# JavaWEB笔记

## 琐碎知识点

什么是WEB：他是一种基于超文本（HTML）和HTTP协议的、全球性的、动态交互的、跨平台的**分布式图形信息系统**，是建立在Internet上的一种网络服务

什么是WEB应用：多个web资源的集合，是提供浏览器访问的程序，一个web应用由多个静态web资源和动态web资源组成

JavaWEB应用开发的相关技术：

1. 表示层： JSP界面（Java Server Pages）
2. 控制层： Servlet
3. 业务逻辑层： JavaBean或EJB
4. 持久层： JDBC

**学习思路：**

1. 我学的是什么技术
2. 我学的技术能干什么
3. 我该怎么用这个技术
   1. 什么时候能用
   2. 在哪里能用
   3. 具体怎么用

### 自定义错误页面

配置当前web项目的错误页，每当程序发生错误时会自动跳转到指定的页面，在当前整个项目中都有效

在web.xml文件中书写以下标签：

<error-page>

<error-code>错误码</error-code>

<location>路径</location>

<error-page>

### 修改JSP默认码表

1、windows--perferences--general--workspace，Text file encoding设置成utf-8

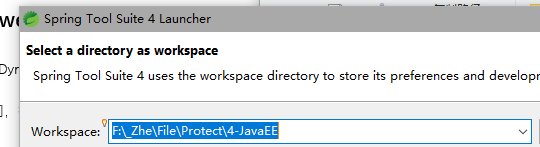
2、windows--perferences--general--content types，把里面text的default encoding utf-8

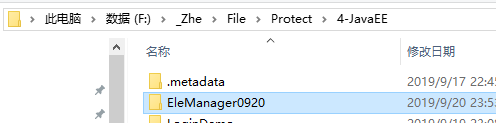
3、windows--perferences--web--jsp files，将encoding设置成utf-8

### STS导入动态web项目

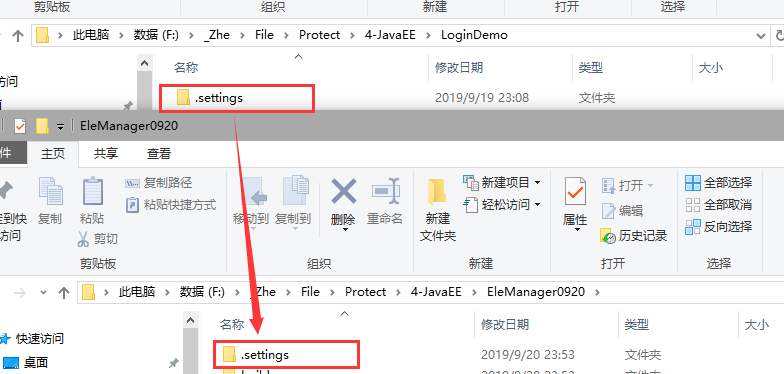
导入动态web项目（Dynamic Web Project），如果项目原本环境与目标环境完全一致（JDK，Tomcat，STS版本），那么第四步及以后的步骤可以省略（如不能运行在尝试修改）

1. 找到你的工作空间，把项目拷贝过去

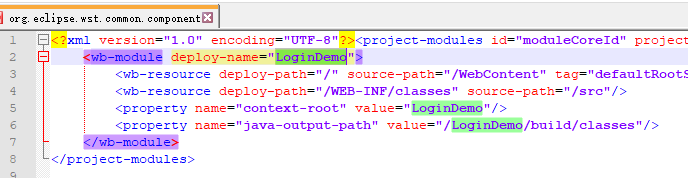




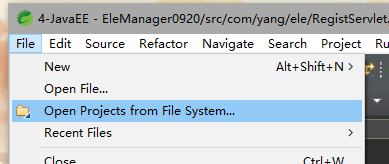
1. 将其他正常项目的.settings文件替换过来



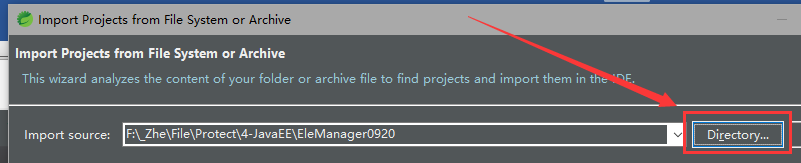
1. 将指定的文件中的项目名修改为当前项目名



1. 导入项目



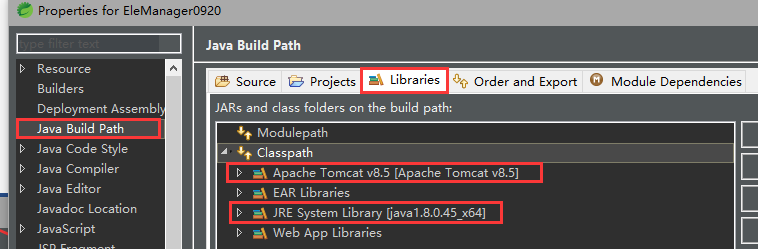
1. 选择你想要导入项目的文件夹，点击finish



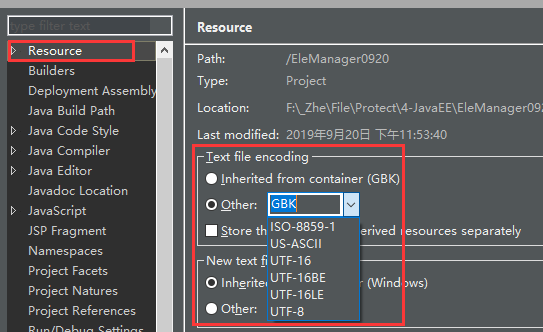
1. 右键刚刚导入的项目，选择Properties选项，修改JDK及Tomcat



