|  |
| --- |
| [公司名称] |
| 标题 |
| 副标题 |

|  |
| --- |
| 姚斯元  2018-10-30 |

目录

[一、 123 1](#_Toc3539499)

[1.1 1 1](#_Toc3539500)

[1.1.1 3 1](#_Toc3539501)

# JavaEE

## 基础知识

### B/S与C/S

B/S:   browser/server    浏览器/服务器

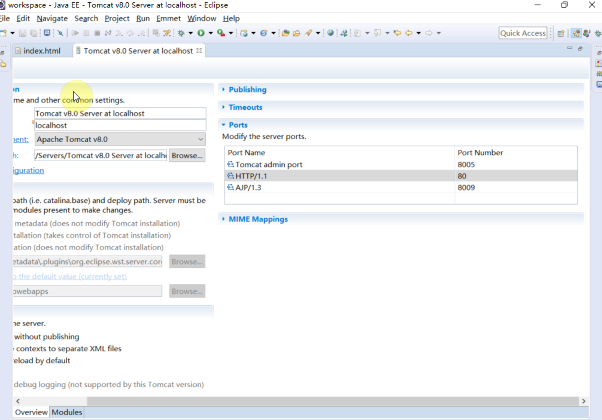
免安装升级，更新快   功能弱化

C/S:    client/client         客户端/服务器

#### C/S 架构和 B/S 架构分别是什么?各有哪些优缺点？

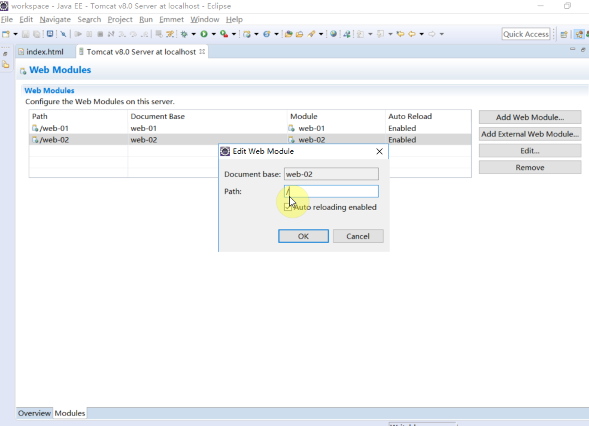
* C/S 是指客户端应用,这些应用是需要客户下载安装才能使用;比如腾讯 QQ 等
* B/S 是指浏览器端的应用;
* C/S 的优点是稳定完善,界面和操作可以很丰富,缺点是不便于更新维护;
* B/S 的优点是更新方便,客户端无需安装,缺点是功能弱化,离线方面功能不完善

使用eclipse修改项目的默认端口



* 使用eclipse修改运行项目时url(去掉项目名)

       修改的是Server文件夹中的server.xml中的配置，不会修改源文件



#### eclipse安装插件  <https://emmet.io/download/>

eclipse安装io插件  在线

选择help-->install new Software -->在打开的界面中选择添加按钮 在name属性中随便起名(有意义) location中填写下载地址(http://download.emmet.io/eclipse/updates/)  然后选择🆗  eclipse会自动下载，然后一路next 最后完成即可

window-->preferects-->出现Emmet即为成功

eclipse安装插件  离线

将插件jar包放入eclipse的plugins文件夹下即可

静态资源都放入WebContent-->static

jar包都放入lib文件夹下

WEB项目有红叉，但能运行，解决办法：

cmd中在该项目的路径下执行   mvn eclipse:clean     mvn eclipse:eclipse

  或(mvn eclipse:clean eclipse:eclipse)

右键项目 configure  选择最后一个即可

## 深入理解Http协议

### Http协议入门

#### 什么是http协议

http协议： 对浏览器客户端 和 服务器端 之间数据传输的格式规范

### 查看http协议的工具

1）使用火狐的firebug插件（右键->firebug->网络）

2）使用谷歌的“审查元素”

#### 2.1 http协议内容

|  |
| --- |
| 请求（浏览器-》服务器）  GET /day09/hello HTTP/1.1  Host: localhost:8080  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  Accept-Language: zh-cn,en-us;q=0.8,zh;q=0.5,en;q=0.3  Accept-Encoding: gzip, deflate  Connection: keep-alive |

|  |
| --- |
| 响应（服务器-》浏览器）  HTTP/1.1 200 OK  Server: Apache-Coyote/1.1  Content-Length: 24  Date: Fri, 30 Jan 2015 01:54:57 GMT  this is hello servlet!!! |

### Http请求

|  |
| --- |
| GET /day09/hello HTTP/1.1 -请求行  Host: localhost:8080 --请求头（多个key-value对象）  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  Accept-Language: zh-cn,en-us;q=0.8,zh;q=0.5,en;q=0.3  Accept-Encoding: gzip, deflate  Connection: keep-alive  --一个空行  name=eric&password=123456 --（可选）实体内容 |

#### 3.1 请求行

GET /day09/hello HTTP/1.1

##### #http协议版本

http1.0：当前浏览器客户端与服务器端建立连接之后，只能发送一次请求，一次请求之后连接关闭。

http1.1：当前浏览器客户端与服务器端建立连接之后，可以在一次连接中发送多次请求。（基本都使用1.1）

##### #请求资源

URL: 统一资源定位符。http://localhost:8080/day09/testImg.html。只能定位互联网资源。是URI 的子集。

URI： 统一资源标记符。/day09/hello。用于标记任何资源。可以是本地文件系统，局域网的资源（//192.168.14.10/myweb/index.html）， 可以是互联网。

##### #请求方式

常见的请求方式： GET 、 POST、 HEAD、 TRACE、 PUT、 CONNECT 、DELETE

常用的请求方式： GET 和 POST

表单提交：

<form action="提交地址" method="GET/POST">

<form>

GET vs POST 区别

**1）GET方式提交**

a）地址栏（URI）会跟上参数数据。以？开头，多个参数之间以&分割。

|  |
| --- |
| **GET** /day09/testMethod.html?name=eric&password=123456 HTTP/1.1  Host: localhost:8080  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  Accept-Language: zh-cn,en-us;q=0.8,zh;q=0.5,en;q=0.3  Accept-Encoding: gzip, deflate  Referer: http://localhost:8080/day09/testMethod.html  Connection: keep-alive |

b）GET提交参数数据有限制，不超过1KB。

c）GET方式不适合提交敏感密码。

d）注意： 浏览器直接访问的请求，默认提交方式是GET方式

**2）POST方式提交**

a）参数不会跟着URI后面。参数而是跟在请求的实体内容中。没有？开头，多个参数之间以&分割。

|  |
| --- |
| **POST** /day09/testMethod.html HTTP/1.1  Host: localhost:8080  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  Accept-Language: zh-cn,en-us;q=0.8,zh;q=0.5,en;q=0.3  Accept-Encoding: gzip, deflate  Referer: http://localhost:8080/day09/testMethod.html  Connection: keep-alive  name=eric&password=123456 |

b）POST提交的参数数据没有限制。

c）POST方式提交敏感数据。

#### 3.2 请求头

|  |
| --- |
| Accept: text/html,image/\* -- 浏览器接受的数据类型  Accept-Charset: ISO-8859-1 -- 浏览器接受的编码格式  Accept-Encoding: gzip,compress --浏览器接受的数据压缩格式  Accept-Language: en-us,zh- --浏览器接受的语言  Host: www.it315.org:80 --（必须的）当前请求访问的目标地址（主机:端口）  If-Modified-Since: Tue, 11 Jul 2000 18:23:51 GMT --浏览器最后的缓存时间  Referer: http://www.it315.org/index.jsp -- 当前请求来自于哪里  User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0) --浏览器类型  Cookie:name=eric -- 浏览器保存的cookie信息  Connection: close/Keep-Alive -- 浏览器跟服务器连接状态。close: 连接关闭 keep-alive：保存连接。  Date: Tue, 11 Jul 2000 18:23:51 GMT -- 请求发出的时间 |

#### 3.3 实体内容

只有POST提交的参数会放到实体内容中

#### 3.4 HttpServletRequest对象

HttpServletRequest对象作用是用于获取请求数据。

核心的API：

请求行：

request.getMethod(); 请求方式

request.getRequetURI() / request.getRequetURL() 请求资源

request.getProtocol() 请求http协议版本

请求头：

request.getHeader("名称") 根据请求头获取请求值

request.getHeaderNames() 获取所有的请求头名称

实体内容:

request.getInputStream() 获取实体内容数据

#### 3.5 service 和 doXX方法区别

|  |
| --- |
| **HttpSevlet类的源码：**  **protected** **void** service(HttpServletRequest req, HttpServletResponse resp)  **throws** ServletException, IOException {  //得到请求方式  String method = req.getMethod();  **if** (method.equals(*METHOD\_GET*)) {  **long** lastModified = getLastModified(req);  **if** (lastModified == -1) {  // servlet doesn't support if-modified-since, no reason  // to go through further expensive logic  doGet(req, resp);  } **else** {  **long** ifModifiedSince;  **try** {  ifModifiedSince = req.getDateHeader(*HEADER\_IFMODSINCE*);  } **catch** (IllegalArgumentException iae) {  // Invalid date header - proceed as if none was set  ifModifiedSince = -1;  }  **if** (ifModifiedSince < (lastModified / 1000 \* 1000)) {  // If the servlet mod time is later, call doGet()  // Round down to the nearest second for a proper compare  // A ifModifiedSince of -1 will always be less  maybeSetLastModified(resp, lastModified);  doGet(req, resp);  } **else** {  resp.setStatus(HttpServletResponse.*SC\_NOT\_MODIFIED*);  }  }  } **else** **if** (method.equals(*METHOD\_HEAD*)) {  **long** lastModified = getLastModified(req);  maybeSetLastModified(resp, lastModified);  doHead(req, resp);  } **else** **if** (method.equals(*METHOD\_POST*)) {  doPost(req, resp);    } **else** **if** (method.equals(*METHOD\_PUT*)) {  doPut(req, resp);    } **else** **if** (method.equals(*METHOD\_DELETE*)) {  doDelete(req, resp);    } **else** **if** (method.equals(*METHOD\_OPTIONS*)) {  doOptions(req,resp);    } **else** **if** (method.equals(*METHOD\_TRACE*)) {  doTrace(req,resp);    } **else** {  //  // Note that this means NO servlet supports whatever  // method was requested, anywhere on this server.  //  String errMsg = *lStrings*.getString("http.method\_not\_implemented");  Object[] errArgs = **new** Object[1];  errArgs[0] = method;  errMsg = MessageFormat.*format*(errMsg, errArgs);    resp.sendError(HttpServletResponse.*SC\_NOT\_IMPLEMENTED*, errMsg);  }  } |

#### 3.6 案例- 什么是时间戳

很多网站在发布版本之前，都会在URL请求地址后面加上一个实现戳进行版本更新。

#### 3.7 案例- 防止非法链接(referer)

第1次 蚂蚁课堂 -> 页面(展示一张图片)

第2次 直接查看展示图片

非法链接：

1）直接访问访问资源

referer： 当前请求来自于哪里。

代码:

|  |
| --- |
| **<filter>**  **<filter-name>ImgFilter</filter-name>**  **<filter-class>com.itmayiedu.filter.ImgFilter</filter-class>**  **</filter>**  **<filter-mapping>**  **<filter-name>ImgFilter</filter-name>**  **<url-pattern>/static/\*</url-pattern>**  **</filter-mapping>** |

|  |
| --- |
| public class ImgFilter implements Filter {  public void destroy() {  // TODO Auto-generated method stub  }  public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain)  throws IOException, ServletException {  HttpServletRequest request = (HttpServletRequest) servletRequest;  HttpServletResponse response = (HttpServletResponse) servletResponse;  String referer = request.getHeader("referer");  System.*out*.println("refer is" + "" + referer);  if (referer == null || !referer.contains(request.getServerName())) {  request.getRequestDispatcher("/static/error.png").forward(request, response);  } else {  filterChain.doFilter(request, response);  }  }  public void init(FilterConfig arg0) throws ServletException {  // TODO Auto-generated method stub  }  } |

#### 传递的请求参数如何获取

GET方式： 参数放在URI后面

POST方式： 参数放在实体内容中

获取GET方式参数：

request.getQueryString();

获取POST方式参数：

request.getInputStream();

问题：但是以上两种不通用，而且获取到的参数还需要进一步地解析。

所以可以使用统一方便的获取参数的方式：

核心的API：

request.getParameter("参数名"); 根据参数名获取参数值（注意，只能获取一个值的参数）

request.getParameterValue("参数名“)；根据参数名获取参数值（可以获取多个值的参数）

request.getParameterNames(); 获取所有参数名称列表

### Http响应

|  |
| --- |
| HTTP/1.1 200 OK --响应行  Server: Apache-Coyote/1.1 --响应头（key-vaule）  Content-Length: 24  Date: Fri, 30 Jan 2015 01:54:57 GMT  --一个空行  this is hello servlet!!! --实体内容 |

#### 5.1 响应行

##### #http协议版本

##### #状态码: 服务器处理请求的结果（状态）

常见的状态：

200： 表示请求处理完成并完美返回

302： 表示请求需要进一步细化。

404： 表示客户访问的资源找不到。

500： 表示服务器的资源发送错误。（服务器内部错误）

##### #状态描述

#### 5.2 常见的响应头

|  |
| --- |
| Location: http://www.it315.org/index.jsp -表示重定向的地址，该头和302的状态码一起使用。  Server:apache tomcat ---表示服务器的类型  Content-Encoding: gzip -- 表示服务器发送给浏览器的数据压缩类型  Content-Length: 80 --表示服务器发送给浏览器的数据长度  Content-Language: zh-cn --表示服务器支持的语言  Content-Type: text/html; charset=GB2312 --表示服务器发送给浏览器的数据类型及内容编码  Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT --表示服务器资源的最后修改时间  Refresh: 1;url=http://www.it315.org --表示定时刷新  Content-Disposition: attachment; filename=aaa.zip --表示告诉浏览器以下载方式打开资源（下载文件时用到）  Transfer-Encoding: chunked  Set-Cookie:SS=Q0=5Lb\_nQ; path=/search --表示服务器发送给浏览器的cookie信息（会话管理用到）  Expires: -1 --表示通知浏览器不进行缓存  Cache-Control: no-cache  Pragma: no-cache  Connection: close/Keep-Alive --表示服务器和浏览器的连接状态。close：关闭连接 keep-alive:保存连接 |

#### 5.3 HttpServletResponse对象

HttpServletResponse对象修改响应信息：

响应行：

response.setStatus() 设置状态码

响应头：

response.setHeader("name","value") 设置响应头

实体内容：

response.getWriter().writer(); 发送字符实体内容

response.getOutputStream().writer() 发送字节实体内容

#### 5.4 案例- 请求重定向（Location）

|  |
| --- |
| **resp.setStatus(302);**  **resp.setHeader("Location", "OtherServlet");** |

