**Servlet实战详解**

拓薪教育

刘洋

1. Servlet概述(了解)
   1. JavaWeb的三大组件

Servlet是JavaWeb三大组件之一，它是我们学习JavaWeb最为基本的组件，也就是说你一定要**100%**的掌握它。

其它两种：Filter（拦截器）、Listener（观察者模式），后续讲解

* 1. Servlet的作用

Servlet，即Server Let的意思，用来处理用户请求。当客户端发出请求后，由Tomcat去找到可以处理这一请求的Servlet来处理。

也就是说，用户的请求是由Servlet来处理的！例如用户发出登录请求，那么就应该由处理登录的Servlet来处理；用户发出登录请求，那么就应该有登录Servlet来处理。



1. Servlet初识(熟练)
   1. 第一个Servlet
      1. Servlet说明

servlet 是运行在 Web 服务器中的小型 Java 程序。servlet 通常通过 HTTP（超文本传输协议）接收和响应来自 Web 客户端的请求。也就是说，Servlet是由我们自己来完成的！但Servlet一定要实现javax.servlet.Servlet接口，并且还要在web.xml文件中部署！不然Tomcat是找不到我们写的Servlet的。

* + 1. Servlet接口

javax.servlet.Servlet接口中方法如下：

* **void init(ServletConfig servletConfig)**：当Tomcat创建Servlet实例后，马上调用init()方法。这个方法只在创建后调用一次！用来做Servlet初始化工作！一个Servlet实例只被创建一次，所以init()方法也只被调用一次！（本方法编写对Servlet的初始化代码）
* **void service(ServletRequest request, ServletResponse response)**：Servlet实例在每次处理请求时都调用service()方法。
* **void destroy()：**当Tomcat要销毁Servlet实例时，会先调用destroy()方法，再销毁它。所谓销毁Servlet，其实就是在Servlet缓存池中把Servlet移除！一般只有Tomcat关闭时，才会销毁Servlet实例！
* **ServletConfig getServletConfig()：**这个方法返回ServletConfig对象，但我们不能自己去创建ServletConfig对象，所以一般我们会在init()方法中把init()方法的参数保存起来，然后再在本方法中返回它。ServletConfig对象对应web.xml中当前Servlet实例的配置信息。
* String getServletInfo()：这个方法只是返回一个字符串，用来说明当前Servlet。**基本没用！**
  + 1. 创建Servlet

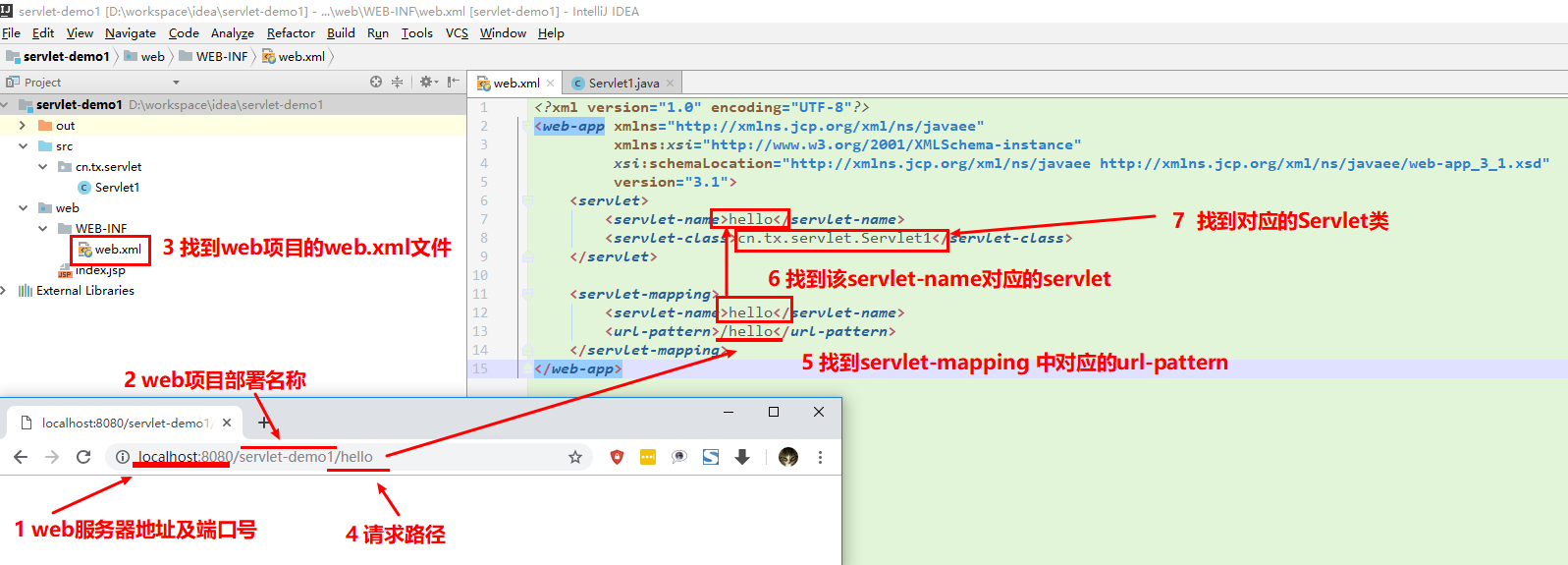
第一步：常见HelloServlet实现Servlet的接口，实现接口中的方法;

