# 更新

8月14日 增加码表的内容：二进制，八进制，十六进制

7月21日增加json模板内容，在其它模块更新get乱码问题第二种方式

7月18日更新了其它WEB-INF目录的问题

7月7日request请求板块中更新文件上传

7月4日增加自定义标签

6月30日添加了过滤器Filter内容

6月29日 在Listener中添加了用户登录列表踢人案例

6月28日 在其它中增加了 国际化 操作 、Listener监听器

6月27日 更新了事务、增加了MD5、更新了码表中一些信息

6月26日 增加其它get乱码问题、cookie技术、session技术、事务

6月25日增加了码表信息、response、获取request客户机

6月25日更新了验证码的操作

# Request

获取客户机信息

1获取客户端请求的完整URL

Request.getRequestURL();打印：http://localhost/wwl/a.jsp

2获取客户端URI（资源部分名称） 就是在响应行中:GET /wwl/a.jsp HTTP/1.1

Request.getRequestURI();打印：/wwl/a.jsp

3获取请求行中参数部分

String qstr=Request.getQueryString();打印：name=wenwenliang&age=199

4获取返回发出请求的客户机的IP地址

String ip=Reuqest.getRemoteAddr();//打印：127.0.0.1 192.168.1.220

5获取客户机的请求方式

String method=Request.getMethod();//打印 GET

6获取当前WEB应用的名称

Request.getContextPath();// 打印 /wwl

获得客户机请求头

1、获得所有的请求头，返回enum 注意：请求行是获取不到的GET /TestServer/TestServlet HTTP/1.1

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

Enumeration<String> e=request.getHeaderNames();

while(e.hasMoreElements()) {

String name= e.nextElement();//只能获得所有的请求头

String value=request.getHeader(name);//通过第二个方法：获得所有请求头的值

System.out.println(name+":"+value);//打印：浏览器请求头内容是怎样打印就怎么样

}

2、获取指定名称的请求头的值

Request.getHeader(“Host”); //打印 localhost:8080

3、获取多个相同头的名称的值，返回enum,

Request.getHeaders(“Host”);

## 防盗链

Referer请求头：http://localhost:8080/wwl/b.jsp:如果有这个头证明：本网页是从：b.jsp页面跳转过来的

注意：点击a标签超链接跳转：会有referer请求头信息。其它像服务器跳转不会有此referer头的

public void doGet(HttpServletRequest request, HttpServletResponse responsethrows ServletException, IOException {

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

String ref = request.getHeader("Referer");//第一步 获取到此头内容

if(ref==null || "".equals(ref) || !ref.startsWith("http://localhost")){ //第二步条件判断

response.sendRedirect(request.getContextPath()+"/index.html");

return; //第三步结束此方法

}

response.getWriter().write("凤姐独家回忆..我在黑马的日子..收货了很多很多......");

}

## 文件上传

Servlet3.0技术

Servlet3.0 与 Servlet2.5：

\* Servlet3.0需要运行在tomcat7以上的服务器中.

\* Servlet3.0以后web.xml就不是必须的.

Servlet3.0支持注解开发.

支持文件上传.

文件上传的抓包分析：

已经修改了enctype属性（部分）:

User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko

Content-Type: multipart/form-data; boundary=---------------------------7e02e526160b66

Accept-Encoding: gzip, deflate

Host: localhost:8080

Content-Length: 224

Connection: Keep-Alive

Cache-Control: no-cache

Cookie: JSESSIONID=99CD51DA9A47D29200168968AD983E9E

-----------------------------7e02e526160b66

Content-Disposition: form-data; name="upload"; filename="C:\Users\apple\Desktop\aaa.txt"

Content-Type: text/plain

Hello shouyi

-----------------------------7e02e526160b66—

【文件上传的技术】

JspSmartUpload: jspSmartUpload组件是应用JSP进行[B/S](http://baike.baidu.com/subview/8039/8039.htm)程序开发过程中经常使用的上传下载组件，它使用简单，方便。现在我又为其加上了下载中文名字的文件的支持，真个是如虎添翼，必将赢得更多开发者的青睐。-Model1年代的文件上传的工具.

FileUpload 是 Apache commons下面的一个子项目，用来实现[Java](http://baike.baidu.com/subview/29/12654100.htm)环境下面的文件上传功能，与常见的SmartUpload齐名.应用在Model2年代了.

Servlet3.0 :

Struts2 :

开发步骤 ①制作JSP页面②开发servlet类

JSP页面：

1.form表单的提交的方式必须是POST、 enctype属性的值multipart/form-data

2.文件上传项:<input type=”file”>,而且文件上传项必须有name属性和值.

