1 自定义 Servlet的三种实现方式

1）实现接口Servlet

2）继承 GenericServlet

3）继承 HttpServlet（只用）

2、使用Servlet（配置）

Servlet2.0 web.xml

<servlet>

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

<servlet-class>包.类</servlet-class>

</servlet>

<servlet-mapping>

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

<url-parttern>/url</ url-parttern>

</servlet-mapping>

Servlet3.0 在servlet类的定义上@WebServlet()

@WebServlet( name=”servletname”, value={“/url”,”/url”})

Class MyServlet{}

简化

@WebServlet(“/url”)

Class MyServlet{}

3、Servlet工作流程（Servlet生命周期）

客户端通过浏览器发送一个请求url，服务器接收请求后，从项目中读取web.xml文件中的配置，是否有url的配置，如果存在url的配置，则根据url找到对应的servlet-name，通过servlet-name找到servlet-class，根据请求类型get/post决定调用Servlet中的doGet()或doPost()，执行完毕后，将结果响应给客户端

默认第一次调用Servlet时，服务器会自动创建Servlet对象，并且执行init()初始化方法，继续执行service()\doGet()\doPost(), 如果再次通过url访问Servlet时，只调用service()\doGet()\doPost(), 当服务器停止时Servlet对象才会调用destroy()销毁方法。

因此Servlet是单例模式，如果想要改变Servlet的创建时机，在服务器加载时就被创建，则需要配置<load-on-startup>num</load-on-startup>或者@WebServlt(loadOnStartup=num),其中num表示Servlet对象的创建顺序。

4、Servlet中常用对象

1） 请求对象HttpServletRequest request

|  |  |
| --- | --- |
| String getParameter（“参数名”） | 获取一个请求参数值 |
| String[] getParameterValues(“参数名”) | 获取一组请求参数值 |
| getRequestDispatcher(“/url”).forward(request,response) | 请求转发给另一个url |
| setCharacterEncoding(“utf-8”) | 设置请求的编码方式，解决post提交时中文乱码 |
| String getServletContext().getRealPath("/url") | 返回/url对应的物理路径 |
| String getMethod() | 返回请求方式 |

2）响应对象HttpServletResponse response

|  |  |
| --- | --- |
| PrintWrite getWrite() | 返回打印流，用于输出 |
| setContentType(“text/html;charset=utf-8”) | 设置响应编码，解决响应中文乱码问题 |
| sendRedirect(“/project/url”) | 重定向 |
|  |  |
|  |  |

3）上下文对象 ServletContext

全局对象，获取ServletContext对象方法，以下常用方法

request.getServletContext()

this.getServletContext()

|  |  |
| --- | --- |
| String getRealPath() | 返回/url对应的物理路径 |
| String contextPath() | 返回服务器中项目的根路径 /project |
| setAttribute(“name”, objvalue) | 保存数据 |
| Object getAttribute(“name”) | 返回数据，如果需要真实的数据类型需要手动类型转换 |

4、url中“\”

客户端“\” 指定服务器下 \项目名\资源

服务器”\” 指定项目下 \资源

5、请求转发和重定向区别

|  |  |  |
| --- | --- | --- |
|  | 请求转发 | 重定向 |
| 对象 | request.getRequstDispatcher(“/url”).forward(request,response) | response.sendRedircte(“/project/url”) |
| \ | 服务器 \资源 | 客户端 \项目名\资源 |
| 可访问的资源 | 仅限于项目中 | 无限制 |
| 请求次数 | 1次 | 2次 |
| 响应次数 | 1次 | 2次 |
| 地址栏变化 | 不变 | 变化 |

6、WEB-INF 特殊性

对外不可访问， 也就是说不能通过地址栏输入的方式访问

可以进行服务器内部访问，通过请求转发访问WEB-INF内的资源

7、Cookie使用

创建Cookie Cookie ck = new Cookie(“name”, “value”);

设置 ck.setMaxAge(秒数);

Ck.setPath(“url”) / ck在项目中的所有资源中都可用

/项目名/url 指定可用路径

如果没有setPath() 当前路径下可用

如果/项目/aaa/bbb/servlet 有效路径/项目/aaa/bbb

保存数据 response.addCookie(ck)

H获取Cookie Cookie[] cks = request.getCookies();

c.getName() cookie名 c.getValue() cookie值

8、Session使用

创建 HttpSession session = request.getSession();

保存数据 session.setAttribute(“name”, objectvalue);

设置时间 session. setMaxInactiveInterval(秒数);

取值 Object obj = session.getAttribute(“name”);

失效 session.invalidate();

Session 全局时间配置，时间单位：分钟

<session-config>

<session-timeout>5</session-timeout>

</session-config>

9、Cookie与Session区别

|  |  |  |
| --- | --- | --- |
|  | Cookie | Session |
| 保存位置 | 客户端 | 服务器 |
| 数据类型 | 字符串，不能保存中文 | 可以保存任何类型 |
| 时间设置（秒） | 最大有效时间 | 发呆时间，可以设置全局session，web.xml  <session-config>  <session-timeout>5</session-timeout>  </session-config> |
| 数据量 | 有限 | 无限 |

10、JSP

1）jsp组成

Html css javascripte

<%%> java代码块 <%@%>指令 <%=%>表达式 <%----%> 注释 <%!%>声明

2）jsp内置对象(9个)

Request response session config page pageContext application out exception

3）jsp动作（6个）

<jsp:useBean> <jsp:setProperty><jsp:getProperty>

<jsp:include><jsp:forward><jsp:param>

4）jsp指令（3个）

<%@page> <%@include%> <%@taglib%>

<%@include%> 静态包含，在翻译成java源码的阶段就把多个文件合并，在编译，即1个源文件和1个class

<jsp:include> 动态包含，各自进行翻译并编译，在执行时才进行合并，即多个源文件和多个class文件

5）jsp作用域（4个）

PageContext pageContext 当前jsp有效

HttpServletRequest request 当前请求有效

HttpSession session 当前会话有效

ServletContext application 当前项目有效

6）el使用

${ xxxxScope.name }

7）jstl使用(jstl.jar)

<%@taglib prefix=”c” uri=””%> <c:xxxx> 判断、循环、声明变量、 连接生成。。。。

<c:if test=”${}”></c:if>

<c:choose>

<c:when test=”${}”></c:when>

<c:otherwise></c:otherwise>

</c:choose>

<c:forEach items=”${集合、数组}” var=”v” varStatus=”vs”>

${v}

</c:forEach>

<%@taglib prefix=”fn” uri=””%> ${fn:xxxx()}

11、过滤器Filter

1)Class xxxxFilter implementer Filter{

Init(){}

Destroy(){}

doFilter(FilterChain chain){

//过滤操作

//放行

chain.doFilter();

}

}

2)过滤器链执行顺序

A）2.0 wel.xml

<filter></filter> <filter-mapping></filter-mapping>

根据配置的顺序进行过滤

B) 3.0 @WebFilter

根据类名排序，进行过滤

3）过滤路径

/\* 项目下所有资源都要过滤

/xxxx/\* xxxx下的所有资源需要过滤

\*.jsp 项目下的所有jsp需要过滤

/xxx.jsp 只有项目根路径下xxx.jsp请求需要过滤

4）过滤请求的类型

REQUEST\INCLUDE\FORWAOR\ERROR

12、监听器Listener

1)ServletContext监听（2个）

2）HttpSession 监听 (5个)

3）HttpServletRequst监听（3个）

每一个种监听有自己要完成的方法，根据需要选择合适的监听器去实现方法

13 、文件下载

14、文件上传