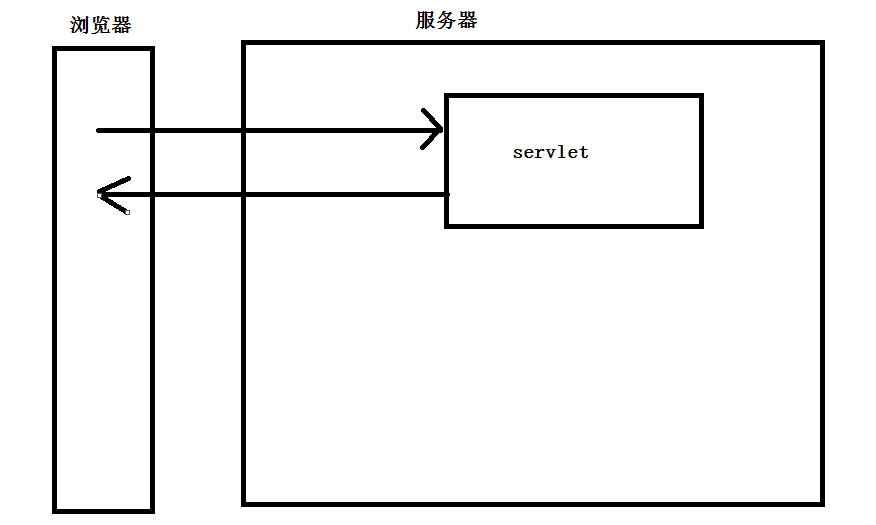
# Servlet

## 第一章Servlet初识

### 1-1 Servlet在web开发中的作用

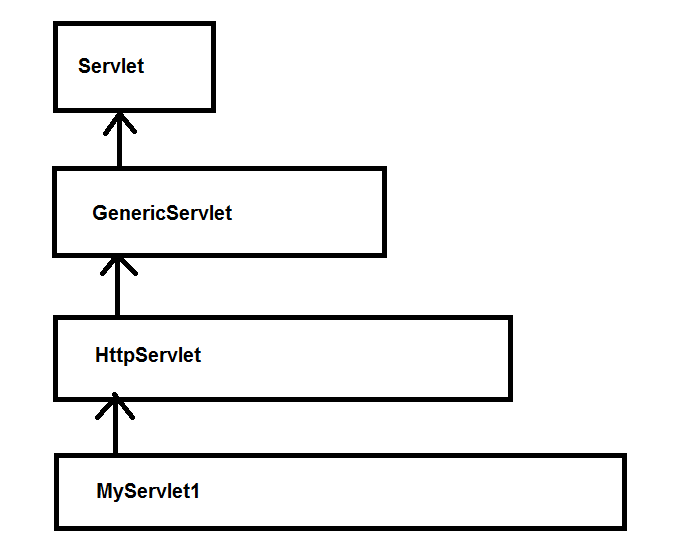


servlet:接收并处理浏览器发出的请求

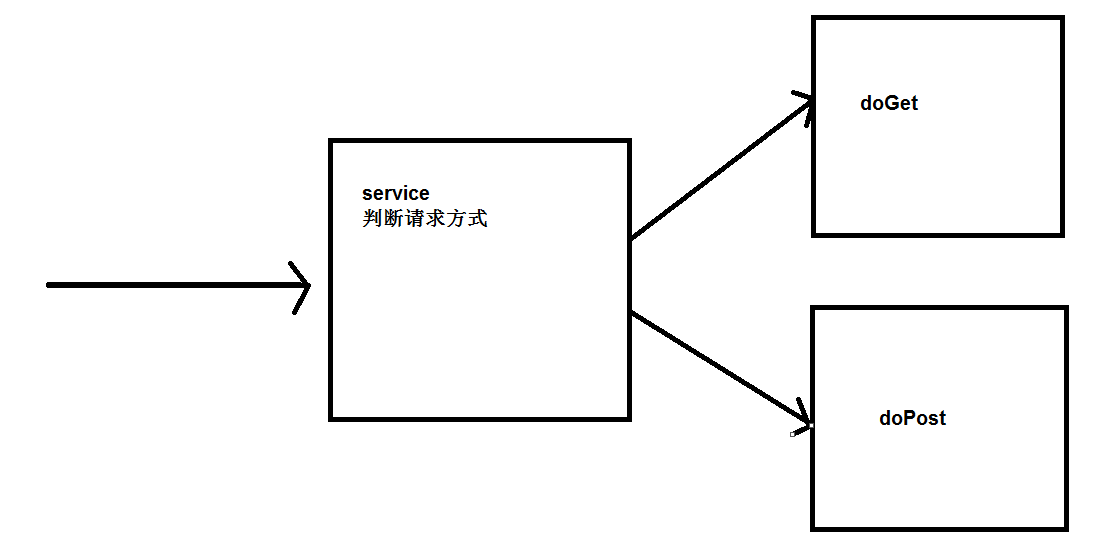
### 1-2 Servlet的创建步骤

见代码注释

### 1-3 Servlet家族继承结构



### 1-4 Servlet中Service方法的应用



service方法不要进行重写,会将分配请求的功能覆盖掉

### 1-5 Servlet的生命周期（以及load-on-startup的用法）

生命周期指的是,类中的对象从创建到销毁的全过程.

servlet的生命周期指的是,servlet对象从创建到销毁的全过程.

