Java Servlet完全教程

本文由 [ImportNew](http://www.importnew.com/) - [进林](http://www.importnew.com/author/8zjl8) 翻译自 [howtodoinjava](http://howtodoinjava.com/2014/12/08/complete-java-servlets-tutorial/" \t "_blank)。欢迎加入[翻译小组](http://group.jobbole.com/category/feedback/trans-team/" \t "_blank)。转载请见文末要求。

Servlet 是一些遵从Java Servlet API的Java类，这些Java类可以响应请求。尽管Servlet可以响应任意类型的请求，但是它们使用最广泛的是响应web方面的请求（me:这是java ee的文档的原话）。 Servlet必须部署在Java servlet容器（me：实际运行时是由servlet的容器来管理server和client之间的servlet）才能使用。虽然很多开发者都使用[Java Server Pages（JSP）](http://en.wikipedia.org/wiki/JavaServer_Pages" \t "_blank)和[Java Server Faces（JSF）](http://en.wikipedia.org/wiki/JavaServer_Faces" \t "_blank)等Servlet框架，但是这些技术都要在幕后通过Servlet容器把页面编译为Java Servlet。也就是说，了解**Java Servlet技术的基础知识**对任何Java web开发者来说是很有用的。

在这个教程里，我们将会通过下面的专题来全面了解Java Servlet技术。

**目录**

* 编写你的第一个Servlet
* Servlet生命周期方法
* 使用@WebServlet注解开发Servlet
* 打包和部署Servlet到Tomcat服务器
* 编写动态的Servlet响应内容
* 处理Servlet请求和响应
* 监听Servlet容器事件
* 传递Servlet初始化参数
* 为特定的URL请求添加Servlet过滤器
* 使用Servlet下载二进制文件
* 使用RequestDispatcher.forward()转发请求到另一个Servlet
* 使用HttpServletResponse.sendRedirect()重定向请求到另一个Servlet
* 使用Servlets读写Cookie

让我们一起来一步步地学习Servlet。

**编写你的第一个Servlet**

我们的第一个Servlet是一个只拥有少量代码的简单Servlet，目的是让你只需关注它的行为。

|  |
| --- |
| package com.howtodoinjava.servlets;    import java.io.IOException;  import java.io.PrintWriter;    import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  @WebServlet  public class MyFirstServlet extends HttpServlet {        private static final long serialVersionUID = -1915463532411657451L;        @Override      protected void doGet(HttpServletRequest request,              HttpServletResponse response) throws ServletException, IOException      {          response.setContentType("text/html;charset=UTF-8");          PrintWriter out = response.getWriter();          try {              // Write some content              out.println("<html>");              out.println("<head>");              out.println("<title>MyFirstServlet</title>");              out.println("</head>");              out.println("<body>");              out.println("<h2>Servlet MyFirstServlet at " + request.getContextPath() + "</h2>");              out.println("</body>");              out.println("</html>");          } finally {              out.close(); #像这种涉及流的对象，在操作时最好的办法就是使用try-catch-finally接口以确保在最后关闭流，数据库的连接也是一样          }      }        @Override      protected void doPost(HttpServletRequest request,              HttpServletResponse response) throws ServletException, IOException {          //Do some other work      }        @Override      public String getServletInfo() {          return "MyFirstServlet";      }  } |

**为了在web容器里注册上面的Servlet，你要为你的应用建一个web.xml入口文件（me：在这个文件中定义servlet的和访问接口的映射关系）。**

|  |
| --- |
| <?xml version="1.0"?>  <web-app version="2.5"      xmlns="http://java.sun.com/xml/ns/javaee"      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee      http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd">      <servlet>          <servlet-name>MyFirstServlet</servlet-name>          <servlet-class>com.howtodoinjava.servlets.MyFirstServlet</servlet-class>      </servlet>      <servlet-mapping>          <servlet-name>MyFirstServlet</servlet-name>          <url-pattern>/MyFirstServlet</url-pattern>      </servlet-mapping>    <welcome-file-list>          <welcome-file>/MyFirstServlet</welcome-file>      </welcome-file-list>  </web-app> |