|  |
| --- |
| ***package cn.tx.servlet*;  *import javax.servlet.*\*; *import java.io.IOException*;  *public class Servlet1 implements*** Servlet **{  @Override  *public void* init(**ServletConfig servletConfig**) *throws ServletException* {    }   @Override  *public*** ServletConfig **getServletConfig() {  *return null*;  }   @Override  *public void* service(**ServletRequest servletRequest**,** ServletResponse servletResponse**) *throws ServletException*, *IOException* {   }   @Override  *public String* getServletInfo() {  *return null*;  }   @Override  *public void* destroy() {   } }** |

第二步：配置servlet的访问路径;

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* **<*web-app* xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xmlns:*xsi*="http://www.w3.org/2001/XMLSchema-instance"  *xsi*:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"  version="3.1">  <*servlet*>  <*servlet-name*>**hello**</*servlet-name*>  <*servlet-class*>**cn.tx.servlet.Servlet1**</*servlet-class*>  </*servlet*>    <*servlet-mapping*>  <*servlet-name*>**hello**</*servlet-name*>  <*url-pattern*>**/hello**</*url-pattern*>  </*servlet-mapping*> </*web-app*>** |

* + 1. JavaWeb请求响应流程





当Tomcat接收到请求（http://localhost:8080/servlet\_pro/logon）后，Tomcat会找到servlet\_pro项目中的web.xml文件，然后通过logon这个请求路径，查找处理这个请求的Servlet类型。这刚好与<url-pattern>/logon</url-pattern>匹配，这说明存在一个可以通过这个请求的Servlet。然后再通过<url-pattern>/logon </url-pattern>查找到<servlet-name>login</servlet-name>，然后再通过<servlet-name>login </servle-name>查找到<servlet-class>com.rl.servlet.LoginServlet</servlet-class>。这时Tomcat已经得到了一个Servlet类名字（一个字符串而已）。