1. 启动服务器,没有创建servlet对象.
2. 当浏览器发出请求,用到了指定的servlet来处理我们请求的时候,servlet对象创建.
3. servlet对象创建完毕后,马上调用init方法执行初始化操作.
4. servlet对象在创建完之后,始终存储在服务器上,是以单例的形式存在的.当浏览器发出第2~n次请求的时候,就不重新创建新对象了,而是使用第一次创建出来的那个.
5. servlet中以doGet/doPost方法来处理浏览器发出的请求.
6. 关闭服务器,servlet对象销毁.
7. 如果是正常关闭服务器,servlet对象销毁之前调用destory方法执行最后的处理工作.
8. 加入load-on-startup,可以在服务器启动期间创建servlet对象,更改servlet对象的创建时机,由以前的懒汉式变为饿汉式.



1. servlet是单例,但是是一种假单例.

真单例:new不出来第二个对象,构造方法私有化

假单例:用法是单例,但是能够new出来其他对象,构造方法没有私有化

## 第二章web开发的常见问题

### 2-1服务器启动时报错，如何进行错误排查

情况1:服务器中没有项目,服务器启动报错.如果不能修复,则迅速移除掉我们之前的tomcat,再重写解压一份全新的tomcat,重新集成到Eclipse中.该情况占未来开发的1%.

情况2:服务器中有项目,服务器启动报错,是项目的问题造成服务器启动报错. 该情况占未来开发的99%.

如果是由项目造成的服务器启动报错,则一定是web.xml中的配置出现了错误.

1. url-pattern前面没有加/
2. servlet-name没有匹配项
3. servlet-class配置错误,不会影响服务器的启动,但是在访问的时候报500错误

### 2-2修改哪些文件，必须重新启动服务器

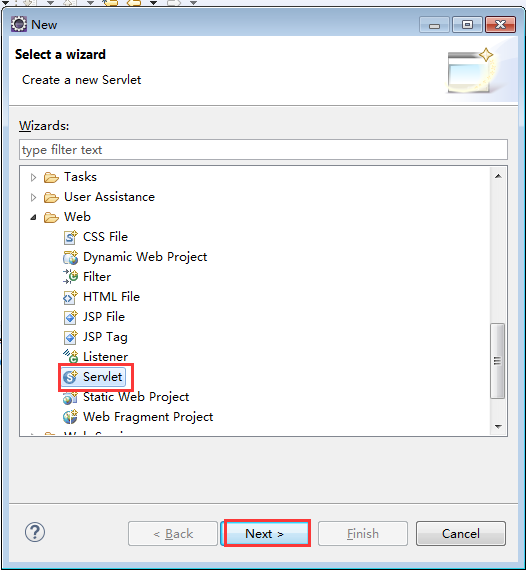
1. 修改前端资源(以html示例),不用重新启动服务器,但是如果浏览器读取的是缓存内容,需要做刷新操作,才能够显示最新的内容.
2. 修改servlet,不用重新启动服务器.但是需要等待java文件重新编译成class文件的过程,才能够访问到最新的内容.

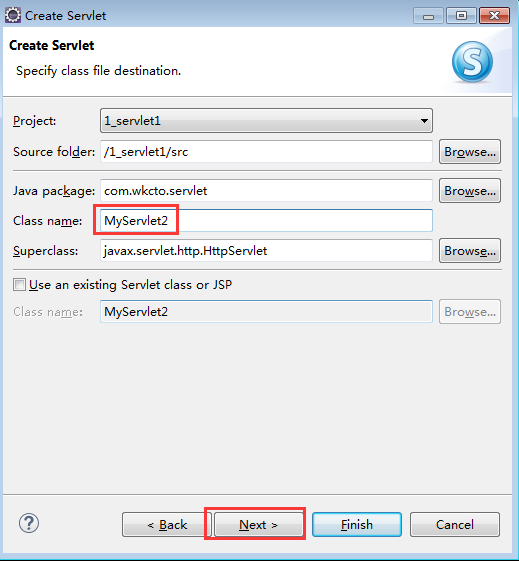
在我们这个阶段,老师的做法是修改完servlet中的内容,每一次都重新启动服务器.

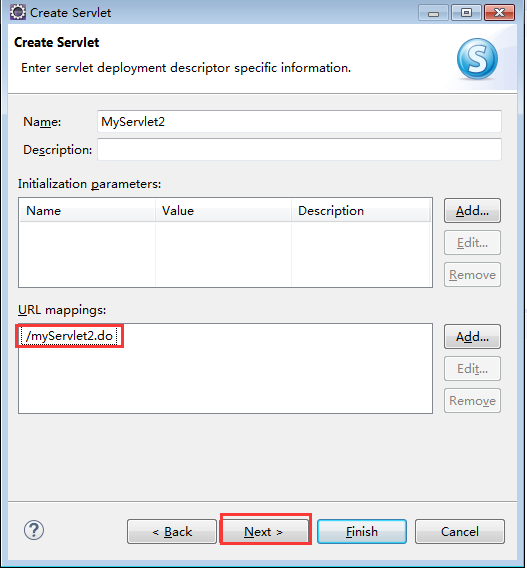
1. 修改web.xml文件,必须重新启动服务器.

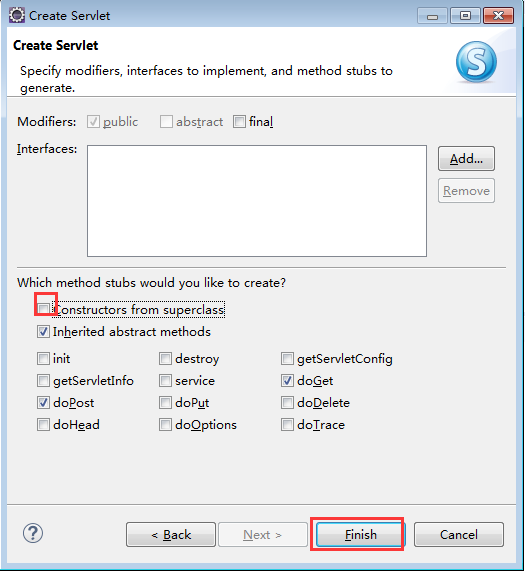
注意:将来我们会大量的做在服务器中对于项目的部署或者移除的工作,在做这些工作之前,必须关闭服务器.