上面的Servlet做了一些重要的事情，你可能想了解的。

（me：每个servlet的定义都需要在web.xml中注册并设置servlet-mapping来关联接口映射）

1. **MyFirstServlet类继承了HttpServlet**。这个继承是必须的，因为所有的Servlet必须是要么继承了 javax.servlet.GenericServlet 的普通Servlet（me：如果不是将servlet作为http服务的话，需要继承自此接口，而且HttpServlet也是继承自此接口），要么是继承了 javax.servlet.http.HttpServlet 的HTTP Servlet（me：主要对应的是将servlet作为http访问接口使用）。
2. **重新 doGet() 和 doPost() 方法。**这两个方法都已在 HttpServlet 类里定义了。当一个GET或POST请求到来时，它就会被映射到相应的方法里。例如，如果你向这个servlet发送一个HTTP GET请求，doGet()方法就会被调用。
3. **这里也有一些其他有用的方法。**你可以重写它们来在运行时控制应用。例如getServletInfo()。
4. **HttpServletRequest 和 HttpServletResponse（me:这两个对象是由servlet容器在接收client请求时创建的） 是所有doXXX()方法的默认参数。**我们会在后面的章节里详细学习这些对象。

以上所有关于简单Servlet的内容就是你需要知道的内容。

**Servlet生命周期方法**

在你的应用加载并使用一个Servlet时，从初始化到销毁这个Servlet期间会发生一系列的事件。这些事件叫做S**ervlet的生命周期事件（或方法）**。让我们一起来进一步了解它们。

Servlet生命周期（me：存在于servlet容器的整个生命周期，所以才能够使用单例模式）的三个核心方法分别是 init() –初始化时调用, service() 和 destroy()-销毁时调用。每个Servlet都会实现这些方法，并且在特定的运行时间调用它们。

1) 在Servlet生命周期的初始化阶段，**web容器通过调用init()方法来初始化Servlet实例**，并且可以传递一个实现 javax.servlet.ServletConfig 接口的对象给它。这个配置对象（configuration object）使Servlet能够读取在web应用的web.xml文件里定义的名值（name-value）初始参数（me：也可以在servlet的类上通过注解的方式来提供此servlet初始化的参数）。**这个方法在Servlet实例的生命周期里只调用一次**。

init方法定义与这类似：

|  |
| --- |
| public void  init() throws ServletException {      //custom initialization code  } |

2) 初始化后，Servlet实例就可以处理客户端请求了。**web容器调用Servlet的service()方法来处理每一个请求**。service() 方法定义了能够处理的请求类型并且调用适当方法来处理这些请求。编写Servlet的开发者必须为这些方法提供实现。如果发出一个Servlet没实现的请求，那么父类的方法就会被调用并且通常会给请求方（requester）返回一个错误信息。

通常，我们不需要重写（override）这个方法（me：我们由这个service方法也可以看出http请求的本质：即实际的server是由一个服务接口来接收所有的http请求的，然后他在此基础上按照http协议，将http的请求分类处理如get请求、post请求和put请求等。然后在类的内部分别定义的对应的方法来处理各自的请求，具体可参看nodejs的实现，能够有更加直观的理解。这也是之所以自定义的servlet类要继承HttpServlet类的原因）。

|  |
| --- |
| protected void service(HttpServletRequest req, HttpServletResponse resp)      throws ServletException, IOException  {  String method = req.getMethod();    if (method.equals(METHOD\_GET)) {      long lastModified = getLastModified(req);      if (lastModified == -1) {      // servlet doesn't support if-modified-since, no reason      // to go through further expensive logic      doGet(req, resp);      } else {      long ifModifiedSince = req.getDateHeader(HEADER\_IFMODSINCE);      if (ifModifiedSince < (lastModified / 1000 \* 1000)) {          // If the servlet mod time is later, call doGet()                  // Round down to the nearest second for a proper compare                  // A ifModifiedSince of -1 will always be less          maybeSetLastModified(resp, lastModified);          doGet(req, resp);      } else {          resp.setStatus(HttpServletResponse.SC\_NOT\_MODIFIED);      }      }    } else if (method.equals(METHOD\_HEAD)) {      long lastModified = getLastModified(req);      maybeSetLastModified(resp, lastModified);      doHead(req, resp);    } else if (method.equals(METHOD\_POST)) {      doPost(req, resp);    } else if (method.equals(METHOD\_PUT)) {      doPut(req, resp);    } else if (method.equals(METHOD\_DELETE)) {      doDelete(req, resp);    } else if (method.equals(METHOD\_OPTIONS)) {      doOptions(req,resp);    } else if (method.equals(METHOD\_TRACE)) {      doTrace(req,resp);    } else {      //      // Note that this means NO servlet supports whatever      // method was requested, anywhere on this server.      //        String errMsg = lStrings.getString("http.method\_not\_implemented");      Object[] errArgs = new Object[1];      errArgs[0] = method;      errMsg = MessageFormat.format(errMsg, errArgs);        resp.sendError(HttpServletResponse.SC\_NOT\_IMPLEMENTED, errMsg);  }  } |