### Https与Http

#### https 与http区别？

　1、https 协议需要到 ca 申请证书，一般免费证书较少，因而需要一定费用。

　2、http 是超文本传输协议，信息是明文传输，https 则是具有安全性的 ssl 加密传输协议。

　3、http 和 https 使用的是完全不同的连接方式，用的端口也不一样，前者是 80，后者是 443。

　4、http 的连接很简单，是无状态的；HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，比 http 协议安全。

#### 6.1.2 https工作原理？

我们都知道 HTTPS 能够加密信息，以免敏感信息被第三方获取，所以很多银行网站或电子邮箱等等安全级别较高的服务都会采用 HTTPS 协议。

　客户端在使用 HTTPS 方式与 Web 服务器通信时有以下几个步骤，如图所示。

　　（1）客户使用 https 的 URL 访问 Web 服务器，要求与 Web 服务器建立 SSL 连接。

　　（2）Web 服务器收到客户端请求后，会将网站的证书信息（证书中包含公钥）传送一份给客户端。

　　（3）客户端的浏览器与 Web 服务器开始协商 SSL 连接的安全等级，也就是信息加密的等级。

　　（4）客户端的浏览器根据双方同意的安全等级，建立会话密钥，然后利用网站的公钥将会话密钥加密，并传送给网站。

　　（5）Web 服务器利用自己的私钥解密出会话密钥。

　　（6）Web 服务器利用会话密钥加密与客户端之间的通信。



#### 6.1.3 https优缺点？

　虽然说 HTTPS 有很大的优势，但其相对来说，还是存在不足之处的：

　　（1）HTTPS 协议握手阶段比较费时，会使页面的加载时间延长近 50%，增加 10% 到 20% 的耗电；

　　（2）HTTPS 连接缓存不如 HTTP 高效，会增加数据开销和功耗，甚至已有的安全措施也会因此而受到影响；

　　（3）SSL 证书需要钱，功能越强大的证书费用越高，个人网站、小网站没有必要一般不会用。

　 （4）SSL 证书通常需要绑定 IP，不能在同一 IP 上绑定多个域名，IPv4 资源不可能支撑这个消耗。

　　（5）HTTPS 协议的加密范围也比较有限，在黑客攻击、拒绝服务攻击、服务器劫持等方面几乎起不到什么作用。最关键的，SSL 证书的信用链体系并不安全，特别是在某些国家可以控制 CA 根证书的情况下，中间人攻击一样可行。

### http请求工具

#### 客户端模拟http请求工具

Postmen(谷歌插件)、RestClient

#### 服务器模拟http请求工具

httpclient、HttpURLConnection

httpCient请求代码

|  |
| --- |
| /\*\*  \* 发送 post请求访问本地应用并根据传递参数不同返回不同结果  \*/  public void post() {  // 创建默认的httpClient实例.  CloseableHttpClient httpclient = HttpClients.*createDefault*();  // 创建httppost  HttpPost httppost = new HttpPost("http://localhost:8080/myDemo/Ajax/serivceJ.action");  // 创建参数队列  List<NameValuePair> formparams = new ArrayList<NameValuePair>();  formparams.add(new BasicNameValuePair("type", "house"));  UrlEncodedFormEntity uefEntity;  try {  uefEntity = new UrlEncodedFormEntity(formparams, "UTF-8");  httppost.setEntity(uefEntity);  System.*out*.println("executing request " + httppost.getURI());  CloseableHttpResponse response = httpclient.execute(httppost);  try {  HttpEntity entity = response.getEntity();  if (entity != null) {  System.*out*.println("--------------------------------------");  System.*out*.println("Response content: " + EntityUtils.*toString*(entity, "UTF-8"));  System.*out*.println("--------------------------------------");  }  } finally {  response.close();  }  } catch (ClientProtocolException e) {  e.printStackTrace();  } catch (UnsupportedEncodingException e1) {  e1.printStackTrace();  } catch (IOException e) {  e.printStackTrace();  } finally {  // 关闭连接,释放资源  try {  httpclient.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }  /\*\*  \* 发送 get请求  \*/  public void get() {  CloseableHttpClient httpclient = HttpClients.*createDefault*();  try {  // 创建httpget.  HttpGet httpget = new HttpGet("http://www.baidu.com/");  System.*out*.println("executing request " + httpget.getURI());  // 执行get请求.  CloseableHttpResponse response = httpclient.execute(httpget);  try {  // 获取响应实体  HttpEntity entity = response.getEntity();  System.*out*.println("--------------------------------------");  // 打印响应状态  System.*out*.println(response.getStatusLine());  if (entity != null) {  // 打印响应内容长度  System.*out*.println("Response content length: " + entity.getContentLength());  // 打印响应内容  System.*out*.println("Response content: " + EntityUtils.*toString*(entity));  }  System.*out*.println("------------------------------------");  } finally {  response.close();  }  } catch (ClientProtocolException e) {  e.printStackTrace();  } catch (ParseException e) {  e.printStackTrace();  } catch (IOException e) {  e.printStackTrace();  } finally {  // 关闭连接,释放资源  try {  httpclient.close();  } catch (IOException e) {  e.printStackTrace();  }  }  } |

#### 前端ajax请求

ajax是异步请求。做的局部刷新。但是也可以设置为同步。async设置成true

|  |
| --- |
| $.ajax({  type : 'post',  dataType : "text",  url : "http://a.a.com/a/FromUserServlet",  data : "userName=YaoSiyuan&userAge=19",  success : **function**(msg) {  alert(msg);  }  }); |

#### 跨域实战解决方案

跨域原因产生：在当前域名请求网站中，默认不允许通过ajax请求发送其他域名。

XMLHttpRequest cannot load 跨域问题解决办法

1. 在相应头设置允许跨域。最简单的。
2. jsonp解决，只能解决get，post无效。
3. 搭建网关系统。适合大型网关。
4. 使用后台服务转发。用的比较少，比较占用带宽。但是不会暴露接口，比较安全。

#### 使用后台response添加header

后台response添加header，**response.setHeader("Access-Control-Allow-Origin", "\*"); 支持所有网站**

#### **使用JSONP**

前端代码:

|  |
| --- |
| $.ajax({  type : "POST",  async : false,  url : "http://a.a.com/a/FromUserServlet?userName=张三",  dataType : "jsonp",//数据类型为jsonp  jsonp : "jsonpCallback",//服务端用于接收callback调用的function名的参数  success : function(data) {  alert(data.result);  },  error : function() {  alert('fail');  }  }); |

后端代码:

|  |
| --- |
| @WebServlet("/FromUserServlet")  public class FromUserServlet extends HttpServlet {  @Override  protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  doPost(req, resp);  }  @Override  protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  resp.setCharacterEncoding("UTF-8");  // resp.setHeader("Access-Control-Allow-Origin", "\*");  String userName = req.getParameter("userName");  String userAge = req.getParameter("userAge");  System.*out*.println(userName + "----" + userAge+"---"+req.getMethod());  // JSONObject JSONObject1 = new JSONObject();  // JSONObject1.put("success", "添加成功!");  // resp.getWriter().write("callbackparam(" + JSONObject1.toJSONString()  // + ")");  try {  resp.setContentType("text/plain");  resp.setHeader("Pragma", "No-cache");  resp.setHeader("Cache-Control", "no-cache");  resp.setDateHeader("Expires", 0);  PrintWriter out = resp.getWriter();  JSONObject resultJSON = new JSONObject(); // 根据需要拼装json  resultJSON.put("result", "content");  String jsonpCallback = req.getParameter("jsonpCallback");// 客户端请求参数  out.println(jsonpCallback + "(" + resultJSON.toJSONString() + ")");// 返回jsonp格式数据  out.flush();  out.close();  } catch (Exception e) {  // TODO: handle exception  }  }  } |

JSONP的优缺点:

JSONP只支持get请求不支持psot请求

jsonp实现原理

生成一个get请求发送到服务器端

因为jsonp底层就是发的get请求。

405，请求方法不对。

面试技巧。

你在项目中遇到了哪些问题，跨域问题，12345解决方案，优缺点。

#### **使用接口网关**

使用nginx转发。

### http抓包工具

fiddler、wireshark

### 本节课程maven坐标

|  |
| --- |
| <dependencies>  <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>javax.servlet-api</artifactId>  <version>3.1.0</version>  <scope>provided</scope>  </dependency>  <!-- https://mvnrepository.com/artifact/org.apache.httpcomponents/httpclient -->  <dependency>  <groupId>org.apache.httpcomponents</groupId>  <artifactId>httpclient</artifactId>  <version>4.3.5</version>  </dependency>  <!-- https://mvnrepository.com/artifact/org.apache.commons/commons-lang3 -->  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-lang3</artifactId>  <version>3.5</version>  </dependency>  <!-- https://mvnrepository.com/artifact/com.alibaba/fastjson -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>1.2.29</version>  </dependency>  </dependencies> |

### 推荐书籍：

head first Servlet Jsp        http权威指南        图解http

http下午茶： <http://www.kancloud.cn/kancloud/tealeaf-http/43837>

### 动态网页和静态网页：

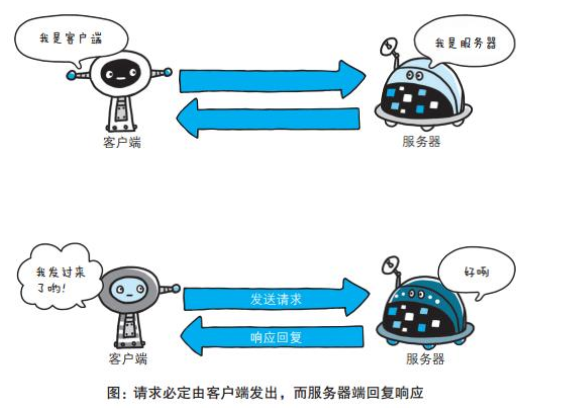
并非网页元素会不会动，而是网页能否和服务器进行交互

#### 区别：

动态网页是指哪些可以和服务端交互的页面, 并且会向后端发送请求的页面;而静态网页是指那些只作为展示的没有交互的普通页面.

#### HTTP协议

   用于客户端和服务器之间的通讯

****

客户端：凡是能够发起http请求或使用http协议的( 提供可访问资源的机器)  浏览器

服务器：凡是能够接收http请求，并作出响应的( 可以进行HTTP访问的设备)

   小至手机，大至机架式服务器，个人PC,笔记本，都可以

#### http协议：

* 请求响应式协议    先请求后响应的形式
* 断开式协议     响应结束后，这次http请求就会断开，不会长久的建立连接
* 无状态协议     当发起请求响应时，服务器不记录任何有关于客户端的信息

#### 发起http请求的方式：

* 在地址栏中输入网址
* 提交表单
* 点击了超链接

### 请求报文：

   客户端发起的内容格式     以键值对的形式存在       

User-Agent: 判断请求的是浏览器还是？  防止黑客攻击

### 响应报文：

* 服务器返回给客户端，按照相应的格式



#### 常见的状态码：

* 200  请求成功
* 302  重定向
* 403  权限不足
* 400  坏的请求
* 404  资源文件找不到
* 500  服务器错误
* 405   请求不一致

#### MiME类型：

描述消息内容类型的因特网标准

            根据响应的元素不同，content内容也不一样     <http://www.w3school.com.cn/media/media_mimeref.asp>

#### Http服务器

nginx(nginx.org)  apache(httpd.apache.org)   性能都极高，但不能过逻辑 ,只处理静态资源，用在分布式部署

# servlet

### 课程目标

动态资源与静态资源区别

1. servlet三及相关接口简介

2. servet 执行过程

3. servlet路径映射

4. servlet生命周期(重点) --理解（重点）

5. Servlet自动加载

6. Servlet线程安全

7. Servlet相关接口详解

8. ServletContext对象 --知识点

### servlet线程安全问题

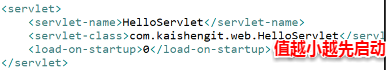
   多线程  同一个对象  同一个属性

servlet不是线程安全的

原因：

* 多次请求的时候，只创建一个servlet对象(即只创建一次Servlet类)//判断调用几次，看构造方法无参
* servlet默认是在第一次请求的时候创建对象，由servlet容器(tomcat)创建
* 多次请求的是同一个Servlet对象
* servlet的设计是线程不安全的，最好不要在servlet中使用有状态的实例变量,必须使用请加锁
  + 实例变量：
    - 有状态：每次调用，其值都会发生改变(int count = 10)
    - 无状态：每次调用，其值不会发生改变 (A a = new A())

Servlet可以在容器启动时创建     两个数值相同，那么根据配置的先后顺序决定，在前者先启动



### Servlet的生命周期

#### 实例化：

调用构造方法     web容器创建Servlet的实例    在容器启动或者第一次请求的时候创建

#### 初始化：

调用init方法 web容器调用Servlet的init() 在容器启动或者第一次请求时调用(创建类之后初始化)

#### 服务：

service方法 由service()决定调用的是dopost还是doget  由service调用dopost.doget   多次

#### 销毁：

destory方法 web容器在销毁Servlet之前调用 在容器关闭时调用

### Web项目结构

|- WebRoot : web应用的根目录

|- 静态资源（html+css+js+image+vedio）

|- WEB-INF ： 固定写法。

|-classes： （可选）固定写法。存放class字节码文件

|-lib： （可选）固定写法。存放jar包文件。

|-web.xml

注意：

1）WEB-INF目录里面的资源不能通过浏览器直接访问

2）如果希望访问到WEB-INF里面的资源，就必须把资源配置到一个叫web.xml的文件中。

练习：

1）在webapps下建立一个mybbs目录

2）创建两个文件

2.1 index.html 里面随便写内容 ，有超链接-连接到test.html

2.2 test.html 里面随便写

3）通过浏览器访问到。

### 手动开发动态资源

#### 2.1静态资源和动态资源的区别

静态资源： 当用户多次访问这个资源，资源的源代码永远不会改变的资源。

动态资源：当用户多次访问这个资源，资源的源代码可能会发送改变。

#### 源的开发技术

Servlet : 用java语言来编写动态资源的开发技术。

Servlet特点：

1）普通的java类，继承HttpServlet类，覆盖doGet方法

2）Servlet类只能交给tomcat服务器运行！！！！（开发者自己不能运行！！！）

Servlet手动编写步骤：

1）编写一个servlet程序，继承HttpServlet

|  |
| --- |
| /\*\*  \* 第一个servlet程序  \* **@author** APPle  \*  \*/  **public** **class** HelloServlet **extends** HttpServlet{  @Override  **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp)  **throws** ServletException, IOException {  //解决中文乱码问题  resp.setCharacterEncoding("utf-8");// 内容编码，防止出现中文乱码  resp.setContentType("text/html;charset=utf-8"); //向浏览器输出内容  resp.getWriter().write("这是第一个servlet程序。当前时间为："+**new** Date());  }  } |

