# response学习

# 学习目标

1.能够使用使用浏览器开发工具查看响应

2.能够理解响应（状态行）的内容

3.能够理解常见的状态码

4.能够使用Response对象操作HTTP响应内容

5.能够处理响应乱码

6.能够完成文件下载案例

7.能够使用servletcontext域对象

8.能够说出servlet生命周期方法执行流程

# 反馈

|  |
| --- |
| 上课的速度稍微有点快, 最后一个案例讲的有点快,跟不上,听的蒙.  讲到主要案例会放慢速度 |
| 谈谈Servlet的生命周期  今天的知识点就有 |
| urlPatterns值设置有什么讲究吗  urlPatterns=”/XXXServlet”,一般在企业中/xxx,不加servlet,因为不想在url暴露技术点  xxx.do/xxx.action,这个老的方式,新的方式都不是,  SSH(会用到xx.do,xxx.action),SSM(/xxx) |
| 数据库忘了  学习方法:一个知识点要复习7次,前两次看着资料复习,后面的主动回想知识点  讲课过程中稍微提一下. |
| BeanUtils封装javaBean对象的时候如果有个表单是多选的，当添加数据进入Mysql里面，mysql里面多选一项只能查到一个值。怎么解决  页面上一个多选框,多个值  Mysql数据库怎么存储,  存一个条记录一个值  存多条记录 |
| 比如@WebServlet(name = "XXX", urlPatterns = "XXXXXX"),加不加name的作用有很大区别吗?  建议加上,因为后面会用到,例如,过滤器可以根据servlet名字或者urlPatterns进行过滤 |
| 上课有用到前面知识的麻烦一起过一下，很多都忘记了  可以有 |
| 笔记代码内容更新的问题,划重点,字体过大  我划的重点,是直接影响后面学习的内容.  我划了解,只需要了解里面的思路就可以了,不需要花太多的在上面. |

# Servlet生命周期

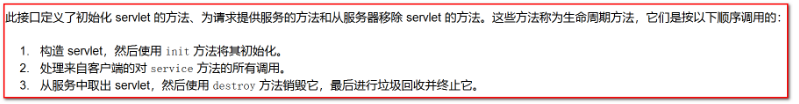
## Servlet生命周期的概述

### 什么是生命周期

​ 一个对象从创建到消亡的过程，就是生命周期。因此，对Servlet生命周期的学习，我们就是研究Servlet什么时候创建，什么时候销毁。

## servlet生命周期相关的方法

​ 首先我们来回顾servlet接口的文档内容，其中一部分如图所示：



### API 介绍

​ 上图中，我们注意到两个点，servlet的创建和销毁由两个相关的方法init方法和destroy方法

1. void destroy() 销毁servlet的方法
2. void init(ServletConfig config)  初始化servlet的方法

我们可以调用测试一下这两个方法：

### 使用步骤

1. 创建LifeCircleServlet初始化
2. 复写init、service、destroy方法
3. 访问servlet测试初始化LifeCircleServlet
4. 关闭服务器测试销毁LifeCircleServlet

### 演示代码

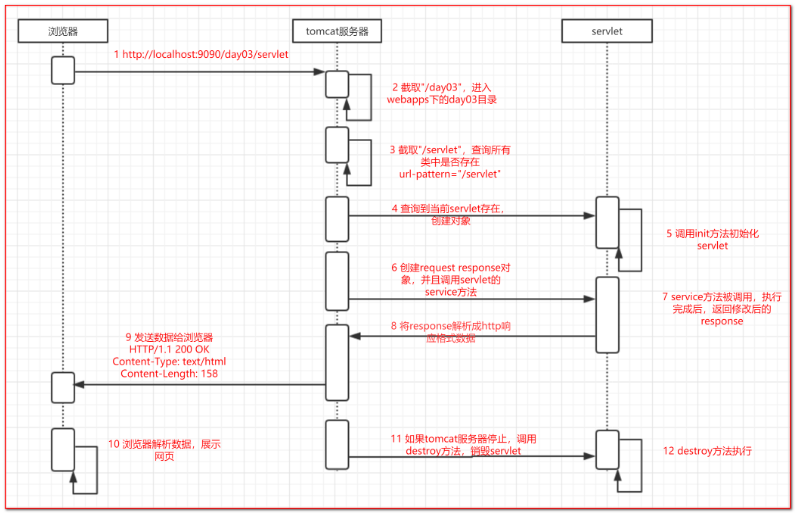
package cn.itcast.web;  
​  
import javax.servlet.ServletException;  
import javax.servlet.ServletRequest;  
import javax.servlet.ServletResponse;  
import javax.servlet.annotation.WebServlet;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
​  
@WebServlet(name = "LifeCircleServlet",urlPatterns = "/life")  
public class LifeCircleServlet extends HttpServlet {  
​  
    @Override  
    public void init() throws ServletException {  
        super.init();  
        System.out.println("LifeCircleServlet初始化。。。");  
   }  
​  
    @Override  
    public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException {  
        System.out.println("LifeCircleServlet执行。。。");  
   }  
​  
    @Override  
    public void destroy() {  
        super.destroy();  
        System.out.println("LifeCircleServlet销毁。。。");  
   }  
}  
​