3) 最后，**web容器调用destroy()方法来终结Servlet**。如果你想在Servlet的生命周期内关闭或者销毁一些文件系统或者网络资源，你可以调用这个方法来实现。destroy() 方法和init()方法一样，在Servlet的生命周期里只能调用一次。

|  |
| --- |
| public void destroy() {  //  } |

在大多数情况下，你通常不需要在你的Servlet里重写这些方法。

*扩展阅读：*[*web服务器是如何运作的？*](http://howtodoinjava.com/2013/10/03/a-birds-eye-view-on-how-web-servers-work/)

**使用@WebServlet注解来开发Servlet**

如果你不喜欢使用xml配置而喜欢注解的话，没关系，Servlets API同样提供了一些注解接口给你。你可以像下面的例子一样使用 [@WebServlet](https://docs.oracle.com/javaee/6/api/javax/servlet/annotation/WebServlet.html) 注解并且不需要在web.xml里为Servlet注册任何信息。**容器会自动注册你的Servlet到运行环境，并且像往常一样处理它（me：原来如此，注解可以代替web.xml配置文件，servlet容器可以根据注解自动为我们注册servlet和它们的映射关系。其实际的工作原理也是根据注解自动为包生成web.xml文件。注意：tomcat容器只认识web.xml，所以其他的无论怎么变化都需要按照tomcat的要求来）。**

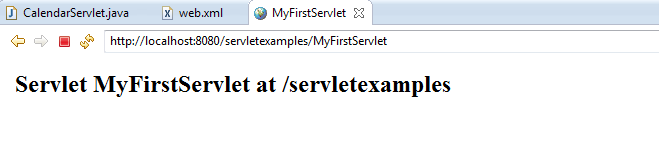
|  |
| --- |
| package com.howtodoinjava.servlets;    import java.io.IOException;  import java.io.PrintWriter;    import javax.servlet.ServletException;  import javax.servlet.annotation.WebServlet;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;    @WebServlet(name = "MyFirstServlet", urlPatterns = {"/MyFirstServlet"})  public class MyFirstServlet extends HttpServlet {        private static final long serialVersionUID = -1915463532411657451L;        @Override      protected void doGet(HttpServletRequest request,              HttpServletResponse response) throws ServletException, IOException      {          //Do some work      }        @Override      protected void doPost(HttpServletRequest request,              HttpServletResponse response) throws ServletException, IOException {          //Do some other work      }  } |

**打包和部署Servlet到Tomcat服务器**

如果你在使用IDE（例如eclipse），那么打包和部署你的应用只需要一个简单的步骤。右击项目> Run As > Run As Server。如果还没配置服务器先配置好服务器，然后就可以准备开干了。

如果你没在使用IDE，那么你需要做一些额外的工作。比如，使用命令提示符编译应用，使用ANT去生成war文件等等。但我相信，现在的开发者都在使用IDE来开发。所以我就不在这方面浪费时间了。

当你把我们的第一个Servlet部署到tomcat上并在浏览器输入“*http://localhost:8080/servletexamples/MyFirstServlet*”，你会得到下面的响应。

[](http://www.importnew.com/?attachment_id=14730)

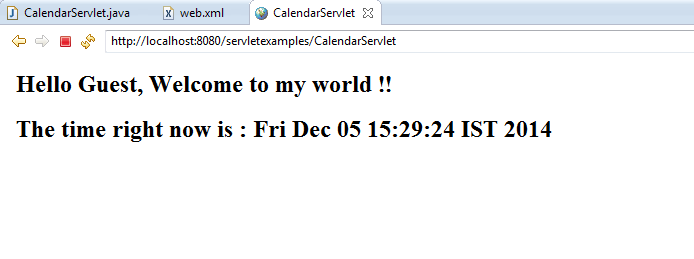
**编写动态的Servlet响应内容**

Java Servlets如此有用的原因之一是Servlet能动态显示网页内容。这些内容可以从服务器本身、另外一个网站、或者许多其他网络可以访问的资源里获取(me：加载其他资源可以使用include或者dispathcer方法或者对象）。Servlet不是静态网页，它们是动态的。可以说这是它们最大的优势（me:原生支持动态的，可以使用静态变量测试一下，由于这个servlet只会在容器启动的时候初始化一次，**因此其实可以使用成员变量来测试，因为它在整个生命周期都是单例的，用户的请求相当于是多线程访问servlet**）。