\* <input type=”file” name=”upload”/>

代码的操作：

@WebServlet("/UploadServlet")

第一步：@MultipartConfig

**public** **class** UploadServlet **extends** HttpServlet {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

// 接收普通项:

request.setCharacterEncoding("UTF-8");

String desc = request.getParameter("desc");

System.***out***.println("文件描述:"+desc);

第二步：通过请求获得Part对象：上传文件的所有内容

Part part = request.getPart("upload");

**long** size = part.getSize();// 获得上传的文件的大小

String type = part.getContentType();

System.***out***.println("文件类型"+type); // text/plain image/jpeg

第三步：获取文件名：Content-Disposition: form-data; name="file"; filename="er.jpg"

String header = part.getHeader("Content-Disposition");

int index=fileName.lastIndexOf("=");

String name=fileName.substring(index+2,fileName.length()-1)

第四步：指定文件要存储的路径

String diskpath=this.getServletContext().getRealPath("/WEB-INF/upLoad");

//UUID配合hashCode创建目录

String rName=UUID.randomUUID().toString()+"\_"+fileName;

String hasName=Integer.toHexString(rName.hashCode());

for(char c :hasName.toCharArray()) {

diskpath+="/"+c;

}

new File(diskpath).mkdirs();

第五步:得到上传文件数据输入流，通过输出流输出到指定位置

InputStream input=part.getInputStream();

OutputStream out= new FileOutputStream(new File(diskpath, rName));

byte[] b= new byte[1024];

int i=0;

while((i=input.read(b)) != -1) {

out.write(b, 0, i);

}

out.close();

input.close(); }

总结:

文件名重名的问题： UUID随机产生一个文件名.

文件上传的目录分离:

按用户分 ：一个用户创建一个或多个路径.

按时间分 ：按月，星期，天进行划分.

按个数分 ：一个路径中存3000个文件.

按分离算法分 ：按照一定的算法进行划分.

# Response

简单入门：向浏览器发送信息

//第一步 告诉服务器要发送的内容编码为utf-8 可以省略，下面第二步底层是有调用此行代码的

Response.setCharacterEncoding(“utf-8”);//注意request也有这个方法：用来得到的请求参数转为utf-8

//第二步 告诉浏览器用utf-8接收

response.setHeader("Content-Type", "text/html;charset=utf-8");//第一种：古老的

Response.setContentType(“text/html;charset=utf-8”);//注意此底层其实是有调用Response.setCharacterEncoding(“utf-8”);的

//第三步 发送信息给浏览器

Response.getWrite().write(‘浏览器你好啊，我是服务器！’);

实现文件下载 响应头 Content-Disposition

//第一步 设置响应头，使用url编码解决中文不显示问题，浏览器只认UTF-8，浏览器会自动解码成utf-8字符

response.setHeader("Content-Disposition", "attachment;filename="+URLEncoder.encode("美女.jpg", "utf-8"));

//第二步 设置输入流（指定服务器中需要下载的文件）和输出流

InputStream in = new FileInputStream(this.getServletContext().getRealPath("1.jpg"));//单1.jpg是在tomcat的bin目录找

OutputStream out = response.getOutputStream();

byte[] bs = new byte[1024];

int i = 0;

while((i=in.read(bs))!=-1){

out.write(bs,0,i);

}

in.close();//输入流需要关闭，输出流是通过响应response拿到的不需要关闭

实现定时刷新页面 响应头refresh

Response.setHeader(“Refresh”,”1”);//每一秒刷新本页面

Response.setHeader(“refresh”,”3;url=”/wwl/.a.jsp”);//3 秒后跳转到阿 a.jsp

在HTML可以利用<meta http-equiv= "" content="">标签模拟响应头的功能

控制浏览器是否缓存 浏览器默认是会话级别的缓存（关闭浏览器就清空）

//不缓存操作

response.setDateHeader("Expires",-1);

response.setHeader("Cache-Control","no-cache");

response.setHeader("Pragma","no-cache");

保存一个月:必须是long型：现在算法已经超出了int的范围会变成负数，当前时间进行加负数就会变成以前时间

response.setDateHeader("Expires", System.currentTimeMillis()+100l\*60\*60\*24\*30);

请求重定向

redirect是服务端根据逻辑,发送一个状态码,告诉浏览器重新去请求那个地址.所以地址栏显示的是新的URL.所以redirect等于客户端向服务器端发出两次不同request对象，同时也接受两次response

Response.setStatus(302);//设置状态302

Response.setHeader(“Location”,”/wwl/a.jsp”);

Response.sendRedirect（”/wwl/a.jsp”）;//底层帮你自动设置响应码，和响应头

## 验证码

Java代码部分：缓存问题要注意，使用万国码找单个字

randNum方法返回int（int 开始,和结束）：第五步画随机直线和第六步的随机字体颜色、随机万国码的字需要用到

//随机生成一个x轴或y轴， 此方法用在第五步直线drawLine参数中

public static int randNum(int begin,int end) {

Random r =new Random();

//begin 到 end之间的值操作方式，Random类没有提供

return r.nextInt(end-begin)+begin;

}

第一步 在内存中构建一张图片

int height=30;

int width=90;

//参数：宽，高，（静态像素老师选择最差的）8位RGB颜色分量的整数像素的图像，

BufferedImage img= new BufferedImage(width, height,BufferedImage.TYPE\_INT\_RGB);

第二步 就像Windows画图工具一样：获取图像上的画布

Graphics2D g = (Graphics2D) img.getGraphics();

第三步 设置此画布的背景色 此步就会在页面形成长方形了

g.setColor(Color.GRAY);//设置颜色

//此方法是填充整个画布的方法：x坐标、y坐标、宽度、高度

g.fillRect(0, 0, width, height);

第四步 （可以省略）设置一个边框

g.setColor(Color.BLUE);

//此方法不填充只是画四周边缘

g.drawRect(0, 0, width-1, height-1);

第五步（可以省略）画干扰线让用户较难辨认，x轴和y轴构建一个点，两点构建一条直线

g.setColor(Color.RED);

