.isNew()){

    title = "Servlet Session 实例 - 菜鸟教程";

    session.setAttribute(userIDKey, userID);

} else {

    visitCount = (Integer)session.getAttribute(visitCountKey);

    visitCount = visitCount + 1;

    userID = (String)session.getAttribute(userIDKey);

}

session.setAttribute(visitCountKey, visitCount);


// 设置响应内容类型

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

PrintWriter out = response.getWriter();

String docType = "<!DOCTYPE html>\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 信息</h2>\n" +

"<table border=\"1\" align=\"center\">\n" +

"<tr bgcolor=\"#949494\">\n" +

"  <th>Session 信息</th><th>值</th></tr>\n" +

"<tr>\n" +

"  <td>id</td>\n" +

"  <td>" + session.getId() + "</td></tr>\n" +

"<tr>\n" +

"  <td>创建时间</td>\n" +

"  <td>" + df.format(createTime) +

"  </td></tr>\n" +

"<tr>\n" +

"  <td>最后访问时间</td>\n" +

"  <td>" + df.format(lastAccessTime) +

"  </td></tr>\n" +

"<tr>\n" +

"  <td>用户 ID</td>\n" +

"  <td>" + userID +

"  </td></tr>\n" +

"<tr>\n" +

"  <td>访问统计: </td>\n" +

"  <td>" + visitCount + "</td></tr>\n" +

"</table>\n" +

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

}

```

编译上面的 **Servlet SessionTrack**，并在 **web.xml** 文件中创建适当的条目。

```
<?xml version="1.0" encoding="UTF-8"?>
```



```
<web-app>

<servlet>

    <!-- 类名 -->

    <servlet-name>SessionTrack</servlet-name>

    <!-- 所在的包 -->

    <servlet-class>com.runoob.test.SessionTrack</servlet-class>

</servlet>

<servlet-mapping>

    <servlet-name>SessionTrack</servlet-name>

    <!-- 访问的网址 -->

    <url-pattern>/TomcatTest/SessionTrack</url-pattern>

</servlet-mapping>

</web-app>
```

在浏览器地址栏输入 `http://localhost:8080/TomcatTest/SessionTrack`，当您第一次运行时将显示如下结果：

Servlet Session 实例 - 菜鸟教程

Session 信息

Session 信息	值
id	2728DD80FEE0935ED601D108FCED54EC
创建时间	2016-08-26 09:56:06
最后访问时间	2016-08-26 09:56:06
用户 ID	Runoob
访问统计：	1

再次尝试运行相同的 Servlet，它将显示如下结果：

Servlet Session 实例 - 菜鸟教程

Session 信息

Session 信息	值
id	2728DD80FEE0935ED601D108FCED54EC
创建时间	2016-08-26 09:56:06
最后访问时间	2016-08-26 09:56:09
用户 ID	Runoob
访问统计：	2

删除 Session 会话数据

当您完成了一个用户的 session 会话数据，您有以下几种选择：

移除一个特定的属性：您可以调用 `public void removeAttribute(String name)` 方法来删除与特定的键相关联的值。

删除整个 **session** 会话：您可以调用 `public void invalidate()` 方法来丢弃整个 **session** 会话。

设置 **session** 会话过期时间：您可以调用 `public void setMaxInactiveInterval(int interval)` 方法来单独设置 **session** 会话超时。

注销用户：如果使用的是支持 **Servlet 2.4** 的服务器，您可以调用 **logout** 来注销 **Web** 服务器的客户端，并把属于所有用户的所有 **session** 会话设置为无效。

web.xml 配置：如果您使用的是 **Tomcat**，除了上述方法，您还可以在 **web.xml** 文件中配置 **session** 会话超时，如下所示：

```
<session-config>

    <session-timeout>15</session-timeout>

</session-config>
```

上面实例中的超时时间是以分钟为单位，将覆盖 **Tomcat** 中默认的 30 分钟超时时间。

在一个 **Servlet** 中的 `getMaxInactiveInterval()` 方法会返回 **session** 会话的超时时间，以秒为单位。所以，如果在 **web.xml** 中配置 **session** 会话超时时间为 15 分钟，那么 `getMaxInactiveInterval()` 会返回 900。

[☐ Servlet Cookie 处理](#)

[Servlet 数据库访问 ☐](#)

[☐ 点我分享笔记](#)

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1

☐

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[☐ Servlet Session 跟踪](#)

[Servlet 文件上传 ☐](#)

Servlet 数据库访问

本教程假定您已经了解了 **JDBC** 应用程序的工作方式。在您开始学习 **Servlet** 数据库访问之前，请访问 [Java MySQL 连接](#) 来设置相关驱动及配置。

注意：

你可以下载本站提供的 **jar** 包：[mysql-connector-java-5.1.39-bin.jar](#)

在 **java** 项目中，只需要在 **Eclipse** 中引入 **mysql-connector-java-5.1.39-bin.jar** 就可以运行 **java** 项目。

但是在 **Eclipse web** 项目中，当执行 `Class.forName("com.mysql.jdbc.Driver");` 时不会去查找驱动的。所以本实例中我们需要把 **mysql-connector-java-5.1.39-bin.jar** 拷贝到 **tomcat** 下 **lib** 目录。

从基本概念下手，让我们来创建一个简单的表，并在表中创建几条记录。

创建测试数据

接下来我们在 **MySQL** 中创建 **RUNOOB** 数据库，并创建 **websites** 数据表，表结构如下：

```
CREATE TABLE `websites` (

  `id` int(11) NOT NULL AUTO_INCREMENT,

  `name` char(20) NOT NULL DEFAULT '' COMMENT '站点名称',

  `url` varchar(255) NOT NULL DEFAULT '',

  `alexa` int(11) NOT NULL DEFAULT '0' COMMENT 'Alexa 排名',

  `country` char(10) NOT NULL DEFAULT '' COMMENT '国家',

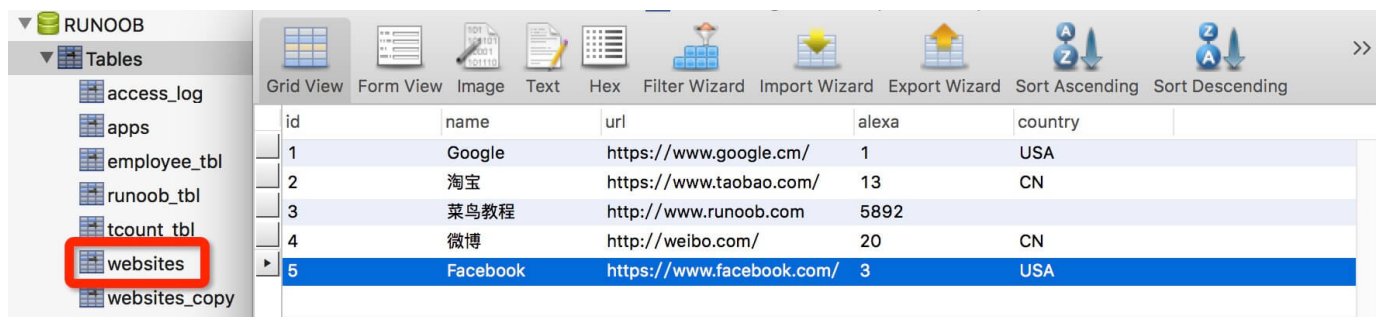
  PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8;
```

插入一些数据:

```
INSERT INTO `websites` VALUES ('1', 'Google', 'https://www.google.cm/', '1', 'USA'), ('2', '淘宝', 'https://www.taobao.com/', '13', 'CN'), ('3', '菜鸟教程', 'http://www.runoob.com', '5892', ''), ('4', '微博', 'http://weibo.com/', '20', 'CN'), ('5', 'Facebook', 'https://www.facebook.com/', '3', 'USA');
```

数据表显示如下:



id	name	url	alexa	country
1	Google	https://www.google.cm/	1	USA
2	淘宝	https://www.taobao.com/	13	CN
3	菜鸟教程	http://www.runoob.com	5892	
4	微博	http://weibo.com/	20	CN
5	Facebook	https://www.facebook.com/	3	USA

访问数据库

下面的实例演示了如何使用 [Servlet](#) 访问 [RUNOOB](#) 数据库。

```
package com.runoob.test;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.*;

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 DatabaseAccess
 */
@WebServlet("/DatabaseAccess")

public class DatabaseAccess extends HttpServlet {

    private static final long serialVersionUID = 1L;

    // JDBC 驱动名及数据库 URL

    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";

    static final String DB_URL = "jdbc:mysql://localhost:3306/RUNOOB";

    // 数据库的用户名与密码，需要根据自己的设置

    static final String USER = "root";

    static final String PASS = "123456";

    /**
     * @see HttpServlet#HttpServlet()
     */

    public DatabaseAccess() {

        super();

        // TODO Auto-generated constructor stub

    }

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

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

        Connection conn = null;

        Statement stmt = null;

        // 设置响应内容类型

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

        PrintWriter out = response.getWriter();
```

```
String title = "Servlet Mysql 测试 - 菜鸟教程";

String docType = "<!DOCTYPE html>\n";

out.println(docType +

"<html>\n" +

"<head><title>" + title + "</title></head>\n" +

"<body bgcolor=\"#f0f0f0\">\n" +

"<h1 align=\"center\">" + title + "</h1>\n");

try{

    // 注册 JDBC 驱动器

    Class.forName("com.mysql.jdbc.Driver");

    // 打开一个连接

    conn = DriverManager.getConnection(DB_URL,USER,PASS);

    // 执行 SQL 查询

    stmt = conn.createStatement();

    String sql;

    sql = "SELECT id, name, url FROM websites";

    ResultSet rs = stmt.executeQuery(sql);

    // 展开结果集数据库

    while(rs.next()){

        // 通过字段检索

        int id = rs.getInt("id");

        String name = rs.getString("name");

        String url = rs.getString("url");

        // 输出数据

        out.println("ID: " + id);

        out.println(", 站点名称: " + name);

        out.println(", 站点 URL: " + url);

        out.println("<br />");

    }

}
```

```

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

        // 完成后关闭

        rs.close();

        stmt.close();

        conn.close();

    } catch(SQLException se) {

        // 处理 JDBC 错误

        se.printStackTrace();

    } catch(Exception e) {

        // 处理 Class.forName 错误

        e.printStackTrace();

    }finally{

        // 最后是用于关闭资源的块

        try{

            if(stmt!=null)

                stmt.close();

        }catch(SQLException se2){

        }

        try{

            if(conn!=null)

                conn.close();

        }catch(SQLException se){

            se.printStackTrace();

        }

    }

}

/**
 * @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);

}

}
```

现在让我们来编译上面的 Servlet，并在 web.xml 文件中创建以下条目：

```
....

<servlet>

    <servlet-name>DatabaseAccess</servlet-name>

    <servlet-class>com.runoob.test.DatabaseAccess</servlet-class>

</servlet>

<servlet-mapping>

    <servlet-name>DatabaseAccess</servlet-name>

    <url-pattern>/TomcatTest/DatabaseAccess</url-pattern>

</servlet-mapping>

....
```

现在调用这个 Servlet，输入链接：<http://localhost:8080/TomcatTest/DatabaseAccess>，将显示以下响应结果：

Servlet Mysql 测试 - 菜鸟教程

ID: 1, 站点名称: Google, 站点 URL: <https://www.google.cm/>
ID: 2, 站点名称: 淘宝, 站点 URL: <https://www.taobao.com/>
ID: 3, 站点名称: 菜鸟教程, 站点 URL: <http://www.runoob.com>
ID: 4, 站点名称: 微博, 站点 URL: <http://weibo.com/>
ID: 5, 站点名称: Facebook, 站点 URL: <https://www.facebook.com/>

☐ Servlet Session 跟踪

Servlet 文件上传 ☐



1 篇笔记
#1



☐ 写笔记

进行数据库插入操作的时候使用 PreparedStatement 更好，好处如下：

- 1.PreparedStatement可以写动态参数化的查询；
- 2.PreparedStatement比 Statement 更快；
- 3.PreparedStatement可以防止SQL注入式攻击

实例：

```
//编写预处理 SQL 语句
```

```
String sql= "INSERT INTO websites1 VALUES(?,?,?,?)";
```

```
//实例化 PreparedStatement

ps = conn.prepareStatement(sql);


//传入参数，这里的参数来自于一个带有表单的jsp文件，很容易实现

ps.setString(1, request.getParameter("id"));

ps.setString(2, request.getParameter("name"));

ps.setString(3, request.getParameter("url"));

ps.setString(4, request.getParameter("alexa"));

ps.setString(5, request.getParameter("country"));


//执行数据库更新操作，不需要SQL 语句

ps.executeUpdate();

sql = "SELECT id, name, url FROM websites1";

ps = conn.prepareStatement(sql);