效果：

访问servlet：  
LifeCircleServlet初始化。。。  
LifeCircleServlet执行。。。  
    
关闭tomcat服务器：  
LifeCircleServlet销毁。。。

## servlet生命周期流程

虽然简单使用过了servlet生命周期相关的方法，但是servlet从创建到销毁的过程对大家来说还是没说清楚，因此，我们以时序图的方式给大家展示了servlet的运行过程，注意图中每一步都由序号，按照序号查看每一个步骤。



附：servlet在初始化一次之后，就不再创建，因此如果多次访问同一个servlet的效果是这样的：

LifeCircleServlet初始化。。。  
LifeCircleServlet执行。。。  
LifeCircleServlet执行。。。  
LifeCircleServlet执行。。。

因此servlet是一个单例对象。​​​

​

# servletContext

## ServletContext的概述

​ **ServletContext是一个容器（域对象）可以存储键值对数据（String key,Object value），保存在ServletContext中的数据不仅可以提供给所有的servlet使用，而且可以在整个项目范围内使用（后期的过滤器、监听器也可以使用ServletContext）。**

**服务器会为每一个工程创建一个ServletContext对象，这个对象就是ServletContext对象。这个对象全局唯一(一个工程一个)，而且工程内部的所有servlet都共享这个对象。所以叫全局应用程序共享对象。**

**服务器启动就创建ServletContext,服务器关闭了才销毁ServletContext**

## ServletContext作用

**ServletContext对象一共两个作用：**

1. **可以读取资源在当前项目中的文件位置**
2. **可以作为域对象在项目全局范围内提供共享数据**

## 作用1: **读取资源在当前项目中的文件位置**

**方法API**

String getServletContext().getRealPath(path),根据相对路径获取项目部署位置上资源的绝对路径

**示例代码**

|  |
| --- |
| **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {    //需求：使用动态资源动态向浏览器输出一张图片    //servletContext可以获取当前项目下的资源文件  //可以根据相对路径获取在服务器上的绝对路径  String realPath = getServletContext().getRealPath("img/2.jpg");  FileInputStream fileInputStream = new FileInputStream(realPath);  **int** length = -1;  **byte**[] bytes = **new** **byte**[1024];  **while**((length = fileInputStream.read(bytes))!=-1){  //将读取到数据的字节数组输出到浏览器  response.getOutputStream().write(bytes, 0, length);  }  //关闭输入流  fileInputStream.close();  } |

## 作用2:ServletContext作为域对象

​ ServletContext中定义了很多方法，在javaweb阶段我最常用的就是将ServletContext作为容器（域对象）使用，因此，接下来我们要学习这个容器（域对象的）API。

### API介绍

GenericServlet：

1. ServletContext getServletContext()   
    获取ServletContext对象

ServletContext：

1. void setAttribute(String name, Object object)  
 往servletcontext容器中存入数据，name为数据名称，object为数据的值  
2. Object getAttribute(String name)  
    从ServletContext中获取数据，根据指定的数据名称  
3. void removeAttribute(String name)  
    从ServletContext中移除数据，根据指定的数据名称

### 使用步骤

1. 创建ServletContextServlet1和ServletContextServlet2
2. ServletContextServlet1调用存方法
3. ServletContextServlet2调用取方法
4. ServletContextServlet2调用删方法
5. ServletContextServlet2调用取方法

### request域与servletContext域的区别

1. request请求域对象,数据有效范围一次请求内,用于转发传递数据
2. servletContext全局域(上下文域)对象,数据有效范围是整个应用程序内,所有资源共享.用于存储全局共享的数据

### 演示代码

ServletContextServlet1：

package cn.itcast.web;  
​  
import javax.servlet.ServletContext;  
import javax.servlet.ServletException;  
import javax.servlet.annotation.WebServlet;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
​  
@WebServlet(name = "ServletContextServlet1" ,urlPatterns = "/context1")  
public class ServletContextServlet1 extends HttpServlet {  
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
        doGet(request, response);  
   }  
​  
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
        //获取容器  
        ServletContext context = getServletContext();  
        //存入数据  
        context.setAttribute("addr","广州");  
   }  
}  
​

ServletContextServlet2：

package cn.itcast.web;  
​  
import javax.servlet.ServletContext;  
import javax.servlet.ServletException;  
import javax.servlet.annotation.WebServlet;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
​  
@WebServlet(name = "ServletContextServlet2" ,urlPatterns = "/context2")  
public class ServletContextServlet2 extends HttpServlet {  
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
        doGet(request, response);  
   }  
​  
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
        //获取容器  
        ServletContext context = getServletContext();  
        //获取数据  
        String addr = (String) context.getAttribute("addr");  
        System.out.println("存入之后，获取数据："+addr);  
        //移除数据  
        context.removeAttribute("addr");  
        //重新获取数据  
        String addr2 = (String) context.getAttribute("addr");  
        System.out.println("移除之后，获取数据："+addr2);  
​  
   }  
}  
​