//画直线 x轴和y轴决定一个点，两点决定一条直线 (int x1,int y1,int x2,int y2)

g.drawLine(0, 0, width, height);//x坐标：0，y坐标：0固定一个点在画布的起始位置

使用到RandNum方法操作

//g.drawLine(randNum(0, width), randNum(0, height), width, height);

第六步 写字 字符串是万国码unicode格式万国码进行操作

String base ="\u7684\u4e86\u662f\u6211\u4e00";//unicode码,万国码一部分

//for循环获得4个字：因为给你的string是一个字的，

for (int i = 0; i < 4; i++) {

//让颜色随机：Color构造方法参数：红、绿、蓝

g.setColor(new Color(randNum(0, 255), randNum(0, 255), randNum(0, 255)));

g.setFont(new Font("黑体",Font.BOLD,10));//设置字体 构造方法参数：名称、样式、大小

//参数:值、宽度、高度

g.drawString(base.charAt(randNum(0, base.length()-1))+"", 5+(20\*i), 22);

//字体旋转这里就不写了

}

第七步 让浏览器不要进行缓存，不然刷新不了永远都是那一张

//response.setDateHeader("Expires",-1);

//response.setHeader("Cache-Control","no-cache");

//response.setHeader("Pragma","no-cache");

第八步 将图片输出到浏览器、或者指定路径

ImageIO.write(img, "jpg",response.getOutputStream());//输出到浏览器上

ImageIO.write(img, "jpg", new FileOutputStream("D:\\afg.jpg"));//指定位置

}

JSP页面 使用IMG标签 javascript代码中内容不一致了才会有效果

Javascript点击更换验证码操作

<script type="text/javascript">

function changeImg(img){ //需要加上时间确保内容不一样，和ajax效果一样

img.src="/Day04/servlet/ValiImg?time="+new Date().getTime();

}

</script>

</head>

<body>

<form action="#" method="POST">

用户名:<input type="text" name="username"/>

密码:<input type="password" name="password"/>

验证码:<input type="text" name="valistr"/>

<img src="/Day04/servlet/ValiImg" style="cursor: pointer;" onclick="changeImg(this)"/>

<input type="submit" value="注册" />

</form>

</body>

# Cookie

简单入门 响应头：Set-Cookie 请求头 Cookie

一 浏览器第一次访问到此代码response.setHeader("Set-Cookie", "wenwenliang=eee");

二 打开浏览器的请求信息和响应信息：响应信息会有Set-Cookie：wenwenliang= eee，请求头无cookie信息

三 再次请求请求头显示：Cookie: wenwenliang ; hibext\_instdsigdipv2=1，如访问servlet有cookie响应头还会有Set-Cookie

注意 不管有没有设置(response.setHeader("Set-Cookie"))，访问jsp页面：

响应头：访问jsp页面会有Set-Cookie名称 JSESSOONID响应浏览器：jsp默认发送session，可以改为false

请求头：请求收到信息：第一次不会有cookie信息

使用Cookie对象操作 (javax.servlet.http.Cookie)

第一步 创建对象，参数：第一个是Cookie名字，第二个是值：用中文使用URL编码

Cookie c = new Cookie("wenwenliang",URLEncoder.encode("我是谁", "utf-8"));

第二步 使用response发送Cookie信息 参数是cookie对象

response.addCookie(c);

第三步 获取Cookie信息

第一种 获取请求头，然后可以通过字符串截取第一个”=”

System.out.println(request.getHeader("Cookie")); //打印wenwenliang=%E6%88%91%E6%98%AF%E8%B0%81;

第二种 sun公司封装好的request获取Cookie数组

Cookie[] cookie=request.getCookies();

for(Cookie c : cookie) {

if("wenwenliang".equals(c.getName())) {

System.out.println(URLDecoder.decode(c.getValue(), "utf-8"));//URL解码 ：打印：我是谁

}

}

使用Cookie技术：显示上次用户访问时间

第一步 把cookie值使用当前系统时间的毫秒值字符串进行存储

Cookie c = new Cookie("wenwenliang",new Date().getTime()+"");

response.addCookie(c);

第二步 在另一个servlet程序中获取这个cookie信息，进行判断获取此毫秒值进行转换为时间

Cookie[] cookie=request.getCookies();

Cookie cookie2=null; //用来接收wenwenliang的cookie信息

if(cookie !=null) {

for(Cookie c : cookie) {

if("wenwenliang".equals(c.getName())) {

cookie2 =c;

}

}

}

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

if(cookie2 ==null) { //判断是否为null

response.getWriter().write("你是第一次访问本网站");

}else {

response.getWriter().write("你上次的访问时间是："+

new Date(Long.parseLong(cookie2.getValue())).toLocaleString());

}

第一：保存Cookie信息一个月；第二:浏览器访问服务器带着cookie信息的条件

Cookie c = new Cookie("wenwenliang","wwl");

c.setMaxAge(3600\*24\*30);//设置保存一个月

response.addCookie(c); //浏览器响应头：Set-Cookie:wenwenliang=wwl;Expires=Thu,26-Jul-2018 03:08:06 GMT ；请求头无变化

注意 ：没有设置路径都是在发送 cookie的所在的路径或子路径Url才会带着Cookie过来

如 http://localhost:8080/Test/wwl/Test此网页是发送cookie信息的路径：wwl/以后的才会带着cookie

第一：http://localhost:8080/Test/wwl这个路径是在wwl下，会带着请求cookie信息过来

而http://localhost:8080/Test/t这个路径不在wwl下，不会带着请求cookie信息过来

设置为web服务器url根路径：就可以全部都会带cookie信息

Cookie c = new Cookie("wenwenliang","wwl");

c.setMaxAge(3600\*24\*30); //设置保存一个月

c.setPath(request.getContextPath());//设置Url根路经，整个web应用都会带着cookie信息过来

response.addCookie(c);//浏览器响应头：Set-Cookie:wenwenliang=wwl;Expires=Thu,26-Jul-2018 03:08:06 GMT ；Path=/Test