### 2-3使用快捷方式创建Servlet类的方式



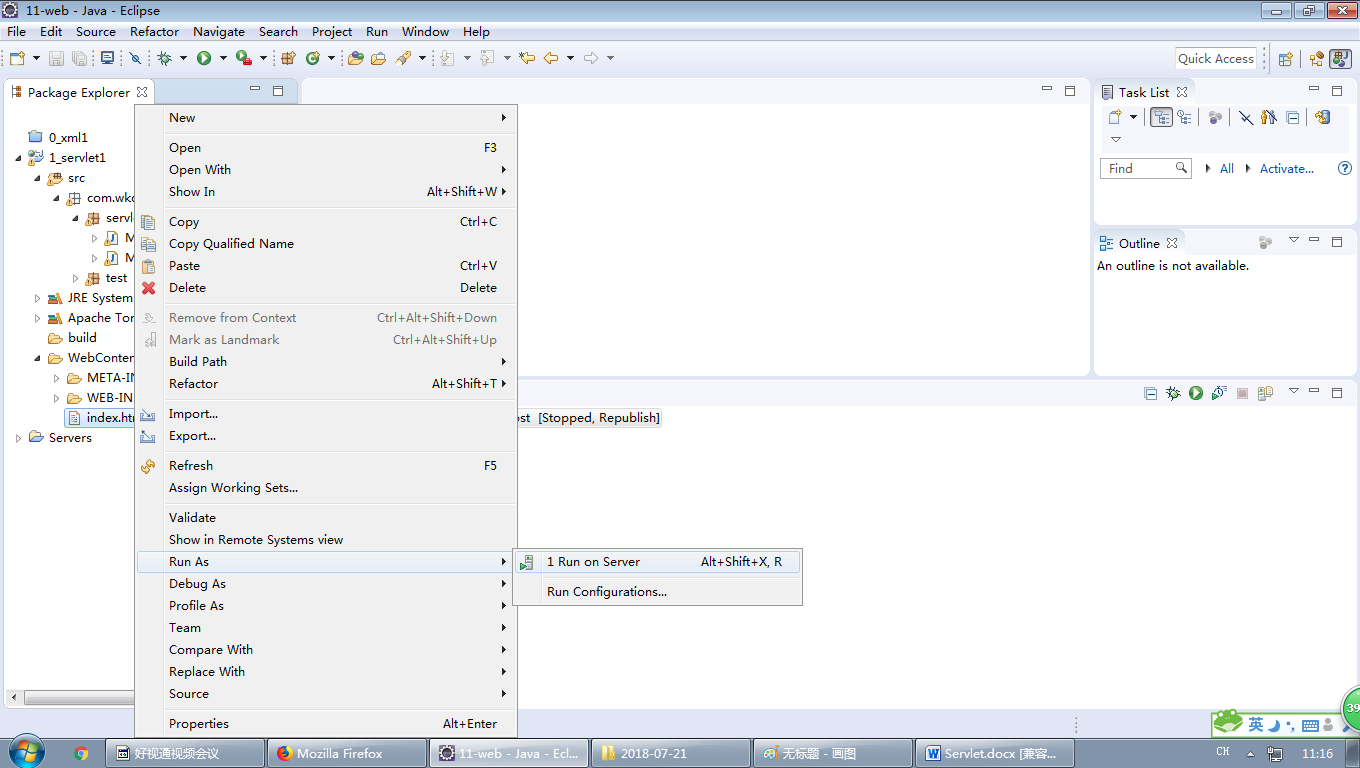






该形式能够快速的创建出来一个servlet,但是以这种形式创建web.xml中的注释会默认的全被干掉,所以这种形式以后用的并不多.我们在搭建servlet模板的时候可以应用到.

### 2-4使用run on server的方式快速发布项目测试简单应用



该形式会快速的发布项目,启动服务器,测试我们指定的资源.

该形式在我们未来的实际项目开发中并不常用.

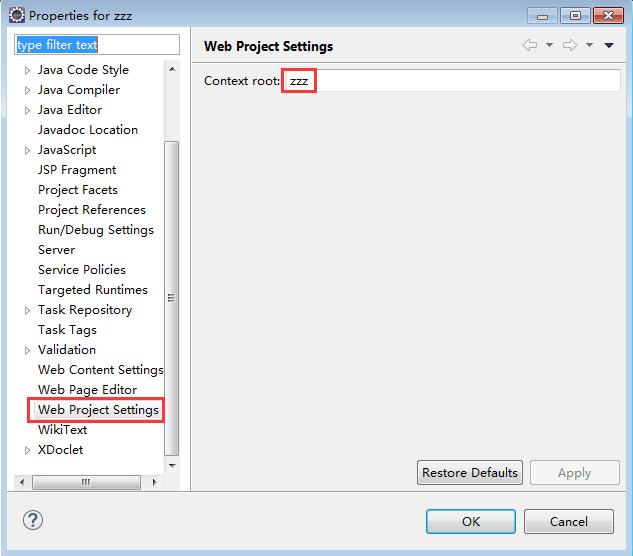
该形式最大的问题在于,Eclipse中自带的浏览器很垃圾,测试简单的应用还可以,测试一些复杂的应用有可能是看不到效果的.更重要的是,在某些情况下,我们的应用程序明明是错误的,使用Eclipse中自带的浏览器会测试成功.所以我们一般情况下不这样来使用.