代码准备好之后，分别访问context1和context2，我们会得到一下结果：

存入之后，获取数据：广州  
移除之后，获取数据：null

通过这个结果，我们发现两次请求虽然访问了不同的servlet，但是依然能通过ServletContext共享数据，而且即使是由一个同学先访问，然后由另一个同学访问也是同样的结果。

总结：保存在ServletContext中的数据是项目全局共享的数据。

# http协议响应数据-response

在前面我们学习过了http协议与请求相关的内容，那么接下来的任务就是要完成响应部分的学习。

## 使用Fillder工具查看响应数据组成

​ 

​ 由上图可知，那么既然分成三个部分，因此，我们的学习也是一步一步来，我们先从响应行（状态行）开始。

## 响应行（状态行）

### 状态行的组成

​ **响应行是http响应内容的第一行。**

​ 响应行一般数据为：HTTP/1.1 200（tomcat8.5） 或者 HTTP/1.1 200 OK（tomcat7）

​ **响应行分为三个部分：**​ ​ **HTTP/1.1：协议版本**

​ **200：响应状态码**

​ **OK：对响应状态码的解释**

### 常见的响应状态码：

1. 200 OK  
   请求已成功，服务器通信正常。
2. 302 Move temporarily 设置重定向页面跳转的动作执行。
3. 304 Not Modified  从浏览器缓存中读取数据，不从服务器重新获取数据。
4. 403 Forbidden  服务器已经理解请求，但是拒绝执行它。由于资源没有执行权限,导致无法执行.
5. 404 Not Found  请求失败，请求所希望得到的资源未被在服务器上发现。一般是用户输错了url导致.
6. 405 Method Not Allowed  请求行中指定的请求方法不存在。例如,发送post请求,服务器没有doPost方法,就会报这个错误.
7. 500 Internal Server Error  服务器发生了错误。一般服务器代码错误

​以上就是我们常用的响应状态码，了解了http协议有关响应行的部分，那么我们接下来要学习，如何使用java对象操作响应。

## 响应头

### 常见的响应头介绍

响应头的数据构成，响应头的数据是响应给浏览器，一般我们不用读取，但是我们可以设置响应头的数据，让浏览器按照我们指定的设置进行执行响应的功能。响应头信息如下：

**Location: http://www.it315.org/index.jsp --跳转方向**

**Server:apache tomcat --服务器型号**

**Content-Encoding: gzip --数据压缩**

**Content-Length: 80 --数据长度**

**Content-Language: zh-cn --语言环境**

**Content-Type: text/html; charset=GB2312 --编码**

**Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT --最后修改时间**

**Refresh: 1;url=http://www.it315.org --定时刷新**

**Content-Disposition: attachment; filename=aaa.zip --下载**

**Set-Cookie:SS=Q0=5Lb\_nQ; path=/search**

**Expires: -1 --缓存**

**Cache-Control: no-cache --缓存**

**Pragma: no-cache --缓存**

**Connection: close/Keep-Alive --连接**

**Date: Tue, 11 Jul 2000 18:23:51 GMT --时间**

1. **refresh:**

**定时刷新跳转页面**

1. **location ：**

重定向操作：通常告知浏览器马上向该地址发送请求，通常和响应码302 一起使用

1. **content-encoding (了解)：**

设置当前数据的压缩格式，告知浏览器以何种压缩格式解压数据

比如像浏览器输出一个压缩文件,可以设置这个响应头通知浏览器解压显示数据,但是目前浏览器只支持”gzip”格式解压

1. **content-disposition ：**

通知浏览器以何种方式获取数据（直接解析数据（网页，图片文本），或者以附件方式（下载文件））

1. **content-type ：**

实体头部用于指示资源的MIME类型（MIME类型：用于提示当前文件的媒体类型，例如图片——（image/png）、音频——（audio/ogg）。它的作用与传统上Windows上的文件扩展名相同。该名称源于 最初用于电子邮件的MIME标准。）

注意：我们content-type常用的设置一般都是——“text/html;charset=utf-8”，其中“text/html;”——设置浏览器以文件格式解析数据；“charset=utf-8”——响应数据的编码表。

​ 响应头虽然学习了，但是简单的介绍肯定记不住，因此，我们准了几个案例，让大家来练习。

### response对象的概念

#### 什么是response对象

我们要学习使用response对象，先要了解它，我们先来看下官方文档截图：



总结：HttpServletResponse对象封装了向客户端发送数据、发送响应头，发送响应状态码的方法。开发人员必须学会使用HttpServletResponse，才能给浏览器发送数据。

#### response常用API

1. void setStatus(int sc) 设置此响应的状态代码
2. void setHeader(String name, String value)  用给定名称和值设置响应头

### 响应头:Location

#### 作用

响应头location的作用是通知浏览器要进行页面跳转的目标地址。

http状态码302的作用是通知浏览器进行页面跳转的动作执行，所以响应头location

和http状态码302配合起来才可以完成页面跳转。

#### 实现跳转代码

//需求：跳转到资源CountServlet

//response.setHeader(name, value); 设置响应头key-value格式

response.setHeader("location", "index.html");

//设置http状态码为302

response.setStatus(302);