//获取查询结果

ResultSet rs = ps.executeQuery();
```

Alan_scut2年前 (2017-03-23)

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

☐ Servlet 数据库访问

Servlet 处理日期 ☐

Servlet 文件上传

Servlet 可以与 HTML form 标签一起使用，来允许用户上传文件到服务器。上传的文件可以是文本文件或图像文件或任何文档。

本文使用到的文件有：

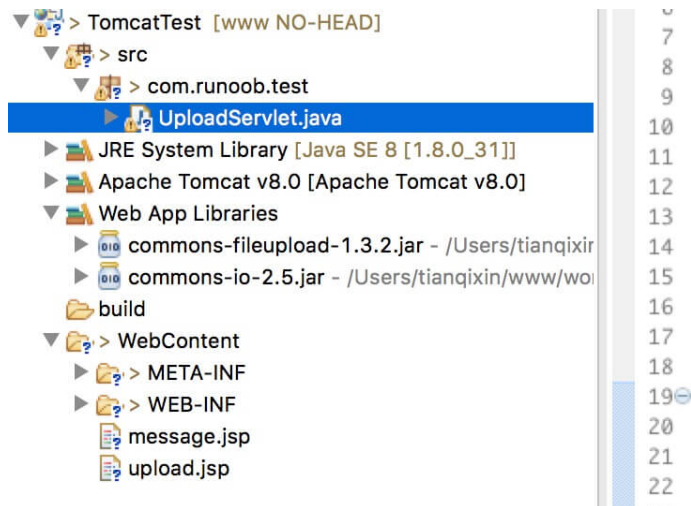
upload.jsp：文件上传表单。

message.jsp：上传成功后跳转页面。

UploadServlet.java : 上传处理 Servlet。

需要引入的 jar 文件: commons-fileupload-1.3.2、commons-io-2.5.jar。

结构图如下所示:



注意: Servlet3.0 已经内置了文件上传这一特性, 开发者不再需要将 Commons FileUpload 组件导入到工程中去。

接下来我们详细介绍。

创建一个文件上传表单

下面的 HTML 代码创建了一个文件上传表单。以下几点需要注意:

表单 **method** 属性应该设置为 **POST** 方法, 不能使用 **GET** 方法。

表单 **enctype** 属性应该设置为 **multipart/form-data**。

表单 **action** 属性应该设置为在后端服务器上处理文件上传的 **Servlet** 文件。下面的实例使用了 **UploadServlet Servlet** 来上传文件。

上传单个文件, 您应该使用单个带有属性 **type="file"** 的 **<input .../>** 标签。为了允许多个文件上传, 请包含多个 **name** 属性值不同的 **input** 标签。输入标签具有不同的名称属性的值。浏览器会为每个 **input** 标签关联一个浏览按钮。

upload.jsp 文件代码如下:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>文件上传实例 - 菜鸟教程</title>

</head>

<body>

<h1>文件上传实例 - 菜鸟教程</h1>

<form method="post" action="/TomcatTest/UploadServlet" enctype="multipart/form-data">
```

选择一个文件：

```
<input type="file" name="uploadFile" />
```

```
<br/><br/>
```

```
<input type="submit" value="上传" />
```

```
</form>
```

```
</body>
```

```
</html>
```

编写后台 Servlet

以下是 `UploadServlet` 的源代码，同于处理文件上传，在这之前我们先确保依赖包已经引入到项目的 `WEB-INF/lib` 目录下：

下面的实例依赖于 `FileUpload`，所以一定要确保在您的 `classpath` 中有最新版本的 `commons-fileupload.x.x.jar` 文件。可以从 <http://commons.apache.org/proper/commons-fileupload/> 下载。

`FileUpload` 依赖于 `Commons IO`，所以一定要确保在您的 `classpath` 中有最新版本的 `commons-io-x.x.jar` 文件。可以从 <http://commons.apache.org/proper/commons-io/> 下载。

你可以直接下载本站提供的两个依赖包：

`commons-fileupload-1.3.2.jar`

`commons-io-2.5.jar`

`UploadServlet` 的源代码 如下所示：

```
package com.runoob.test;

import java.io.File;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.List;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

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.disk.DiskFileItemFactory;

import org.apache.commons.fileupload.servlet.ServletFileUpload;
```

```

/**
 * Servlet implementation class UploadServlet
 */
@WebServlet("/UploadServlet")

public class UploadServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    // 上传文件存储目录

    private static final String UPLOAD_DIRECTORY = "upload";

    // 上传配置

    private static final int MEMORY_THRESHOLD   = 1024 * 1024 * 3;  // 3MB
    private static final int MAX_FILE_SIZE      = 1024 * 1024 * 40; // 40MB
    private static final int MAX_REQUEST_SIZE   = 1024 * 1024 * 50; // 50MB

    /**
     * 上传数据及保存文件
     */
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {

        // 检测是否为多媒体上传
        if (!ServletFileUpload.isMultipartContent(request)) {

            // 如果不是则停止

            PrintWriter writer = response.getWriter();

            writer.println("Error: 表单必须包含 enctype=multipart/form-data");

            writer.flush();

            return;

        }

        // 配置上传参数

        DiskFileItemFactory factory = new DiskFileItemFactory();

```

```
// 设置内存临界值 - 超过后将产生临时文件并存储于临时目录中

factory.setSizeThreshold(MEMORY_THRESHOLD);

// 设置临时存储目录

factory.setRepository(new File(System.getProperty("java.io.tmpdir")));


ServletFileUpload upload = new ServletFileUpload(factory);


// 设置最大文件上传值

upload.setFileSizeMax(MAX_FILE_SIZE);


// 设置最大请求值 (包含文件和表单数据)

upload.setSizeMax(MAX_REQUEST_SIZE);


// 中文处理

upload.setHeaderEncoding("UTF-8");


// 构造临时路径来存储上传的文件

// 这个路径相对当前应用的目录

String uploadPath = request.getServletContext().getRealPath("/") + File.separator + UPLOAD_DIRECTORY;


// 如果目录不存在则创建

File uploadDir = new File(uploadPath);

if (!uploadDir.exists()) {

    uploadDir.mkdir();

}


try {

    // 解析请求的内容提取文件数据

    @SuppressWarnings("unchecked")

    List<FileItem> formItems = upload.parseRequest(request);
```

```

        if (formItems != null && formItems.size() > 0) {

            // 迭代表单数据

            for (FileItem item : formItems) {

                // 处理不在表单中的字段

                if (!item.isFormField()) {

                    String fileName = new File(item.getName()).getName();

                    String filePath = uploadPath + File.separator + fileName;

                    File storeFile = new File(filePath);

                    // 在控制台输出文件的上传路径

                    System.out.println(filePath);

                    // 保存文件到硬盘

                    item.write(storeFile);

                    request.setAttribute("message",

                        "文件上传成功!");

                }

            }

        }

    } catch (Exception ex) {

        request.setAttribute("message",

            "错误信息: " + ex.getMessage());

    }

    // 跳转到 message.jsp

    request.getServletContext().getRequestDispatcher("/message.jsp").forward(

        request, response);

}

}