但是我们可以使用该方式,快速的取得访问路径,拿到路径后在本地的主流浏览器中进行测试就可以了.

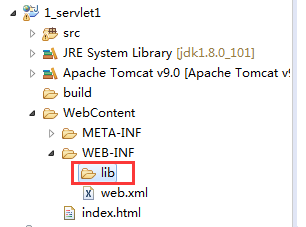
但是我们这个阶段不允许使用该方式,练习在地址栏敲路径.

### 2-5 copy项目及修改项目名的注意事项





### 2-6 web项目导入jar包的方式及注意事项



将jar包粘贴到lib下之后,不需要构建,会自动构建.

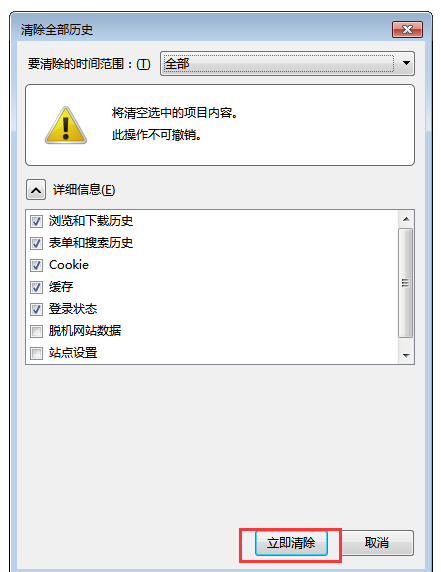
注意:观察是否已经自动构建成功了,如果没有自动构建的话,需要我们自己手动的 add build path操作.

### 2-7清理开发环境及清理浏览器缓存的方式

清理开发环境指的都是将所有的项目从tomcat中移除,重新启动服务器的过程.