#### 优化跳转资源代码(重点)

//一句搞定页面跳转，实现原理就是上面的2句代码

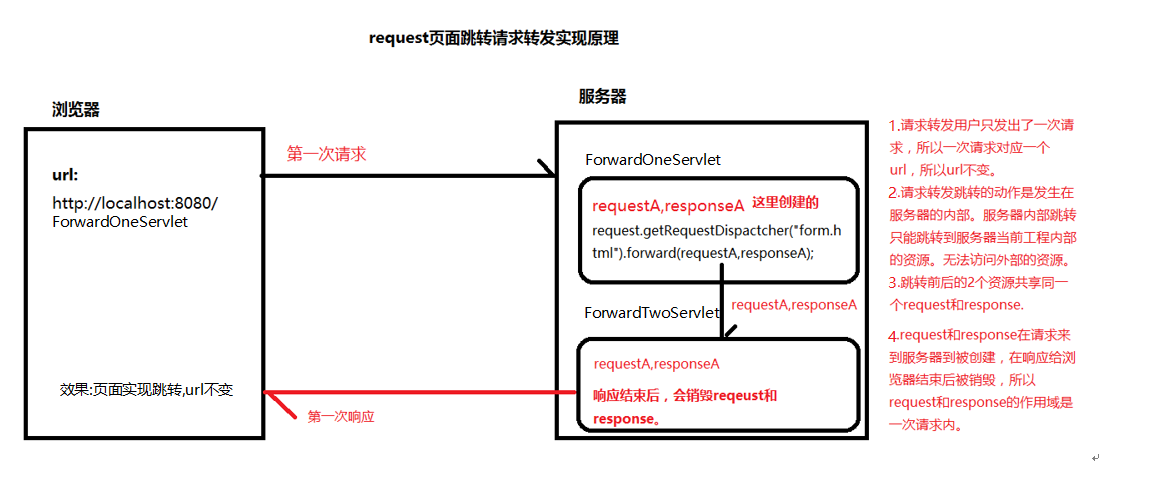
//day37不能写死在代码中，防止修改工程名字，建议使用servletContext获取工程名字

response.sendRedirect(“index.html”);

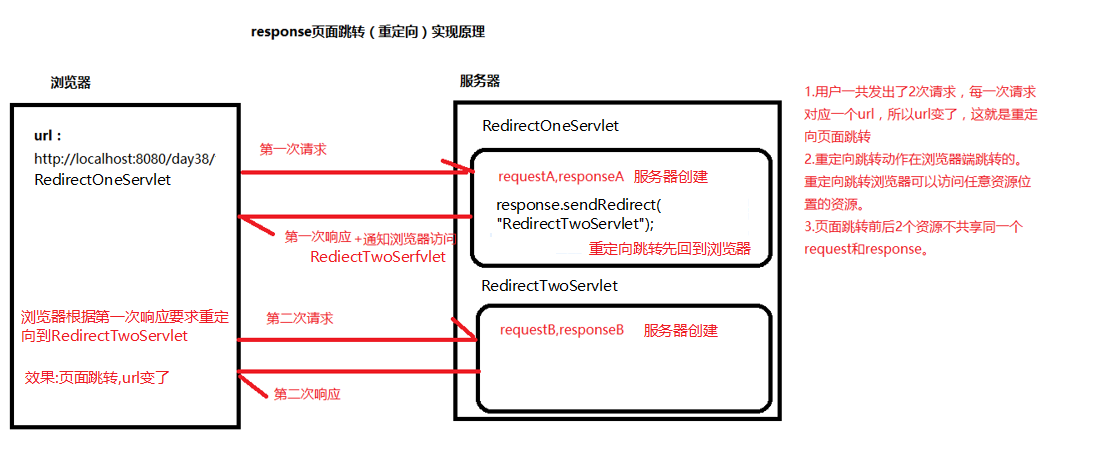
#### 请求转发与请求重定向的区别(重点)

​ 完成重定向操作之后，同学们心里有一个疑问：之前学习的转发和现在的重定向都可以完成跳转，那么他们之间有什么区别呢？

​request页面跳转请求转发实现原理



response页面跳转重定向实现原理



​ 通过上图实现原理，我们可以简单总结几点转发和重定向的区别：

1. 转发在一次请求中完成，重定向是两次请求
2. 转发操作发生在服务器内部，重定向是浏览器执行操作
3. 转发地址栏不变（只有一次请求，一个地址），重定向，地址栏变化（两次请求，两个地址）
4. 转发可以在一次请求中共享数据，重定向不行（重定向两次请求）。

#### 什么时候用请求转发,什么时候用重定向呢?

根据实际情况需要使用

由于request是请求域对象,如果页面跳转在一次请求域内需要传递数据使用请求转发,否则建议使用重定向

### 响应头:Content-Type

#### 格式

**Content-Type: text/html; charset=GB2312 --编码**

#### 介绍

官方叫法，设置**响应正文类型**，报文类型，一共包含2部分内容。

**第一部分text/html**,设置响应的数据类型（**Mime-Type**），这里设置的是响应文本字符串html代码。服务器可以响应任何类型的资源给客户端，资源不同，Mime-Type不同。例如

|  |  |
| --- | --- |
| text/html | Html代码 |
| text/plain | Txt文本文件 |
| image/jpeg | Jpg图片文件 |
| Application/json | Json数据 |