```

message.jsp 文件代码如下:

```

<%@ page language="java" contentType="text/html; charset=UTF-8"

    pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

    "http://www.w3.org/TR/html4/loose.dtd">

```

```
<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>文件上传结果</title>

</head>

<body>

    <center>

        <h2>${message}</h2>

    </center>

</body>

</html>
```

编译和运行 Servlet

编译上面的 Servlet `UploadServlet`，并在 `web.xml` 文件中创建所需的条目，如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xmlns="http://java.sun.com/xml/ns/javaee"

    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"

    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"

    id="WebApp_ID" version="2.5">

    <servlet>

        <display-name>UploadServlet</display-name>

        <servlet-name>UploadServlet</servlet-name>

        <servlet-class>com.runoob.test.UploadServlet</servlet-class>

    </servlet>

    <servlet-mapping>

        <servlet-name>UploadServlet</servlet-name>

        <url-pattern>/TomcatTest/UploadServlet</url-pattern>

    </servlet-mapping>

</web-app>
```

现在尝试使用您在上边创建的 HTML 表单来上传文件。当您在浏览器中访问：<http://localhost:8080/TomcatTest/upload.jsp>，演示如下所示：



☐ Servlet 数据库访问

Servlet 处理日期 ☐



1 篇笔记
#1



☐ 写笔记

对于一般的文件直接用 `a` 标签的话，如下代码所示。由于浏览器可以解析 `jpg` 和 `txt` 文件，故不会直接下载而是在其他网页打开：

```
<a href="/IT/download/1.gif" rel="nofollow">下载图片</a>

<a href="/IT/download/day10.doc" rel="nofollow">下载文档</a>

<a href="/IT/download/day10.txt" rel="nofollow">下载笔记</a>
```

如果想要完成直接下载的目的，可以通过 `Servlet` 进行操作，做了一个简单的 `html` 页面

```
<a href="/IT/download?name=1.gif" rel="nofollow">下载图片1</a>

<a href="/IT/download?name=day10.doc" rel="nofollow">下载文档1</a>

<a href="/IT/download?name=day10.txt" rel="nofollow">下载笔记1</a>
```

我为 `download` 注册了一个 `servlet`，`xml` 描写如下：

```
<servlet>

    <servlet-name>DownloadServlet</servlet-name>

    <servlet-class>com.response.download.DownloadServlet</servlet-class>

</servlet>

<servlet-mapping>
```

```
<servlet-name>DownloadServlet</servlet-name>

<url-pattern>/download</url-pattern>

</servlet-mapping>
```

由于我的请求方式是get方式，所以只需在DownloadServlet这个类中重写doGet方法，代码实现如下：

```
public void doGet(HttpServletRequest request, HttpServletResponse response)

    throws ServletException, IOException {

    //获取文件名

    String filename=request.getParameter("name");

    //防止读取name名乱码

    filename=new String(filename.getBytes("iso-8859-1"),"utf-8");

    //在控制台打印文件名

    System.out.println("文件名: "+filename);

    //设置文件MIME类型

    response.setContentType(getServletContext().getMimeType(filename));

    //设置Content-Disposition

    response.setHeader("Content-Disposition", "attachment;filename="+filename);

    //获取要下载的文件绝对路径，我的文件都放到WebRoot/download目录下

    ServletContext context=this.getServletContext();

    String fullFileName=context.getRealPath("/download/"+filename);

    //输入流为项目文件，输出流指向浏览器

    InputStream is=new FileInputStream(fullFileName);

    ServletOutputStream os =response.getOutputStream();

    /*

    * 设置缓冲区

    * is.read(b)当文件读完时返回-1

    */

    int len=-1;
```



```
byte[] b=new byte[1024];

while((len=is.read(b))!=-1){

    os.write(b,0,len);

}

//关闭流

is.close();

os.close();

}
```

tianqixin2年前 (2017-04-01)

反馈/建议



Servlet 处理日期

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

`Date` 类支持两个构造函数。第一个构造函数初始化当前日期和时间的对象。

```
Date( )
```

下面的构造函数接受一个参数，该参数等于 1970 年 1 月 1 日午夜以来经过的毫秒数。

```
Date(long millisec)
```

一旦您有一个可用的 `Date` 对象，您可以调用下列任意支持的方法来使用日期：

序号	方法 & 描述
1	boolean after(Date date) 如果调用的 <code>Date</code> 对象中包含的日期在 <code>date</code> 指定的日期之后，则返回 <code>true</code> ，否则返回 <code>false</code> 。

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 日午夜以来经过的时间（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很容易的。您可以使用一个简单的 **Date** 对象的 *toString()* 方法来输出当前的日期和时间，如下所示：

```
package com.runoob.test;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.Date;

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 CurrentDate

*/

@WebServlet("/CurrentDate")

public class CurrentDate extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public CurrentDate() {

        super();

    }

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

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

        PrintWriter out = response.getWriter();

        String title = "显示当前的日期和时间";

        Date date = new Date();

        String docType = "<!DOCTYPE html> \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，让我们来编译上面的 **Servlet**，并在 **web.xml** 文件中创建适当的条目：

```

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

<web-app>

    <servlet>

```

```
<servlet-name>CurrentDate</servlet-name>

<servlet-class>com.runoob.test.CurrentDate</servlet-class>

</servlet>

<servlet-mapping>

    <servlet-name>CurrentDate</servlet-name>

    <url-pattern>/TomcatTest/CurrentDate</url-pattern>

</servlet-mapping>

</web-app>
```

然后通过访问 <http://localhost:8080/TomcatTest/CurrentDate> 来调用该 **Servlet**。这将会产生如下的结果：

尝试刷新 URL <http://localhost:8080/TomcatTest/CurrentDate>，每隔几秒刷新一次您都会发现显示时间的差异。

日期比较

正如上面所提到的，您可以在 **Servlet** 中使用所有可用的 **Java** 方法。如果您需要比较两个日期，以下是方法：

您可以使用 `getTime()` 来获取两个对象自 1970 年 1 月 1 日午夜以来经过的时间（以毫秒为单位），然后对这两个值进行比较。

您可以使用方法 `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`。

您可以使用 `compareTo()` 方法，该方法由 **Comparable** 接口定义，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** 允许您选择任何用户定义的日期时间格式化的模式。

让我们修改上面的实例，如下所示：

```
package com.runoob.test;

import java.io.IOException;

import java.io.PrintWriter;

import java.text.SimpleDateFormat;

import java.util.Date;

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 CurrentDate
 */
@WebServlet("/CurrentDate")
public class CurrentDate extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public CurrentDate() {

        super();

    }

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

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

        PrintWriter out = response.getWriter();

        String title = "显示当前的日期和时间";

        Date dNow = new Date( );

        SimpleDateFormat ft =

            new SimpleDateFormat ("yyyy.MM.dd hh:mm:ss E a ");

        String docType = "<!DOCTYPE html> \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>");

    }

}