1. 找到Java Build Path，修改Tomcat及JDK版本



必要时也可以修改项目编码格式，一般不用修改



项目导入完成

### 协议

HTTP协议：

1. 他是TCP/IP协议的一个应用层协议
2. 定义客户端与服务器通讯的格式
3. 位于应用层简历在TCP/IP协议之上
4. 采用可靠地TCP协议默认为80端口
5. 他是一种无状态的协议（交流一次后忘记对方，下次连接需要重新认识）

HTTP协议组成：

1. 请求行 请求头 请求体
2. 相应行 响应头 响应体

## Tomca服务器

什么是Tomcat：Tomcat是免费的开源的Servlet容器；用来运行Servlet程序的服务器

### Tomcat的安装与配置

安装Tomcat:

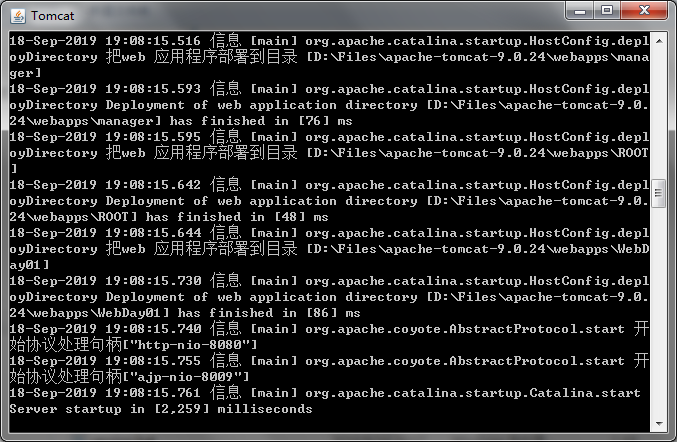
访问tomcat.apache.org下载任意版本Tomcat服务器，下载后为压缩文件，选择目录（建议目录为非C盘，且没有中文和空格的目录）解压即可。

Tomcat环境变量：

添加系统环境变量：CATALINA\_HOME，变量值为Tomcat安装根目录，配置完成后再Path变量中添加%CATALINA\_HOME%/bin，配置完成

打开Tomcat服务器：

在Tomcat根目录下的bin文件夹中，双击startup.bat开启服务器，当弹出黑窗口时，只要出现如下样式的代码，即为开启成功



开启成功后，双击同级目录中的shutdown.bat文件即可停止Tomcat服务器

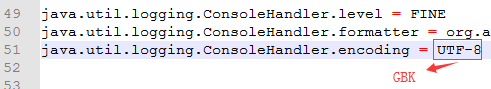
**常见问题**

1. 双击startup.bat后，黑窗口一闪而过，不知道是否开启成功

右键编辑startup.bat文件，在文件最后一行添加pause关键字即可

1. 开启Tomcat后出现中文乱码状况

打开Tomcat根目录 → conf文件夹 → 右键编辑logging.properties文件，修改编码为GBK即可



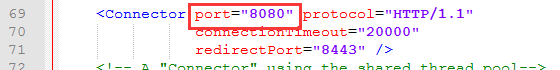
1. Tomcat打不开：请检查环境变量是否配置正确，端口号是否冲突

使用Tomcat发布项目：直接向项目文件拷贝到webapps目录下

### 修改Tomcat配置

修改服务器启动端口号：

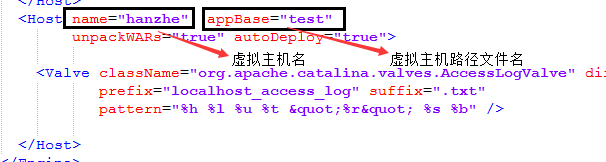
Tomcat根目录下 → conf文件夹 → server.xml文件右键编辑，找到Connector标签：



其中port属性中对应的就是端口号，修改保存即可

修改虚拟主机配置：

1. Tomcat根目录下 → conf文件夹 → server.xml文件右键编辑 滑到最下面，复制host标签，自定义主机名和路径名



1. 然后将默认主机名修改为你自己的主机名（还是在server.xml文件下）：

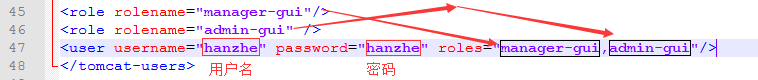
之后还要在Tomcat中新建一个文件夹，以自己的虚拟主机名称命名，将localhost中默认文件复制进去，就可以使用127.0.0.1:8080直接访问了

1. 如果想用域名直接访问的话，那么需要修改本地的host文件的域名绑定

修改/配置Tomcat管理员权限

* 1. 修改/修改Server Status，Manager App账户 ---对应Manager-gui
  2. 配置/修改Host Manager账户 ---对应Admin-gui

Tomcat根目录 → conf文件夹 → tomcat-user.xml ，在</tomcat-users>标签之前加上账户配置信息：



## Servlet

什么是Servlet:

运行在web服务器端的Java类,他能处理web客户端的HTTP的请求，并产生HTTP的响应

HttpServlet继承关系：

HttpServlet继承于GenreicServlet抽象类，同时GenreicServlet抽象类又实现了Servlet接口和ServletConfig接口

Servlet声明周期：从创建到响应客户端请求，直到销毁的过程

Servlet几个重要的接口：

1. ServletConfig配置信息：

用来获取Servlet初始化参数和获取ServletContext对象

1. ServletContext上下文信息：

代表当前Servlet运行环境，可以通过ServletContext访问Servlet容器中各种资源

1. HttpServletRequest：

用于封装HTTP请求信息

1. HttpServletResponse：

用于封装HTTP相应信息

Servlet中常用的方法：

init(): Servlet初始化方法，只能在首次被访问的时候被调用一次，如果此方法没有正常结束，就会抛出ServletException异常

service()： 当用户对Servlet发出请求时被调用，可以理解为每次访问都会被调用

destroy()： Servlet的销毁方法，容器在终止Servlet时调用，可以理解为关闭服务器时调用

-----------------------------------

getServletConfig()：获取ServletConfig对象

getServletInfo()： 返回String类型对象，包含当前Servlet信息

### Request 请求

HttpServletRequest实现了ServletRequest（封装请求信息）接口，用于封装HTTP请求信息

**常用方法：**

getProtocol() 获取使用协议及版本号 例HTTP/1.1

getMethod() 获取请求提交方法get/set

getLocalAddr() 获取请求服务端IP地址

getRemoteAddr() 获取请求客户端IP地址

getRequestURL() 获取请求行中的全路径名

getRequestURI() 获取请求行中主机后面项目名称开始的资源名

getServletName() 获取服务器名称

getContextPath() 获取当前项目名称

getServletPath() 获取URL中项目名称后面的的路径+资源名(访问的文件名)

getServerPort() 获取服务端口号

getQueryString() 获取地址栏?后面的参数，仅对get提交方式有效

**获取请求正文方法：**

获取请求正文方法：

getParameter(String name) 返回有name指定的用户请求参数的值

getParameterNames() 返回所有客户请求的参数名（Enumeration对象）

getParameterMap() 返回一个请求参数的Map

getParameterValues(String name) 返回由name指定的用户请求参数的值（String[] 数组）

请求参数乱码问题：

1. **原因**：浏览器会按照当前页面字符集对请求的中文数据进行编码，在请求响应的过程中对数据进行解码时，会按照默认的ISO-8859-1进行解码，于是导致中文参数乱码问题
2. **解决**：在调用getParameter方法之前先调用**setCharacterEncoding（charset）**方法设定与页面请求编码相同的解码字符集

### Response响应

#### 了解状态码

状态码由三位数字组成，第一位定义了状态码的类型，后面的无分类作用，类型分别为：

* 1\*\*：服务器已经收到请求，需要请求者继续执行操作
* 2\*\*：请求已经收到、理解、并接受
* 3\*\*：需要客户端采取进一步操作才能完成请求
  + 一般用来重定向
* 4\*\*：表示客户端错误，请求语法错误或无法完成请求
* 5\*\*：服务器处理请求时发生错误

常见的状态码

404： 找不到资源文件

302： 资源暂时转移到其他URL

404： 请求的资源不存在

500： 服务器内部错误

#### HttpServletResponse

HttpServletResponse实现了ServletResponse（创建响应消息）接口，用于封装HTTP响应消息，他的常用方法如下  
**设置响应消息头**

setContentType( Stirng mime ) 设置content-type消息头

setContentLength( int length ) 设置content-Length消息头

addHeader( String name，String value ) 新增String类型头信息

addIntHeader( String name，int value ) 新增int类型头信息

addIntHeader( String name，long date ) 新增long类型头信息

addCookie( cookie c ) 为Set - Cookie消息头添加一个值

**设置响应正文**

getOutputStream() 获取字节输出流对象

gerWriter() 获取字符输出流对象

**响应输出中文乱码问题解决：**

resp.setContentType(“text/html;charset-UTF-8”)

### XML中配置Servlet

直接在web项目中书写Servlet是不可以被浏览器访问的，需要配置该Servlet的信息

通过配置项目中的web.xml文件：

<servlet>

<servlet-name>类名</servlet-name>

<servlet-class>包名.类名</servlet-class>

<load-on-startup>10</load-on-startup> 运行优先级，值越低越高

</servlet>

<servlet-mapping> mapping：映射

<servlet-name>类名</servlet-name>

<url-pattern>/虚拟主机</url-pattern>

</servlet-mapping>

Servlet声明配置有两种方法，另外一种是通过注解的方法，具体[**跳到注解**](#_注解)查看

### 重定向&转发

**重定向：**

重定向是服务端接受到请求后，返还给客户端一个新的URL地址，两次请求两次响应，且地址栏改变，状态码改变为302

首先通过resp.setStatus(302)来修改状态码，表示要使用重定向，然后调用resp.Header(“location”,”URL”)更改头信息实现重定向

由于重定向十分受用，所以将修改状态码为302和重定向放在一起使用，形成一个新的方法为：sendRedirect()方法，具体体现为

**转发：**

转发是调用resp.getRequestDispatcher()方法，紧接着调用rd.forward(req, resp)，具体表现为：

客户端向服务器请求A，A将请求发给了B，然后将B的结果响应给客户端，一共发生了一次请求和相应，地址栏不产生变化

**重定向与转发的区别：**

1. 重定向是两个全新的对象，两次请求，两次响应，而转发是使用的相同的对象
2. 重定向可以跳转外网部资源，而转发只能跳转本应用资源
3. 重定向和转发不能同时出现在同一个Servlet中