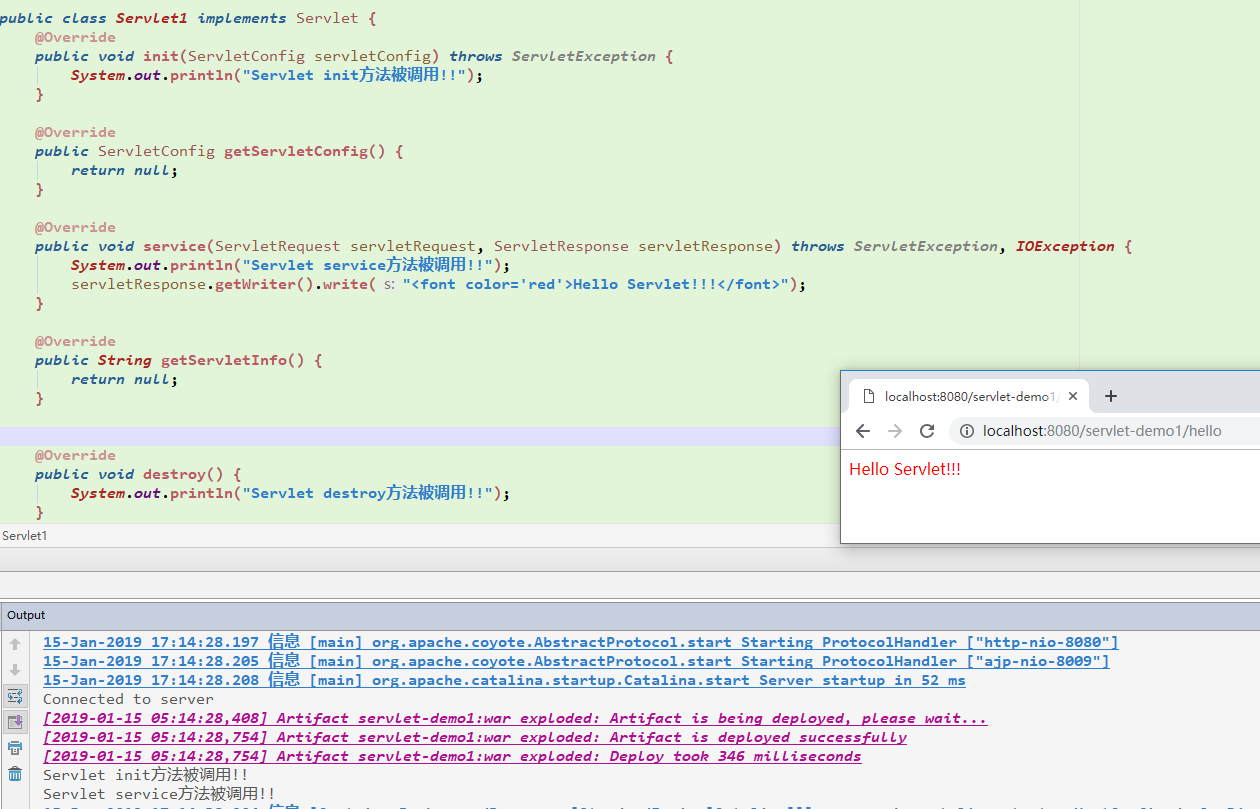
Tomcat通过Servlet类名字去查找内存中是否存在Servlet对象，如果存在，那么就不用再去创建，直接获取这个Servlet实例，调用它的service()方法完成请求！

如果这个Servlet不存在，那么Tomcat会通过反射来创建Servlet实例，并把Servlet实例存放到Servlet池中，再去调用Servlet的service方法处理请求。

* 1. Servlet生命周期

javax.servlet.Servlet接口中，有三个方法说明了Servlet的生命周期：

* void init(ServletConfig)：创建后马上调用init()完成初始化；
* void service(ServletRequest,ServletResponse)：每次处理请求时调用service()方法；
* void destroy()：当Tomcat要销毁Servlet实例时，先调用destroy()方法。



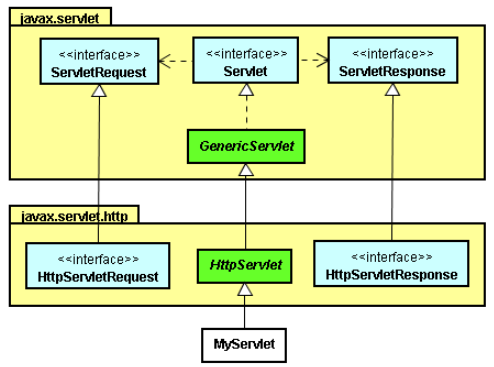
Servlet对象的实例默认情况下是在浏览器第一次调用servlet时候被创建的（可以修改其创建时机后续讲解）