```

再次编译上面的 **Servlet**，然后通过访问 <http://localhost:8080/TomcatTest/CurrentDate> 来调用该 **Servlet**。这将会产生如下的结果：

简单的日期格式的格式代码

使用事件模式字符串来指定时间格式。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	Era 指示器	AD
y	四位数表示的年	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	所在的周是这个月的第几周	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	Delimiter
"	单引号	`

如需查看可用的处理日期方法的完整列表，您可以参考标准的 **Java** 文档。

☐ Servlet 文件上传

Servlet 网页重定向 ☐

☐ 点我分享笔记

反馈/建议

Servlet 网页重定向

当文档移动到新的位置，我们需要向客户端发送这个新位置时，我们需要用到网页重定向。当然，也可能是为了负载均衡，或者只是为了简单的随机，这些情况都有可能用到网页重定向。

重定向请求到另一个网页的最简单的方式是使用 `response` 对象的 `sendRedirect()` 方法。下面是该方法的定义：

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

throws IOException
```

该方法把响应连同状态码和新的网页位置发送回浏览器。您也可以通过把 `setStatus()` 和 `setHeader()` 方法一起使用来达到同样的效果：

```
....

String site = "http://www.runoob.com" ;

response.setStatus(response.SC_MOVED_TEMPORARILY);

response.setHeader("Location", site);

....
```

实例

本实例显示了 **Servlet** 如何进行页面重定向到另一个位置：

```
package com.runoob.test;

import java.io.IOException;

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 PageRedirect
 */
@WebServlet("/PageRedirect")
public class PageRedirect extends HttpServlet{

    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");

        // 要重定向的新位置

        String site = new String("http://www.runoob.com");

        response.setStatus(response.SC_MOVED_TEMPORARILY);

        response.setHeader("Location", site);

    }

}

```

现在让我们来编译上面的 **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>/TomcatTest/PageRedirect</url-pattern>

</servlet-mapping>

```


....

现在通过访问 URL <http://localhost:8080/PageRedirect> 来调用这个 **Servlet**。这将把您转到给定的 URL <http://www.runoob.com>。

☐ Servlet 处理日期

Servlet 点击计数器 ☐

☐ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](http://www.runoob.com) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

☐ Servlet 网页重定向

Servlet 自动刷新页面 ☐

Servlet 点击计数器

网页点击计数器

很多时候，您可能有兴趣知道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所在的容器控制的。

以下是实现一个简单的基于 **Servlet** 生命周期的网页点击计数器需要采取的步骤：

在 `init()` 方法中初始化一个全局变量。

每次调用 `doGet()` 或 `doPost()` 方法时，都增加全局变量。

如果需要，您可以使用一个数据库表来存储全局变量的值在 `destroy()` 中。在下次初始化 **Servlet** 时，该值可在 `init()` 方法内被读取。这一步是可选的。

如果您只想对一个 **session** 会话计数一次页面点击，那么请使用 `isNew()` 方法来检查该 **session** 会话是否已点击过相同页面。这一步是可选的。

您可以通过显示全局计数器的值，来在网站上展示页面的总点击量。这一步是可选的。

在这里，我们假设 **Web** 容器将无法重新启动。如果是重新启动或 **Servlet** 被销毁，计数器将被重置。

实例

本实例演示了如何实现一个简单的网页点击计数器：

```
package com.runoob.test;

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 PageHitCounter
 */
@WebServlet("/PageHitCounter")

public class PageHitCounter extends HttpServlet {

    private static final long serialVersionUID = 1L;

    private int hitCount;

    public void init()

    {

        // 重置点击计数器

        hitCount = 0;

    }

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

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

        // 增加 hitCount

        hitCount++;

        PrintWriter out = response.getWriter();

        String title = "总点击量";

        String docType = "<!DOCTYPE html> \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()

    {
        // 这一步是可选的，但是如果需要，您可以把 hitCount 的值写入到数据库

    }

}

```

现在让我们来编译上面的 **Servlet**，并在 **web.xml** 文件中创建以下条目：

```

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

<web-app>

    <servlet>

        <servlet-name>PageHitCounter</servlet-name>

        <servlet-class>com.runoob.test.PageHitCounter</servlet-class>

    </servlet>

    <servlet-mapping>

        <servlet-name>PageHitCounter</servlet-name>

        <url-pattern>/TomcatTest/PageHitCounter</url-pattern>

    </servlet-mapping>

</web-app>

```

现在通过访问 <http://localhost:8080/TomcatTest/PageHitCounter> 来调用这个 **Servlet**。这将会在每次页面刷新时，把计数器的值增加 1，结果如下所示：

总点击量

6

网站点击计数器

很多时候，您可能有兴趣知道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做到这一点。

以下是实现一个简单的基于过滤器生命周期的网站点击计数器需要采取的步骤：

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，都增加全局变量。

如果需要，您可以在过滤器的 **destroy()** 中使用一个数据库表来存储全局变量的值。在下次初始化过滤器时，该值可在 **init()** 方法内被读取，这一

步是可选的。

在这里，我们假设 **Web** 容器将无法重新启动。如果是重新启动或 **Servlet** 被销毁，点击计数器将被重置。

实例

本实例演示了如何实现一个简单的网站点击计数器：

```
// 导入必需的 java 库

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{

        // 重置点击计数器

        hitCount = 0;

    }

    public void  doFilter(ServletRequest request,

        ServletResponse response,

        FilterChain chain)

        throws java.io.IOException, ServletException {

        // 把计数器的值增加 1

        hitCount++;

        // 输出计数器

        System.out.println("网站访问统计: "+ hitCount );

        // 把请求传回到过滤器链

        chain.doFilter(request,response);
```

```

    }

    public void destroy()

    {

        // 这一步是可选的，但是如果需要，您可以把 hitCount 的值写入到数据库

    }

}

```

现在让我们来编译上面的 **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>

....

```

现在访问网站的任意页面，比如 <http://localhost:8080/>。这将会在每次任意页面被点击时，把计数器的值增加 1，它会在日志中显示以下消息：

```

网站访问统计: 1

网站访问统计: 2

网站访问统计: 3

网站访问统计: 4

网站访问统计: 5

.....

```

Servlet 自动刷新页面

假设有一个网页，它是显示现场比赛成绩或股票市场状况或货币兑换率。对于所有这些类型的页面，您需要定期刷新网页。

Java Servlet 提供了一个机制，使得网页会在给定的时间间隔自动刷新。

刷新网页的最简单的方式是使用响应对象的方法 **setIntHeader()**。以下是这种方法的定义：

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

此方法把头信息 "Refresh" 连同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

本实例演示了 Servlet 如何使用 **setIntHeader()** 方法来设置 **Refresh** 头信息，从而实现自动刷新页面。