### 四大域对象介绍

JavaWeb中一共有四大作用域，按照范围大小排序为：

对象 范围

1. HttpServletContext 当前整个web项目（作用域最大）
   1. 生命周期为当前项目发布开始，直到项目移除或者服务器关闭
2. HttpSession 一次会话
   1. 生命周期为调用reques.getSession方法开始，未使用到30分钟（默认值），或者调用invalidate销毁方法，或者服务器关闭
3. HttpServletRequest 一次请求
   1. 生命周期在调用Service方法之前由服务器创建，传入Service方法后结束
4. PageContext（Page） 当前JSP页面 （作用域最小）
   1. 生命期为当前JSP页面，超过当前页面便不能使用

### ServletConfig

作用：

获取当前Servlet中初始化参数和获取ServletContext对象

参数都包括：

1. web.xml下的当前Servlet标签下的init-param标签内的值
2. @WebServlet下的初始化参数

配置ServletContext初始化参数：

在当前项目下的web.xml下添加web-app标签，设置name和value属性

主要方法：

getInitParameter(String param)： 传入初始化的参数名称，返回匹配的值

getInitParameterNames()： 返回一个Enumeration对象，里面包含所有初始化参数

getServletContext()： 返回一个ServletContext对象

getServletName()： 返回当前Servlet名字（@WebServlet中的name）

### ServletContext

也被称为上下文接口，代表当前Servlet运行环境，是Servlet与Servlet之间通信的接口

操作域属性：

getInitParameter(String name) 返回Web应用范围内匹配的初始化参数值。

getInitParameterNames() 返回一个包含初始化参数的Enumeration对象

setAttribute( String name，Object obj ) 把name+value绑定并存放

getAttribute(String name) 给定一个name，返回一个value

getAttributeNames() 返回一个Enumeration集合对象，

removeAttribute(String name) 删除指定的name-value

操作Web应用信息：

getContextPath() 返回当前web应用的根路径

getServletContextName() 返回web应用名字（web.xml中设置的name）

getRealPath( String url) 返回系统中URI对应的绝对路径，如果不能获取则返回null

getResourceAsStream(路径) 得到web应用中的资源文件image等

getResourceDispatcher( String path ) 返回用于其他组件转发请求的RequestDispatcher对象

## 会话跟踪技术

什么是会话：【打开浏览器 → 访问服务端 → 关闭浏览器】这个过程可以简单理解为会话，客户端和浏览器交互的过程就叫做会话

会话跟踪技术： 是一种让服务端和客户端保持HTTP状态的技术，使上一次发出的请求能够维持到下一次请求，并且辨认出是否是相同的客户端所发出来的----因为HTTP是一种无状态的协议（交流一次后忘记对方，下次连接需要重新认识）

会话跟踪技术主要体现为： 【Cookie】-【Session】-【URL重写】-【隐藏表单域】

### Cookie技术

1. 是一种基于浏览器让服务端和客户端保持HTTP状态的技术，会话数据保存在客户端浏览器
2. Cookie在用户首次访问服务器时通过响应头发给客户端浏览器，当用户再次访问发出请求时会附带文本信息
3. 使用Cookie时需要保证客户端浏览器没有禁用Cookie

Cookie常用方法：

1. new Cookie(key,URLEncoder.encode(value,”UTF-8”)) 创建Cookie对象，存储value及设置编码
2. setPath() 设置Cookie的有效路径
3. setMaxAge( int time ); 设置Cookie在浏览器缓存保存时长

如果参数为正数，则设置的数值为保存时长（毫秒值）

如果参数为负数。则为临时Cookie，关闭浏览器时自动删除

如果参数为0，则表示立即删除该Cookie

Cookie的默认保存时间为-1，也就是临时参数

1. setValue( value ) 设置Cookie的值
2. getName() 获取存储Name
3. getValue() 获取存储Value

注意：创建Cookie结束之后，一定要调用Response中的addCookie方法，将Cookie响应给客户端

请求响应中对应的方法：

req.getCookies() 获取Cookie对象

resp.addCoocies() 追加Cookie对象

Cookie局限性：

1. Cookie可能被禁用客户端浏览器禁用
2. Cookie可能被删除
3. Cookie的大小和格式受限制（看浏览器，一般是4095字节，50个**/**每个域名）
4. Cookie安全性不够高，他是以纯文本方式记录与文件中

### Session技术

1. 他是HttpSession接口的对象，也被称为会话对象，是一种让服务端和客户端保持HTTP状态的技术
2. 客户端访问访问服务端动态资源的时候服务器自动创建Session对象
3. 服务器在创建会话对象时，会分配一个全球唯一的会话标识SessionId，以JsessionId的属性保存到Cookie中

Session和Cookie的主要区别：

Cookie是将数据写入到客户端浏览器中

Session是将数据写到用户独占的session中

Cookie是通过创建的对象，而Session是通过获取的对象（也存在创建过程）

常用方法：

req.getSession( boolean b ) 获取Session（检测是否携带sessionId），当没有session则创建一个Session（b=true，default）