**第二部分charset=GB2312，**响应的字符码表gb2312

通知浏览器以什么码表解码数据

#### 作用

content-type用于服务器通知浏览器采用什么码表对服务器响应的数据进行解码。由于服务器响应数据默认采用iso8859-1码表，然而中国大陆浏览器默认采用GBK码表，所以通过设置响应头content-type来统一码表，解决响应中文数据乱码问题。

#### 服务器端输出中文数据有2种方式

##### 字节流输出

response.getOutputStream().write("传智".getBytes());

##### 字符流输出

|  |
| --- |
| *// 字符打印流,直接输出字符的* PrintWriter out = response.getWriter(); *//out.write(String);输出字符串 //out.print(Object);将object转换为字符串输出* out.print(**"hello 传智"**); |

#### 解决服务器输出字符流中文乱码(重点)

##### 乱码介绍

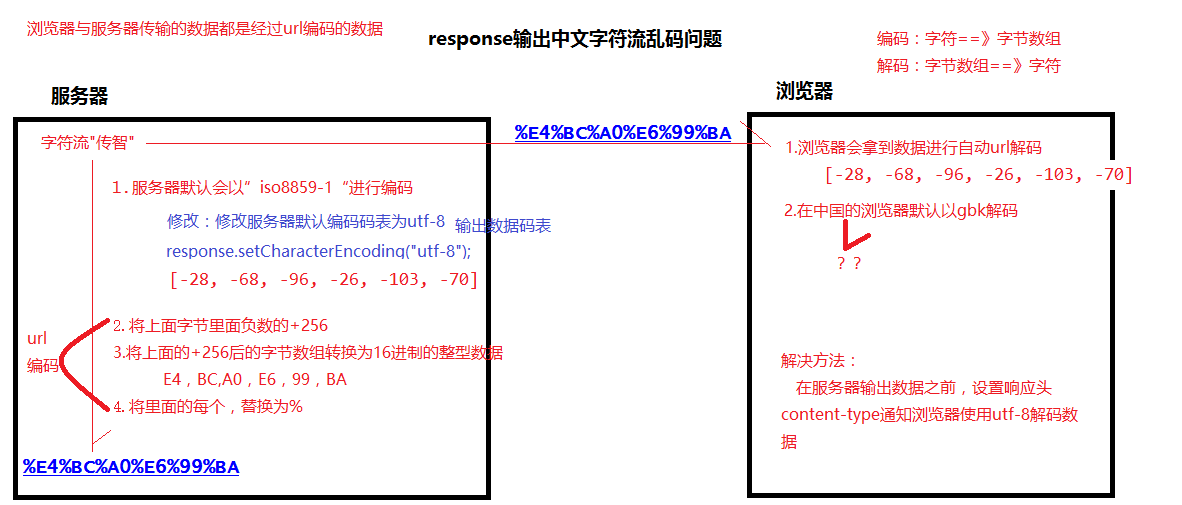
浏览器与服务器传输中文数据会乱码，原因就是服务器响应数据默认采用iso8859-1码表，浏览器默认采用GBK码表，所以就会乱码

##### 解决服务器输出字符流中文乱码问题实现原理

说明：浏览器与服务器传输中文数据需要进行url编码，url编码过程如下：

1. 对字节数组中的负数加256
2. 将加后的字节数组转换为十六进制整型数据
3. 之后在每个16进制数据前加%

例如浏览器与服务器传输“传智”两个汉字，真实传递的是url编码数据“%E4%BC%A0%E6%99%BA”，如下图就是模拟服务器输出“传智”给浏览器最终显示结果“??”乱码了的过程



解决上图输出中文字符流乱码问题需要2个步骤：

1. 服务器默认输出中文编码iso8859-1码表修改为utf-8编码码表
2. 服务器通知浏览器采用utf-8解码码表解析数据

##### 使用响应头content-type解决乱码代码

|  |
| --- |
| **package** com.itheima.response.head;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException; **import** java.io.PrintWriter;  @WebServlet(name = **"ContentTyepServlet"**, urlPatterns = **"/contentType"**) **public class** ContentTyepServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  */\*  \* 响应头Content-tyep  \* 作用  \* 设置响应数据类型  \* 通知浏览器以什么码表解码  \* \*/   //修改response默认输出中文数据码表* response.setCharacterEncoding(**"utf-8"**);  *//通知浏览以utf-8解码* response.setHeader(**"content-type"**,**"text/html;charset=utf-8"**);      *//服务器端输出中文数据有2种方式  // 字节流,response.getOutPutStream();  // 字符打印流,直接输出字符的* PrintWriter out = response.getWriter();  *//out.write(String);输出字符串  //out.print(Object);将object转换为字符串输出* out.print(**"hello 传智"**);  } } |

