链滴

# 学习笔记 --Servlet 技术

作者：KioLuo

原文链接：https://ld246.com/article/1492567951161

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

**//Servlet概述**

一个servlet就是一个java类，并提供基于请求-响应模式的web服务。

**//创建Servlet**

新建继承HttpServlet的类，覆盖init(), destroy(), service(HttpServletRequest req, HttpServletResponse res), doGet(HttpServletRequest req, HttpServletResponse res)方法

**//配置Servlet**

在web.xml中servlet项加入项，如：

data1

value1

data2

value2

TestServlet

com.netease.server.example.web.controller.TestServlet

代码中相应方法：

```
ServletConfig config = this.getServletConfig();
String v1 = config.getInitParameter("data1");
```

全局配置：

在servlet项外加入项，如：

globalData1

123

代码中相应方法：

```
ServletContext context = this.getServletContext();
String globalV1 = context.getInitParameter("globalData1");
```

ServletMapping匹配规则：

支持模糊匹配

精确路径匹配（完全匹配）---> 最长路径匹配（最长前缀匹配）---> 扩展名匹配 ---> default servlt或放弃

更改servlet默认启动时间：

在元素中增加元素，负数为第一次请求时启动该servlet，大于等于0则在servlet容器启动时启动servle，数字越小，优先级越高，如：

0

配置错误页面：

404

/404.html

配置欢迎页面：

元素，可包含多个元素，顺序加载

MIME类型映射

定义扩展文件名映射类型，包含两个子元素：和

**//Servelt共享动态属性**

一个servlet代码中：

```
ServletContext context = this.getServletContext();
context.setAttribute("attribute1", "123");
```

另一个servlet代码中：

```
String globalV1 = (String)context.getAttribute("attribute1");
```

**//Servlet读取外部资源配置文件信息**

三种方法：

1. getResource

代码如：

```
ServletContext context = this.getServletContext();
URL url = context.getResource("/WEB-INF/log4j.properties");
InputStream in = url.openStream();
String propertyValue = GeneralUtil.getPropery("log4j.rootCategory", in);
```

2. getResourceAsStream

代码如：

```
InputStream in2 = context.getResourceAsStream("/WEB-INF/log4j.properties");
String p2 = GeneralUtil.getPropery("log4j.rootCategory", in2);
```

3. getRealPath

代码如：

```
String path = context.getRealPath("/WEB-INF/log4j.properties");
File file = new File(path);
InputStream in3 = new FileInputStream(file);
String p3 = GeneralUtil.getPropery("log4j.rootCategory", in3);
```

**//HttpServletRequest实现类方法：**

1. 获得客户机信息

getRequestURL方法返回客户端发出请求时的完整URL。

getRequestURI方法返回请求行中的资源名部分。

getQueryString 方法返回请求行中的参数部分。

getRemoteAddr方法返回发出请求的客户机的IP地址

getRemoteHost方法返回发出请求的客户机的完整主机名

getRemotePort方法返回客户机所使用的网络端口号

getLocalAddr方法返回WEB服务器的IP地址。

getLocalName方法返回WEB服务器的主机名

getMethod得到客户机请求方式

getServerPath()获取请求的文件的路径

2. 获得客户机请求头

getHeader(string name)方法

getHeaders(String name)方法

getHeaderNames方法

3. 获得客户机请求参数(客户端提交的数据)

getParameter(name)方法 获取请求中的参数，该参数是由name指定的

getParameterValues（String name）方法 获取指定名称参数的所有值数组。它适用于一个参数名应多个值的情况。如页面表单中的复选框，多选列表提交的值。

getParameterNames方法 返回一个包含请求消息中的所有参数名的Enumeration对象。通过遍历这Enumeration对象，就可以获取请求消息中所有的参数名。

getCharacterEncoding() 返回请求的字符编码方式

getAttributeNames()返回当前请求的所有属性的名字集合赋值:setAttribute()

getAttribute(String name) 返回name指定的属性值

getsession()返回和客户端相关的session，如果没有给客户端分配session，则返回null

getParameterMap():返回一个保存了请求消息中的所有参数名和值的Map对象。Map对象的key是符串类型的参数名，value是这个参数所对应的Object类型的值数组

addHeader(String name,String value) 将指定的名字和值加入到响应的头信息中

**//HttpServletResponse实现类方法：**

encodeURL(String url) 编码指定的URL

sendError(int sc) 使用指定状态码发送一个错误到客户端

setDateHeader(String name,long date) 将给出的名字和日期设置响应的头部

setHeader(String name,String value) 将给出的名字和值设置响应的头部

setStatus(int sc) 给当前响应设置状态码

setContentType(String ContentType) 设置响应的MIME类型 ,页面的设置文本类型,获取或设置输流的 HTTP MIME 类型。输出流的 HTTP MIME 类型。默认值为“text/html”

getOutputStream() 字节输出流对象

getWriter() 字符的输出流对象

**//Cookie和Session**

关于Cookie

cookie数量限制：20个，每个大小限制：4kb

```
Cookie userNameCookie = new Cookie("userName", userName);  //创建cookie
userNameCookie.setMaxAge(10 * 60);        //设置cookie有效时间
response.addCookie(userNameCookie);       //在响应中加入cookie
Cookie[] cookies = request.getCookies();       //从请求中获取cookie
cookie.getValue();          //从cookie中获取值
cookie.getName();          //从cookie中获取名字
```