```
package com.runoob.test;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.Calendar;

import java.util.GregorianCalendar;

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 Refresh

*/

@WebServlet("/Refresh")

public class Refresh extends HttpServlet {

    private static final long serialVersionUID = 1L;

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

        // 设置刷新自动加载的事件间隔为 5 秒

        response.setIntHeader("Refresh", 5);

        // 设置响应内容类型

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

        // 获取当前的时间

        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 = "使用 Servlet 自动刷新页面";

        String docType = "<!DOCTYPE html> \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");

    }

}

```

现在让我们来编译上面的 **Servlet**，并在 **web.xml** 文件中创建以下条目：

```

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

<web-app>

    <servlet>

        <servlet-name>Refresh</servlet-name>

        <servlet-class>com.runoob.test.Refresh</servlet-class>

    </servlet>

    <servlet-mapping>

        <servlet-name>Refresh</servlet-name>

        <url-pattern>/TomcatTest/Refresh</url-pattern>

    </servlet-mapping>

</web-app>

```

现在通过访问 <http://localhost:8080/TomcatTest/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。运行该 **Servlet**，并等待查看结果：

使用 **Servlet** 自动刷新页面

当前时间是：9:44:50 PM

☐ Servlet 点击计数器

Servlet 发送电子邮件 ☐

☐ 点我分享笔记

反馈/建议

Servlet 发送电子邮件

使用 Servlet 发送一封电子邮件是很简单的，但首先您必须在您的计算机上安装 **JavaMail API** 和 **Java Activation Framework**（**JAF**）。

您可以从 **Java** 网站下载最新版本的 **JavaMail**，打开网页右侧有个 **Downloads** 链接，点击它下载。

您可以从 **Java** 网站下载最新版本的 **JAF**（版本 **1.1.1**）。

你也可以使用本站提供的下载链接：

JavaMail **mail.jar** **1.4.5**

JAF（版本 **1.1.1**） **activation.jar**

下载并解压缩这些文件，在新创建的顶层目录中，您会发现这两个应用程序的一些 **jar** 文件。您需要把 **mail.jar** 和 **activation.jar** 文件添加到您的 **CLASSPATH** 中。

发送一封简单的电子邮件

下面的实例将从您的计算机上发送一封简单的电子邮件。这里假设您的**本地主机**已连接到互联网，并支持发送电子邮件。同时确保 **Java Email API** 包和 **JAF** 包的所有的 **jar** 文件在 **CLASSPATH** 中都是可用的。

```
// 文件名 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

    {

        // 收件人的电子邮件 ID

        String to = "abcd@gmail.com";
```

```
// 发件人的电子邮件 ID

String from = "web@gmail.com";


// 假设您是从本地主机发送电子邮件

String host = "localhost";


// 获取系统的属性

Properties properties = System.getProperties();


// 设置邮件服务器

properties.setProperty("mail.smtp.host", host);


// 获取默认的 Session 对象

Session session = Session.getDefaultInstance(properties);


// 设置响应内容类型

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

PrintWriter out = response.getWriter();


try{

    // 创建一个默认的 MimeMessage 对象

    MimeMessage message = new MimeMessage(session);

    // 设置 From: header field of the header.

    message.setFrom(new InternetAddress(from));

    // 设置 To: header field of the header.

    message.addRecipient(Message.RecipientType.TO,

                           new InternetAddress(to));

    // 设置 Subject: header field

    message.setSubject("This is the Subject Line!");

    // 现在设置实际消息

    message.setText("This is actual message");
```

```

// 发送消息

Transport.send(message);

String title = "发送电子邮件";

String res = "成功发送消息...";

String docType = "<!DOCTYPE html> \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面的 **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>

....

```

现在通过访问 URL <http://localhost:8080/SendEmail> 来调用这个 **Servlet**。这将会发送一封电子邮件到给定的电子邮件 ID *abcd@gmail.com*，并将显示下面所示的响应：

发送电子邮件

成功发送消息...

如果您想要发送一封电子邮件给多个收件人，那么请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
```

下面是对参数的描述:

type: 这将被设置为 TO、CC 或 BCC。在这里，CC 代表抄送，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，您需要使用 `InternetAddress()` 方法。

发送一封 HTML 电子邮件

下面的实例将从您的计算机上发送一封 HTML 格式的电子邮件。这里假设您的本地主机已连接到互联网，并支持发送电子邮件。同时确保 Java Email API 包和 JAF 包的所有的 jar 文件在 CLASSPATH 中都是可用的。

本实例与上一个实例很类似，但是这里我们使用 `setContent()` 方法来设置第二个参数为 "text/html" 的内容，该参数用来指定 HTML 内容是包含在消息中的。

使用这个实例，您可以发送内容大小不限的 HTML 内容。

```
// 文件名 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

    {

        // 收件人的电子邮件 ID

        String to = "abcd@gmail.com";
```

```
// 发件人的电子邮件 ID

String from = "web@gmail.com";


// 假设您是从本地主机发送电子邮件

String host = "localhost";


// 获取系统的属性

Properties properties = System.getProperties();


// 设置邮件服务器

properties.setProperty("mail.smtp.host", host);


// 获取默认的 Session 对象

Session session = Session.getDefaultInstance(properties);


// 设置响应内容类型

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

PrintWriter out = response.getWriter();


try{

    // 创建一个默认的 MimeMessage 对象

    MimeMessage message = new MimeMessage(session);

    // 设置 From: header field of the header.

    message.setFrom(new InternetAddress(from));

    // 设置 To: header field of the header.

    message.addRecipient(Message.RecipientType.TO,

                           new InternetAddress(to));

    // 设置 Subject: header field

    message.setSubject("This is the Subject Line!");


    // 设置实际的 HTML 消息, 内容大小不限

    message.setContent("<h1>This is actual message</h1>",
```

```

        "text/html" );

    // 发送消息

    Transport.send(message);

    String title = "发送电子邮件";

    String res = "成功发送消息...";

    String docType = "<!DOCTYPE html> \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**，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

下面的实例将从您的计算机上发送一封带有附件的电子邮件。这里假设您的**本地主机**已连接到互联网，并支持发送电子邮件。同时确保 **Java Email API** 包和 **JAF** 包的所有的 **jar** 文件在 **CLASSPATH** 中都是可用的。

```

// 文件名 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

{

    // 收件人的电子邮件 ID

    String to = "abcd@gmail.com";

    // 发件人的电子邮件 ID

    String from = "web@gmail.com";

    // 假设您是从本地主机发送电子邮件

    String host = "localhost";

    // 获取系统的属性

    Properties properties = System.getProperties();