设置Domain现代浏览器不会接收：参数用来放域名：www.baidu.com

删除cookie：需要名字、path、Domain(一般都不用了)、3样都要一致

Cookie c = new Cookie("wenwenliang","wwl");

c.setMaxAge(0); 设置为0；覆盖掉之前的cookie

c.setPath(request.getContextPath());//设置Url根路经，整个web应用都会带着cookie信息过来

response.addCookie(c);

# Session

简单入门 浏览器不会主动创建session

第一步 当浏览器第一次访问到有Request.getSeeion();代码//浏览器响应头会有Set-cookie：JSESSIONID和独一无二的值

Set –Cookie：JSESSIONID=FF95C76911632721BD4BF886B233F0C1;Path=/Test/;HttpOnly

第二步 当第二次访问到有Request.getSeeion();代码，浏览器响应头不会再有Set-Cookie

因为服务器是先判断请求头有无JSESSIONID名称的Cookie，有：不再此创建session，无：进行创建

保存Session信息一个月，原理和保存Cookie信息一个月一样的

因为session就是基于cookie技术的：它的名字是JSESSIONID，需要同名、同path才能进行设置

HttpSession session=request.getSession(); 第一步 创建session对象

session.setAttribute("A", "B");

Cookie c = new Cookie("JSESSIONID",session.getId()); 第二步 name要和session名字一样，它的值通过getId()获得

c.setMaxAge(60\*60\*30\*24); 第三步 设置时间为30天

c.setPath(request.getContextPath()); 第四步 设置路径上下文路径，因为session就是上下文路径

response.addCookie(c); 第五步 发送cookie信息

响应头变成带时间：Set-Cookie: JSESSIONID=FF95C76911632721BD4BF886B233F0C1;26-Jul-2018 03:08:06 GMT ;Pth=/Test

Session\_URL重写：被设置成浏览器拒绝接受Cookie（了解即可）

对Url路径进行改造：

Request.getSession(); 先有此代码 此方法支持URL重写带session

String url1 = request.getContextPath()+”/wwl/t?name=电视”;

Response.encodeURL(url1); //把路径放入此方法中即可，了解即可，很麻烦，要把全部URL进行改造

<a href=”/Test/Test/t;jsessionid=B81EB72CF092436462K?name=电视”>电视机</a> 看源码会发现拼接了session信息

session的钝化和活化 条件：①需要tamcat正常关闭②有未过时session对象

需要①正常关闭服务器②session还没过时：称为钝化

D:\apache-tomcat-7.0.78\work\Catalina\localhost\Test目录下有文件SESSIONS.ser

服务器开启此工程如有此文件SESSIONS.ser会把它加载到内存中 同时此文件消失

注意 对象的类（实体类）如果要被钝化必须系列化：实体类需实现Serializable

# Filter

简单入门

第一步 创建类实现接口Filter类

@WebFilter("/\*") //注解直接在这里写/\*拦截URL工程下得所有路径

public class TestFilter implements Filter{

public void destroy() {}

public void doFilter(ServletRequest arg0, ServletResponse arg1, FilterChain arg2)throws IOException, ServletException {

System.out.println("被我拦截了");

arg2.doFilter(arg0, arg1);//经测试没有此句话进来都是空白的网页，不显示内容也不报错，控制台打印”被我拦截了”

}

public void init(FilterConfig arg0) throws ServletException {

System.out.println("我是过滤器");}

第二步 web.xml配置 一个Filter可以多个filter-mapping、多个url-partten

<filter-mapping>

<filter-name>Demo1</filter-name> //跟Fileter标签相同的名字

**第一种** 一个Filtermapping中可以配置多个url-partten,这个url-partten的写法和servlet-mapping中的写法相同

<url-pattern>/servlet/A</url-pattern> //指定单独 A类

<url-pattern>/servlet/\*</url-pattern> //指定servlet/路径下得所有类

<url-pattern>/\*</url-pattern> //工程下得所有

<url-pattern>\*.do</url-pattern> //后缀是do的类

**第二种**  配置多个servlet-name,其中填入servlet的名字明确的通知要拦截哪个名字的Servlet

<servlet-name>Demo3Servlet</servlet-name>

**第三种** 指定拦截REQUEST、FORWARD、INCLUDE、ERROR四个值,可以配置多个dispatcher,如果不配默认是REQUEST

<dispatcher>REQUEST</dispatcher>

<dispatcher>FORWARD</dispatcher>

</filter-mapping>

测试 结果：过滤器只认URL路径 只要URL符合要求，不管路径是否正确都会拦截，

第一种 /\* 代表工程名下URL路径全部拦截，只认URL路径，不管此路径对不对；如工程Test：

http://localhost:8080/Test/会拦截，不管有没有资源都会执行doFilter方法

http://localhost:8080/login.jsp 不会拦截，不是Test路径下的文件

第二种 测试默认配置：转发不会拦截，因为默认只是request请求的拦截（重定向也会拦截）

request.getRequestDispatcher("/login.jsp").forward(request, response);//打印一次：我被调用了，（注意请求servlet需要一次）

response.sendRedirect("/Test/login.jsp");//打印两次：我被调用了

第三种 arg2.doFilter(arg0, arg1);方法等于就是进行调用另一个方法，另一个方法执行完，再来执行后续的代码

测试：两个拦截器执行arg2.doFilter(arg0, arg1)前后打印A、B和C、D；最后访问的资源：打印”资源”

public void doFilter(ServletRequest arg0, ServletResponse arg1, FilterChain arg2) throws IOException, ServletException {