2）找到HelloServlet类的class字节码，然后把拷贝到tomcat的一个web应用中WEB-INF/classes目录下。

3）在当前web应用下的web.xml文件配置Servlet。

|  |
| --- |
| <!-- 配置一个servlet程序 -->  <servlet>  <!-- servlet的内部名称 ，可以自定义-->  <servlet-name>HelloServlet</servlet-name>  <!-- servlet类名： 包名+简单类名-->  <servlet-class>com.itmayiedu.HelloServlet</servlet-class>  </servlet>  <servlet-mapping>  <!-- servlet的内部名称，和上面的名称保持一致！！！-->  <servlet-name>HelloServlet</servlet-name>  <!-- servlet的访问名称： /名称 -->  <url-pattern>/hello</url-pattern>  </servlet-mapping> |

4）启动tomcat服务器，运行访问

访问servlet: http://localhost:8080/myweb/ hello

### 工具开发动态资源

1）创建web project （javaweb工程）

2）在WebRoot下建立静态资源文件，

3）在src下建立动态资源文件

3.1 new -> Servlet( servlet的代码生成器)

3.2 写pacakge -> class名 -> 修改mapping url

4）关联tomcat服务器

4.1 window-> Preferences - > MyEcplise -> servers -> Tomcat 6.x (注意一定要enable)

5）部署web project应用。（拷贝web应用到tomcat的webapps目录下）

6）启动tomcat服务器

7）访问servlet

http://localhost:8081/myWeb /hello

### 如何开发一个Servlet

1.1 步骤：

1）编写java类，继承HttpServlet类

2）重新doGet和doPost方法

3）Servlet程序交给tomcat服务器运行！！

3.1 servlet程序的class码拷贝到WEB-INF/classes目录

3.2 在web.xml文件中进行配置

|  |
| --- |
| <!-- 配置一个servlet -->  <!-- servlet的配置 -->  <servlet>  <!-- servlet的内部名称，自定义。尽量有意义 -->  <servlet-name>FirstServlet</servlet-name>  <!-- servlet的类全名： 包名+简单类名 -->  <servlet-class>com.itmayiedu.FirstServlet</servlet-class>  </servlet>      <!-- servlet的映射配置 -->  <servlet-mapping>  <!-- servlet的内部名称，一定要和上面的内部名称保持一致！！ -->  <servlet-name>FirstServlet</servlet-name>  <!-- servlet的映射路径（访问servlet的名称） -->  <url-pattern>/first</url-pattern>  </servlet-mapping> |

问题：访问次URL： http://localhost:8080/myweb /first

前提： tomcat服务器启动时，首先加载webapps中的每个web应用的web.xml配置文件。

http://: http协议

localhost： 到本地的hosts文件中查找是否存在该域名对应的IP地址

127.0.0.1

8080： 找到tomcat服务器

/day10 在tomcat的webapps目录下找 day10的目录

/first 资源名称。

1）在myweb的web.xml中查找是否有匹配的url-pattern的内容（/first）

2）如果找到匹配的url-pattern,则使用当前servlet-name的名称到web.xml文件中查询是否相同名称的servlet配置

3）如果找到，则取出对应的servlet配置信息中的servlet-class内容：

字符串： **com.itmayiedu.a\_servlet.FirstServlet**

**通过反射：**

**a）构造FirstServlet的对象**

**b）然后调用FirstServlet里面的方法**

### Servlet 注解版本

Servlet3.0以上使用 注解自动映射@webServlet.

### Sevlet的生命周期（重点）

#### 引入

Servlet的生命周期： servlet类对象什么时候创建，什么时候调用什么方法，什么时候销毁。

以前的对象： new Student（）； stu.study(); stu=null;

Servlet程序的生命周期由tomcat服务器控制的！！！！

#### Servlet重要的四个生命周期方法

构造方法： 创建servlet对象的时候调用。默认情况下，第一次访问servlet的时候创建servlet对象 只调用1次。证明servlet对象在tomcat是单实例的。

init方法： 创建完servlet对象的时候调用。只调用1次。

service方法： 每次发出请求时调用。调用n次。

destroy方法： 销毁servlet对象的时候调用。停止服务器或者重新部署web应用时销毁servlet对象。

只调用1次。

#### 伪代码演示servlet的生命周期

Tomtcat内部代码运行：

1）通过映射找到到servlet-class的内容，字符串： **com.itmayiedu.a\_servlet.FirstServlet**

2）通过反射构造FirstServlet对象

2.1 得到字节码对象

Class clazz = class.forName("**com.itmayiedu.a\_servlet.FirstServlet**");

2.2 调用无参数的构造方法来构造对象

Object obj = clazz.newInstance(); ---1.servlet的构造方法被调用

3）创建ServletConfig对象，通过反射调用init方法

3.1 得到方法对象

Method m = clazz.getDeclareMethod("init",ServletConfig.class);

3.2 调用方法

m.invoke(obj,config); --2.servlet的init方法被调用

4）创建request，response对象，通过反射调用service方法

4.1 得到方法对象

Methodm m =clazz.getDeclareMethod("service",HttpServletRequest.class,HttpServletResponse.class);

4.2 调用方法

m.invoke(obj,request,response); --3.servlet的service方法被调用

5）当tomcat服务器停止或web应用重新部署，通过反射调用destroy方法

5.1 得到方法对象

Method m = clazz.getDeclareMethod("destroy",null);

5.2 调用方法

m.invoke(obj,null); --4.servlet的destroy方法被调用

#### 用时序图来演示servlet的生命周期



### Servlet的自动加载

默认情况下，第一次访问servlet的时候创建servlet对象。如果servlet的构造方法或init方法中执行了比较多的逻辑代码，那么导致用户第一次访问sevrlet的时候比较慢。

改变servlet创建对象的时机： 提前到加载web应用的时候！！！

在servlet的配置信息中，加上一个<load-on-startup>即可！！

|  |
| --- |
| <servlet>  <servlet-name>LifeDemo</servlet-name>  <servlet-class>com.itmayiedu.life.LifeDemo</servlet-class>  <!-- 让servlet对象自动加载 -->  <load-on-startup>1</load-on-startup> 注意： 整数值越大，创建优先级越低！！  </servlet> |

### Servlet的多线程并发问题

注意： servlet对象在tomcat服务器是单实例多线程的。

怎么证明：

servlet的无惨构造函数只被执行了一次，说明servlet是单例的。

因为servlet是多线程的，所以当多个servlet的线程同时访问了servlet的共享数据，如成员变量，可能会引发线程安全问题。

解决办法：

1）把使用到共享数据的代码块进行同步（使用synchronized关键字进行同步）

2）建议在servlet类中尽量不要使用成员变量。如果确实要使用成员，必须同步。而且尽量缩小同步代码块的范围。（哪里使用到了成员变量，就同步哪里！！），以避免因为同步而导致并发效率降低。

servlet处理多个请求的时候只执行一次，说明是单例，

流程：

1. tomcat加载web.xml文件，
2. 请求来的时候，如果是servlet。找到配置，
3. 解析class内容
4. 通过java的反射机制，调用class.forName.newInstance方法，通过无参构造函数创建servlet对象。
5. 先走service方法，进行判断是什么请求方法，然后执行具体的方法，
6. 服务器被停止掉后，走destory()

Servlet学习：

HttpServletRequest 请求对象：获取请求信息

HttpServletResponse 响应对象： 设置响应对象

ServletConfig对象 servlet配置对象

ServletContext对象； servlet的上下文对象

#### 线程安全代码:

|  |
| --- |
| **package** com.servlet;  **import** java.io.IOException;  **import** java.util.Date;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  /\*\*  \*  \* **@classDesc**: 功能描述:(线程安全演示)  \* **@author**: YaoSiyuan  \* **@createTime**: 2017年8月31日 下午11:47:20  \* **@version**: v1.0  \* **@copyright**:  \*/  **public** **class** ServetlDemo4 **extends** HttpServlet {  **private** **int** i = 1;  @Override  **public** **void** init() **throws** ServletException {  System.***out***.println("ServetlDemo4...init()");  }  @Override  **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  // 设置编码格式  // resp.setContentType("text/html;charset=utf-8");  resp.setCharacterEncoding("utf-8");// 内容编码，防止出现中文乱码  resp.setContentType("text/html;charset=utf-8");  **synchronized** (ServetlDemo4.**class**) {  // 向浏览器输出内容  resp.getWriter().write("这是第" + i + "次访问...");  **try** {  Thread.*sleep*(5000);  } **catch** (Exception e) {  // **TODO**: handle exception  }  i++;  }  }  @Override  **public** **void** destroy() {  System.***out***.println("ServetlDemo4...destroy()");  }  } |

### ServletContext对象

#### 得到web应用路径

java.lang.String getContextPath() 用在请求重定向的资源名称中

#### 域对象有关的方法

域对象：作用是用于保存数据，获取数据。可以在不同的动态资源之间共享数据。

案例：

Servlet1

Servlet2

name=eric

response.sendRedirect("/Servlet2?name=eric")

String request.getParameter("name");

保存到域对象中 从域对象获取

Student

方案1： 可以通过传递参数的形式，共享数据。局限：只能传递字符串类型。

方案2： 可以使用域对象共享数据，好处：可以共享任何类型的数据！！！！！

ServletContext就是一个域对象！！！！

保存数据：void setAttribute(java.lang.String name, java.lang.Object object)

获取数据： java.lang.Object getAttribute(java.lang.String name)

删除数据： void removeAttribute(java.lang.String name)

ServletContext域对象：作用范围在整个web应用中有效！！！

所有域对象：

HttpServletRequet 域对象

ServletContext域对象

HttpSession 域对象

PageContext域对象

#### 转发与重定向

RequestDispatcher getRequestDispatcher(java.lang.String path)

1）转发

a）地址栏不会改变

b）转发只能转发到当前web应用内的资源

c）可以在转发过程中，可以把数据保存到request域对象中

2）重定向

a）地址栏会改变，变成重定向到地址。

b）重定向可以跳转到当前web应用，或其他web应用，甚至是外部域名网站。

c）不能再重定向的过程，把数据保存到request中。

自定义重定向:

|  |
| --- |
| response.setStatus(302);  response.setHeader("Location","OtherServlet"); |

结论： 如果要使用request域对象进行数据共享，只能用转发技术！！！

总结：

Servlet编程:

Servlet生命周期（重点）

其他都是应用的东西（敲代码练习）

第一个熟悉作用域区别

第二个目标就是熟悉重定向底层原理。

.

### Servlet3..0  新特性

在servlet3.0以上的版本是支持注解的

注解：

* @WebServlet("/请求路径")  / 解决重名  @WebServlet(name="xxx",value="/请求路径")  写在servlet类的上面        D:\1 - 副本\Evernote_files\Image [188].png
* @WebFilter("拦截路径")   配置在过滤器类上

D:\1 - 副本\Evernote_files\Image [189].png

* @WebListener  配置监听器    3.1新特性：在需要session时再创建，不会一请求就创建session

2.x和3.x的schema不同，因此不能相互切换

### 硬编码

   修改起来困难，配置上的值  在打好的包中配置有可能改变

 配置硬编码的配置值：

在web.xml配置文件中，servlet节点标签中添加   可以配置多个

<init-param>

<param-anme>name<param-anme>

<param-value>root</param-value>

</init-param>

#### 获取配置文件中编写的配置：

通过重写servlet的init方法，解决硬编码问题    **ServletConfig接口**

##### 无参的init方法

              @Override

              public void init() throws ServletException {

            ServletConfig config = getServletConfig();

String value = config.getInitParameter("name");

         }

##### 有参的init方法



## Servlet容器

* 性能比http服务器偏低，处理业务逻辑
* tomcat  (不仅可以处理后端，也可以访问html,css)jetty    jBoss(已过时)
* **http服务器和Servlet容器的区别：**

Http 可以处理 Http 请求

Servlet 容器是一种程序服务器,实现了 servlet 和 jsp,同时具有 Http 的功能

基金会

apache基金会    http服务器   servlet容器(apache)

eclipse基金会     ide:eclipse      servlet容器(jetty)

### Ip地址：

* 外网ip: 域名:ip局域网ip：  使用同一个路由器下
* 网段： 每个路由器有一个网段，例如：192.168.1.x   192.168.1就是网段(c类网段)    网段分为A.B.C.D四个等级
* 没有网络时，ip为127.0.0.1(localhost)  即本机ip
* 没有路由器，光有交换机，只要网段不变，还是局域网
* 路由器在对外请求时，会有一个全球唯一的Mac地址用于请求，然后将请求到的资源返回到局域网中
* 虚拟机：NAT模式    不同笔记本 网段一样也不行，   内部

 桥接模式     自动变为局域网网段

## tomcat:

配套jdk版本使用

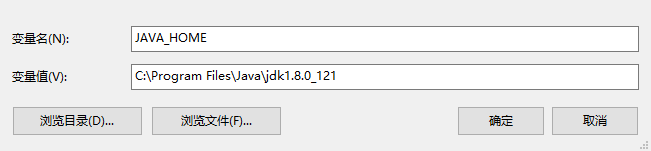
下载：

选择镜像下载(Mirrors)   <http://tomcat.apache.org/download-80.cgi>

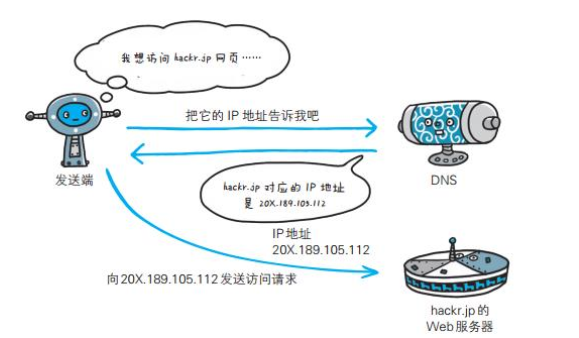
startup.sh  运行在linux的启动项         startup.bat  运行在window的启动项

shutdown.sh 运行在linux的关闭         shutdown.bat 运行在window的关闭项

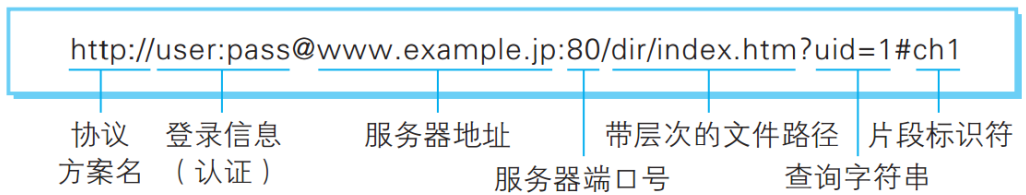
使用tomcat需要配置环境变量：  JAVA\_HOME           jdk的根目录