setAttribute( String key , Object obj ) 向Session域中存放数据

getAttribute( String key ) 获取session数据

removeAttribute( String key ) 删除session数据

invelidate() 销毁该Session

setMaxInactiveInterval( int s ) 长时间不产生会话定时销毁 /单位 秒

getMaxInactiveInterval() 获取活动有效时间，返回int值/单位 秒，设为-1永不过期

getId() 获取当前Session对象表示sessionId

getCreationTime() 获取对象的产生时间

getLastAccessedTime() 获取最后通过这个HTTP Session对象发发出请求的时间

### URL重写技术

什么是URL重写：

URL重写是通过硬编码将id传到地址栏追加到URL的尾部

他的作用时什么：

防止客户端禁用Cookie,导致id无法传入服务器

重写URL需要用到HTTP Servlet Response中的：

encodeURL（URL）方法： 对任意请求的URL进行重写

encodeRedirectURL（URL）方法： 对重定向进行URL重写

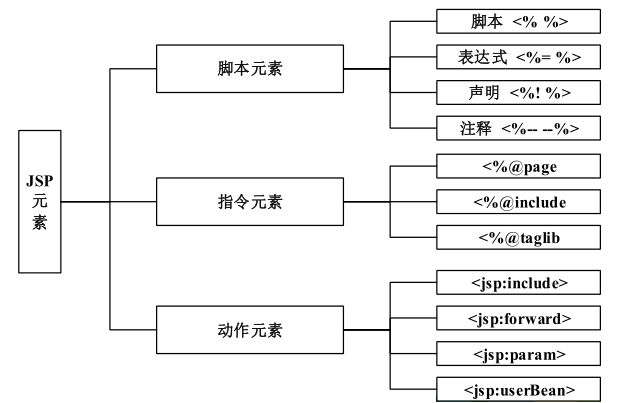
如果重写方法中的请求信息中包含set-Cookie请求头，则将URL原样输出，如禁用了Cookie且请求信息中年不包含则通过硬编码将SessionId传到地址栏追加到URL的尾部

## JSP技术

1. JSP(Java Server Pages)是服务器端的脚本语言,是带有JSP元素的常规web页面
2. JSP同Servlet样,都是运行在Servlet容器(Toncat服务器)中
3. JSP是首先被翻译为Servlet后才编译运行的,本质上其实就是Servlet
4. 相比HTML而言,HTML只能为用户提供静态数据,而JSP允许在页面中嵌套java代码,为用户提供动态数据

JSP由三种类型的元素:

【脚本元素】 【指令元素】 【动作元素】



### JSP指令元素

JSP脚本元素都包括：

【脚本】 【声明】 【注解】 【表达式】

脚本元素的生命格式为：

<%脚本符号%>

**JSP的脚本：**

<%! %> 声明的变量是全局变量，翻译为Servlet后是方法外的变量

<% %> 书写Java代码的区域，翻译后为Servlet中Service方法内的代码

<%= %> 书写表达式的java代码区域

<%-- --%> JSP代码的注释

什么是JSP指令：

他并不直接产生任何可见的输出，而是告诉Servlet容器如何处理JSP页面中的其余部分

JSP指令语法：

【<%@ 指令 属性名 = “值” %>】

JSP指令的元素包括三种：

【page指令】 【include指令】 【taglib指令】

#### page指令

描述了页面相关的信息，例如导包，指定输出内容类型等，在一个JSP页面中，page指令可以出现多次，但是在page指令中，每种属性只能出现一次，重复设置之前的会被覆盖掉（import属性除外）

**page指令的属性都包含：**

language 设定JSP页面默认的脚本语言，默认为Java

contentType 设置tomcat服务器发给浏览器时的编码表

pageEncoding 设置当前JSP文件在磁盘中保存的编码格式（可省略不写，默认使用contentType中的编码表）

import 导入Java包，引用多个请用，逗号隔开

session 指定页面中是否可以使用Session，true/false，true为默认值

isELIgnored 指定页面是否忽略EL表达式true/false，默认false

errorPage 指定当前页面的错误处理页面，和isErrorPage配合使用，但是不能在同一个页面上

isErrorPage 指定当前页面是否为错误处理页面true/false，错误页面可以使用exception内置对象，而一般页面不可以使用

buffer 指定输出流是否需要缓冲，默认为8kb，一般与autoFlush一起使用

autoFlush 如果为页面缓存区满时要自动刷新输出，则为true，否则，缓存区满时为抛出异常则为flash

isThreadSafe 指定执行JSP程序的模式（true单线程/false多线程），一般建议使用多线程

#### include指令

1. include又被称为包含指令，静态包含，在由JSP翻译为Servlet时引入包含文件
2. 被包含的文件可以是JSP、HTML或文本文件
3. 当包含文件时，必须保证新合成的文件符合JSP语法规则，例如当前文件和被包含文件不能同时定义同名变量
4. 使用语法为 **<%@include file=”文件” %>**

**静态包含** 是将所有包含的页面放在一起编译，无论有多少JSP文件，编译过后只有那一个主要的class文件，因为其他的都被包含进去了

使用<%@include file=””%>的时候，要在被包含的文件中删除多余的HTML代码，不然包含后，同一个页面会出现多个HTML的声明， 只可删除HTML的代码，但是JSP的第一行属于JSP的声明标签，不要删除

#### taglib指令

taglib指令被叫做标签库指令

他的作用是可以在JSP页面中使用标签库中的标签

书写格式<%@ taglib uri=”标签库URI” prefix=”标签前缀” %>

### 动作元素

JSP页面仅做数据显示模块，不希望嵌套任何的Java业务逻辑代码，但在实际开发中不引入业务逻辑是不可能的，引入还会降低代码的维护性、可读性

因此Sun公司允许用户开发自定义标签封装页面的Java代码，以便jsp页面不出现Java代码

JSP标签也称之为Jsp Action元素，用于在JSP页面中提供业务逻辑功能

动作标签大致分为三种：

1. include 当前页面包含其他资源
2. forward 转发到其他资源
3. param 传参

#### include动态包含

如果包含的是普通文本文件，就将这个文件发送到客户端，如果是JSP文件，那么就将JSP容器执行后的结果发送给客户端，客户端负责显示

动态包含 仅仅是将输出结果放在一个页面上显示，其实内部是把各自的JSP单独运行，包含了多少JSP，就会有多少个class文件

代码体现:

<jsp:include page=”资源uri”>

<%- 在引入文件的时候传入参数，如无需要可不写 -%>

<jsp:param name=”name” value=”value” />

<jsp:include>

动态包含和静态包含的区别：

对包含文件处理的方式不同

1. 静态包含是在翻译阶段（JSP→Servlet）就已经开始引入所包含的文件，被包含的文件依赖于当前JSP页面，优点是执行速度快
2. 动态包含是在JSP页面运行时引入包含文件产生的结果，被包含的文件都是独立的，特点是可以使用param子标签更加灵活的处理信息

执行速度不同：

静态包含相对于动态包含执行速度更快

#### forward转发

forward标签用于将请求转发给另一个资源

他的使用语法为:<jsp:forward page=”资源名称”>

forward标签内部也可以嵌入<jsp:param />标签用来传值

代码体现：

<jsp:forward page=”text1.jsp”>

<jsp:param name=”name” value=”<%= 表达式 %>” />

<jsp:forward>

### JSP内置对象

JSP内置对象就是在页面中不用声明就可以在脚本和表达式中直接使用的对象，说白了就是不用new或者获取，就可以直接调用其中的方法

JSP的内置对象也被称为隐含对象，他有以下特点：

1. 由Web容器自动载入，不需要实例化
2. 通过Web容器来实现和管理
3. 每一个JSP在第一次被访问的时候，JSP引擎先将JSP翻译为一个\_jspServlet文件，然后按照Servlet的方法调用
4. 在JSP引擎调用\_JSPService时创建或传递内置对象

**JSP中一共有9种内置对象：**

1. page 指向当前JSP翻译后的Servlet对象，发生了向上转型，所以没有任何方法
2. pageContext 当前页面范围域
3. request 请求范围域
4. response 响应对象
5. out 输出对象
6. session 会话存储域
7. application 当前应用存储域
8. config 配置对象
9. exception 异常对象

注意：

JSP被翻译成Servlet文件后，内置对象都是在方法区内获取的，这也就意味着在方法外无法调用内置对象，也就说明了在<%!%>的声明中不可以使用内置对象

#### out对象

out对象就是输出对象，基本等同于writer对象，需要注意的就是针对缓存区处理的方法

缓存区处理方法：

clear() 清除缓存区内容，如果缓存区为空，则产生IO异常

clearBuffer() 清除缓存区内容，不会产生异常

flush() 将目前暂存区缓存区的数据刷新输出

close() 关闭输出流，一旦关闭流就不能使用out对象

getBufferSize 获取使用后的缓存区剩余大小（kb）

isAutoFlush() 返回boolean值，true代表缓存区满自动刷新输出

#### page&pageContext对象

page对象，指向当前JSP页面（指向当前JSP页面翻译后的那个Servlet对象），是四大域对象之一，可以理解为Java中的this关键字

虽然page属于作用域，但是他并没有类似其他域对象中的getAttribute或setAttribute方法，因为page对象发生了向上转型，它属于Object类型

有一个对象专门针对page对象存储信息：pageContext对象

pageContext又被称为页面上下文对象，作用域为当前JSP页面

常用方法：

getOut() 获取当前JSP页面的输出流

getPage() 可理解为this/获取当前JSP页面的实体

getRequest() 获取当前JSP页面请求对象

getResponse() 获取当前JSP页面响应对象

getSession() 获取和当前页面有联系的会话对象

getServletConfig() 获取当前页面ServletConfig对象

getServletContext() 获取当前项目上下文对象

getException() 获取异常对象、设置当前页为异常页

getAttribute(String name，int scopr) 获取指定作用域指定name的value

setAttribute(String name，Object value，int scope) 在指定作用域中存储键值对

removeAttribute() 删除页面上下文的name+value

findAttribute(String name) 从低到高获取指定name对应的value值，如果没有则返回null

getAttributeNamesInScope() 获取页面所有上下文的name

在pageContext中有一个特殊的方法：setAttribute( String name , Object value , int scope)向指定的域中存储数据