System.out.println("A");

arg2.doFilter(arg0, arg1);

System.out.println("B");

}//两个Filter一样的操作另一个是：C、D进行打印

执行方法后打印顺序是：A、C、资源、D、B // 等同于方法调用了另一个方法，另一个方法之行完了回来再执行下面代码

## 解决乱码

第一步 使用装饰模式：创建类继承HttpServletRequestWrapper，（测试了动态代理也可以操作）

注意getParameterMap第一次获取就缓存在内存中，后续就从缓存中取解决过乱码的内容

public class A extends HttpServletRequestWrapper{

boolean flag=true; // 设置此成员布尔值是为了只进行一次解决乱码 的代码执行

public A(HttpServletRequest request) { super(request); } //继承的(父类没有无参构造)

public Map<String, String[]> getParameterMap() {

HttpServletRequest request=(HttpServletRequest)getRequest();

if(request.getMethod().equals("POST")) {

request.setCharacterEncoding("utf-8");

return super.getParameterMap();

}else if(request.getMethod().equals("GET")) {

Map<String, String[]> v=request.getParameterMap();

//只执行一次解决乱码,因为此方法有缓存机制,后续导致获取相同参数会把正确解码内容给解码成乱码

if(flag) { Collection<String[]> l=v.values();

for(String[] e:l) {

//增强for循环是不能修改的，只能进行赋值后进行重新保存；

for(int i=0;i<e.length;i++) {

e[i]= new String(e[i].getBytes("iso8859-1"),"utf-8");}}

flag=false;//设置为false，不再进行第二次

return v; }}

return super.getParameterMap();}

public String getParameter(String name) {return getParameterValues(name)[0]; }//调Values的方法

public String[] getParameterValues(String name) {return getParameterMap().get(name); }//调解决乱码的Map方法

第二步 Filter类中的调用 测试过直接动态代理proxy，可以操作

public void doFilter(ServletRequest arg0, ServletResponse arg1, FilterChain arg2) throws IOException, ServletException {

response.setContentType("text/html;charset=utf-8"); //--解决响应乱码

arg2.doFilter(new A((HttpServletRequest)arg0), arg1); //使用A类的request对象，

}

//此初始方法可以获取web中的init配置文件：utf-8可以不用写死，可以通过此方法获取：A类弄个两参数的构造方法传值

public void init(FilterConfig arg0) throws ServletException {}

# Listener

简单入门 8大监听器接口类

**第一步** 创建类实现监听器接口类；**第二步** ①web.xml进行注册或 ②注解形式@WebListener

监听三大作用域的创建和销毁监听器：3个监听器接口

ServletContextListener 监听WEB工程的启动和关闭销毁

@WebListener //注解

public class TestListen implements ServletContextListener{

public void contextDestroyed(ServletContextEvent arg0) {

System.out.println("销毁了"+arg0.getServletContext());

}

public void contextInitialized(ServletContextEvent arg0) {

System.out.println("创建了"+arg0.getServletContext());

}

当启动web工程加载完成创建了当前SetvletContext对象后立即调用了初始化方法

六月 28, 2018 10:16:26 上午 org.apache.catalina.startup.TaglibUriRule body

信息: TLD skipped. URI: http://java.sun.com/jsp/jstl/xml is already defined

创建了org.apache.catalina.core.ApplicationContextFacade@604da796

六月 28, 2018 10:16:26 上午 org.apache.catalina.core.StandardContext reload

信息: Reloading Context with name [/Test] is completed

HttpSessionListener 三种死法①默认30分钟未使用②调用invalidate()③未正常关闭工程

第一步 第一次触发了request.getSession();第二次是不会再触发下面方法，因为已经有session对象了，不会再创建

public class TestListen implements ServletContextListenerrequest{

public void sessionCreated(HttpSessionEvent arg0) {

System.out.println("创建了session对象");

}

第二步 ①默认30分钟未使用②调用invalidate()③未正常关闭工程 都会触发下面的方法

public void sessionDestroyed(HttpSessionEvent arg0) {

System.out.println("销毁了session对象");

}

ServletRequestListener 监听ServletRequest创建和销毁

注意 请求的创建和销毁之间是很快的：一打开servlet类就直接两个都打印出到控制台了

public void requestDestroyed(ServletRequestEvent arg0) {

System.out.println("销毁了request对象");

}

public void requestInitialized(ServletRequestEvent arg0) {

System.out.println("创建了request对象");