    // 设置邮件服务器

    properties.setProperty("mail.smtp.host", host);

    // 获取默认的 Session 对象

    Session session = Session.getDefaultInstance(properties);

    // 设置响应内容类型

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

    PrintWriter out = response.getWriter();

    try{

        // 创建一个默认的 MimeMessage 对象

        MimeMessage message = new MimeMessage(session);

        // 设置 From: header field of the header.

        message.setFrom(new InternetAddress(from));
```

```
// 设置 To: header field of the header.

message.addRecipient(Message.RecipientType.TO,

                        new InternetAddress(to));


// 设置 Subject: header field

message.setSubject("This is the Subject Line!");


// 创建消息部分

BodyPart messageBodyPart = new MimeBodyPart();


// 填写消息

messageBodyPart.setText("This is message body");


// 创建一个多部分消息

Multipart multipart = new MimeMultipart();


// 设置文本消息部分

multipart.addBodyPart(messageBodyPart);


// 第二部分是附件

messageBodyPart = new MimeBodyPart();

String filename = "file.txt";

DataSource source = new FileDataSource(filename);

messageBodyPart.setDataHandler(new DataHandler(source));

messageBodyPart.setFileName(filename);

multipart.addBodyPart(messageBodyPart);


// 发送完整的消息部分

message.setContent(multipart );


// 发送消息

Transport.send(message);
```



```
String title = "发送电子邮件";

String res = "成功发送电子邮件...";

String docType = "<!DOCTYPE html> \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**，在给定的电子邮件 **ID** 上发送带有文件附件的消息。

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** 和密码进行身份认证，那么您可以设置如下属性：

```
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讲解的保持一致。

☐ [Servlet 自动刷新页面](#)

[Servlet 包](#) ☐

☐ [点我分享笔记](#)

[反馈/建议](#)

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

Servlet 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，并由 Servlet API 规范指定。给定一个顶级目录名 myapp，目录结构如下所示：

```
/myapp

  /images

  /WEB-INF

    /classes

    /lib
```

WEB-INF 子目录中包含应用程序的部署描述符，名为 web.xml。所有的 HTML 文件都位于顶级目录 *myapp* 下。对于 admin 用户，您会发现 ROOT 目录是 myApp 的父目录。

创建包中的 Servlet

WEB-INF/classes 目录包含了所有的 Servlet 类和其他类文件，类文件所在的目录结构与他们的包名称匹配。例如，如果您有一个完全合格的类名称 **com.myorg.MyServlet**，那么这个 Servlet 类必须位于以下目录中：

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

下面的例子创建包名为 *com.myorg* 的 MyServlet 类。

```
// 为包命名

package com.myorg;

// 导入必需的 java 库

import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

@WebServlet("/MyServlet")

public class MyServlet 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;charset=UTF-8");

    // 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

}

}
```

编译包中的 Servlet

编译包中的类与编译其他的类没有什么大的不同。最简单的方法是让您的 `java` 文件保留完全限定路径，如上面提到的类，将被保留在 `com.myorg` 中。您还需要在 `CLASSPATH` 中添加该目录。

假设您的环境已正确设置，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` 依赖于其他库，那么您必须在 `CLASSPATH` 中也要引用那些 `JAR` 文件。这里我只引用了 `servlet-api.jar` `JAR` 文件，因为我在 `Hello World` 程序中并没有使用任何其他库。

该命令行使用内置的 `javac` 编译器，它是 `Sun Microsystems Java` 软件开发工具包（`JDK`，全称 `Java Software Development Kit`）附带的。 `Microsystems` 的 `Java` 软件开发工具包（`JDK`）。为了让该命令正常工作，必须包括您在 `PATH` 环境变量中所使用的 `Java SDK` 的位置。

如果一切顺利，上述编译会在同一目录下生成 **`MyServlet.class`** 文件。下一节将解释如何把一个已编译的 `Servlet` 部署到生产中。

Servlet 打包部署

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，您需要在位于 <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/MyServlet**。如果一切顺利，您会看到下面的结果：



Servlet 调试

测试/调试 Servlet 始终是开发使用过程中的难点。Servlet 往往涉及大量的客户端/服务器交互，可能会出现错误但又难以重现。这里有一些提示和建议，可以帮助您调试。

System.out.println()

`System.out.println()` 是作为一个标记来使用的，用来测试一段特定的代码是否被执行。我们也可以打印出变量的值。此外：

由于 **System** 对象是核心 **Java** 对象的一部分，它可以在不需要安装任何额外类的情况下被用于任何地方。这包括 **Servlet**、**JSP**、**RMI**、**EJB's**、普通的 **Beans** 和类，以及独立的应用程序。

与在断点处停止不同，写入到 **System.out** 不会干扰到应用程序的正常执行流程，这使得它在时序是至关重要时候显得尤为有价值。

下面是使用 `System.out.println()` 的语法：

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

通过上面的语法生成的所有消息将被记录在 **Web** 服务器日志文件中。

消息日志

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，这是非常好的想法，推荐使用 [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：

```
// 导入必需的 java 库

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");

        // 调用两个 ServletContext.log 方法

        ServletContext context = getServletContext( );

        if (par == null || par.equals(""))

            // 通过 Throwable 参数记录版本

            context.log("No message received:",

                new IllegalStateException("Missing parameter"));

        else

            context.log("Here is the visitor's message: " + par);