让我们来举个Servlet例子，这个Servlet会显示当前日期和时间给用户并且会显示用户名和一些自定义的信息。让我们来为这个功能编写代码吧。

|  |
| --- |
| package com.howtodoinjava.servlets;    import java.io.IOException;  import java.io.PrintWriter;  import java.util.Date;  import java.util.HashMap;  import java.util.Map;    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(name = "CalendarServlet", urlPatterns = {"/CalendarServlet"})  public class CalendarServlet extends HttpServlet {        private static final long serialVersionUID = -1915463532411657451L;        @Override      protected void doGet(HttpServletRequest request,              HttpServletResponse response) throws ServletException, IOException      {            Map<String,String> data = getData();            response.setContentType("text/html;charset=UTF-8");          PrintWriter out = response.getWriter();          try {              // Write some content              out.println("<html>");              out.println("<head>");              out.println("<title>CalendarServlet</title>");              out.println("</head>");              out.println("<body>");              out.println("<h2>Hello " + data.get("username") + ", " + data.get("message") + "</h2>");              out.println("<h2>The time right now is : " + new Date() + "</h2>");              out.println("</body>");              out.println("</html>");          } finally {              out.close();          }      }        //This method will access some external system as database to get user name, and his personalized message      private Map<String, String> getData()      {          Map<String, String> data = new HashMap<String, String>();          data.put("username", "Guest");          data.put("message",  "Welcome to my world !!");          return data;      }  } |

当你在tomcat里运行上面的Servlet并在浏览器里输入“*http://localhost:8080/servletexamples/CalendarServlet*”，你会得到下面的响应。

[](http://www.importnew.com/?attachment_id=14731)

**处理Servlet请求和响应**

Servlet可以轻松创建一个基于请求和响应生命周期的web应用。它们能够提供HTTP响应并且可以使用同一段代码来处理业务逻辑。处理业务逻辑的能力使Servlet比标准的HTML代码更强大。

现实世界里的应用，一个HTML网页表单包含了要发送给Servlet的参数。Servlet会以某种方式来处理这些参数并且 返回一个客户端能够识别的响应。在对象是HttpServlet的情况下，客户端是web浏览器，响应是web页面。<form>的 action属性指定了使用哪个Servlet来处理表单里的参数值。

为了获取请求参数，需要调用 HttpServletRequest 对象的 getParameter() 方法，并且传递你要获取的输入参数的id给该方法。

|  |
| --- |
| String value1 = req.getParameter("param1");  String value1 = req.getParameter("param2"); |

一旦获取了参数值，它们就会在需要时被处理。对客户端的响应和我们上面部分讨论的一样。我们使用 HttpServletResponse 对象给客户端发送响应。

request和response处理的基本使用可以是这样的：

|  |
| --- |
| @Override  protected void doGet(HttpServletRequest request,          HttpServletResponse response) throws ServletException, IOException  {        response.setContentType("text/html;charset=UTF-8");      PrintWriter out = response.getWriter();        String username = request.getParameter("username");      String password = request.getParameter("password");        boolean success = validateUser(username, password);        try {          // Write some content          out.println("<html>");          out.println("<head>");          out.println("<title>LoginServlet</title>");          out.println("</head>");          out.println("<body>");            if(success) {              out.println("<h2>Welcome Friend</h2>");          }else{              out.println("<h2>Validate your self again.</h2>");          }            out.println("</body>");          out.println("</html>");      } finally {          out.close();      }  } |

为了发送内容给客户端，你需要使用从 HttpServletResponse 里获取的 PrintWriter 对象。任何写到这个对象的内容都会被写进outputstream里，并会把内容发送回给客户端。

**监听Servlet容器事件**

有时候，知道应用服务器容器（the application server container）里某些事件发生的时间是很有用的。这个概念适用于很多情况，但它通常用在开启应用时初始化应用或者关闭应用时清理应用。可以在应用里注册一个监听器（listener）来显示应用什么时候开启或者关闭。因此，通过监听这些事件，Servlet可以在一些事件发生时执行相应的动作。

为了创建一个基于容器事件执行动作的监听器，你必须创建一个实现 ServletContextListener 接口的类。这个类必须实现的方法有 contextInitialized() 和 contextDestroyed()。这两个方法都需要 ServletContextEvent 作为参数，并且在每次初始化或者关闭Servlet容器时都会被自动调用。