} 控制台同时打印了上面两句话

监听三大作用域属性：和上面的几乎一致 3个监听器接口

**1、**ServletContextAttributeListener**2、**HttpSessionAttributeListener、**3、**ServletRequestAttributeListener

它们3个都有三个方法：**加入、更改、移除**，这三大监听器就不写了

Javabean(实体类)感知自己在Session中状态变化的最后两个监听器

触发条件 ①被绑定到session②被移除session③随着session被钝化④随着session被活化

这两个监听器不用在web.xml注册，只需实体类实现这两个接口即可

绑定和移除session接口监听器：HttpSessionBindingListener

活化和钝化session接口接口监听器：HttpSessionActivationListener

public class T implements HttpSessionBindingListener,HttpSessionActivationListener,Serializable{

private Integer id;

public Integer getId() {return id; }

public void setId(Integer id) {this.id = id; }

public void valueBound(HttpSessionBindingEvent arg0) {

System.out.println("我被绑定到session了");

}

public void valueUnbound(HttpSessionBindingEvent arg0) {

System.out.println("我被移除了session");

}

public void sessionDidActivate(HttpSessionEvent arg0) {

System.out.println("我活化了");

}

public void sessionWillPassivate(HttpSessionEvent arg0) {

System.out.println("我钝化了");

}

用户登录列表踢人案例 注意 ①hashMap存储机制②使用了两个监听器对象

第一步 自定义类实现ServletContextListener 监听器进行域属性添加：msp<String, HttpSession>集合

….省略了一些代码；

public void contextInitialized(ServletContextEvent sce) {

ServletContext context= sce.getServletContext(); //获得ServletContext对象

context.setAttribute("usermap", new LinkedHashMap<Integer,HttpSession>());//把空Map存进application属性域中

第二步 登录操作：通过数据库查询出来的对象存进session后再存储到第一步map中，实体类有绑定监听器看第三步

//这里需要注意如果相同用户登录：需要顶掉登录相同用户

ServletContext context=this.getServletContext(); **//获得servletContext对象**

Map<String,HttpSession> map=(Map<String, HttpSession>) context.getAttribute("usermap"); **//获取存到域中的空map**

HttpSession session=map.get(u.getName());  **//这步是查询map中有没有这个值，如有：进行移除操作**

if(session != null) {

session.invalidate(); **//一旦移除会触发另一个监听器 HttpSessionBindingListener**

}

request.getSession().setAttribute("user", u); //一旦添加实体类触发：实体类绑定监听器中的方法看第三步

response.sendRedirect(request.getContextPath()+"/b.jsp");

第三步 实体类中实现了HttpSessionBindingListener监听器：一旦移除、绑定session触发事件

用户绑定到属性session中：request.getSession().setAttribute("user", u)触发此方法

public void valueBound(HttpSessionBindingEvent event) {

HttpSession session=event.getSession(); //第一步 获取session对象

ServletContext servlet=session.getServletContext(); //第二步 通过session获取ServletContext对象

Map<String,HttpSession> map=(Map<String, HttpSession>) servlet.getAttribute("usermap"); //第三步 获取第一步中map对象

map.put(this.getName(), session); //第四步 存入session对象：当前用户直接this.名字

}

用户从session中移除：session.invalidate()此代码会触发此方法、超时下线

public void valueUnbound(HttpSessionBindingEvent event) {

HttpSession session=event.getSession();

ServletContext servlet=session.getServletContext();

Map<String,HttpSession> map=(Map<String, HttpSession>) servlet.getAttribute("usermap");

map.remove(this.getName()); //移除当前对象的名字

}

第四步 踢人操作 页面获取到name值，获取此name的HttpSession值，再进行杀死session，触发销毁监听方法

String s=request.getParameter("name");

byte[] b1=s.getBytes("iso8859-1");

String name =new String(b1, "utf-8");

Map<Integer,HttpSession> map=(Map<Integer, HttpSession>) request.getServletContext().getAttribute("usermap");

HttpSession session=map.get(name);

if(session != null) {

session.invalidate();

}

response.sendRedirect("userList.jsp");

# 事务

四大特性 ACID

**1、原子性(Atomicity)** 事务是一个不可分割的工作单位，多条语句操作数据库要么都成功，要么都失败

**2、一致性(Consistency)** 事务前后数据的完整性必须保持一致，如： 不要有员工却没有部门表

**3、持久性(Durability)** 一个事务一旦提交，数据库中的数据改变就是永久性，哪怕数据库故障也不受影响

**4、隔离性(Isolation)：**当多个用户并发访问数据库操作同一张表时，为每一个用户开启事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。

隔离性(Isolation) 需要自己操心的，当并发操作数据库有下面3个情况：

1、两个线程并发修改或增加: 数据库都会加锁(排他锁)，等一个修改完，另一个才能进行修改，没有线程安全问题

2、两个线程并发查询:没有线程安全问题(数据都不变)

3、两个线程一个修改或插入，一个查询：就有可能会导致下面3个问题发生

**脏读：**一个事务读到另一个事务未提交的事务

B事务进行修改或提交，还未commit(提交事务)，A事务就能查询到B事务未commit的数据

**不可重复读：**在一个事务内读取表中的某一行数据，多次读取结果不同**(线程并发一个修改一个查询)**

A事务先查询数据是2 ,并发中B事务把A查询的数据2修改为3进行了commit;A还没有commit：再次查询变成3：这就是不可重复读

**虚读(幻读)：**在一个事务内查询到了别的事务插入的数据，导致前后读取不一致**(线程并发一个插入一个查询)**

A事务查询到数据只有一个2，并发中B事务插入了一条3进行了commit;A还没有commit：再次查询变成了2,3的数据，这就是虚读(幻读)