DNS: 域名系统 [**https://baike.baidu.com/item/DNS/427444**](https://baike.baidu.com/item/DNS/427444)   ip138.com



url组成



## javaEE项目

### 开发web项目步骤：

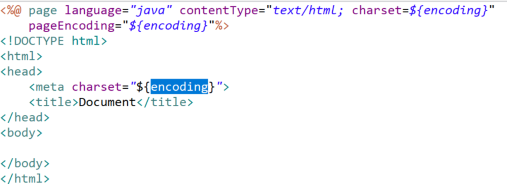
#### 选择新建Dynamic Web project项目

    编写代码-->发布到服务器中-->运行(http://localhost:8080/servlet在web.xml中的配置路径)

被浏览器执行解析的文件 html,css,js

jsp-----模板语言   html+java

使用eclipse建立jsp模板    preferences-->web-->jsp files -->editor-->template-->new jsp file(html)



配置web.xml  WEB-INF文件夹下面的web.xml文件   修改了web.xml文件，必须要重启服务器

配置欢迎页  可以配置多个欢迎页，从上往下依次查找对应的页面，如果都找不到，则404

  <welcome-file-list>

       <welcome-file>xxx.jsp</welcome-file>

  <welocme-file-list>

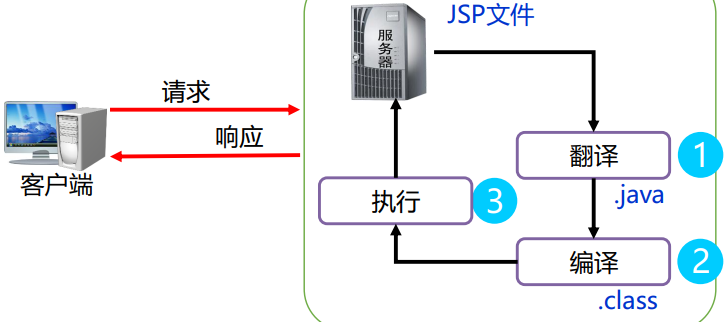
线上环境开发：将项目放入到tomcat安装目录的webapp文件夹中即可，然后启动tomcat即可(startup.bat双击)

如果是jsp文件，放入Root文件夹中，不写的话默认的是Root文件夹中的index.jsp

jsp文件的执行过程  把html标签以out.write("")的形式输出 全部在\_jspService方法中书写这些代码

1. 客户端向服务器发送请求;
2. 服务器接受到客户端的请求后将 JSP 文件编译为.java 文件;
3. 将.java 文件并编译为 class 文件;
4. 执行该文件并响应给客户端;
5. 第二次加载则会找到对应的.class 文件直接调用执行,如果修改了 jsp 文件则还需要编译执行啊

            第一次执行



下次执行



### jsp页面包含的内容：

#### 指令部分

例如page指令

<%@ page langeuage="java" contentType="text/html;charset=UTF-8"

      pageEncoding="UTF-8">  contentType为mime类型，告诉浏览器

<%@ page import="java.util.List">  导入Java包

#### 表达式部分

<%=表达式 %>

例如：<%=username %>  jack          <%=11+22 %> 33

#### 注释部分

<%-- 服务器注释 --%>   只能在服务器端(写程序时)看见，网页源代码看不见

<!--客户端注释 -->   网页源代码也可以看见

#### 静态内容部分

html  css js

声明部分

方法的声明：

<%!   public void xxx(){}>这样会在xxx\_jsp.java文件中生成一个方法(\_jspService方法外部)

#### 变量的声明：

<%!    %> //声明的内容为成员变量

<%  %> //声明的时局部变量

#### scriptlet

Servlet   一个标准的服务端的类，提供了被客户端访问的服务

#### 自定义Servlet类：

必须继承HttpServlet抽象类

常用的访问方式有：get post put delete

#### 开发Servlet步骤：

##### 创建servlet类并继承HttpServlet类

##### 添加可序列化ID

##### 根据Http请求的方式，至少提供一个doxxx()方法

* 提供doget()  方法，支持get请求
* 提供doPost() 方法，支持post请求
* 提供doPut() 方法，支持put请求
* 提供doDelete()方法，支持delete请求

##### 在web.xml中配置：

##### 配置节点

       <servlet>

  <servlet-name></servlet-name>  //自定义，一般而言和servlet类名相同

         <servlet-class></servlet-class> //servlet类的完全限定名

          </servlet>

     <servlet-mapping> //这时servlet的子节点 与上面servlet节点成对出现

<servlet-name></servlet-name> //自定义，必须和servlet中配置的servlet-name一致

<url-pattern></url-pattern> //客户端请求的路径，必须以/开头  不写会包nullpoitnerException

          </servlet-mapping>

##### 在Java中写html代码：   之前的写法(会自动部署)   使用jsp代替这种写法

           PrintWriter out = resp.getWriter();

           out.print("<html>");

           out.print("<head><title>hello</title></head>");

           out.print("<body>hello</body>");

           out.print("</html>");

           out.flush();

           out.close();

##### 修改tomcat端口号：

打开 Tomcat 的安装目录下的 conf 文件夹中的配置文件 server.xml,找到connector prot 节点默认值为 8080 的参数,修改后重启 Tomcat 服务器

* 常用类：
  + **HttpServletRequest   用于接收客户端请求，获取客户端信息**

通过getParameter(String name)来获得客户端表单或者url中对应的值(url是使用字符串拼接)

jsp内置对象之一，在jsp页面中叫做request

  <%  String name = request.getParameter("xxx")  %>

  <h3><%=name %></h3>  //获得接收到的值

通过req.setAttribute(String key,Object value)方法向跳转目标进行传值   servlet中

通过getAttribute(String key)方法获得setAttribute()方法设置的值**(需强转)**

通过req.getRequestDispatcher(String urlName)方法取得RequestDispatcher对象，并掉用forward(req.resp)方法进行请求转发

HttpServletResponse  给客户端做出响应

通过sendRedirect(String urlName)来重定向到目标页面(以/开头)

sendError(int errorCode)    手动引发404   例：  通过sendError(404,"错误信息")  //错误信息可不写

jsp内置对象之一： 叫做response

HttpSession   抽象类的对象 session执行过程(会话)

 jsp内置对象之一，叫session

setAttribute(String key,String value)向session中存放数据

getAttribute(String key)根据键值获得session空间中的值

removeAttribute(String key)根据key值从session控件中删除数据

getId()  获取sessionID的值

invalidate() 强制过期

HttpSession session = req.getSession(); //获取session对象

#### ServletContext类

   (servlet上下文)   公用的一个柜子，而session是一个小柜子

作用：又称为Application,是服务器中最大的存储空间，并且在服务器启动时便存在，在服务器关闭时释放，所有用户共享一个，因此存放公用的数据

表示所有客户端都可以存放和读取的数据空间    随着容器启动和关闭而产生和消失

ServletContext context = req.getServletContext();//获取ServletContext对象

      = session.getServletContext();//获取ServletContext对象

      = getServletContext();//获取ServletContext对象

setAttribute(String key,Object value)向servletContext空间中存储

getAttribute(String key)从servletContext中取值   需强转

removeAttribute(String key)从ServletContext中删除数据

jsp的内置对象之一，叫做application

四种存值空间：        jsp中的叫法            空间前缀作用域

pageContext      -->  pageScope       -->  用于当前页面

HttpServletRequest --> request         -->  requestScope   -->  用于一次请求响应期间

HttpSession             --> session         -->  sessionScope   --> 用于会话期间

ServletContext         --> application  -->  applicationScope ->用于容器启动期间

  如果是url传过来的值，在jsp页面上使用param前缀，param.xxx

get和post请求方式的区别

get:

如果表单中没有method属性或者method属性为get，以及超链接。地址栏输入url才发起get请求

如果使用get请求，表单内容将全部显示在地址栏中

get请求数据长度有限制

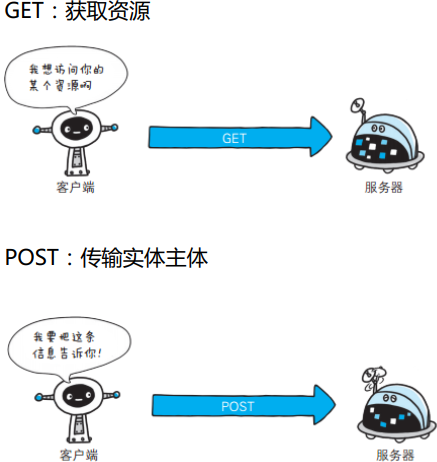
post:

当且仅当from表单的method属性为post的时候，发起post请求

如果使用post请求，则不会显示在地址栏中

大多数情况下，表单使用post请求，一般只有搜索表单使用get请求

post没有字节数限制



#### 请求转发和重定向：

请求转发：req.getRequestDispatchar("/xxx.jsp").forward(req.resp);   客户端的二次请求

·本质：302响应，get请求

流程：浏览器发送http请求-->服务器接受请求并发送302状态吗和新的对应的url到浏览器-->浏览器接收响应并自动请求新的url-->服务器接受请求并群招客户所需的资源响应到浏览器

请求次数：做了两次请求，及产生了两个response对象，重定向会导致request对象丢失

请求资源：可以重定向到任意一个url,而不仅仅是同一个应用。重定向的源组件与目标组件不共用同一个HttpServletRequest对象，因此不能再request范围内共享数据

重定向：resp.sendRedirect("/xxx.jsp");      服务器行为

流程：浏览器发出http请求-->服务器接收请求-->服务器调用内部的一个方法在容器内完成请求处理和转发动作-->将客户所需资源发送到浏览器

请求次数：转发做了一个请求，浏览器的地址栏一直是第一次请求的地址，转发是服务器内部request/response控制权的移交

请求资源：转发的源组件必须在同一个应用汇总，两者可以在request范围内共享数据

### 请求转发跳转和重定向跳转有哪些区别？如何实现？

#### 重定向：

重定向跳转使用 URL 重写的方式进行值的传递,值显示在浏览器地址栏中

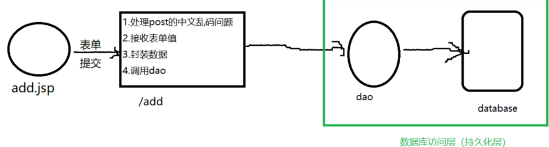
* 重定向适合传递不敏感的数据以及简单的字符串数字等基本数据类型
* 重定向跳转后浏览器的地址显示的是跳转目标的 URL（地址栏发生了改变）
* 重定向不会引起表单的重复提交
* 重定向本质上是服务器产生 302 响应，在消息报文中增加 location 键值对(包含二次请求的地址)，客户端获得 location 的值并向服务器发起第二次请求。
* 实现方式调用 request 的 sendRedirect()方法,不可以通过 setAttribute()设置值

#### 请求转发：

* 请求转发使用HttpServletRequest对象的setAttribute方法进行值传递，值不会显示在地址栏中
* 请求转发跳转传值方式适合传递敏感数据以及对象以及对象、数组、集合等类型的数据
* 请求转发跳转后地址栏不会显示跳转的目标的 URL(地址栏不会发生改变)
* 请求转发跳转会引起表单的重复提交
* WEB-INF 下的 jsp 页面不能被客户端直接访问，只能通过服务器跳转访问，因此访问页面只能通过请求转发跳转访问，不能通过重定向（本质是客户端请求）
* 实现方式是调用 request 的 getRequestDispatch.forward()方法

### 中文乱码问题：

* + **post提交中文乱码：  req.setCharacterEncoding("UTF-8");**
  + **get请求中文乱码 ：new String(fileName.getBytes("ISO8859-1"),"UTF-8");**
  + **解决往前端传值时的中文乱码：传之前是UTF-8，传之后是ISO8859-1**
    - **new String(name.getBytes("UTF-8"),"ISO8859-1");**
* **MVC**
  + 分层  示例



* mvc   示例       
* **MVC定义：**
  + MVC模型指的是模型-视图-控制器（ Model-View-Controller）。 MVC模型有助于将应用程序分割为三个组件，使得程序设计更加的容易，并使各个组件之间的耦合度降到最低。 MVC中三个组件分别为：模型（ Model）、视图（ View）和控制器（ Controller）。
  + 模型   处理业务逻辑

     模型代表应用程序的数据以及用于访问控制和修改这些数据的业务规则

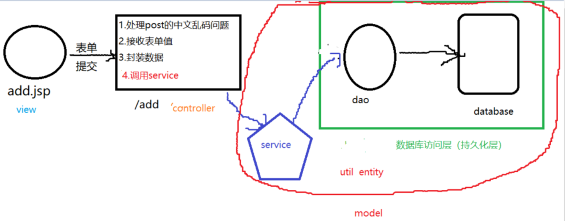
* 视图   显示结果，并收集用户信息  一般是jsp

     视图用来组织模型的内容。视图对象使用对象模型的查询方法以获得数据，并将数据显示给用户

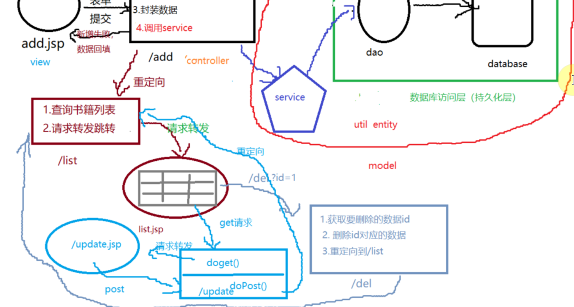
* 控制器  接受view请求，并将请求转交给model处理，最后向客户端做出响应     一般是servlet

    控制器定义了应用程序的行为。它负责对来自视图的用户请求进行解析，并把这些请求映射成相应的行为，这些行为由模型负责实现

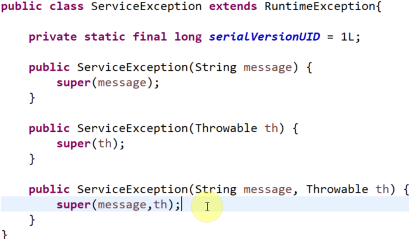
* **服务层 service     用于处理某类的业务逻辑    也属于模型层**