为了在容器注册监听器，你可以使用下面其中一个方法：

1) 利用 @WebListener 注解。  
2) 在web.xml应用部署文件里注册监听器。  
3) 使用 ServletContext 里定义的 addListener() 方法

请注意，ServletContextListener 不是Servlet API里唯一的监听器。这里还有一些其他的监听器，比如

|  |
| --- |
| javax.servlet.ServletRequestListener  javax.servlet.ServletRequestAttrbiteListener  javax.servlet.ServletContextListener  javax.servlet.ServletContextAttributeListener  javax.servlet.HttpSessionListener  javax.servlet.HttpSessionAttributeListener |

根据你要监听的事件选择他们来实现你的监听器类。比如，每当创建或销毁一个用户session时，HttpSessionListener 就会发出通知。

**传递Servlet初始化参数(me：[可以使用web.xml配置initParam或者使用在注解@WebServlet](mailto:可以使用web.xml配置initParam或者使用在注解@WebServlet)中添加参数initParam来实现)**

现在的大多数应用都需要设置一些在应用/控制器（controller）启动时可以传递的配置参数（configuration parameters）。Servlet同样可以接受初始化参数，并在处理第一个请求前来使用它们来构建配置参数。

显然，你也可以在Servlet里硬编码配置值。但是这样做的话，在Servlet发生改动时你需要再次重新编译整个应用。没有人喜欢这样做。

|  |
| --- |
| <web-app>      <servlet>          <servlet-name>SimpleServlet</servlet-name>          <servlet-class>com.howtodoinjava.servlets.SimpleServlet</servlet-class>            <!-- Servlet init param -->          <init-param>              <param-name>name</param-name>              <param-value>value</param-value>          </init-param>        </servlet>    </web-app> |

设置后，你就可以在代码里调用 getServletConfig.getInitializationParameter() 并传递参数名给该方法来使用参数。就像下面展示的代码一样：

|  |  |
| --- | --- |
| 1 | String value = getServletConfig().getInitParameter("name"); |

**为特定的URL请求添加Servlet过滤器（Filter）**

Web过滤器(me:Filter主要用来对request进行预处理并且对response进行进一步的处理)在给定的URL被访问时对请求进行预处理并调用相应的功能是很有用的。相 比于直接调用给定URL请求的Servlet，包含相同URL模式的过滤器（filter）会在Servlet调用前被调用。这在很多情况下是很有用的。 或许最大的用处就是执行日志，验证或者其他不需要与用户交互的后台服务。

过滤器必须要实现 javax.servlet.Filter 接口。这个接口包含了init()，descriptor()和doFilter()这些方法。init()和destroy()方法会被容器调用。 doFilter()方法用来在过滤器类里实现逻辑任务。如果你想把过滤器组成过滤链（chain filter）或者存在多匹配给定URL模式的个过滤器，它们就会根据web.xml里的配置顺序被调用。

在filter里面一定要调用chain.doFilter方法，否则request不会被进一步处理。

为了在web.xml里配置过滤器，需要使用<filter>和<filter-mapping> XML元素以及相关的子元素标签。

|  |
| --- |
| <filter>      <filter-name>LoggingFilter</filter-name>      <filter-class>LoggingFilter</filter-class>  </filter>  <filter-mapping>      <filter-name>LogingFilter</filter-name>      <url-pattern>/\*</url-pattern>  </filter-mapping> |

*如果你要使用注解来为特定的servlet配置过滤器，你可以使用@WebFilter注解。*

**使用Servlet下载二进制文件**

几乎所有的web应用都必须有下载文件的功能。为了下载一个文件，Servlet必须提供一个和下载文件类型匹配的响应类型（me：只需要在content-type指定为流类型就可以了）。同样，必须在响应头里指出该响应包含附件。就像下面的代码。

|  |
| --- |
| String mimeType = context.getMimeType( fileToDownload );  response.setContentType( mimeType != null ? mimeType : "text/plain" );  response.setHeader( "Content-Disposition", "attachment; filename=\"" + fileToDownload + "\"" ); |