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

```
        java.io.PrintWriter out = response.getWriter( );

        String title = "Context Log";

        String docType = "<!DOCTYPE html> \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。对于 **Tomcat**，这些日志可以在 **<Tomcat-installation-directory>/logs** 目录中找到。这些日志文件确实对新出现的错误或问题的频率给出指示。正因为如此，建议在通常不会发生的异常的 **catch** 子句中使用 **log()** 函数。

使用 JDB 调试器

您可以使用调试 **applet** 或应用程序的 **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** 来响应浏览器端的 **HTTP** 请求。这与调试 **applet** 小程序非常相似。与调试 **applet** 不同的是，实际被调试的程序是 **sun.applet.AppletViewer**。

大多数调试器会自动隐藏如何调试 **applet** 的细节。同样的，对于 **servlet**，您必须帮调试器执行以下操作：

设置您的调试器的类路径 **classpath**，以便它可以找到 **sun.servlet.http.Http-Server** 和相关的类。

设置您的调试器的类路径 **classpath**，以便它可以找到您的 **servlet** 和支持的类，通常是在 **server_root/servlets** 和 **server_root/classes** 中。

您通常不会希望 **server_root/servlets** 在您的 **classpath** 中，因为它会禁用 **servlet** 的重新加载。但是这种包含规则对于调试是非常有用的。它允许您的调试器在 **HttpServer** 中的自定义 **Servlet** 加载器加载 **Servlet** 之前在 **Servlet** 中设置断点。

如果您已经设置了正确的类路径 **classpath**，就可以开始调试 **sun.servlet.http.HttpServer**。可以在您想要调试的 **Servlet** 代码中设置断点，然后通过 **Web** 浏览器使用给定的 **Servlet** (**http://localhost:8080/servlet/ServletToDebug**) 向 **HttpServer** 发出请求。您会看到程序执行到断点处会停止。

使用注释

代码中的注释有助于以各种方式进行调试。注释可用于调试过程的很多其他方式中。

该 **Servlet** 使用 **Java** 注释和单行注释 (**//...**)，多行注释 (**/* ...*/**) 可用于暂时移除部分 **Java** 代码。如果 **bug** 消失，仔细看看您刚才注释的代码并找出问题所在。

客户端和服务端头信息

有时，当一个 **Servlet** 并没有像预期那样时，查看原始的 **HTTP** 请求和响应是非常有用的。如果您熟悉 **HTTP** 结构，您可以阅读请求和响应，看看这些头信息究竟是什么。

重要的调试技巧

下面列出了一些 **Servlet** 调试的技巧：

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新加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，请使用 **Shift-Reload**，在 **Internet Explorer** 中，请使用 **Shift-Refresh**。

请确认 `servlet` 的 `init()` 方法接受一个 `ServletConfig` 参数，并调用 `super.init(config)`。

[Servlet 包](#)

[Servlet 国际化](#)

[点我分享笔记](#)

[反馈/建议](#)

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[Servlet 调试](#)

[Servlet 有用的资源](#)

Servlet 国际化

在我们开始之前，先来看看三个重要术语：

国际化（i18n）：这意味着一个网站提供了不同版本的翻译成访问者的语言或国籍的内容。

本地化（l10n）：这意味着向网站添加资源，以使其适应特定的地理或文化区域，例如网站翻译成印地文（Hindi）。

区域设置（locale）：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下划线和一个国家符号。例如 "en_US" 表示针对 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。本教程不会讲解这些注意事项的完整细节，但它会通过一个很好的实例向您演示如何通过差异化定位（即区域设置）来让网页以不同语言呈现。



Servlet 可以根据请求者的区域设置拾取相应版本的网站，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` 对象中返回 `Locale` 对象的方法。

```
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，您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

本实例演示了如何显示某个请求的语言和相关的国家：

```
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

    {

        // 获取客户端的区域设置

        Locale locale = request.getLocale();

        String language = locale.getLanguage();

        String country = locale.getCountry();

        // 设置响应内容类型

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

        PrintWriter out = response.getWriter();

        String title = "检测区域设置";

        String docType = "<!DOCTYPE html> \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（如英语、西班牙语、德语、法语、意大利语、荷兰语等）编写的页面。在这里，为了能正确显示所有的字符，设置 **Content-Language** 头是非常重要的。

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

    {

        // 设置响应内容类型

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

        PrintWriter out = response.getWriter();

        // 设置西班牙语语言代码

        response.setHeader("Content-Language", "es");

        String title = "En Espa&ntilde;ol";

        String docType = "<!DOCTYPE html> \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

您可以使用 `java.text.DateFormat` 类及其静态方法 `getDateTimeInstance()` 来格式化特定于区域设置的日期和时间。下面的实例演示了如何格式化特定于某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  
  
    {  
  
        // 设置响应内容类型  
  
        response.setContentType("text/html;charset=UTF-8");  
  
        PrintWriter out = response.getWriter();  
  
        // 获取客户端的区域设置  
  
        Locale locale = request.getLocale( );  
  
        String date = DateFormat.getDateTimeInstance(  
  
                                DateFormat.FULL,  
  
                                DateFormat.SHORT,  
  
                                locale).format(new Date( ));  
  
        String title = "特定于区域设置的日期";  
  
        String docType = "<!DOCTYPE html> \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

您可以使用 `java.text.NumberFormat` 类及其静态方法 `getCurrencyInstance()` 来格式化数字（比如 `long` 类型或 `double` 类型）为特定于区域设置的货币。下面的实例演示了如何格式化特定于某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

    {

        // 设置响应内容类型

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

        PrintWriter out = response.getWriter();

        // 获取客户端的区域设置

        Locale locale = request.getLocale( );

        NumberFormat nft = NumberFormat.getCurrencyInstance(locale);

        String formattedCurr = nft.format(1000000);
    }
}

```

```

String title = "特定于区域设置的货币";

String docType = "<!DOCTYPE html> \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

您可以使用 `java.text.NumberFormat` 类及其静态方法 `getPercentInstance()` 来格式化特定于区域设置的百分比。下面的实例演示了如何格式化特定于某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

    {

        // 设置响应内容类型

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

        PrintWriter out = response.getWriter();

        // 获取客户端的区域设置

        Locale locale = request.getLocale( );

        NumberFormat nft = NumberFormat.getPercentInstance(locale);

```

```
String formattedPerc = nft.format(0.51);

String title = "特定于区域设置的百分比";

String docType = "<!DOCTYPE html> \n";

out.println(docType +

"<html>\n" +

"<head><title>" + title + "</title></head>\n" +

"<body bgcolor=\"#f0f0f0\">\n" +

"<h1 align=\"center\">" + formattedPerc + "</h1>\n" +

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

}

}
```

[☐ Servlet 调试](#)

[Servlet 有用的资源](#) ☐

[☐ 点我分享笔记](#)

[反馈/建议](#)

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[☐ Servlet 国际化](#)

Servlet 有用的资源

本章列出了 *Servlet* 网站、书籍和文章。

Java Servlet 有用的网站

[Sun's Site on Servlets](#) - Sun 的官方网站上关于 **Servlet** 的相关资料。

[JSP Engine - Tomcat](#) - [Apache Tomcat](#) 是一个开源软件，实现了对 **Java Servlet** 和 **JSP**（**JavaServer Pages**）技术的支持。

[MySQL Connector/J](#) - [MySQL Connector/J](#) 是 [MySQL](#) 官方 **JDBC** 驱动程序。

[The Java™ Tutorials](#) - 该 **Java** 教程是为那些想用 **Java** 编程语言创建应用程序的编程人员提供的实用指南。

[Java™ 2 SDK, Standard Edition](#) - [Java™ 2 SDK, Standard Edition](#) 的官网。

[Free Java Download](#) - 为您的桌面计算机下载 **Java**！

[Sun Developer Network](#) - [Sun Microsystems](#) 的官方网站，上面列出了所有的 **API** 文档，最新的 **Java** 技术、书籍和其他资源。

Java Servlet 有用的书籍



☐ Servlet 国际化

☐ 点我分享笔记

反馈/建议