****

* **一个简单的增删改查：**

****

* **定义一个非强制捕获型异常  常用于提示信息**



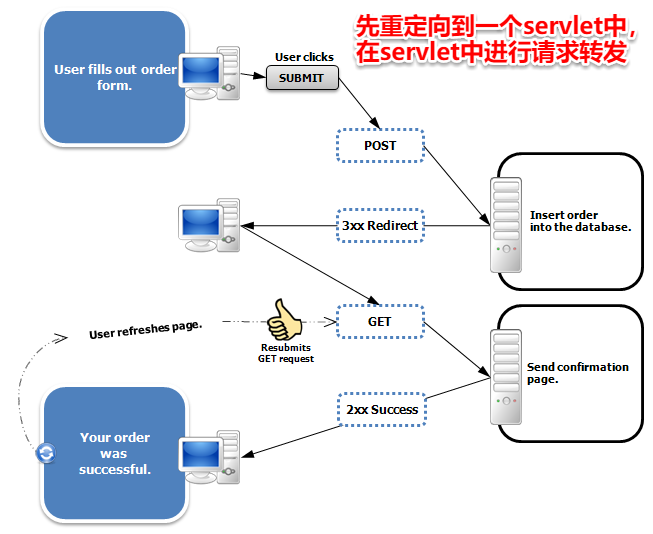
* 定义之后，在其他类的方法中使用：手动抛出异常时，需要在方法签名处添加throws ServlceException 声明要抛出的异常

  throw new ServlceException("sss");   //手动抛出异常    服务层

  在servlet中处理异常，使用请求转发

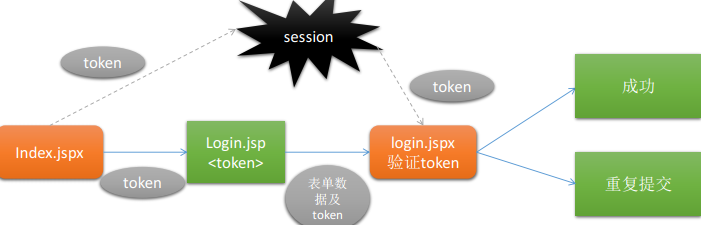


解决表单重复提交的方式

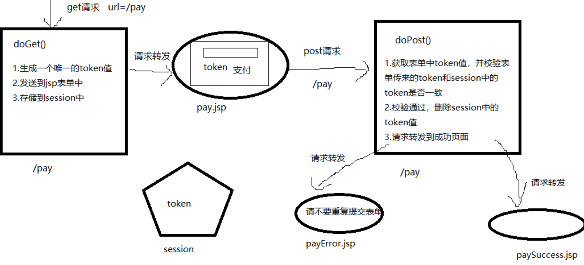
prg(post redirect get)     //重定向相当于发起二次的get请求   

* 1. token(令牌)
     + 步骤:
       1. 接收客户请求，产生Token，并放入session和request中
       2. 在表单中接收Token
       3. 提交表单

处理表单请求前验证Token是否和Session中的一致，如果一致则不是重复提交，删除Session中Token，如果不

一致则提示提交失败      

* 支付举例应用：



* 代码实现:
  + 简述：
    1. java.util.UUID uuid = UUID.randomUUID()  // 获得全球唯一的值，返回一个数
    2. 将之发送到jsp中
    3. 将之发送到session中 然后进行请求转发判断，jsp中的和session中的是否一致，一致后，完成后删除session中的

         表单中的action不写代表从哪转过来就从哪里传过去

* PayServlet.java

    @Override

    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

        // 生成全球唯一的token值,然后转换为字符串

        String token = UUID.randomUUID().toString();

        // 发送到jsp表单中

        req.setAttribute("token", token);

        // session中存储

        HttpSession session = req.getSession();

        session.setAttribute("token", token);

        req.getRequestDispatcher("/WEB-INF/views/pay.jsp").forward(req, resp);

    }

    @Override

    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

        String money = req.getParameter("money");

        String token = req.getParameter("token");

        HttpSession session = req.getSession();

        String sessionToken = (String)session.getAttribute("token");

        if(sessionToken != null && sessionToken.equals(token)) {

            System.out.println("支付了" + money + "元");

            // 删除session中的token值

            session.removeAttribute("token");

            req.getRequestDispatcher("/WEB-INF/views/paySuccess.jsp").forward(req, resp);//正常时跳转到的页面

        } else {

            req.getRequestDispatcher("/WEB-INF/views/payError.jsp").forward(req, resp);//重复提交时跳转的页面

        }

    }

* pay.jsp

<div class="container">

           <form action="/pay" id="payForm" method="post">

<div class="form-group">

  <input type="hidden" name="token" value="${token}"/>

  <input type="text" class="form-control" name="money" id="money"/>

</div>

            <button class="btn btn-primary">支付</button>

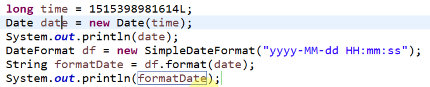
           </form>

       </div>

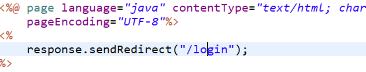
* 在javascript中
  + 需要使用js控制的a标签，href属性的值一律写为javascript:;
  + 使用location.href="url地址"   //修改浏览器中地址栏中的值
  + 通常将id属性值赋值给rel属性值     例：$(".del").attr("rel");   //获得当前对象的值



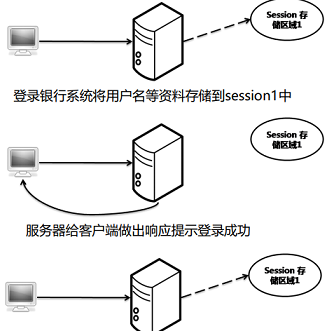
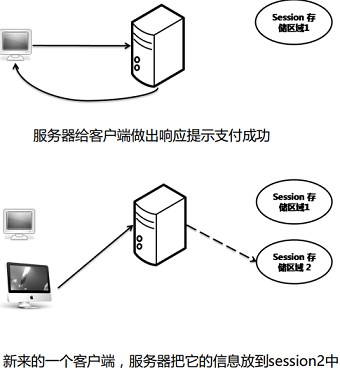
* 插件: commons-lang3-3.5.jar   //使用该插件进行校验传值
  + CharSequence接口，如果该接口作为方法的参数时，可以传入StringBuffer,StringBuilder
  + StringUtils.isNumeric(xxx) //判断xxx是否为纯数字
  + StringUtils.isNotEmpty(xxx)//判断是否为空和是否为空字符串   为空是true
* 路径：
  + 存放在WEB-INF文件夹下的文件对客户端是不可见的  (所有的页面都必须经过servlet，服务器做请求转发，才能访问页面)
* 时间转换格式   long-->日期类型    java.util.Date  java.text.DateFormat   java.text.SimpleDateFormat

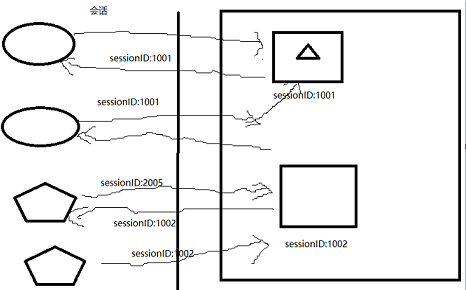


* url传参方式：
  + 使用get调用，只是可以显示的将表单以键值对的形式展示在url中，而post则会隐藏，本质还是传参
* 欢迎页的设置：在WebContent文件夹下新建index.jsp文件

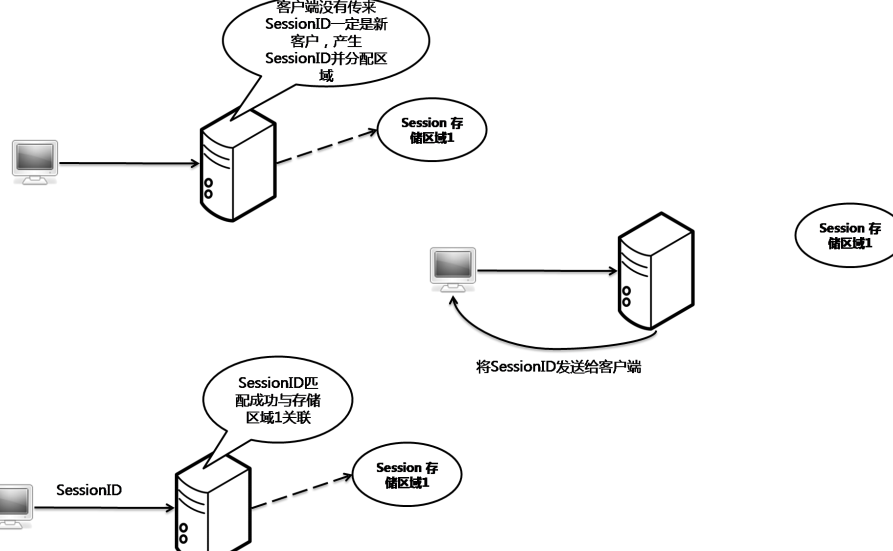


* **Session**



* **sessionID    不同浏览器sessionID不同**



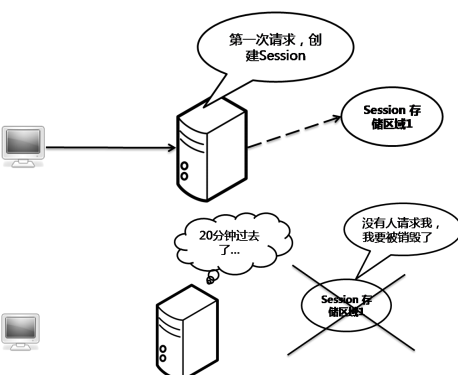
* **session死亡：**
  + 过期（默认30分钟）
* 调用setMaxInactiveInterval()方法设置

     session.setMaxInactiveInterval(2); //单位是秒

* 在web.xml中设置

D:\1 - 副本\Evernote_files\Image [85].png

* 调用invalidate()方法强制其死亡
* Web容器(服务器)关闭或重启



* **session常用方法**

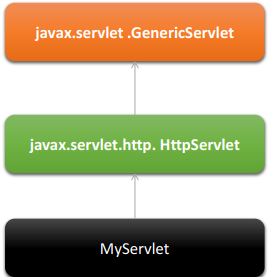
****

* **Servlet深层次学习**
  + **eclipse中学习：tomcat.apache.org/download-80.cgi**
    - 进入网址，选择源文件代码，zip格式下载   下载好后不要解压

D:\1 - 副本\Evernote_files\Image [88].png

* + 将其添加到eclipse中即可
  + ctrl+o可以查看所有类，方法，常量

### Servlet继承体系

****

* + HttpServletRequest接口的唯一父接口是：ServletRequest接口
  + servlet中的doget与dopost是由servlet容器调用(tomcat/jetty)
  + 只用Servlet容器才有权利创建HttpServletRequest/HttpServletResponse接口的实现类对象
  + 大多数ServletRequest接口的实现类对象都可以直接强制类转换为HttpServletRequest,ServletResponse同理
    - 是ServletRequest接口的，强制类型转化只能转为HttpServetRequest,大多数情况下直接转

### Servlet执行过程

1. 客户端发出请求
2. 容器(Tomcat)生成 request 和 response 对象
3. 容器根据 URL 找到合适的 Servlet 并分配线程//多线程-->访问Service方法，同时多个线程进行，一个请求一个线程
4. 访问 service 方法根据请求头决定调用 doGet 或 doPost//service()中获取方法，req.getMethod()请求头,向服务器发起请求，服务器调用Service
5. Servlet 使用响应对象通过容器给客户端做出响应,service 方法执行结束,访问对线程和 request,response 对象被销毁;
6. 注：子类继承父类，那么子类就拥有父类中所有的方法，在使用父类的方法时，就可以和使用子类当中的方法一样，无需调用，直接使用

## 分页

### 分页的原因：

没有必要返回全部内容，客户也看不到

对计算机的损耗也是比较高的

### 分页的实现：

#### ListServlet.java

public class ListServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

        String p = req.getParameter("p");

        //默认页码

        int pageNo = 1;

        if(StringUtils.isNumeric(p)){

            pageNo = Integer.parseInt(p);

        }

        //创建服务层对象

        BookService bookService = new BookService();

     //调用服务层的方法查询数据，返回一个page对象

        Page<Book> page = bookService.findByPage(pageNo);

    //将对象封装到req中

        req.setAttribute("page", page);

        //请求转发到前端页面

        req.getRequestDispatcher("/WEB-INF/views/list.jsp").forward(req, resp);

    }

}

#### BookService.java

    public Page<Book> findByPage(int pageNo){

        int total = bookDao.countBook(); //获得书籍的总数量

        Page<Book> page = new Page<>(total,pageNo);

        List<Book> booklist = bookDao.findByPage(page.getStart(),page.getPageSize());

        page.setItems(booklist);

        return page;

    }

#### page.java  工具类，处理页面的

public class Page<T>{

    private int totalpage;//总数量

    private int pageNo;//当前页码

    private List<T> items;//页面数据

    private int pageSize = 5;

    private int start;

    publlic Page(){}

    public Page(int total, int pageNo){

        totalpage = total/pageSize;   //总页码

        if(total % pageSize != 0){

            totalpage++;

        }

        //解决总数据为0的情况

        if(totalpage < 1){

            totalpage = 1;

        }

        //解决pageNo是0的问题，以及末页的问题

        if(pageNo > totalpage){

  pageNo = totalpage;

 }

        //如果客户端传来的pageNo比1小，则默认显示第一页

        if(pageNo < 1){

            pageNo = 1;

        }

        this.pageNo = pageNo;//当前页码

        this.start = (pageNo - 1) \* pageSize; //起始行号

    }

    //实例属性的getter/setter方法

}

#### BookDao.java

public int countBook(){   //计算总数

    String sql = "select count(\*) from t\_book";  //count()底层实现 返回的是一个long类型

    return DbHelp.query(sql,new ScalarHandler<Long>()).intValue();

}

public List<Book> findByPage(int start,int pageSize){  //根据页码查询数据

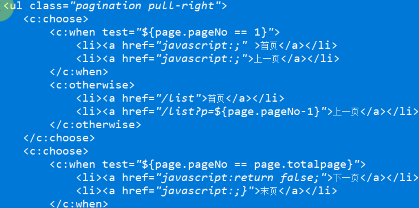
    String sql = "select \* from t\_book limit ?,?";

    return DbHelp.query(sql,new BeanListHandler<>(Book.class),start,pageSize);

}

#### 前端页面：list.jsp

##### 第一种常规方法



##### 第二种，

使用jQuery插件<https://esimakin.github.io/twbs-pagination/>

使用:

先下载zip 解压之后，只需将jquery.tebsPagination.js文件导入项目即可

在页面中添加一个ul标签，然后，使用js控制

<ul id="page" class="pagination-sm"></ul>

<script src="/static/js/jquery-3.2.1.js"></script>

<script src="/static/js/jquery.tebsPagination.js"></script>

<script>

    $(function(){

        $('#page').twbsPagination({

            totalPages: $(page.totalpage), //总页码

        visiblePages: 3, //可显示多少页

            first:'首页',

            last:'末页',

            prev:'上一页',

            next:'下一页',

            href: '/list?p={{number}}'//这里必须是number,根据点击哪个自动获取页码

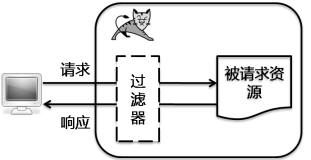
        })