通过调用 ServletContext.getResourceAsStream() 方法并传递文件路径给该方法，你可以获取要下载的文件（文件保存在文件系统）的引用。这个方法会返回一个输入流（InputStream）对 象，我们可以用这个对象来读取文件内容。当读取文件时，我们创建一个字节缓存区（byte buffer）从文件里获取数据块。最后的工作就是读取文件内容并且把它们复制到输出流。我们使用while循环来完成文件的读取，这个循环直到读取了文 件的所有内容才会跳出循环。我们使用循环来读进数据块并把它写进输出流。把所有数据写进输出流后，ServletOutputStream 对象的flush方法就会被调用并且清空内容和释放资源。

看这段简单的代码：

|  |
| --- |
| private void downloadFile(HttpServletRequest request, HttpServletResponse response, String fileToDownload) throws IOException      {          final int BYTES = 1024;          int length = 0;            ServletOutputStream outStream = response.getOutputStream();          ServletContext context = getServletConfig().getServletContext();            String mimeType = context.getMimeType( fileToDownload );          response.setContentType( mimeType != null ? mimeType : "text/plain" );          response.setHeader( "Content-Disposition", "attachment; filename="" + fileToDownload + """ );            InputStream in = context.getResourceAsStream("/" + fileToDownload);            byte[] bbuf = new byte[BYTES];            while ((in != null) && ((length = in.read(bbuf)) != -1)) {              outStream.write(bbuf, 0, length);          }            outStream.flush();          outStream.close();      } |

**使用RequestDispatcher.forward()转发请求到另一个Servlet**

有时候，你的应用需要把一个Servlet要处理的请求转让给另外的Servlet来处理并完成任务。而且，转让请求时不能重定向客户端的URL。即浏览器地址栏上的URL不会改变(me:类似server端做了一次负载均衡，或者相当于server端的这个接口充当了http的普通代理)。

这里说明了forward做转发client不知道，而使用sendRedirect时client端知道，因为它收到了302的状态码，并且浏览器在此默认请求的话变为get请求

在 ServletContext 里已经内置了实现上面需求的方法。所以，当你获取了 ServletContext 的引用，你就可以简单地调用getRequestDispatcher() 方法去获取用来转发请求的 RequestDispatcher 对象。当调用 getRequestDispatcher() 方法时，需要传递包含servlet名的字符串，这个Servlet就是你用来处理转让请求的Servlet。获取 RequestDispatcher 对象后，通过传递 HttpServletRequest 和HttpServletResponse 对象给它来调用转发方法。转发方法负责对请求进行转发。

|  |
| --- |
| RequestDispatcher rd = servletContext.getRequestDispatcher("/NextServlet");  rd.forward(request, response); |

**使用HttpServletResponse.sendRedirect()重定向请求到另一个Servlet**

尽管有时候，你不想在Servlet发送重定向时通知用户，就像我们在上面那段看到的一样。但是在某些情况下，我们确实想要通知用户。当应用内的特定URL被访问时，你想把浏览器的URL重定向到另外一个。

要实现这种功能，你需要调用 HttpServletResponse 对象的sendRedirect()方法。

|  |
| --- |
| httpServletResponse.sendRedirect("/anotherURL"); |

*这个简单的重定向，与servlet链（servlet chaining）相反，不需要传递目标地址的HttpRequest对象。*

**使用Servlet读写Cookie**

很多应用都想在客户端机器里保存用户当前的浏览历史。目的是当用户再次使用应用时，他能够从上次离开的地方开始浏览。为了实现这个需求，通常使用cookies。你可以把它看作是保存在客户端机器里的键值对基本数据。当使用浏览器打开应用时，应用可以对这些数据进行读写。

为了创建cookie，需要实例化一个新的 javax.servlet.http.Cookie 对象并且为它分配名称和值。实例化cookie后，可以设置属性来配置cookie。在这个例子里，我们使用 setMaxAge() 和 setHttpOnly() 方法来设置cookie的生命周期和防范客户端脚本。

*从Servlet3.0 API开始，已经可以把cookie标记为HTTP only了。这使cookie可以防范客户端脚本的攻击，使cookie更加安全。*

|  |
| --- |
| Cookie cookie = new Cookie("sessionId","123456789");  cookie.setHttpOnly(true);  cookie.setMaxAge(-30);  response.addCookie(cookie); |

这里的response是传递给doXXX()方法的 HttpServletResponse 实例。

要读取服务端的cookie信息，使用下面代码：

|  |
| --- |
| Cookie[] cookies = request.getCookies();  for(Cookie cookie : cookies)  {      //cookie.getName();      //cookie.getValue()  } |

这就是这篇教程里关于Servlet技术的全部内容了。欢迎评论和回馈。