##### 优化解决乱码代码

|  |
| --- |
| **package** com.itheima.response.head;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException; **import** java.io.PrintWriter;  @WebServlet(name = **"ContentTyepServlet"**, urlPatterns = **"/contentType"**) **public class** ContentTyepServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  */\*  \* 响应头Content-tyep  \* 作用  \* 设置响应数据类型  \* 通知浏览器以什么码表解码  \* \*/   //修改response默认输出中文数据码表 // response.setCharacterEncoding("utf-8"); // //通知浏览以utf-8解码 // response.setHeader("content-type","text/html;charset=utf-8");    //解决乱码不需要这么没法,优化升级,一句搞定,原理就是上面2句代码* response.setContentType(**"text/html;charset=utf8"**);    *//服务器端输出中文数据有2种方式  // 字节流,response.getOutPutStream();  // 字符打印流,直接输出字符的* PrintWriter out = response.getWriter();  *//out.write(String);输出字符串  //out.print(Object);将object转换为字符串输出* out.print(**"hello 传智"**);    *//思考  //为什么字节流输出中文不乱码呢?  //response.getOutputStream().write("传智".getBytes());是一般不会乱码  //"传智".getBytes(charset);在中国无参默认使用gbk进行编码,在中国浏览器默认也是使用GBK解码  //但是部分少数浏览器没有使用gbk  //通过上面分析,得知,使用字节流输出中文数据不安全.* } } |

### 响应头:refresh

#### 格式

**Refresh: 1;url=http://www.it315.org --定时刷新**

描述：1秒以后页面跳转到http://www.it315.org

#### 作用

设置浏览器定时刷新页面或定时跳转到指定的资源

#### 代码

|  |
| --- |
| /\*需求：3秒以后跳转到demo.html\*/  response.setHeader("refresh","3;url="+getServletContext().getContextPath()+"/demo.html"); |

#### 案例-页面倒计时进行跳转到index.html资源

实现步骤：

1. 准备静态倒计时页面wait.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"*>  <title>Insert title here</title>  <script type=*"text/javascript"*>  //定义倒计时数字  **var** count = 3;  //当页面加载完成时调用  window.onload=**function**(){  //倒计时  tick();  }  //启动倒计时  **function** tick(){  //判断倒计时数字大于0继续进行倒计时操作  **if**(count>0){  //获得页面中显示倒计时的标签对象  **var** p = document.getElementById("one");  //页面显示倒计时数字,显示后数字减1  p.innerHTML=count--;  //设置定时在一秒后继续调用当前方法  window.setTimeout(tick, 1000);  }  }  </script>  </head>  <body>  页面在<p id=*"one"*>3</p>秒后跳转...  </body>  </html> |

1. Servlet代码实现定时跳转资源

|  |
| --- |
| //要求2：页面倒计时3秒之后跳转到demo.html  //分析：第一次跳转到倒计时页面，第二次跳转3秒以后跳转到demo.html  response.setHeader("refresh", "3;url="+getServletContext().getContextPath()+"/demo.html");  //直接跳转倒计时页面wait.html（页面跳转必须时候转发）  request.getRequestDispatcher("/wait.html").forward(request, response); |

### 响应头:Content-Disposition

#### 格式

**Content-Disposition: attachment; filename=aaa.zip --下载**

**attachment,通知浏览器不要显示数据要以附件形式下载**

**filename=aaa.zip,下载的文件名字**

#### 作用

通知浏览器不要直接显示数据，以附件形式下载数据。默认浏览器查看数据是直接显示数据，有的资源下载网站需要下载资源数据而不是直接显示，所以需要通过设置响应头content-disposition来通知浏览器以附件下载数据。

#### 示例代码

|  |
| --- |
| **package** com.itheima.response.head;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.FileInputStream; **import** java.io.IOException; **import** java.io.InputStream; **import** java.net.URLEncoder; **import** java.util.Base64;  @WebServlet(name = **"ContentDispositionServlet"**, urlPatterns = **"/contentDisposition"**) **public class** ContentDispositionServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  */\*  \* 响应头:Content-Disposition  \* 作用:通知浏览器不要直接显示数据，以附件形式下载数据。  \* 格式:  \* Content-Disposition: attachment; filename=aaa.zip --下载  attachment,通知浏览器不要显示数据要以附件形式下载  filename=aaa.zip,下载的文件名字  \*  \* \*/   //响应头传递中文,服务器不会进行url编码操作,需要手动进行url编码操作  //手动url编码2种方式  // ie浏览器,直接进行url编码  // 其他浏览器,使用base64进行编码  // 获取浏览器类型，通过请求头中的User-Agent来判断* String fileName = **"传智.jpg"**;   String ua = request.getHeader(**"User-Agent"**);  **boolean** IE\_LT11 = ua.contains(**"MSIE"**); *// IE11以下版本* **boolean** IE11 = ua.contains(**"rv:11.0) like Gecko"**); *// IE11* **boolean** Edge = ua.contains(**"Edge"**); *// win10自带的Edge浏览器  // 如果是微软的浏览器，直接进行UTF-8编码* **if** (IE\_LT11 || IE11 || Edge) {  fileName = URLEncoder.*encode*(fileName, **"UTF-8"**);  *// java的编码方式和浏览器有略微的不同：对于空格，java编码后的结果是加号，  // 而浏览器的编码结果是%20，因此将+替换成%20, 这样浏览器才能正确解析空格* fileName = fileName.replace(**"+"**, **"%20"**);  }  *// 标准浏览器使用Base64编码* **else** {  Base64.Encoder encoder = Base64.*getEncoder*();  fileName = encoder.encodeToString(fileName.getBytes(**"utf-8"**));  *// =?utf-8?B?文件名?= 是告诉浏览器以Base64进行解码* fileName = **"=?utf-8?B?"** + fileName + **"?="**;  }    response.setHeader(**"content-disposition"**,**"attachment;filename="**+fileName);    *//要求:动态向浏览器输出一张图片,要求浏览器不要显示要以附件下载进行保存* String realPath = getServletContext().getRealPath(**"img/4.jpg"**);   InputStream inputStream = **new** FileInputStream(realPath);  **int** length =-1;  **byte**[] bytes = **new byte**[1024];  *//循环读取* **while**((length=inputStream.read(bytes))!=-1){  *//将字节数组写出到浏览器上* response.getOutputStream().write(bytes,0,length);  }  *//关闭输入流* inputStream.close();  } } |