关于Session

默认有效期30分钟，setMaxInactiveInterval设置有效期，invalidate使session失效

```
HttpSession session = request.getSession();
String name = (String) session.getAttribute("userName");
session.setMaxInactiveInterval(2 * 60);
session.setAttribute("userName", userName);
session.invalidate();
```

在部署描述符web.xml中配置有效期：

5

**//请求转发与重定向**

请求转发：

获取RequestDispatcher：

方法1：

RequestDispatcher rd = req.getRequestDispatcher("/forwardExample");　　//可以是绝对路径者相对路径

方法2：

RequestDispatcher rd = this.getServletContext().getNameDispatcher("ServletForwardExample"; //转发到的servlet名称

或者

RequestDispatcher rd = this.getServletContext().getRequestDispatcher("/forwardExample"); //只能是绝对路径

转发：

rd.forward(req, resp);

请求重定向：

resp.sendRedirect("redirectExample");　　//相对路径则访问本web，绝对路径可以访问其他web

**//过滤器与监听器**

启动顺序：ServletContext--->listener--->filter--->servlet

过滤器：

创建实现Filter接口的类作为过滤器，在web.xml中增加项和，可定义多个，可配置FilterConfig，如：

filterParam

111

TestFilter

com.netease.server.example.web.controller.filter.TestFilter

TestFilter

/hello/world/*

代码中：

```
String value = filterConfig.getInitParameter("filterParam");
```

在doFilter方法中可对request进行处理，如：

```
@Override
public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain) throws IOException, ServletException {
    System.out.println("filter doFilter method");
    HttpServletRequest req = (HttpServletRequest) request;
    HttpSession session = req.getSession();
    if (session.getAttribute("username") == null) {
        HttpServletResponse res = (HttpServletResponse) response;
        res.sendRedirect("../index.html");
    } else {
        chain.doFilter(request, response);
    }
}
```

监听器：

分类

1. 监听应用程序环境 ServletContext

ServletContextListener

ServletContextAttributeListener

2. 监听用户请求对象 ServletRequest

ServletRequestListener

ServletRequestAttributeListener

3. 监听用户会话对象 HttpSession

HttpSessionListener

HttpSessionAttributeListener

HttpSessionActivationListener

HttpSessionBindingListener

创建实现相关接口的类，如实现HttpSessionAttributeListener，ServletContextListener, ServletReuestListener接口的类

在web.xml中增加项：

com.netease.server.example.web.controller.listener.TestListener

**//Servlet并发处理**

特征：单实例，多线程，线程不安全

变量的线程安全：

--参数变量本地化

--使用同步块synchronized

属性的线程安全：

--SerletContext线程不安全

--HttpSession理论上线程安全

--ServletRequest线程安全

避免在Servlet中创建线程

多个Servlet访问外部对象加锁

在代码中加入

synchronized (this) {

...

}

**//JSP**

动态网页技术标准

简化的servlet

JSP = Html + Java + JSP tag

JSP处理流程：

客户端--->发送JSP请求--->JSP文件--->转换为Servlet文件--->编译为class文件--->执行并载入servlet实例--->返回响应到客户端

JSP基本语法：

1. 静态内容

模板数据

2. 指令

page指令

--定义页面的依赖属性，比如脚本语言、error页面、缓存需求等

include指令

--包含其他文件

taglib指令

--引入标签库的定义

3. 注释

<%-- 注释 --%>

4. 表达式

<%= 表达式 %>

如下：

Today's date: <%= (new java.util.Date()).toLocaleString() %>

5. 声明

<%! declaration; [declaration; ] + ... %>

如： <%! int a, b, c; %>

6. 脚本

<% 代码片段 %>

如下：

<% out.println("Your IP address is : " + request.getRemoteAddr()); %>

JSP内置对象

request　　HttpServletRequest类的实例

| | |
|---|---|
| response | HttpServletResponse类的实例 |
| out | PrintWriter类的实例，用于把结果输出至网页上 |
| session | HttpSession类的实例 |
| applicaiton | ServletContext类的实例，与应用上下文有关 |
| config | ServletConfig类的实例 |
| page | 类似于Java中的this关键字 |
| pageContext | PageContext类的实例，提供对JSP页面的所有对象以及所有命名空间的访问 |
| Exception | Exception类的对象，代表发生错误的JSP页面对应的异常对象 |