其中的int scope参数是用来指定存储/获取数据的作用域，他的参数如下：

pageContext.PAGE\_SCOPE 对应pageContext域

pageContext.REQUEST\_SCOPE 对应request域

pageContext.SESSION\_SCOPE 对应session域

pageContext.CONTEXT\_SCOPE 对应context域

## EL表达式

EL表达式主要的作用是：

替换JSP页面中的脚本表达式、可以在JSP页面中执行一些基本的关系运算、逻辑运算及算数运算

EL表达式含有11个内置对象，可以轻松获得对web常用对象的引用、可以调用用户自定义的Java函数

EL表达式隐含对象：

pageContext

pageScope

requestScope

sessionScope

paramValues

cookie

initParam

param

### EL基本语法

EL表达式可以当作输出语句使用，直接打印在客户端浏览器上的输出流

EL表达式不能直接操作对象，需要将对象先一步存到域中，然后通过域的name属性进行操作

基本运算操作符



empty userid userid为空返回true

not empty userid userid不为空返回true

EL表达式的基本语法：

${“字符串内容”} // 可以直接输出字符串

${ 2+3 } // 可直接输出表达式结果 5，如果是两个字符串类型的数字相加，也会产生结果，如果类似a+b（字符串相加），会出现类型转换异常

${ 2>3 } // 可以直接输出表达式结果true/false

${empty person } // 如果对象为空，返回true（包括空字符串）

${not empty person } // 如果对象为空，返回false

### EL表达式11个内置对象

pageScope page作用域

requestScope request作用域

sessionScope session作用域

applicationScope application作用域

param request请求参数

paramValues request请求参数字符串集合

header HTTP信息头

headerValues HTTP信息头的字符串集合

initParam 获取ServletContext初始化参数

cookie 操作cookie的值

pageContext 当前页面的作用域

### 使用EL表达式操作域

获取域中的数据：

通用格式：${域对象.name }----即为获取到指定域中指定name对应的value值，例如：

${pageScope.userid } // 获取pageContext中userid对应的value值

pageScope： 对应pageContext域

requestScope： 对应request域

sessionScope： 对应session域

applicationScope：对应application域

如果不书写域作为前缀直接书写name属性的话，那么他会默认从小（pageScope）到大（applicationScope）的顺序找，如果都没有则返回空

使用EL表达式获取自定义类的数据：

${person.name } // 使用对象名.属性名的方法获取到对应的值

细节：其实他并不是直接访问person中的name属性，而是访问了name属性对应的getName方法

也就是说如果类中有name属性，但是没有getName方法，或者有getName方法但是命名不符合要求，一样还是访问不到数据

使用EL表达式操作域中存储的JavaBean

pageSession.setAttribute(name,new JavaBean(key，value)) // 将JavaBean存储在域中

/\*

\* sessionScope： 指定了获取范围是session域

\* sessionScope.name： 获取到指定name的JavaBean对象

\* sessionScope.name.key： 获取到了JavaBean中的一个具体的属性

\*/

${ sessionScope.name.key }

使用EL表达式操作存储数组的值

pageRequest.setAttirbute(name,new int[]{11,22,33}) // 将数组对象存储在域中

${ requestScope.name[0] } // 访问了数组中下标为0的值

操作符 . 和 []

在上面使用了[]来指定了访问的下标，除开这种情况下使用，在获取的name中含有特殊符号的时候也可以使用

pageSession.setAttribute(“good-boy”,’zhang’) // 当域中存储的信息中的name含有特殊符号时

${ sessionScope[good-boy] }

使用EL表达式获取请求参数

URL：localhost:8080/项目名/文件路径/资源名?userid=”zhang”&username=”hanzhe”