清理浏览器缓存

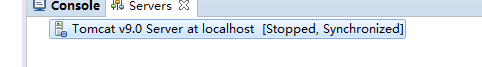
ctrl+shift+delete

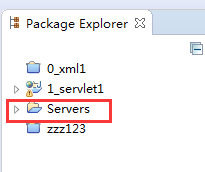


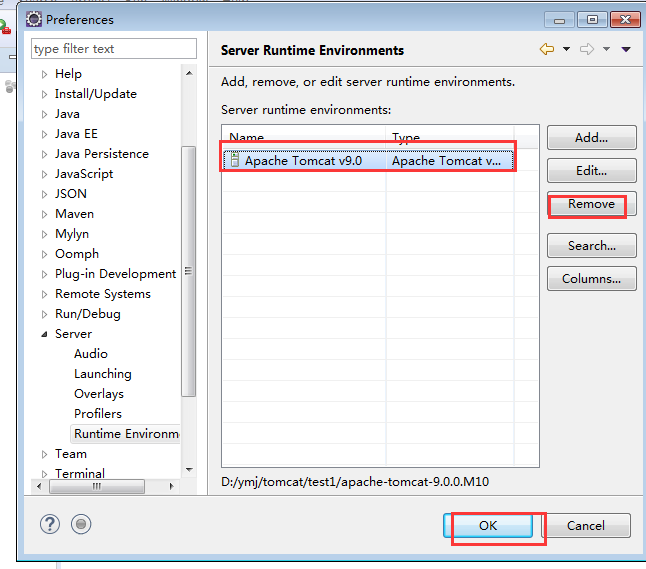
### 2-8原有项目丢失tomcat开发环境，如何重新导入

删除服务器的流程

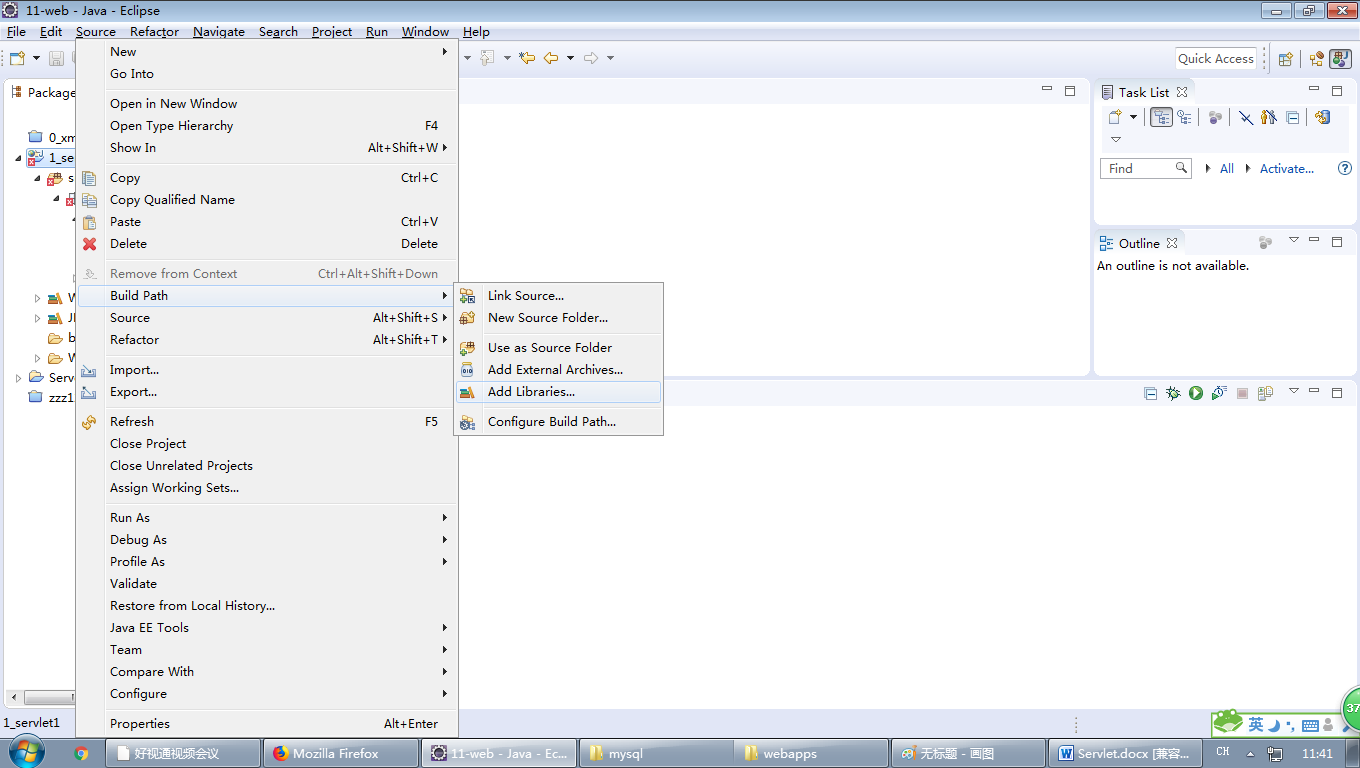
以下操作,见到复选框,都调√

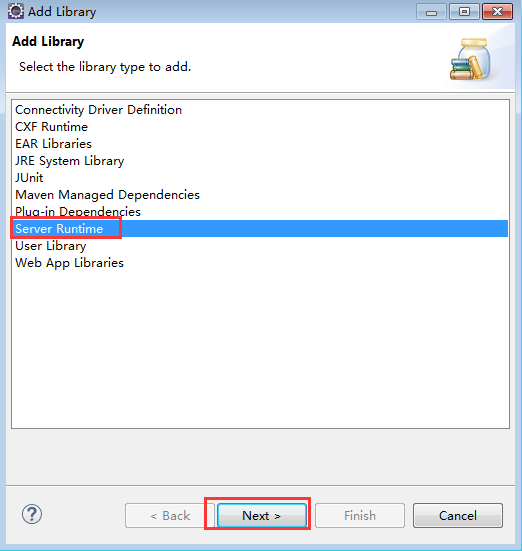


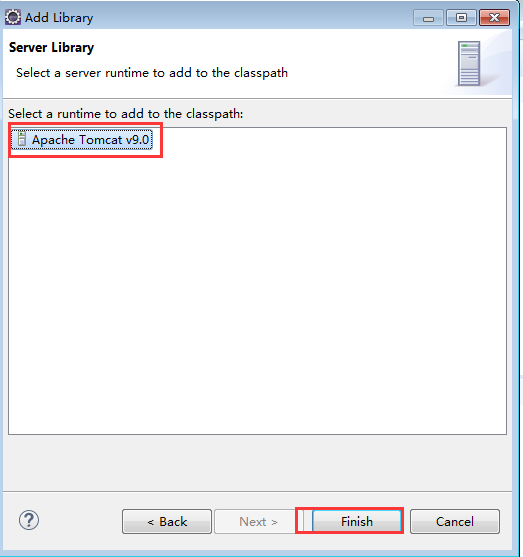




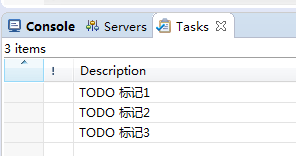
项目重新添加tomcat的开发环境







### 2-9实际项目开发中TODO的使用(tasks视图的使用)



### 2-10 360浏览器的访问问题

启动服务器,第一次发送的get请求,默认执行两次

## 第三章get与post提交方式

### 3-1 get与post提交方式

基于HTTP协议,有两种请求方式,一种为get请求方式,一种为post请求方式.

get:

(1)打开浏览器,地址栏输入访问路径,敲回车,为get请求方式,servlet中由doGet方法来处理请求.

(2)点击超链接,为get请求方式,servlet中由doGet方法来处理请求.

(3)提交表单,默认为get请求方式,servlet中由doGet方法来处理请求.

post:

1. 提交表单,将表单设置为 method=”post”

为post请求方式,servlet中由doPost方法来处理请求.

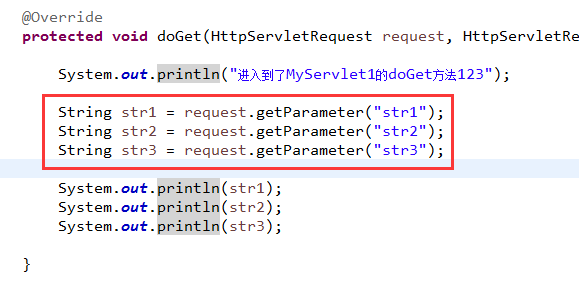
### 3-2 请求参数的传递与接收

参数的传递:

url?key1=value1&key2=value2&key3=value3



参数的接收:



对于表单参数的提交

表单元素input中的name属性值为key

我们在文本框中输入的内容为value

### 3-3 get提交和post提交的区别

get提交

1. 参数的传递

显示的传递参数,参数会显示的挂在url路径的后面

1. 安全性 低
2. 大量的参数传递,有可能会超出浏览器地址栏的上限,导致参数传递的不全
3. 参数的中文支持性

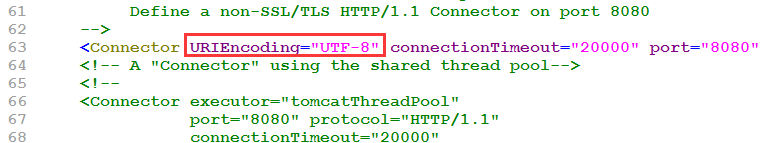
从示例来看,get提交的参数,后台接收后没有出现中文乱码问题.