    })

</script>

资料参考：https://gist.github.com/fankay/4449871

## 过滤器  filter



### 什么是Fileter？

Filter 也称之为过滤器，它是 Servlet 技术中最实用的技术，Web 开发人员通过 Filter 技术，对 web 服务器管理的所有 web 资源：例如 Jsp, Servlet, 静态图片文件或静态 html 文件等进行拦截，从而实现一些特殊的功能。例如实现 URL 级别的权限访问控制、过滤敏感词汇、压缩响应信息等一些高级功能。

它主要用于对用户请求进行预处理，也可以对 HttpServletResponse 进行后处理。使用 Filter 的完整流程：Filter 对用户请求进行预处理，接着将请求交给 Servlet 进行处理并生成响应，最后 Filter 再对服务器响应进行后处理。

### 什么是Fileter使用

|  |
| --- |
| import java.io.IOException;  import java.util.Map;  import javax.servlet.Filter;  import javax.servlet.FilterChain;  import javax.servlet.FilterConfig;  import javax.servlet.ServletException;  import javax.servlet.ServletRequest;  import javax.servlet.ServletResponse;  import javax.servlet.annotation.WebFilter;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  /\*\*  \* 使用Filter 打印参数  \* @author Administrator  \*  \*/  public class FilterDemo implements Filter {  public FilterDemo(){  System.*out*.println("FilterDemo 构造函数被执行...");  }  /\*\*  \* 销毁  \*/  public void destroy() {  System.*out*.println("destroy");  }  public void doFilter(ServletRequest paramServletRequest, ServletResponse paramServletResponse,  FilterChain paramFilterChain) throws IOException, ServletException {  System.*out*.println("doFilter");  HttpServletRequest request = (HttpServletRequest) paramServletRequest;  HttpServletResponse response = (HttpServletResponse) paramServletResponse;  // 请求地址  String requestURI = request.getRequestURI();  System.*out*.println("requestURI:"+requestURI);  // 参数  Map<String, String[]> parameterMap = request.getParameterMap();  for (String key : parameterMap.keySet()) {  String[] arr=parameterMap.get(key);  }  }  /\*\*  \* 初始化  \*/  public void init(FilterConfig paramFilterConfig) throws ServletException {  System.*out*.println("init");  }  } |

web.xml配置

|  |
| --- |
| <filter>  <filter-name>FilterDemo</filter-name>  <filter-class>com.itmayiedu.servlet.FilterDemo</filter-class>  </filter>  <filter-mapping>  <filter-name>FilterDemo</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

### 定义过滤器：

创建一个自定义的filter类并实现javax.servlet.Filter接口，并实现其所有抽象方法

public class MyFilter implements Filter {

    //在容器关闭时调用

    public void destroy() {

      //销毁方法

    }

    //在每次请求的时候调用

    public void doFilter(ServletRequest request, ServletResponse response,

             FilterChain chain) throws IOException, ServletException {

        //过滤方法

        //请求过滤

        //FilterChain接口中的doFilter()方法就是让请求经过过滤器，并且是请求过滤和响应过滤的分割

        chain.doFilter(request,response);

        //响应过滤

    }

    //在容器启动时调用

    public void init(FilterConfig arg0) throws ServletException {

        //初始化方法

    }

}

### 过滤器配置：web.xml中注册

<filter>

    <filter-name>myfilter</filter-name>

    <filter-class>com.kaishengit.util.MyFilter</filter-class>

</filter>

<filter-mapping>

    <filter-name>myfilter</filter-name>

    <!--配置拦截的请求  /\*代表所有请求都要经过该过滤器-->

    <url-pattern>/\*</url-pattern>

</filter-mapping>

### 优化过滤器

使用适配器模式：定义一个父类实现Filter接口，并实现所有抽象方法，如果有需要子类必须重写的，将该方法定义为抽象方法，子类继承该父类即可

### 例：登录过滤器

**@Override**

  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)

            throws IOException, ServletException {

        HttpServletRequest req = (HttpServletRequest)request;

        HttpServletResponse resp = (HttpServletResponse) response;

        // 获取请求路径

        String uri = req.getRequestURI();

        // 如果是如下路径资源，则直接通过，不用校验是否登录

        if("/".equals(uri) || "/login".equals(uri) || "/index.jsp".equals(uri)

 || uri.startsWith("/static") ) {

            chain.doFilter(req, resp);

        } else {

            // 如果是诸如/add /list等资源需要判断是否登录，如果session中有admin对象，则已经登录   过，直接通过过滤器

            // 否则视为还没有登录，重定向到登录页面

            HttpSession session = req.getSession();

            Admin admin = (Admin)session.getAttribute("admin");

            if(admin != null) {

 chain.doFilter(req, resp);

            }else {

          //记录当前页面的地址然后赋值给callback属性，然后前端获取url中callback的值，并放入一个类型为hidden的控件中，servlet获取表单中的元素，就可以获取到登录前的地址，这样就可以回到登录前的地址。

 resp.sendRedirect("/login?callback=" + uri);

            }

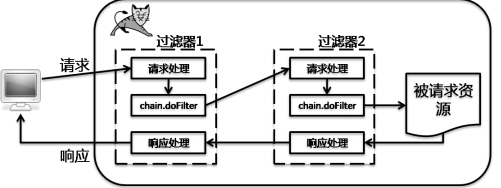
        }

    }

### 过滤器链

过滤器链的顺序和web.xml中配置的顺序相同(从上往下)

配置过滤器时：所有的**字符过滤器必须放在第一位**，其他的依次往后写，，，，，这样会组成过滤器链



### 使用Fileter防止XSS攻击

#### 什么是XSS攻击？

XSS攻击使用Javascript脚本注入进行攻击

例如在表单中注入: <script>location.href='http://www.itmayiedu.com'</script>

实例:

演示:

代码: fromToXss.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>  <form action="XssDemo" method="post">  <input type="text" name="userName"> <input type="submit">  </form>  </body>  </html> |

代码: XssDemo

|  |
| --- |
| **import** java.io.IOException;  **import** javax.servlet.ServletException;  **import** javax.servlet.annotation.WebServlet;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  @WebServlet("/XssDemo")  **public** **class** XssDemo **extends** HttpServlet {  @Override  **protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  String userName = req.getParameter("userName");  req.setAttribute("userName", userName);  req.getRequestDispatcher("showUserName.jsp").forward(req, resp);  }    } |

代码: showUserName.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>userName:${userName}  </body>  </html> |

#### 解決方案

使用Fileter过滤器过滤器注入标签

FilterDemo

|  |
| --- |
| import java.io.IOException;  import java.util.Map;  import javax.servlet.Filter;  import javax.servlet.FilterChain;  import javax.servlet.FilterConfig;  import javax.servlet.ServletException;  import javax.servlet.ServletRequest;  import javax.servlet.ServletResponse;  import javax.servlet.annotation.WebFilter;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  /\*\*  \* 使用Filter 打印参数  \*  \* @author Administrator  \*  \*/  public class FilterDemo implements Filter {  public FilterDemo() {  System.*out*.println("FilterDemo 构造函数被执行...");  }  /\*\*  \* 销毁  \*/  public void destroy() {  System.*out*.println("destroy");  }  public void doFilter(ServletRequest paramServletRequest, ServletResponse paramServletResponse,  FilterChain paramFilterChain) throws IOException, ServletException {  System.*out*.println("doFilter");  HttpServletRequest request = (HttpServletRequest) paramServletRequest;  XssAndSqlHttpServletRequestWrapper xssRequestWrapper = new XssAndSqlHttpServletRequestWrapper(request);  // HttpServletResponse response = (HttpServletResponse)  // paramServletResponse;  // // 请求地址  // String requestURI = request.getRequestURI();  // System.out.println("requestURI:" + requestURI);  // // 参数  // Map<String, String[]> parameterMap = request.getParameterMap();  // for (String key : parameterMap.keySet()) {  // String[] arr = parameterMap.get(key);  // System.out.print("key:");  // for (String string : arr) {  // System.out.println(string);  // }  // }  paramFilterChain.doFilter(xssRequestWrapper, paramServletResponse);  }  /\*\*  \* 初始化  \*/  public void init(FilterConfig paramFilterConfig) throws ServletException {  System.*out*.println("init");  }  } |

**XssAndSqlHttpServletRequestWrapper**

|  |
| --- |
| **import javax.servlet.http.HttpServletRequest;**  **import javax.servlet.http.HttpServletRequestWrapper;**  **import org.apache.commons.lang3.StringEscapeUtils;**  **import org.apache.commons.lang3.StringUtils;**  **/\*\***  **\* 防止XSS攻击**  **\*/**  **public class XssAndSqlHttpServletRequestWrapper extends HttpServletRequestWrapper {**  **HttpServletRequest request;**  **public XssAndSqlHttpServletRequestWrapper(HttpServletRequest request) {**  **super(request);**  **this.request = request;**  **}**  **@Override**  **public String getParameter(String name) {**  **String value = request.getParameter(name);**  **System.*out*.println("name:" + name + "," + value);**  **if (!StringUtils.*isEmpty*(value)) {**  **// 转换Html**  **value = StringEscapeUtils.*escapeHtml4*(value);**  **}**  **return value;**  **}**  **}** |

|  |
| --- |
| **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletRequestWrapper;  **import** org.apache.commons.lang3.StringEscapeUtils;  **import** org.apache.commons.lang3.StringUtils;  /\*\*  \* 防止XSS攻击  \*/  **public** **class** XssAndSqlHttpServletRequestWrapper **extends** HttpServletRequestWrapper {  HttpServletRequest request;  **public** XssAndSqlHttpServletRequestWrapper(HttpServletRequest request) {  **super**(request);  **this**.request = request;  }  @Override  **public** String getParameter(String name) {  String value = request.getParameter(name);  System.***out***.println("name:" + name + "," + value);  **if** (!StringUtils.*isEmpty*(value)) {  // 转换Html  value = StringEscapeUtils.*escapeHtml4*(value);  }  **return** value;  }  } |

作业:使用Cookie实现自动登录效果。

本节课用到maven坐标

|  |
| --- |
| <!-- https://mvnrepository.com/artifact/org.apache.commons/commons-lang3 -->  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-lang3</artifactId>  <version>3.4</version>  </dependency> |

作业题： 完成自动登录。

## 加密 Codec

加密算法的历史背景(黑客)

### 一些名词：

* 拖库:  刚开始无良商家  用户名密码以明文形式存储到数据库
* 撞库：将数据库拖走之后，挨个尝试
* 彩虹表：使用MD5加密的一些常用的简单的明文，以键值对的形式存储进彩虹表中，破解的
* http协议以明文形式传输
* https协议在传输时进行加密，到服务器进行解密   需要购买SSL证书

### 分类：

* 可逆算法：  明文-->密文-->明文
* 不可逆算法： 明文-->密文    用在登录上
* MD5   SHA

第3方插件  加密(apache基金会)  [http://commons.apache.org](http://commons.apache.org/proper/commons-codec/download_codec.cgi)

* 下载  <http://commons.apache.org/proper/commons-codec/download_codec.cgi>
  + 选择commons-codec-1.11-bin.zip进行下载
  + 下载好之后，文件夹选择commons-codec-1.11.jar复制进项目里(一般放进lib文件夹中)

### MD5使用：

加密：String code = DigestUtils.md5Hex("xxx");

生成的密文不管明文的长度为多少一般是32位的

明文可以加盐  salt=""   pass = password+salt 然后再对pass进行加密

盐值的存放位置，一般放在接口中，定义一个常量存放颜值

public interface Config {

  String PASSWORD\_SALT = "#$@#DFSA^%$#asdfas^%^";

}

使用时直接Config.PASSWORD\_SALT即可获取到

* + 啊
* 应用：网盘  MD5秒传  文件内容校验
  + 实现原理：
    - md5加密，密文一样，不需要重复存储
  + 文件内容校验代码实现：

InputStream in1 = new FileInputStream("d:/1.txt");

InputStream in2 = new FileInputStream("d:/2.txt");

String code1 = DigestUtils.md5Hex(in1);

String code2 = DigestUtils.md5Hex(in2);

System.out.println(code1.equals(code2));  //true false

* + SHA使用：
    - 加密：String code = DigestUtils.sha1Hex("xxx")//使用sha1的加密。
      * sha2,sha3,sha4,数字越大，生成的密文越长
* javascript加密库   <https://code.google.com/archive/p/crypto-js/>   谷歌的(需翻墙)
  + 下载：选择版本，下载之后，将cryptojs文件夹拷贝进项目中即可
  + 使用：
    - 先引入js

<script src="/static/js/cryptojs/rollups/md5.js"></script>

* 使用js控制进行加密   宁可不使用盐值，也要保证安全(加密之后，再加盐，之后再加密)

 var password = $("#password").val(); //获取输入的密码明文

 password = CryptoJS.MD5(password); //进行MD5加密

 $("#password").val(password);   //重新赋值

* 啊

## 监听器   listener

分类：

ServletContextListener接口的实现类监听servletContext对象的创建和销毁

### 作用：

常用于容器启动就要执行的代码，且只执行一次

### 自定义监听器类：

import javax.servlet.ServletContext;

import javax.servlet.ServletContextEvent;

import javax.servlet.ServletContextListener;

 public class MyServletContextListener implements ServletContextListener{

    //初始化

    @Override

    public void contextInitialized(ServletContextEvent sce) {

        ServletContext context = sce.getServletContext();//通过ServletContextEvent获取ServletContext对象

        String value = context.getInitParameter("name");//获取web.xml中配置的<context-param>节点中的值

        System.out.println("name-->" + value);

        System.out.println("context init...");

    }

    //销毁

    @Override

    public void contextDestroyed(ServletContextEvent sce) {

        System.out.println("context destroyed...");

    }

}

### 在tomcat中注册，即在web.xml中配置监听器

  <listener> <listener-class>com.kaishengit.web.listener.MyServletContextListener</listener-class></listener>

### 解决硬编码问题：

在配置文件中配置，获取ServletContext的配置文件

    <context-param>        <!--ServletContext的配置文件-->

        <param-name>name</param-name>

        <param-value>root</param-value>

    </context-param>

HttpSessionListener接口的实现类  用于监听HttpSession对象的创建和销毁

### 自定义监听器类：

import javax.servlet.http.HttpSession;

import javax.servlet.http.HttpSessionEvent;

import javax.servlet.http.HttpSessionListener;

public class MyHttpSessionListener implements HttpSessionListener{

    @Override

    public void sessionCreated(HttpSessionEvent se) {

        HttpSession session = se.getSession(); //获取HttpSession对象

        session.setAttribute("name", "jack");

        System.out.println("session create..");

    }

    @Override

    public void sessionDestroyed(HttpSessionEvent se) {

        System.out.println("session destroyed..");