解决方案：脏读、虚读、幻读的问题数据库提供了四大隔离级别

Read ybcimmutted 不防止任何隔离性问题。

Read committed 可以防止脏读问题，但是不能防止不可重复读/虚读(幻读)的问题

Repeatable read 可以防止脏读、不可重复读问题，不能防止虚读(幻读)问题

A事务查询到数据是2 ,B事务进行修改跟A一样的数据为3进行了commit，虽然是真的修改了数据库内容，但A还没有commit：再次查询还是2，这就是**解决了不可重复读问题**

Serializable 数据库被设计为单线程数据库，可以防止上述所有问题，但是效率最低

A事务查询到数据的期间，B事务要进行插入或修改操作，都必须等A事务commit结束后才能操作

数据库中的锁机制：

**共享锁：**两个条件：①Serializable隔离级别下、② 查询操作，才会加共享锁

特点：共享锁和共享锁可以共存(两个查询可以共同操作)，但是共享锁和排他锁不能共存；

**排他锁：**两个条件：①所有的隔离级别、②增删改操作 。都会加排它锁：只要有一个线程进行了增删改操作，其它线程就没法进行增删改

特点：和任意的锁都不能共存

# 自定义标签

第一步 自定义类继承javax.servlet.jsp.tagext.SimpleTagSupport

实现标签体内<a>这里内容循环</a>根据需求次数进行循环操作步骤

public class TestTag extends SimpleTagSupport{ 第一步 继承SimpleTagSupport类实现doTag()方法

**private Integer cc;** 第二步 设置属性成员变量，需set进行封装

**public void setCc(Integer cc) {this.cc = cc; }**//属性需要封装set方法

**public void doTag() throws JspException, IOException {**

**for(int i=0;i<cc;i++)** 第三步 在方法内进行for循环即可

**getJspBody().invoke(null); //此方法执行方法体内的内容显示到页面**

doTag方法里要获取标签体内的内容需要配合StringWriter（字符串流）

让结束标签以后jsp所有内容不输出：throw new SkipPageException();

第二步 写tld文件引用第一步的类，必须在WEB-INF (不能在lib和classes文件夹)下

<?xml version="1.0" encoding="UTF-8"?>

<taglib version="2.0" xmlns="http://java.sun.com/xml/ns/j2ee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary\_2\_0.xsd">

<tlib-version>1.0</tlib-version>

<short-name>A</short-name> <!--第一步 名称空间缩写相当于引入C标签的C-->

<uri>http://www.wwl.com/AA</uri> <!--第二步 名称空间相当于映入c标签的uri -->

<tag>

<name>D</name> <!--第三步 jsp页面调用名字 <A:D/>-->

<tag-class>com.wwl.tag.TestTag</tag-class> <!--第四步 引入类全路径 -->

<body-content>empty</body-content> <!--第五步 empty单标签 scriptless双标签tagdependent后台使用如<sql> -->

<attribute> <!--第六步 类中属性定义 -->

<name>cc</name> <!-- 属性名字 -->

<required>true</required> <!-- 属性是否必须：true必须-->

<rtexprvalue>true</rtexprvalue> <!-- 是否支持el表达式 -->

<type>java.lang.Integer</type> <!-- 类型 -->

</attribute>

</tag>

</taglib>

第三步 调用 引入标签库

引入标签库：<%@taglib uri="http://www.wwl.com/AA" prefix="A" %>

使用 <A:D cc="5">呵</A:D> 页面显示：呵呵呵呵呵

第四步 打包 让别人用我们开发好的标签库

建议建一个新java工程包(没有乱七八糟的东西)注意一定要没有错误情况下打jar包

把tld文件：放到META-INF的文件夹下(没有自己创建名字一定要对)。Class类：还是按原来的全路径放

Eclipse打包：右键工程名Expor-> jar file -> 把工程中都勾上(browse按钮选择目标位置)然后就全部默认

# JSON

格式 大括号 {}代表一组：如xml的<a></a>双标签，[]代表数组：组中相同名称进行使用

注意：{}一组中不能有相同的key的值，如有相同的key键使用数组[]

{"p": {"国家": "中国","省份": {"广东": {"市": ["梅州", "深圳", "广州"]},"name": "江西"}}}

上面的json格式和下面xml格式一致：

<p>

<国家>中国</国家>

<省份>

<广东>

<市>梅州</市>

<市>广州</市>

<市>深圳</市>

</广东>

<name>江西</name>

</省份>

</p>

JSON操作

目前最新的是2.4的版本：json-lib-2.4-jdk15.jar ，还需要以下依赖包：

jakarta commons-lang 2.5，jakarta commons-beanutils 1.8.0，jakarta commons-collections 3.2.1

jakarta commons-logging 1.1.1，ezmorph 1.0.6

第一步 创建对象 添加数据

JSONObject j = new JSONObject(); //大括号 {} 的对象

j.put("1", "第一");

j.put("2", "第二");

j.put("3", "第三");

JSONArray ja = new JSONArray();//中括号 [] 的对象

ja.add("第一");

ja.add("第二");

ja.add("第三");

System.out.println("JSONObject格式是："+j);

//打印：JSONObject格式是：{"1":"第一","2":"第二","3":"第三"}

System.out.println("JSONArray格式是："+ja);

//打印：JSONArray格式是：["第一","第二","第三"]

j.put("8", ja); //把中括号对象put为值

System.out.println("JSONObject格式是："+j);

//打印：JSONObject格式是：{"1":"第一","2":"第二","3":"第三","8":["第一","第二","第三"]}

//使用类自带的方法来转换成String类型

String str\_ = j.toString();