之所以成功了,是因为我们现在的服务器的版本号较高(tomcat9).

如果应用的是较低版本的服务器,那么一定会出现乱码.

如果出现乱码了,解决方式为

conf/server.xml



注意,以上设置方式仅针对get请求参数乱码问题.

1. 执行效率

执行效率相对较高.

post提交

1. 参数的传递

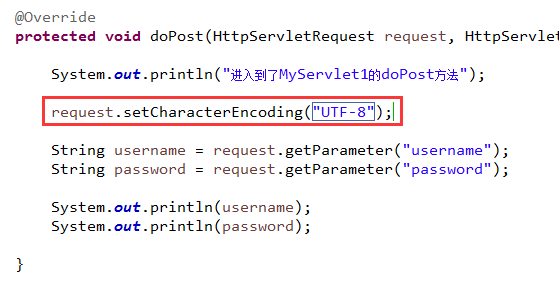
参数是以http协议请求体的方式传输到后台的.

参数是以隐式传递的,在地址栏上是没有体现的.

1. 安全性 高
2. 不用担心参数量多大的问题
3. 参数的中文支持性

通过示例,提交中文参数,后台接收后,是乱码.

我们在后台接收参数前,需要对我们接收的参数进行字符编码的处理.



注意:以上代码仅针对post参数有效.

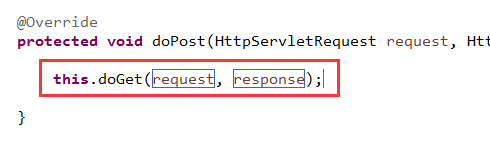
(3)执行效率

执行效率与get请求比较,相对较低.

### 3-4 实际项目开发中对于get和post提交方式的使用

如果我们的需求,不涉及到参数的安全性问题,不涉及到大量的参数的传递问题,一律使用get方式提交,执行效率较高.否则,一律创建表单使用post方式提交.

对于代码结构的处理:



## 第四章url-pattern的使用

### 4-1 url-pattern的表现方式

/abc/myServlet1.do: 精确路径匹配

/abc/\*: 部分路径匹配

/\*: 通配符匹配

\*.do: 后缀名匹配

注意:后缀名匹配,前面不加/

不存在以下形式

/\*.do

/abc/\*.do

abc/\*.do

### 4-2 url-pattern的优先级

如果浏览器发出的请求,路径与web.xml中的多个url-pattern都匹配,则只能选择一个servlet来处理我们的请求.选择的servlet是按照优先级的顺序选择的,优先级较高的优先执行.

精确路径匹配 > 部分路径匹配 > 通配符匹配 > 后缀名匹配

## 第五章响应流

### 5-1 响应流的创建和基本使用



### 5-2 使用响应流响应中文信息



### 5-3 练习：使用响应流响应学生信息列表

domain:领域模型

实体类 pojo

### 5-4 实战：完成登录操作，如果登录成功，则在页面上打印学生信息列表，如果登录失败，则在页面打印失败信息。

用户表 tbl\_user

id int(6)

username varchar(255)

password varchar(255)

学生表 tbl\_student

id varchar(255)

name varchar(255)

age int(3)

## 第六章 转发和重定向

当servlet处理完请求之后,为浏览器做出响应

以前只能使用响应流来完成,使用响应流拼接html非常麻烦

我们可以准备好一张html页面,在servlet处理完请求后,直接跳转到这个html页面就可以了.

使用转发和重定向技术来实现.