    }

}

### 在tomcat中注册，即在web.xml中配置监听器

<listener><listener-class>com.kaishengit.web.listener.MyHttpSessionListener</listener-class></listener>

* HttpSessionAttributeListener

可以实现此侦听器接口，以便获取对此web应用程序内会话的属性列表所做更改的通知

HttpSessionActivationListener

ServletContextAttributeListener

## 文件上传

文件上传组件分类：apache-commons(fileupload组件)/servlet3.0新特性两种

### fileupload

实现：

apache-commons  **fileupload组件**  <http://commons.apache.org/>

下载   <http://commons.apache.org/proper/commons-fileupload/>

点击fileupload按钮可以阅读文档，也可以选择版本进行下载

<https://gist.github.com/4368621>  github上对组件文档的一些总结

#### 2.1　fileupload概述

fileupload是由apache的commons组件提供的上传组件。它最主要的工作就是帮我们解析request.getInputStream()。

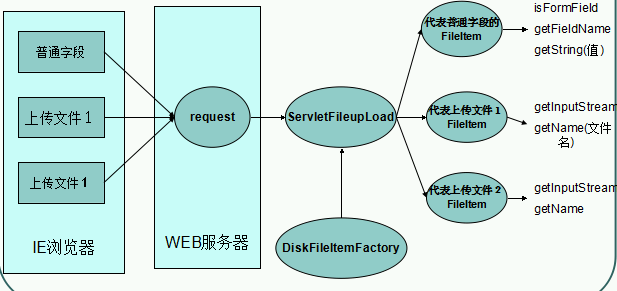
导入commons-fileupload相关jar包

* commons-fileupload.jar，核心包；
* commons-io.jar，依赖包。

#### fileupload的核心类有：

DiskFileItemFactory、ServletFileUpload、FileItem。

a、解析原理



#### fileupload简单应用

#### 导入jar包：

* fileupload.jar  文件上传
* commons-io.jar   文件上传依赖的io流问题 下载过程一样
* commons-lang3.jar  字符串问题
* jstl-1.2.jar   解决EL表达式

#### 前端：

* 文件上传的表单的method属性必须是post
* 文件上传的表单控件使用的是input元素的type属性为file
* 设置文件上传表单的format元素的enctype属性的值为mutilpart/from-data
* action属性可以不写，不写默认还是提交到原servlet中



##### enctype属性

作用：告知服务器请求正文的MIME类型。（请求消息头：Content-Type作用是一致的）

可选值：

* application/x-www-form-urlencoded(默认)：

正文：name=admin&password=123

服务器获取数据：String name = request.getParameter("name");

* multipart/form-data:

正文：



服务器获取数据：**request.getParameter(String)方法获取指定的表单字段字符内容，但文件上传表单已经不在是字符内容，而是字节内容，所以失效。**

文件上传：解析请求正文的每部分的内容。

#### 后端：

使用fileupload组件的步骤如下：

1. 创建工厂类DiskFileItemFactory对象：

DiskFileItemFactory factory = new DiskFileItemFactory()

1. 使用工厂创建解析器对象：

ServletFileUpload fileUpload = new ServletFileUpload(factory)

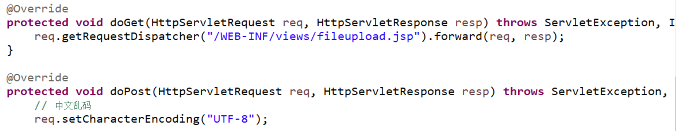
1. 使用解析器来解析request对象：

List<FileItem> list = fileUpload.parseRequest(request)

FileItem对象对应一个表单项（表单字段）。可以是文件字段或普通字段

* boolean isFormField()：判断当前表单字段是否为普通文本字段，如果返回false，说明是文件字段；
* String getFieldName()：获取字段名称，例如：<input type=”text” name=”username”/>，返回的是username；
* String getString()：获取字段的内容，如果是文件字段，那么获取的是文件内容，当然上传的文件必须是文本文件；
* String getName()：获取文件字段的文件名称；（a.txt）
* String getContentType()：获取上传的文件的MIME类型，例如：text/plain。
* int getSize()：获取上传文件的大小；
* InputStream getInputStream()：获取上传文件对应的输入流；
* void write(File)：把上传的文件保存到指定文件中。
* delete();

##### 先通过get请求请求转发到页面

然后post接收请求，先解决中文乱码问题，然后开始文件上传   

##### 文件上传步骤：

**import java.io.File;**

**import java.io.FileOutputStream;**

**import java.io.IOException;**

**import java.io.InputStream;**

**import java.io.OutputStream;**

**import java.util.List;**

**import java.util.UUID;**

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 [org.apache.commons.fileupload.FileItem](http://org.apache.commons.fileupload.fileitem/);

import [org.apache.commons.fileupload.FileUploadException](http://org.apache.commons.fileupload.fileuploadexception/);

import [org.apache.commons.fileupload.disk.DiskFileItemFactory](http://org.apache.commons.fileupload.disk.diskfileitemfactory/);

import [org.apache.commons.fileupload.servlet.ServletFileUpload](http://org.apache.commons.fileupload.servlet.servletfileupload/);

import [org.apache.commons.io.IOUtils](http://org.apache.commons.io.ioutils/);

import [org.apache.commons.lang3.StringUtils](http://org.apache.commons.lang3.stringutils/);

  @Override

    protected void doPost(HttpServletRequest req, HttpServletResponse resp)

             throws ServletException, IOException {

        //将表单的enctype属性设置为multipart/form-data导致无法通过req.getparameter("name")来获取表单的值

       // String desc = req.getParameter("desc");

        //System.out.println("desc:" + desc);

        //1.设置文件上传路径    File类是java.io.File

        File saveDir = new File("d:/upload");

        //2.判断是否存在该路径，不存在就创建出来

        if(!saveDir.exists()) {

            saveDir.mkdir();

        }

        //3.设置临时文件夹路径   在servlet3.0中不需要设置，但在servlet2.5是需要设置临时路径

        File tempDir = new File("d:/temp");

        //4.判断该路径是否存在，不存在的话就去创建

        if(!tempDir.exists()) {

            tempDir.mkdir();

        }

        //5.判断表单是否设置enctype属性为mutilpart/form-data

        if(ServletFileUpload.isMultipartContent(req)) {

            //6.创建工厂    **基础**

            DiskFileItemFactory itemFactory = new DiskFileItemFactory();

            // 工厂的一些属性：设置缓冲区大小

            itemFactory.setSizeThreshold(1024);

            // 工厂的一些属性：设置临时文件夹

            itemFactory.setRepository(tempDir);

            //7.将创建的工程对象传入，创建  ServletFileUpload对象

            ServletFileUpload servletFileUpload = new ServletFileUpload(itemFactory);

            //  ServletFileUpload的属性 设置最大上传文件大小    **以字节为单位**

            servletFileUpload.setSizeMax(1024 \* 1024 \* 10);

            try {

 // 8.获取所有表单元素（包括普通元素和文件元素）包装成FileItem对象的List集合

 List<FileItem> itemList = servletFileUpload.parseRequest(req);

 for(FileItem item : itemList) {

   if(item.isFormField()) {//判断是否为普通元素，满足为普通，不满足为文件元素

       // 普通元素

       System.out.println("fildName:" + item.getFieldName()); // 获取表单name的属性值

       System.out.println("getString:" + item.getString("UTF-8")); // 获取表单普通元素的value值

   } else {

       // 文件元素

       System.out.println("fileName:" + item.getFieldName()); // 获取表单文件元素的name属性值

       System.out.println("name:" + item.getName()); // 获取上传文件的文件名

       //判断上传的文件名是否为空

       if(StringUtils.isNotEmpty(item.getName())) {

           // 获得文件输入流

           InputStream in = item.getInputStream();

           // 重命名文件

           String fileName = item.getName(); // 1.jpg

           //UUID转换为字符串+.jpg

           String newFileName = UUID.randomUUID().toString() +           fileName.substring(fileName.lastIndexOf("."));

           // 获取文件输出流   d:/upload/1.jpg

           OutputStream out = new FileOutputStream(new File(saveDir, newFileName));

          //第二种，使用commons-io工具包实现  输入-->输出

           IOUtils.copy(in, out);

           /\*第一种：小推车  输入-->输出

           byte[] buffer = new byte[1024];

           int len = -1;

           while((len = in.read(buffer)) != -1) {

out.write(buffer, 0, len);

           }\*/

          //关闭

           out.flush();

           out.close();

           in.close();

       }

   }

 }

            } catch (FileUploadException e) {

 throw new RuntimeException("文件上传异常");

            }

        } else {

            throw new RuntimeException("enctype属性异常！");

        }

    }

### 基于servlet3.0新特性

前端不变

后端有区别

@WebServlet("/upload2")

@MultipartConfig   //servlet3.0才有的特性，增加这个注解就可以用简单的方式上传

public class FileuploadServlet2 extends HttpServlet{

        private static final long serialVersionUID = 1L;

    //跳转到上传页面

        @Override

        protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

            //请求转发到页面

            req.getRequestDispatcher("/WEB-INF/views/fileupload2.jsp").forward(req, resp);

        }

        //文件上传，接收表单值

        @Override

        protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

       //1.解决请求的中文乱码问题

         req.setCharacterEncoding("UTF-8");

        // 获得普通表单元素的值，根据表单的name属性值

        //System.out.println(req.getParameter("desc"));

        // 获得文件元素的属性值

        Part part = req.getPart("file1");

        System.out.println("name:" + part.getName()); // name属性值

        System.out.println("contentType:" + part.getContentType()); // 文件MIME类型

        System.out.println("size:" + part.getSize()); // 文件大小   字节数大小

        System.out.println("readSize:" + FileUtils.byteCountToDisplaySize(part.getSize())); //人类可读的文件大小

        System.out.println("submittedFileName:" + part.getSubmittedFileName()); // 上传文件的文件名

        // 2.获得输入流

        InputStream in = part.getInputStream();

        //3. 创建上传文件路径

        File saveDir = new File("d:/upload");

        //4.获取上传文件的文件名

        String fileName = part.getSubmittedFileName(); // 1.jpg

        //5.重命名

        String newFileName = UUID.randomUUID().toString()+ fileName.substring(fileName.lastIndexOf("."));

        //6.获取输出流     new File(父文件，子文件)

        OutputStream out = new FileOutputStream(new File(saveDir,newFileName));

        //7.流的拷贝，输入-->输出

        IOUtils.copy(in, out);

        //8.关闭流

        out.flush();

        out.close();

        in.close();

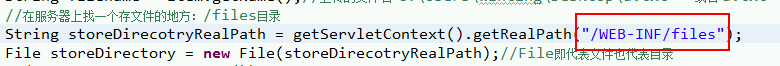
    }

}

### 文件上传时要考虑的几个问题（经验分享）

a、保证服务器的安全

把保存上传文件的目录放在用户直接访问不到的地方。



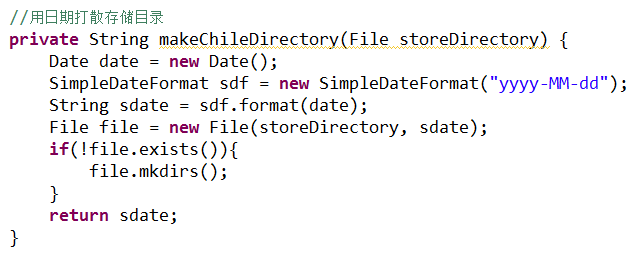
b、避免文件被覆盖

让文件名唯一即可



c、避免同一个文件夹中的文件过多

方案一：按照日期进行打散存储目录



方案二：用文件名的hashCode计算打散的存储目录：二级目录



d、限制文件的大小：web方式不适合上传大的文件

单个文件大小：

ServletFileUpload.setFileSizeMax(字节)

总文件大小：（多文件上传）

ServletFileUpload.setSizeMax(字节)

e、上传字段用户没有上传的问题

通过判断文件名是否为空即可

f、临时文件的问题

DiskFileItemFactory:

作用：产生FileItem对象

内部有一个缓存，缓存大小默认是10Kb。如果上传的文件超过10Kb，用磁盘作为缓存。

存放缓存文件的目录在哪里？默认是系统的临时目录。

如果自己用IO流实现的文件上传，要在流关闭后，清理临时文件。

FileItem.delete();

## 文件下载

本质：从url中获取，get请求跳转到服务器，使用url传值的方式发送键值对，然后获取资源进行 下载

### 代码实现:

* 前端：

<a href="/download?fileName=1.pdf">我的PDF</a>  //谷歌浏览器会直接尝试对简单文件类型打开，

<a href="/download?fileName=a.zip">我的文档</a> //对于复杂文件类型会点击下载，打开一个窗口进行选择

<!--解决下载和预览-->

<a href="/download?fileName=1.jpg&name=我的照片.jpg">我的照片</a>

<a href="/download?fileName=1.jpg">预览我的照片</a>

* 后端：

@WebServlet("/download")

public class DownloadServlet extends HttpServlet{

    private static final long serialVersionUID = 1L;

    @Override

    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

        // 1.获取下载的文件名

        String fileName = req.getParameter("fileName");

        //获取url中的name属性值，用来判断下载/预览，没有name属性值时，不设置mime类型，浏览器尝试打开

        String name = req.getParameter("name"); // 我的照片

        //2.创建下载文件对象，根据文件名去服务器中寻找

        //2.1下载文件夹

        File saveDir = new File("d:/upload");

        //2.2下载文件名

        File file = new File(saveDir, fileName);

        //3.判断该文件是否存在

        if(file.exists()) {

            // 下载文件

            //4. 获取文件输入流

            InputStream in = new FileInputStream(file);

            //5. 获取输出流 (响应输出流)   输出流给响应

            OutputStream out = resp.getOutputStream();

            //5.1判断url中是否带有name属性， name不为空 下载

            if(StringUtils.isNotEmpty(name)) {

             // 5.2设置MIME类型  **application/octet-stream会当成字节流去处理  (默认的下载mime类型)**

**//设置mime类型目的：给客户端一个不认识的类型让其不默认打开而进行下载，服务端认识的类型**

 resp.setContentType("application/octet-stream");

//解决往前端传值时的中文乱码

 name = new String(name.getBytes("UTF-8"),"ISO8859-1");

 //5.3 设置文件名 要下载的文件名与下载好的文件名一样

 resp.addHeader("Content-Disposition", "attachment; filename=\""+name + "\"");

 // 5.4 设置文件总大小

 resp.setContentLengthLong(file.length());

            }

            //6.文件拷贝

            IOUtils.copy(in, out);

            //7.关闭

            out.flush();

            out.close();

            in.close();

        } else {

            resp.sendError(404, "参数异常");