String strs\_ = ja.toString();

打印给浏览器

第一步 代码

@RequestMapping(value = "/a ", method = RequestMethod.GET,)

public static void getLYHPartLinkDoc(HttpServletRequest requset,HttpServletResponse response) throws Exception {

SONObject json= new JSONObject();

json.put("1", "第一");

json.put("2", "第二");

json.put("3", "第三");

response.setCharacterEncoding("utf-8");

response.setContentType("application/json;charset=utf-8");

response.getWriter().write(json.toString());

}

response.setCharacterEncoding("utf-8");

response.setContentType("application/json;charset=utf-8");

response.getWriter().write(json.toString());

第二步 浏览器地址栏访问：localhost:8080?/wwl/a/lyhPartId=162143

如果是空的页面：ie会提示你进行保存xxx.json, 火狐浏览器会直接页面显示：

{"1":"第一","2":"第二","3":"第三"}

一般都是通过js进行获取打过来的数据进行接收

# 其它

1、get的参数乱码解决

第一种方式

Request.setCharacterEncoding(“utf-8”) 通知服务器以什么编码解码 —>只对post有效，下面对get和post都有效

String u = request.getParameter(“userName”); 第一步

// 因浏览器默认是iso8859-1请求过来的，所以先转换iso8859-1二进制字节码，再通过utf-8转换

byte[] b=u.getBytes("iso8859-1"); //第二步 把获得的值用iso8859-1转换成二进制的字节

String name = new String(b, "utf-8"); //第三步 使用Strin构造方法把二进制字节数组用utf-8转换为字符串

第二种方式 修改tomcat（默认编码方式为iso8859-1）编码

第一步 找到tomcat下的conf/server.xml文件，（eclipse的servlets工程目录下也可以找到）

第二步添加一个属性：URIEncoding

<Connector port="8080" protocol="HTTP/1.1"connectionTimeout="20000" redirectPort="8443" URIEncoding="系统编码" />

1. 访问jsp页面默认会发送session的Cookie信息，可以更改

默认

第一次访问jsp页面：响应头会有：Set –Cookie：JSESSIONID=FF95C76911632721BD4BF886B233F0C1;Path=/Test/;HttpOnly

更改

<%@ page language="java" contentType="text/html; charset=utf-8"

pageEncoding="utf-8" session="false"%>改为 false 那么就不会有Set-Cookie的session信息了

1. 国际化 internationalization 开头字母i + 加中间18字母+尾数n 简称：i18n

ResourceBundle在查找资源文件时,首先找指定的,如果找不到找操作系统语言环境的,如果还找不到用默认的 必须在web环境下操作

第一步创建3个资源文件properties：开头统一：resource\_zh.properties、resource\_en.properties、resource.properties

resource.properties 内容：username=default username

resource\_en.properties 内容：username=username

resource\_zh.properties 内容：username=\u7528\u6237\u540D（用户名）

第二步 代码 资源文件都要在类根目录

<h1>java方式实现的页面国际化</h1><hr>

<% Locale local = request.getLocale();

ResourceBundle bundle = ResourceBundle.getBundle("resource",local); %>

<form action="#"> <%=bundle.getString("username") %>:<input type="text"/>

<%=bundle.getString("password") %>:<input type="password"/>

<input type="submit" value="<%=bundle.getString("submit") %>"/> </form>

<h1>fmt标签方式实现的页面国际化</h1><hr>

<fmt:setBundle basename="resource" var="bundle" scope="page"/> 第一步 获取资源文件，设置域

<form action="#"><fmt:message bundle="${bundle}" key="username"/>:<input type="text"/>第二步 使用message获取操作

<fmt:message bundle="${bundle}" key="password"/>:<input type="password"/>

<input type="submit" value="<fmt:message bundle="${bundle}" key="submit"/>"/> </form>

4 、web项目启动顺序：监听器-->过滤器-->Servlet.

项目启动后,容器会首先创建声明的各种监听器,为后继的各个事件监听做准备,然后创建过滤器,最后是Servlet.

项目销毁：过滤器-->Servlet-->监听器

5、jsp页面在第一次被访问到时会被jsp翻译引擎翻译成一个Servlet,从此对这个jsp页面的访问都是由这个Servlet执行后进行输出:

1、a.jsp访问后：

会在tomcat中的work目录中生成：Catalina引擎文件夹\虚拟主机目录localhost\工程名称\ org\apache\jsp下：a\_jsp.java:

2、此类继承、实现了：

public final class a\_jsp extends org.apache.jasper.runtime.**HttpJspBase** implements

org.apache.jasper.runtime.JspSourceDependent, org.apache.jasper.runtime.JspSourceImports

3、org.apache.jasper.runtime.HttpJspBase：

此抽象类是tomcat写的继承了HttpServlet实现了HttpJspPage

4、此类的方法：public void \_jspService就是用来执行方法显示的：

Html标签的输出：out.write("<head>\r\n");

out.write("<font color=\"red\">\r\n");

out.print(new Date().toLocaleString() );// jsp页面写的代码：<%=new Date().toLocaleString() %>

out.write("\r\n");

为什么out在网页可以直接使用呢？：在此方法内有定义局部变量：JspWriter out = null;

6、注意URL中?后参数没有双引号的，http://www.wwl.com/a.jsp?name=你好

7、注意WEB-INF目录下的文件只能通过请求转发来访问，不能通过页面重定向访问。