现在你应该已经清楚了，Servlet的实例不由我们创建，Servlet的方法不由我们来调用，这一切都是由Tomcat来完成！！！这就是说由Tomcat来管理Servlet，而我们只需要去编写Servlet实现类，并将其部署到web.xml文件中去！

**再次提醒，只有这三个方法是生命周期中的方法**。也就是说，生命周期方法会被Tomcat在不同的时间点来调用！而其它方法就不会被调用了！！！如果你在自己写的Servlet中添加了其他方法，那么Tomcat也是不会去调用它们的！但你可以让生命周期方法去调用你自己写的方法就OK了！

1. HttpServlet(精通)
   1. HttpServlet介绍

因为现在我们的请求都是基于HTTP协议的，所以我们应该专门为HTTP请求写一个Servlet做为通用父类。



由上图我们可以看出,以后再写Servlet 可以直接继承HttpServlet

* Servlet 一个标准
* GenericServlet 是Servlet接口子类
* HttpServlet 是GenericServlet子类，一个**专门处理Http请求**的Servlet
  1. Http请求方法

HTTP请求方法不只是GET和POST，还有其他的方法，但基本上用不上。这里只是简单介绍一下。你自己心里有个数，HTTP请求除了GET和POST之外还有别的就行了。

* **GET** 通过请求URI得到资源
* **POST** 用于添加新的内容
* PUT 用于修改某个内容
* DELETE 删除某个内容
* CONNECT 用于代理进行传输，如使用SSL
* OPTIONS 询问可以执行哪些方法
* PATCH 部分文档更改
* RACE 用于远程诊断服务器
* HEAD 类似于GET, 但是不返回body信息，用于检查对象是否存在，以及得到对象的元数据
* TRACE 用于远程诊断服务器
  1. 创建HttpServlet
     1. 第一种方法

创建一个类继承HttpServlet

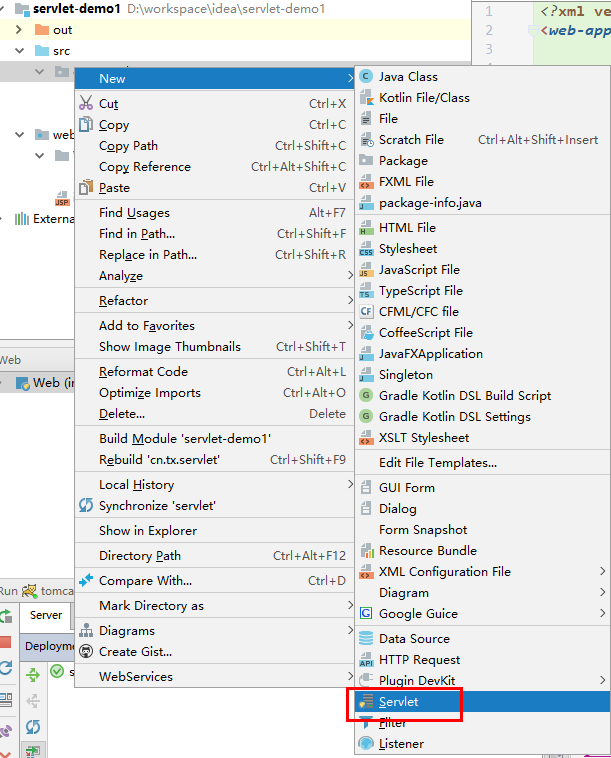
|  |
| --- |
| ***package cn.tx.servlet*;  *import javax.servlet.http.HttpServlet*;  *public class Servlet2 extends HttpServlet*{   }** |

配置Servlet的映射路径

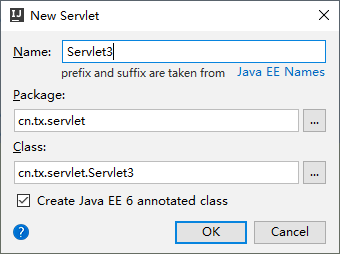
|  |
| --- |
| **<*servlet*>  <*servlet-name*>**hello2**</*servlet-name*>  <*servlet-class*>**cn.tx.servlet.Servlet2**</*servlet-class*> </*servlet*>  <*servlet-mapping*>  <*servlet-name*>**hello2**</*servlet-name*>  <*url-pattern*>**/hello2**</*url-pattern*> </*servlet-mapping*>** |