#### 案例：以附件形式下载不同类型文件(了解)

##### 案例需求

​ 完成文件下载功能。

##### 案例效果

​ 用户点击页面的链接，浏览器开始下载文件。

##### 案例分析

1. 创建一个页面，展示所有要被下载文件的链接
2. 链接将要下载的文件名称，发送给服务器的servlet，让servlet进行处理
3. 服务器加载文件资源
4. 提示浏览器，以下载的方式，获取服务器资源
5. 使用IO的方式，将文件数据输出到浏览器（response.getOutputStream();）

##### 代码实现

导入资源到项目内容

|  |
| --- |
|  |

1. html页面

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**>  <**script**>   **function** *isIE*(){  *//获取当前浏览器相关信息* **var** explorer = ***window***.navigator.userAgent.toLowerCase() ;  *//判断是否是ie浏览器* **if** (explorer.indexOf(**"msie"**) >= 0 || explorer.indexOf(**"rv:11.0) like gecko"**) >= 0) {  **return true**;  }**else** {  **false**;  }  }  ***window***.onload = **function** () {  **if**(*isIE*()){  *//在是IE浏览器的情况下，对中文请求参数编码* **var** str = ***document***.getElementById(**"ww"**).**href**;  **var** str = *encodeURI*(str);  ***document***.getElementById(**"ww"**).**href** = str;  }  };  </**script**> </**head**> <**body**> <**a href="download?fileName=1.jpg"**>1.jpg</**a**> <**a href="download?fileName=2.jpg"**>2.jpg</**a**> <**a href="download?fileName=3.txt"**>3.txt</**a**> <**a id="ww" href="download?fileName=传智播客.txt"**>传智播客.txt</**a**> <**a href="download?fileName=heidisql.rar"**>heidisql.rar</**a**>  </**body**> </**html**> |

​

1. servlet演示代码

|  |
| --- |
| **package** com.itheima.servlet;  **import** javax.servlet.ServletContext; **import** javax.servlet.ServletException; **import** javax.servlet.ServletOutputStream; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.File; **import** java.io.FileInputStream; **import** java.io.IOException; **import** java.net.URLEncoder; **import** java.util.Base64;  @WebServlet(name = **"DownloadServlet"**, urlPatterns = **"/download"**) **public class** DownloadServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  *//1 获取要下载的文件名称* String fileName = request.getParameter(**"fileName"**);  System.***out***.println(fileName);  *//2 加载当前文件  // 注意：需要动态的获取当前文件的目录位置（即使服务器所在目录发生变化，我也可以获取到准确位置）  // 我们需要使用servletContext，获取资源路径* ServletContext context = getServletContext();  String realPath = context.getRealPath(**"/download"**);  File file = **new** File(realPath,fileName);   *//3 提示浏览器，以下载的方式，获取服务器资源  //响应消息头设置：  //Content-Type 设置文件媒体格式 getMimeType：1.txt 2.jpg 获取文件的后缀名* response.setContentType(getServletContext().getMimeType(fileName));   *//4 处理中文文件名乱码问题  // 获取浏览器类型，通过请求头中的User-Agent来判断* String ua = request.getHeader(**"User-Agent"**);  **boolean** IE\_LT11 = ua.contains(**"MSIE"**); *// IE11以下版本* **boolean** IE11 = ua.contains(**"rv:11.0) like Gecko"**); *// IE11* **boolean** Edge = ua.contains(**"Edge"**); *// win10自带的Edge浏览器  // 如果是微软的浏览器，直接进行UTF-8编码* **if** (IE\_LT11 || IE11 || Edge) {  fileName = URLEncoder.*encode*(fileName, **"UTF-8"**);  *// java的编码方式和浏览器有略微的不同：对于空格，java编码后的结果是加号，  // 而浏览器的编码结果是%20，因此将+替换成%20, 这样浏览器才能正确解析空格* fileName = fileName.replace(**"+"**, **"%20"**);  }  *// 标准浏览器使用Base64编码* **else** {  Base64.Encoder encoder = Base64.*getEncoder*();  fileName = encoder.encodeToString(fileName.getBytes(**"utf-8"**));  *// =?utf-8?B?文件名?= 是告诉浏览器以Base64进行解码* fileName = **"=?utf-8?B?"** + fileName + **"?="**;  }  *//Content-Disposition 设置要被下载的文件名* response.setHeader(**"Content-Disposition"**, **"attachment;filename="** + fileName);  *//5 将指定文件使用IO技术，向浏览器输出* FileInputStream in = **new** FileInputStream(file);  ServletOutputStream out = response.getOutputStream();   *//6 标准IO代码* **byte**[] buf = **new byte**[1024];  **int** len = -1;  **while**((len = in.read(buf)) != -1) {  out.write(buf, 0, len);  }  in.close();   } } |