### 6-1 转发代码示例

（1）转发到普通页面

（2）转发到WEB-INF下面的页面

（3）转发到另一个servlet

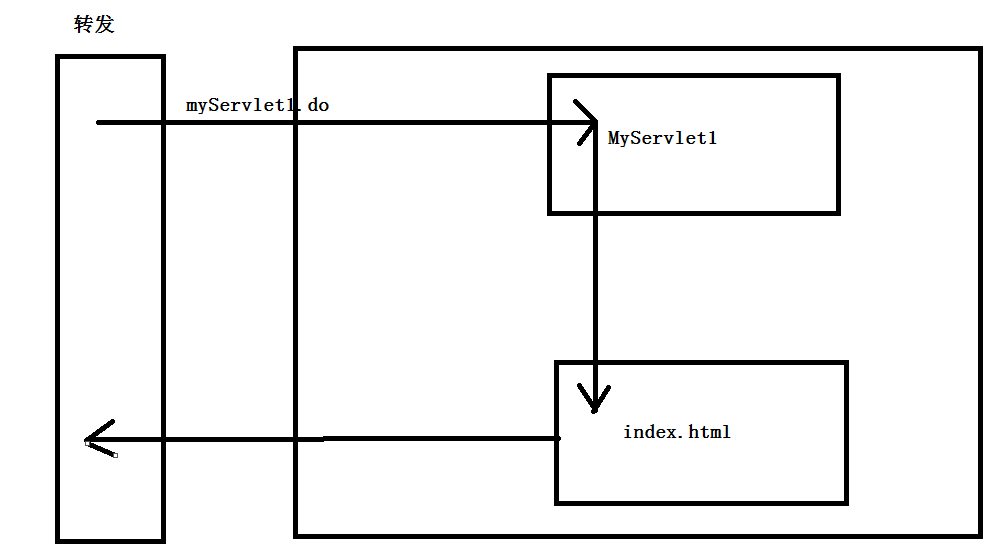
### 6-2 重定向代码示例

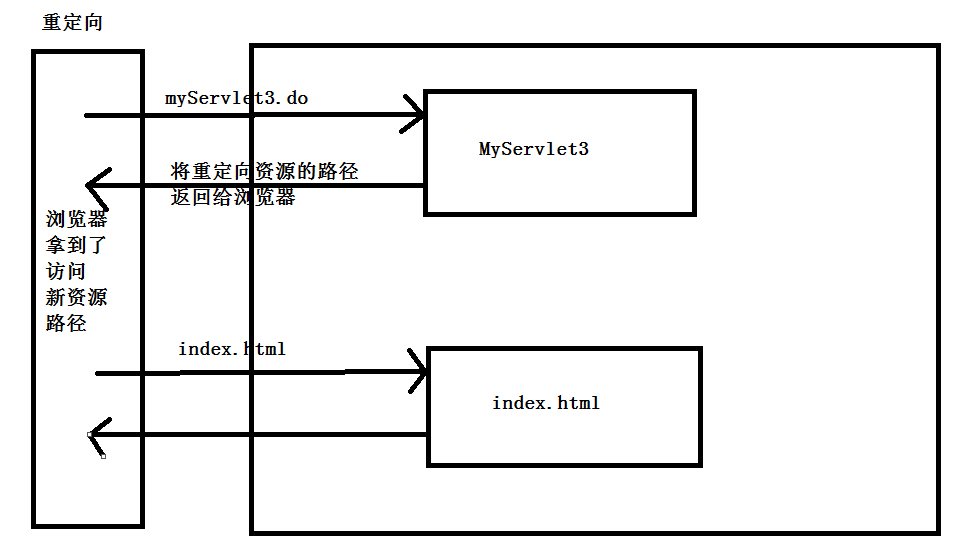
（1）重定向到普通页面

（2）重定向到WEB-INF下面的页面

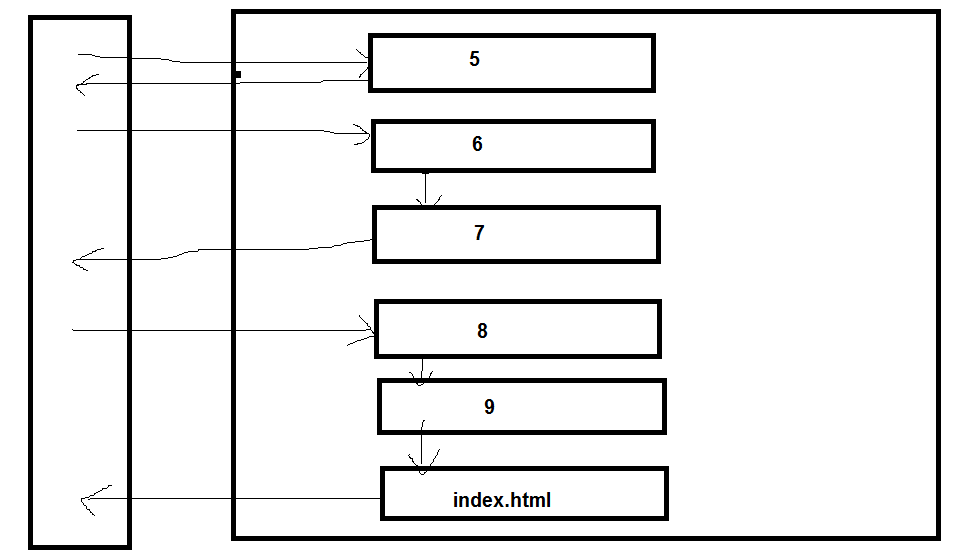
（3）重定向到另一个servlet

### 6-3 转发和重定向原理





### 6-4 练习：使用Servlet做连续转发和重定向的操作



## 第七章ServletContext对象（上下文对象）

### 7-1 上下文对象概述及生命周期

上下文对象在项目中,是一个全局管理者.

在服务器启动的时候,服务器会自动的为我们创建出来一个上下文对象.这个上下文对象始终存储在服务器上.当服务器关闭,上下文对象销毁.

上下文对象作为项目的全局管理者,可以服务于每一个servlet.

**每一个servlet**都可以随时随地的取得并使用上下文对象.

### 7-2 上下文对象的取得方式

### 7-3 上下文对象的常用方法

（1）获得全局初始化参数

getInitParameter(String key)

搭配web.xml中的配置

<context-param>

<param-name>encoding</param-name>

<param-value>utf-8</param-value>

</context-param>

（2）作为域对象存取数据

见代码注释

（3）获取项目发布路径

getRealPath("/")

## 第八章ServletConfig对象

### 8-1 ServletConfig对象概述

在Servlet对象创建出来之后,搭配每一个Servlet对象,会伴随着一个ServletConfig对象的产生.

Servlet对象是我们要用的对象,ServletConfig对象是用来装饰Servlet对象的配置对象.

Servlet对象就相当于人,ServletConfig对象相当于衣服.