* + 1. 第二种方法

IDEA直接创建servlet



填写Servlet名称



创建完成,自己填写映射路径

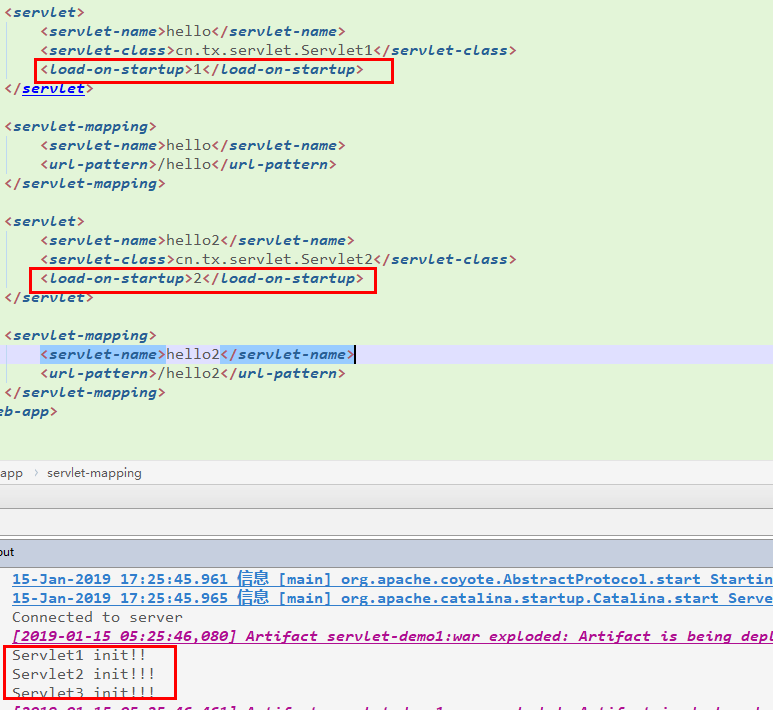


* + 1. Servlet创建顺序

有些Servlet需要在Tomcat启动时就被创建，而不是第一次访问时被创建，那么可以在web.xml文件中配置<servlet>元素。

在<servlet>元素中添加子元素<load-on-startup>元素！

|  |
| --- |
| **<*servlet*>  <*servlet-name*>**hello**</*servlet-name*>  <*servlet-class*>**cn.tx.servlet.Servlet1**</*servlet-class*>  <*load-on-startup*>**1**</*load-on-startup*> </*servlet*>** |



当值为0或者大于0时，表示容器在应用启动时就加载这个servlet；

所有添加了<load-on-startup>子元素的Servle，都会在Tomcat启动时被创建！当然，只是被创建，但没有处理请求！但我们知道在Servlet生命周期中init()方法会在创建后被调用，所以你可以在init()方法中做一些输出，查看是否在Tomcat启动时调用了它。

<load-on-startup>元素的值是一个序号，Tomcat会使用这个序号给多个Servlet排序！然后在Tomcat启动时会按这个顺序来创建Servlet实例对象！

1. ServletConfig(熟练)

ServletConfig对象对应web.xml文件中的<servlet>元素。例如你想获取当前Servlet在web.xml文件中的配置名，那么可以使用servletConfig.getServletName()方法获取！

你不能自己去创建ServletConfig对象，Servlet的init()方法的参数就是ServletConfig类型的。Tomcat在调用init()方法时，会传递ServletConfig对象。你可以在init()方法中使用它！

|  |
| --- |
| **<*servlet*>  <*servlet-name*>**hello**</*servlet-name*>  <*servlet-class*>**cn.tx.servlet.Servlet1**</*servlet-class*>  <*init-param*>  <*param-name*>**username**</*param-name*>  <*param-value*>**root**</*param-value*>  </*init-param*>  <*init-param*>  <*param-name*>**password**</*param-name*>  <*param-value*>**txjava**</*param-value*>  </*init-param*>  <*load-on-startup*>**1**</*load-on-startup*> </*servlet*>** |

|  |
| --- |
| ***String username* =** servletConfig**.getInitParameter("username"); *System*.*out*.println("username:"+*username*);** Enumeration**<*String*> *parameterNames* =** servletConfig**.getInitParameterNames(); *while* (*parameterNames*.hasMoreElements()){  *String element* = *parameterNames*.nextElement();  *System*.*out*.println(*element*+":"+**servletConfig**.getInitParameter(*element*)); }** |