$(param.userid) // 获取userid的值

$(paramValues.username[0]) // 获取请求数组下标为0的值

### 其他隐含对象的常用方法

pageContext隐含对象常用方法：

${ pageContext.servletContext.serverinfo } 获取服务信息

${ pageContext.request.requestURL } 获取请求地址

${ pageContext.session.creationTime } 获取session创建时间

${ pageContext.response.contentType } 获取响应的文档类型

header隐含对象常用方法：

${ header.host } 获取请求头host的值

${ headerValues[“name”] } 获取请求头accept的

cookie隐含对象的用法

${ cookie.JSESSIONID } 获取cookie中的JSESSIONID对象

${ cookie.JSESSIONID.name } 获取指定对象的名

${ cookie.JSESSIONID.value } 获取指定对象的值

## JSTL标准标签库

通过JSTL表达式以标签的形式书写流程控制语句(循环;分支;判断),进而摆脱JSP页面内套用java代码

标签库：

核心标签： **c** http://java.sun.com/jsp/jstl/core

web应用的基本操作标签，输出、循环、判断等

I18N格式化标签： **fmt** http://java.sun.com/jsp/jstl/fmt

web应用程序国际化标签，时区转换等

SQL标签库： **sql** http://java.sun.com/jsp/jstl/sql

访问数据库和针对数据库操作的标签

XML标签库: **x** http://java.sun.com/jsp/jstl/xml

对XML文件的数据进行操作的标签

JSTL函数标签库： **fn** http://java.sun.com/jsp/jstl/funct

使用JSTL提供的一套EL自定义函数

如果想使用JSTL包括他的标签库，必须要下载4个JSTL的第三方jar包，然后将jar包拷贝到当前项目下的WebContent→WEB—INF→lib文件夹中



### 核心标签库C

#### c标签库介绍

JSTL标签库中比较重要的就是核心标签库c，在使用他之前首先要将他引入进来：

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

他可按照功能分为四类：

通用标签： 操作变量

条件标签： 流程控制

迭代标签： 循环遍历

URL标签： URL操作

#### c标签库---通用标签

<c:set></c:set> set标签内的属性有：

var：变量名

value：修改的值

scope：存储在那个域中

target：指定操作的对象（被操作的对象必须放在域内，然后通过域的name属性操作）

property：对象内的属性（对应的set方法）

<c:out><c/:out> out标签内的属性有：

value：输出的值/变量/表达式结果

default：如果输出的变量不存在时将会输出这个值 ,

escapeXml：是否忽略xml格式标签，默认为忽略true，如不忽略在value中输出HTML代码时会进行解析

<c:remove var=”key” scope=”request”> // 删除request域中name为key的键值对

<c:catch> // 捕获嵌套在标签体中的异常

#### c标签库---条件标签

// c : if标签—相当于Java中的if判断

<c:if test=”el表达式-boolean” var=”存储结果的变量” scope=”存储范围” >

if判断通过后执行的代码

<c:if>

// c : choose标签—相当于Java中的switch分支语句

<c:choose>

<c:when test=”bool”> 满足条件后执行的代码—相当于case </c:when>

<c:when test=”bool”> 满足条件后执行的代码—相当于case </c:when>

<c:otherwise> 当所有条件不满足时所执行的代码吗—相当于default </c:otherwise>

</c:chose>

#### c标签库---迭代标签

普通for循环：

<c:forEach></c:forEach> // 类似Java中的普通for循环，他有以下属性：

var：循环的变量，可理解为 int i；

begin：循环的初始值，可理解为i = 1

end：循环的终止值，可以理解为 i<=10

step：每此循环时的变化，可理解为i++或者i—

其中还有一个特殊的属性：varStatus=“a“，他实际上时一个对象，通过他调用各种属性

a.count：循环的次数

a.index：普通循环代表变量的值，遍历容器代表下表

a.first：判断是否是第一次循环

a.last：判断是否是最后一次循环

a.begin：循环的初始值，遍历容器时为空

a.end：循环的结束值，遍历容器时为空

a.step：每次循环时变量的变化，遍历容器时为空

遍历功能：

// 遍历数组and集合

<c:forEach items="${arr }" var="a"> ${a } </c:forEach> 他有很多属性：

items：需要遍历的那个容器的对象

var：容器中每个元素的对象

scope：指定遍历哪个域中的对象

// 唯一不同的便是Map接口下的集合，它属于双列存储，对与Map集合的遍历有两种方法：

<!--通过keySet方法获取到set集合的方式来遍历map集合-->

<c:forEach items="${set }" var="s" >

${s }${map[s]}

</c:forEach>

<hr />

<!--通过entry对象的方法来遍历map集合-->

<c:forEach items="${map }" var="m" >

${m.key }${m.value }

</c:forEach>

forTokens字符串切割

// 将字符串按照对应的分隔符进行分割

<c:forTokens items="111-222-333-444-555" delims="-" var="a" >

${a }

</c:forTokens>

#### c标签库---其他标签

<c:catch>： 捕获异常

<c:import url=”访问路径”>： 在当前页面引入其他页面，类似于jsp:include

<c:redirect context=”项目名” url =“资源路径“>： 重定向

<c:param name=”” value=”” > 在重定向的时候携带参数

## JavaBean简介

JavaBean是JavaSE中的类它具有以下特点:

1. 这个Java类必须具有一个无参的构造方法
2. 属性必须私有化
3. 私有属性必须对外提供改查功能

// 创建或查找JavaBean

<jsp:useBean id=”” class=”” >

// 设置属性值

<jsp:setProperty property=”” name=”” value=”” useBean的id” >

// 获取属性值

<jsp:getProperty property=”” name=”” value=”” useBean的id” >

## Filter过滤器

Filter过滤器也被称为拦截器，可以在用户访问某个web资源之前，对访问的请求和响应进行拦截，进行一些处理

请求达到指定网页之前处理被称为“预处理”，请求到达后，在响应给用户之前处理，被称为“后处理”

Filter接口下的方法：

init() 服务器加载时运行的方法

doFilter() 收到符合条件的请求时运行的方法

destroy() 当服务器停止运行时运行的方法

FilterConfig接口下的方法：

getFilterName() 获取当前过滤器的名字

getServletContext() 获取ServletContext的实例

getInitParamter(String name) 获取指定name配置信息中的初始化值

getInitParamterNames() 获取到所有name的枚举集合

## Listener监听器

监听器的作用：

监听web容器中有效期事件（增删改查），他可以监听：

1. ServletContext → ServletContextListener

HttpSession → HttpSessionListener

ServletRequest

三个域对象

监听的行为可分为以下三种：

1. 创建和销毁的事件监听
2. 域对象中属性的增删事件监听
3. HttpSession域中的对象状态

其实监听器就是一个Servlet，他实现了监听器的接口后成为了一个监听器的类，在web.xml配置中声明监听器：

<listener>

<listener-class>com.neuedu.listener.MyServletContextListener</listener-class>

</listener>

1. @WebListener通过注解方式声明监听器

@WebListener

## 文件的上传和下载

文件的上传需要使用form表单中的<input type=”file”>

如果form表单中包含上传文件的功能，那么要注意一下几点：

* form表单中的提交方式必须改成post
* form表单中必须要设置enctype=“multipart/form-data”作为文件上传的标识
  + 【application/x-www-form-urlencoded】将value值处理为URL编码方式
  + 【multipart/form-data】以二进制流的方式处理表单数据
  + 【text/plain】：适合通过表单发送邮件
* 处理上传文件的Servlet类需要标识@MultipartConfig

在Servlet3.0之前，实现文件的上传和下载功能需要借助Apache的第三方组件来完成，但是在Servlet3.0之后，通过注解的方式就可以解决

-- 添加注解代码体现：

@MultipartConfig

-- 可选择性常用参数：

maxFileSize： 单个文件最大限制

maxRequestSize： 所有文件最大限制

Location： 设置临时文件存储的路径

fileSizeThreshold：默认值0，如果文件大于这个值，就以文件的形式存储，否则存储在内存中

方法摘要：

request中的方法：

getPart(“String file”); 通过文件上传域中的name属性获取到Part对象

getParts（）； 获取到所有的文件上传域的Part类型的集合对象

Part中的方法：

getSize（）； 获取文件大小（单位kb）

getContentType（）； 获取文件类型

getSubmittedFileName（）； 获取文件名

getName（）； 获取input标签中name属性值

getHeader（String name）； 获取请求头

getInputStream（）； 获取文件输入流

write（String path）； 将文件保存到指定目录下