## 响应体

### 介绍

就是服务器输出数据给用户看，浏览器直接要显示给用户，就是服务器输出数据

### 作用

1. 输出字符数据、字节数据
2. 输出资源文件数据（资源图片）
3. 输出缓存（内存中）图片（资源没有对应的物理资源）--验证码

### 案例：验证码

#### 案例需求

​ 在页面展示登录验证码

#### 案例效果

​ 用户点击验证码图片可以更换新的验证码



#### 案例分析

1. 创建一个登录页面页面，展示验证码图片
2. 配置今天资料中提供的servlet输出验证码图片
3. 设置页面的点击事件，触发点击事件就重新获取验证码图片

#### 代码实现

1. html页面

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**> <**form action="LoginServlet" method="post"**>  <**table**>  <**tr**>  <**td**>用户名</**td**>  <**td**><**input type="text" name="username"**/></**td**>  </**tr**>  <**tr**>  <**td**>密码</**td**>  <**td**><**input type="password" name="password"**/></**td**>  </**tr**>  <**tr**>  <**td**>验证码：</**td**>  <**td**><**input type="password" name="checkcode"**/></**td**>  </**tr**>  <**tr**>  <**td**></**td**>  <**td**><**img id="img" onclick="***changeCode*(**this**);**" src="CheckCodeServlet"**/></**td**>  </**tr**>  <**tr**>  <**td**></**td**>  <**td**><**input type="submit" value="登录"**/></**td**>  </**tr**>  <**script type="text/javascript"**>  **function** *changeCode*(imgObj) {  *//点击验证码图片重写请求CheckCodeServlet  //浏览器对静态资源img,src值不变的话,浏览器会一直取缓存数据,只有src值改变,才会重新请求服务器获取资源* imgObj.**src**=**"CheckCodeServlet?t="**+**new *Date***().getTime();  }    </**script**>  </**table**> </**form**> </**body**> </**html**> |

​

1. 配置验证码servlet

|  |
| --- |
| **package** com.itheima.response.body;  **import** javax.imageio.ImageIO; **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.awt.\*; **import** java.awt.image.BufferedImage; **import** java.io.IOException; **import** java.util.Random;  @WebServlet(name = **"CheckCodeServlet"**, urlPatterns = **"/CheckCodeServlet"**) **public class** CheckCodeServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }    *//随机类* **private** Random **random** = **new** Random();   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  */\*验证码实现原理  \*  \* 输出一张内存图片,不是资源图片  \* \*/   //BufferedImage,内存图片对象  //new BufferedImage(int width, int height, int imageType);  // width,宽度  // height,高度  // imageType,图片类型,图片模式,RGB* BufferedImage image = **new** BufferedImage(90, 30,BufferedImage.***TYPE\_INT\_RGB***);  *//获取画笔* Graphics g= image.getGraphics();  *//设置画笔颜色* g.setColor(Color.***white***);  *//填充区域* g.fillRect(0,0,90,30);  *//设置边框线颜色* g.setColor(Color.***red***);  *//设置边框线* g.drawRect(0,0,89,29);   *//画干扰线,随机画2条* **for**(**int** i=0;i<2;i++) {  *//每条线改变画笔的颜色* g.setColor(getRandomColor());  **int** x1 = **random**.nextInt(89);  **int** y1 = **random**.nextInt(29);  **int** x2 = **random**.nextInt(89);  **int** y2 = **random**.nextInt(29);  g.drawLine(x1,y1,x2,y2);  }   *//画验证码  //准备验证码范围* String data = **"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890"**;  *//遍历字符串,随机取4个,并且将每个字符画到图片上去  //定义变量接收4个验证的字符串* StringBuilder checkCodeStr = **new** StringBuilder();  **for**(**int** i=0;i<4;i++){  **int** index = **random**.nextInt(data.length());  **char** item = data.charAt(index);  *//拼接* checkCodeStr.append(item);*//后面用,验证码验证  //一个字一个随机颜色* g.setColor(getRandomColor());  *//画到图片上去* g.drawString(item+**""**,10+i\*20,20);  }    *//ImageIO对象将内存图片输出到浏览器* ImageIO.*write*(image,**"png"**,response.getOutputStream());  }   */\*\*  \* 随机颜色  \** ***@return*** *\*/* **private** Color getRandomColor(){  *//Color(int r, int g, int b)* **int** r=**random**.nextInt(256);*//red* **int** g=**random**.nextInt(256);;*//green* **int** b=**random**.nextInt(256);;*//blue* **return new** Color(r,g,b);  } } |

​