添加了两个初始化参数，第一个参数的名称为paramName1，第一个参数的值为paramValue1；第二个参数的名称为paramName2，第二个参数值为paramValue2。

在<servlet>元素中可以加载多个<init-param>，每个<init-param>表示一个参数。<init-param>下有两个子元素：<param-name>和<param-value>，其中<param-name>表示参数的名称，而<param-value>元素参数的值。

注意，<init-param>是添加到<servlet>元素中，而不是<servlet-mapping>中。

　　使用ServletConfig对象的getInitParameter(String paramName)方法可以获取指定参数名称的参数值。getInitParameterNames()方法返回所有参数的名字，返回值类型为Enumeration。

1. Servlet路径映射(熟练)

关于Url-Pattern的配置：

* 完全路径匹配 以/开头 例如 /aaa /aaa/bbb
* 目录匹配 以/开头 例如 /aaa/\* /\*
* 扩展名匹配 不能以/开头 例如 \*.do \*.action ....

**优先级： 完全路径匹配 > 目录匹配 > 扩展名匹配**

**经典错误： /\*.do**

举例：

对于如下的一些映射关系：

* Servlet11 映射到 /abc/\*
* Servlet22 映射到 /\*
* Servlet33 映射到 /abc
* Servlet44 映射到 \*.do

问题：

* URL为“/abc/a.html”，“/abc/\*”和“/\*”都匹配，哪个servlet响应

Servlet引擎将调用Servlet1。同样是目录匹配所限定范围更精确的先匹配

* URL为“/abc”时，“/abc/\*”和“/abc”都匹配，哪个servlet响应

Servlet引擎将调用Servlet3。

* URL为“/abc/a.do”时，“/abc/\*”和“\*.do”都匹配，哪个servlet响应

Servlet引擎将调用Servlet1。

* URL为“/a.do”时，“/\*”和“\*.do”都匹配，哪个servlet响应

Servlet引擎将调用Servlet2.

* URL为“/xxx/yyy/a.do”时，“/\*”和“\*.do”都匹配，哪个servlet响应

Servlet引擎将调用Servlet2。

服务器端路径 /demo4 （转发、包含...）

1. ServletContext(熟练)
   1. ServletContext介绍

ServletContext是一个全局的储存信息的空间，服务器开始，其就存在，服务器关闭，其才释放。request，一个用户可有多个；session，一个用户一个；而servletContext，所有用户共用一个。所以，为了节省空间，提高效率，ServletContext中，要放必须的、重要的、**所有用户需要共享的线程又是安全**的一些信息。



* WEB容器在启动时，它会为每个WEB应用程序都创建一个对应的ServletContext对象，它代表当前web应用。
* 由于一个WEB应用中的所有Servlet共享同一个ServletContext对象，因此Servlet对象之间可以通过ServletContext对象来实现通讯。ServletContext对象通常也被称之为context域对象。
* 查看ServletContext API文档，了解ServletContext对象的功能。
  1. ServletContext API

|  |  |
| --- | --- |
| **Object** | **getAttribute**(**String** name)            Returns the servlet container attribute with the given name, or null if there is no attribute by that name. |
| **String** | **getContextPath()            Returns the context path of the web application.** |
| **String** | **getInitParameter(String name)            Returns a String containing the value of the named context-wide initialization parameter, or null if the parameter does not exist.** |
| **String** | **getRealPath(String path)            Returns a String containing the real path for a given virtual path** |
| **void** | **setAttribute(String name, Object object)            Binds an object to a given attribute name in this servlet context.** |
| **InputStream** | **getResourceAsStream(String path)            Returns the resource located at the named path as an InputStream object.** |