### 8-2 ServletConfig对象取得方式

this.getServletConfig()

### 8-3 ServletConfig对象常用方法

（1）获得web.xml中配置的servlet-name

getServletName()

（2）根据key获取value

getInitParameter(String key)

配合web.xml中的配置（注意配置在servlet-class的下方）

（3）取得上下文对象

### 8-4 搭配init方法的注意事项

## 第九章web开发中文乱码解决方式总结

### 9-1 get请求参数字符编码的处理

conf/server.xml 6x URIEncoding=”UTF-8”

### 9-2 post请求参数字符编码的处理

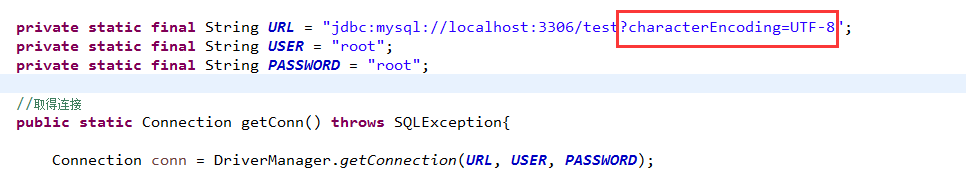
**request.setCharacterEncoding("UTF-8");**

### 9-3响应流字符编码的处理

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

### 9-4通过jdbc往数据库里面存值的字符编码的处理

? characterEncoding=UTF-8



### 9-5通用处理方式

String str11= new String(str1.getBytes("ISO8859-1"),"utf-8");

## 第十章web站点的默认欢迎页的设置



如果请求发送到了服务器中的指定的项目,但是没有指定具体的资源路径,那么项目默认执行配置的欢迎页的内容.

1. 对于欢迎页的设置,我们习惯于使用index系列的名字.这是我们一种对于前端约定俗成的规矩.只要看到了index,就说明他是一个项目的初始页或者是欢迎页.

index.html

index.htm

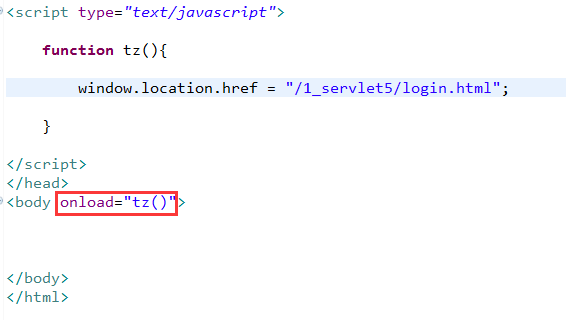
index.jsp

index.jspx

1. 欢迎页可以同时配置多个,按照上下的配置顺序,先有哪个,就执行哪个.
2. 欢迎页可以配置前端资源,也可以配置后端servlet



1. 常用方式



## 第十一章Servlet线程的安全性问题

在我们的实际项目开发中,没有需求会使用到servlet中的成员变量,所以为了线程的安全性问题,我们不在servelt中使用成员变量就可以了.

## 第十二章 实战

### 12-1验证登录，当我们登录失败,提示"用户名密码不匹配",3秒以后跳转到登录页面，若登录成功，可查看学生列表

1\_servlet3 🡪 1\_servlet6

方式1：

常见的响应头-refresh

响应头格式:

refresh:秒数;url=跳转的路径

设置定时刷新:

response.setHeader("refresh","3;url=/z1/login.htm");

这种方式只能搭配响应流的形式来完成

response.setHeader("refresh","3;url=/z1/index.html");

PrintWriterout = response.getWriter();

out.write("<html>");

out.write("<body>");

out.write("3秒钟之后，跳转到登录页");

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

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

方式2：在页面完成头信息的设置

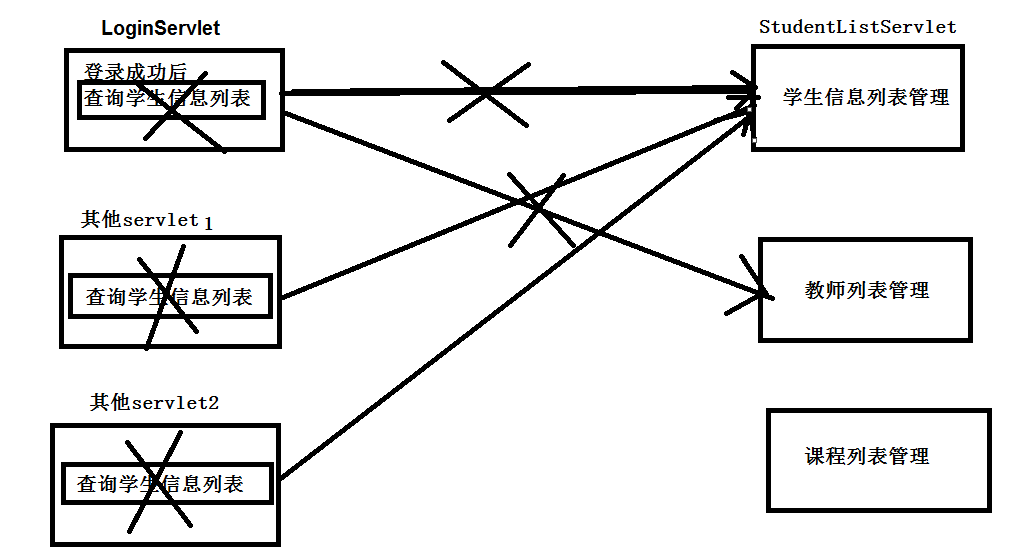
首先重定向到一个倒计时页面

response.sendRedirect("/z1/ts.html");

在该页面设置

<meta http-equiv="refresh" content="3;url=/z1/index.html">

登录成功



### 12-2统计在服务器启动期间，系统被访问的总次数

### 12-3 完成注册操作

### 12-4 完成对于学生信息列表的删除操作