* 1. ServletContext使用
     1. 常规应用:
* 获取WEB应用的全局初始化参数
* 通过ServletContext对象实现数据共享

|  |
| --- |
| **<*context-param*>  <*param-name*>**company**</*param-name*>  <*param-value*>**拓薪教育**</*param-value*> </*context-param*>** |

|  |
| --- |
| ServletContext ***servletContext* =** config**.getServletContext(); *String company* = *servletContext*.getInitParameter("company"); *System*.*out*.println("company:"+*company*);** |

示例:

统计站点访问次数

|  |
| --- |
| **@Override *protected void* doGet(**HttpServletRequest request**,** HttpServletResponse response**) *throws ServletException*, *IOException* {  doPost(**request**,** response**); }  @Override *protected void* doPost(**HttpServletRequest request**,** HttpServletResponse response**) *throws ServletException*, *IOException* {** ServletContext ***servletContext* = getServletContext();  *Integer visitNums* = (*Integer*)(*servletContext*.getAttribute("visitNums"));  *visitNums* = *visitNums* == *null* ? *0* : *visitNums*;  *servletContext*.setAttribute("visitNums",++*visitNums*);** response**.setCharacterEncoding("UTF-8");** response**.getWriter().write("当前网站访问人次:"+*visitNums*); }** |

* + 1. 通过ServletContext对象读取资源文件

第一种方法,通过ServletContext.getRealPath()方法;

|  |
| --- |
| // 获得到ServletContext对象 ServletContext ***servletContext* =** config**.getServletContext();** // 获得工程目录web下文件的绝对路径 D:\workspace\idea\servlet-demo1\out\artifacts\servlet\_demo1\_war\_exploded\tx.properties // getRealPath的参数内容不会被校验，只有真正要用这个路径的时候才知道路径对不对 // String path = servletContext.getRealPath("tx.properties");  ***String path* = *servletContext*.getRealPath("tx.properties"); *System*.*out*.println(*path*); *try* {  *InputStream is* = *new* FileInputStream(*path*);  *Properties p* = *new* Properties();  *p*.load(*is*);  *String username* = *p*.getProperty("username");  *System*.*out*.println(*username*); } *catch* (*Exception*** e**) {** e**.printStackTrace(); }** |

第二种方法,使用servletContext.getResourceAsStream()方法;

|  |
| --- |
| //获得工程目录web下文件的流第一个/代表项目的根目录 ***InputStream is* =** ***servletContext*.getResourceAsStream("/WEB-INF/tx1.properties");** |

第三种方法,使用getClass().getClassLoader().getResourceAsStream()方法;

|  |
| --- |
| //使用类加载器的方式来读取classpath下的资源文件，好处不依赖与ServletContext，任何类都可以获得classpath下的资源文件， //不需要再自己指定/WEB-INF/classes ***InputStream is* = *this*.getClass().getClassLoader().getResourceAsStream("tx2.properties");** |

1. Servlet相关资料(熟练)
   1. 面试题
      1. 单例的Servlet

因为Servlet实例是由Tomcat来创建的，但Tomcat只会**创建一个Servlet实例**，所以Servlet就是单例的！这与我们自己写的单例模式不太一样。因为这种单例是通过**容器**tomcat来管理而实现的！

一个实例需要在同一个时间点上处理多个请求！

同步就是安全，但效率太低！

Servlet是线程**不安全**的！

* 不写属性；
* 不写可以存储数据的属性！
  + 1. Servlet的生命周期
* 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。

1. 学习目标

* 了解Servlet概念和JavaWeb的三大组件都包含哪些组件;
* 创建一个Servlet实现javax.servlet.Servlet接口,并自行验证Servlet的生命周期;
* 创建一个Servlet继承javax.servlet.http.HttpServlet类,提供一个初始化属性,并在Servlet初始化时打印它;
* 自己实现统计网址访问人次的功能;
* 熟练掌握两道面试题