        }

    }

}

* 资料参考：  <https://gist.github.com/fankay/4368629>

## 项目应用

### 记住我的功能实现：

#### 客户端实现

添加复选框(input type="checkbox")并添加name,value,id属性，js控制当点击按钮时触发该事件  if(选中该复选框){设置cookie属性}

默认勾选：html：在input框中增加checked属性    js端：  $("#remember")[0].checked="checked"

先获取复选框对象，判断该框是否被勾选

方法一：if($("#remember")[0].checked){ alert("checked") }  //然后将jQuery对象变成原生js对象

方法二：if($("#remember").is(":checked")){} //使用纯jQuery的方式

如果被勾选了，设置cookie

数据回显，获得input框的对象设置从cookie中获取的值

 <div class="checkbox">

       <label>

           <input type="checkbox" name="rememberme"  id="rememberme"/> 记住帐号

       </label>

   </div>

       $(function(){

            $("#username").val(Cookies.get("username"));//用于回显

            $("#loginBtn").click(function(){

 /\* if($("#rememberme")[0].checked){ //第一种

   alert("checked");

 } \*/

if($("#rememberme").is(":checked")) { //第二种

   Cookies.set("username",$("#username").val(),{ expires: 7, path: '/' }) //设置cookie

 }

            });

        })

#### 服务端实现：

根据该复选框的name属性值，获得复选框的value值，选中复选框代表有值，不为空，未选中代表没值，为空，根据value是否为空，进行增加cookie与删除cookie

为框设置name属性，表单提交时一块提交即可

servlet获取之后，判断该值是否为空即可

    @Override

    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

        Cookie[] cookies = req.getCookies();//获取所有的cookie

        String userName = "";//赋予初始值

        for(Cookie cookie : cookies) {

            if("username".equals(cookie.getName())){

 userName = cookie.getValue();

 break;

            }

        }

        req.setAttribute("username", userName);//将获取到的cookie存进req

        req.getRequestDispatcher("/WEB-INF/views/login.jsp").forward(req, resp);//请求转发到前端

    }

    @Override

    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

            String rememberme = req.getParameter("rememberme"); //获取框中的值

            HttpSession session = req.getSession();

            session.setAttribute("admin", admin);

            if(StringUtils.isNotEmpty(rememberme)) { //如果打勾了设置cookie

 Cookie cookie = new Cookie("username",userName);

 cookie.setDomain("localhost");

 cookie.setPath("/");

 cookie.setMaxAge(60 \* 60 \* 24 \* 30);

 cookie.setHttpOnly(true);

 resp.addCookie(cookie);

            } else {  //当没打勾时删除cookie

 Cookie[] cookies = req.getCookies();

 for(Cookie cookie : cookies) {

   if("username".equals(cookie.getName())) {

       cookie.setDomain("localhost");

       cookie.setPath("/");

       cookie.setMaxAge(0);

       resp.addCookie(cookie);

       break;

   }

 }

            }

    }

## 异常处理

### 异常的处理步骤：

* 将异常转换为非强制捕获型异常
* 将异常向上层抛出，并统一处理   throw new XxxException()
* 在错误页面给出客户友好提示，并通知管理员处理异常信息

### 配置错误页面：

#### 第一种：根据错误码，配置页面   全面

* + web.xml配置

             <error-page>

   <error-code>500</error-code>  //错误码  当抛出500异常时展示错误页

   <location>/WEB-INF/views/error/500.jsp</location> //错误页的路径

            </error-page>

写错误页xxx.jsp   例：500.jsp

#### 第二种： 根据异常，配置页面   细化

web.xml配置

<error-page>

      <exception-type>com.kaishengit.exception.DataAccessException</exception-type>//自定义异常的完全限定名

      <location>/WEB-INF/views/error/dataAccessError.jsp</location> //错误页的异常

   </error-page>

写错误页xxx.jsp   例：dataAccessError.jsp

#### 错误页的其他配置：

在page指令上添加isErrorPage属性且设置为true   标明该jsp为错误页   这时会开启一个内置对象exception

<%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8" isErrorPage="true" %>

在jsp页面获取一些信息(即客户端)

exception.getMessage();  //使用内置对象exception调用方法获取异常信息

javax.servlet.error.status\_code: Integer HTTP协议的状态代码

        例：     Integer code = (Integer)request.getAttribute("javax.servlet.error.status\_code");   //  获取错误码  需要进行强转

* javax.servlet.error.exception\_type: Class 未捕获异常的Class类的对象
* javax.servlet.error.message: String 传递给sendError()方法的消息
* javax.servlet.error.exception: Throwable 调用错误页面的未捕获异常
* javax.servlet.error.request\_uri: String 当前请求的URI
* javax.servlet.error.servlet\_name: String 导致错误页面被调用的Servlet的名字

# 会话

## 生活中会话

我： 小张，你会跳小苹果码？

小张： 会，怎么了？

我： 公司年会上要表演节目，你教教我把

小张：没问题，一顿饭而已。

我： OK。

。。。。。。。。

在这次生活中的会话中产生通话记录（会话数据）

## 软件中的会话

一次会话： 打开浏览器 -> 访问一些服务器内容 -> 关闭浏览器

登录场景：

打开浏览器 -> 浏览到登陆页面 -> 输入用户名和密码 -> 访问到用户主页(显示用户名)

修改密码（输入原密码）

修改收货地址

.......

问题：在此处登录会话过程中产生的数据（用户会话数据）如何保存下来呢？

购物场景：

打开浏览器 -> 浏览商品列表 -> 加入购物车(把商品信息保存下来) -> 关闭浏览器

打开浏览器-> 直接进入购物车 -> 查看到上次加入购物车的商品 -> 下订单 -> 支付

问题： 在购物会话过程中，如何保存商品信息？？

会话管理： 管理浏览器客户端 和 服务器端之间会话过程中产生的会话数据。

域对象： 实现资源之间的数据共享。

request域对象

context域对象

登录场景：

小张： 输入“张三” （保存数据： context.setAttribute("name","张三")） -> 用户主页（显示“张三”）

小李： 输入“李四”(保存数据：context.setAttribute("name","李四")) -> 用户主页（显示“李四”）

问题： context是所有用户公有的资源！！！会覆盖数据。

小张： 输入“张三”（保存数据： request.setAttribute("name","张三")）- > 用户主页（显示“张三”） 问题： 一定要使用转发技术来跳转页面！！！

解决办法： 可以使用session域对象来保存会话数据！！！

## Cookie  缓存

### 作用：

 一些不太重要的信息存放进cookie进行一些操作   一个客户端一个cookie，一个cookie有多个域名

### 特点

Cookie技术：会话数据保存在浏览器客户端。

### Cookie技术核心

Cookie类：用于存储会话数据

1）构造Coo kie对象

Cookie(java.lang.String name, java.lang.String value)

2）设置cookie

void setPath(java.lang.String uri) ：设置cookie的有效访问路径

void setMaxAge(int expiry) ： 设置cookie的有效时间

如果是负数，浏览器关闭就失效，如果是正数，以秒为单位保存。

cookie.setMaxAge(60 \* 60 \* 24);//cookie保存一天，没有一家公司能做到永久保存。

void setValue(java.lang.String newValue) ：设置cookie的值

3）发送cookie到浏览器端保存

void response.addCookie(Cookie cookie) : 发送cookie

4）服务器接收cookie

Cookie[] request.getCookies() : 接收cookie

### Cookie原理

1）服务器创建cookie对象，把会话数据存储到cookie对象中。

new Cookie("name","value");

2） 服务器发送cookie信息到浏览器

response.addCookie(cookie);

举例： set-cookie: name=eric (隐藏发送了一个set-cookie名称的响应头)

3）浏览器得到服务器发送的cookie，然后保存在浏览器端。

4）浏览器在下次访问服务器时，会带着cookie信息

举例： cookie: name=eric (隐藏带着一个叫cookie名称的请求头)

5）服务器接收到浏览器带来的cookie信息

request.getCookies();

### Cookie的细节

1）void setPath(java.lang.String uri) ：设置cookie的有效访问路径。有效路径指的是cookie的有效路径保存在哪里，那么浏览器在有效路径下访问服务器时就会带着cookie信息，否则不带cookie信息。

2）void setMaxAge(int expiry) ： 设置cookie的有效时间。

正整数：表示cookie数据保存浏览器的缓存目录（硬盘中），数值表示保存的时间。

负整数：表示cookie数据保存浏览器的内存中。浏览器关闭cookie就丢失了！！

零：表示删除同名的cookie数据

3）Cookie数据类型只能保存非中文字符串类型的。可以保存多个cookie，但是浏览器一般只允许存放300个Cookie，每个站点最多存放20个Cookie，每个Cookie的大小限制为4KB。

### 使用：

     每访问一次，cookie都会得到更新，重新创建

#### 设置cookie： 服务端使用

Cookie cookie = new Cookie("playid", "1001"); //创建cookie对象并存储键值对

        // 设置常用属性

        // 所属域名  域名没有/ 也可以写ip地址

        cookie.setDomain("localhost");

        //设置路径  根目录

        cookie.setPath("/");

        // 生存周期  以秒为单位   生存周期为负数，为Session cookie，浏览器关闭就消失

        cookie.setMaxAge(60 \* 60 \* 24 \* 30);

        //某些敏感的cookie值，只能在服务端读取，不能在客户端读取，需要在服务端设置加上该属性

        cookie.setHttpOnly(true);//打勾

        resp.addCookie(cookie);//发送到客户端

#### 读取cookie：

        Cookie[] cookies = req.getCookies();

        for(Cookie cookie : cookies) {

            System.out.println(cookie.getName() + "-->" + cookie.getValue());

        }

#### 删除cookie

        Cookie[] cookies = req.getCookies();

        for(Cookie cookie : cookies) {

            if("playid".equals(cookie.getName())) {

 cookie.setDomain("localhost");

 cookie.setPath("/");

 cookie.setMaxAge(0);

 resp.addCookie(cookie);

            }

        }

### jQuery中cookie插件

<https://github.com/js-cookie/js-cookie>    客户端使用

<https://github.com/js-cookie/js-cookie/blob/master/src/js.cookie.js>

选择Raw按钮在浏览器打开，复制全选 ，新建一个文件cookie.js，粘贴即可  2.2.0版本

将该文件拷贝到项目中，依赖jQuery

#### 使用：

#### 先引入js

    <script src="/static/js/jquery-3.2.1.js"></script>

    <script src="/static/js/cookie.js"></script>

#### 客户端设置cookie:

            //键值对           生命周期，天为单位，路径

  Cookies.set('name', 'jack', { expires: 7, path: '/' });

#### 客户端获取cookie：

    //第一种

   var nameValue = Cookies.get('name');

    //第二种

    var cookies = Cookies.get(); //获得cookie数组，cookie既像数组又像对象

    alert(cookies.productId);//使用cookies.key值获得

    alert(cookies.playid);

#### 客户端删除cookie:

        Cookies.remove('productId');//根据key值删除cookie

#### cookie安全相关的文章

[**http://zh.wikipedia.org/zh-cn/Cookie**](http://zh.wikipedia.org/zh-cn/Cookie)

[**http://www.infoq.com/cn/articles/cookie-security**](http://www.infoq.com/cn/articles/cookie-security)

[**http://coolshell.cn/articles/5353.html**](http://coolshell.cn/articles/5353.html)

### 案例- 显示用户上次访问的时间

|  |
| --- |
| @WebServlet("/LastAccessTime")  public class LastAccessTime extends HttpServlet {  @Override  protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  resp.setContentType("text/html;charset=utf-8");// 防止浏览器显示乱码  String lastAccessTime = null;  Cookie[] cookies = req.getCookies();  for (Cookie cookie : cookies) {  String name = cookie.getName();  if (name.equals("lastAccessTime")) {  lastAccessTime = cookie.getValue();  break;  }  }  if (StringUtils.*isEmpty*(lastAccessTime)) {  resp.getWriter().print("您是首次访问!");  } else {  resp.getWriter().print("你上次访问时间:" + lastAccessTime);  }  // 保存访问时间  // 创建cookie 将当前时间作为cookie保存到浏览器  String currenttime = new SimpleDateFormat("yyyy-mm-dd hh:mm:ss").format(new Date());  Cookie cookie = new Cookie("lastAccessTime", currenttime);  cookie.setMaxAge(60 \* 60 \* 24);  // 发送cookie  resp.addCookie(cookie);  }  String addCookie(String key, String value, HttpServletResponse resp) {  return key;  }  } |

## 会话技术

Cookie技术：会话数据保存在浏览器客户端。

Session技术：会话数据保存在服务器端。

## Session技术

### 引入

Cookie的局限：

1）Cookie只能存字符串类型。不能保存对象

2）只能存非中文。

3）1个Cookie的容量不超过4KB。

如果要保存非字符串，超过4kb内容，只能使用session技术！！！

Session特点：

会话数据保存在服务器端。（内存中）

### Session技术核心

HttpSession类：用于保存会话数据

1）创建或得到session对象

HttpSession getSession()

HttpSession getSession(boolean create)

2）设置session对象

void setMaxInactiveInterval(int interval) ： 设置session的有效时间

void invalidate() ： 销毁session对象

java.lang.String getId() ： 得到session编号

3）保存会话数据到session对象

void setAttribute(java.lang.String name, java.lang.Object value) ： 保存数据

java.lang.Object getAttribute(java.lang.String name) ： 获取数据

void removeAttribute(java.lang.String name) ： 清除数据

### Session原理

问题： 服务器能够识别不同的浏览者！！！

现象：

前提： 在哪个session域对象保存数据，就必须从哪个域对象取出！！！！

浏览器1：(给s1分配一个唯一的标记：s001,把s001发送给浏览器)

1）创建session对象，保存会话数据

HttpSession session = request.getSession(); --保存会话数据 s1

浏览器1 的新窗口（带着s001的标记到服务器查询，s001->s1,返回s1）

1）得到session对象的会话数据

HttpSession session = request.getSession(); --可以取出 s1

新的浏览器1：(没有带s001,不能返回s1)

1）得到session对象的会话数据

HttpSession session = request.getSession(); --不可以取出 s2

浏览器2：(没有带s001,不能返回s1)

1）得到session对象的会话数据

HttpSession session = request.getSession(); --不可以取出 s3

代码解读：HttpSession session = request.getSession();

1）第一次访问创建session对象，给session对象分配一个唯一的ID，叫JSESSIONID

new HttpSession();

2）把JSESSIONID作为Cookie的值发送给浏览器保存

Cookie cookie = new Cookie("JSESSIONID", sessionID);

response.addCookie(cookie);

3）第二次访问的时候，浏览器带着JSESSIONID的cookie访问服务器

4）服务器得到JSESSIONID，在服务器的内存中搜索是否存放对应编号的session对象。

if(找到){

return map.get(sessionID);

}

Map<String,HttpSession>]

<"s001", s1>

<"s001,"s2>

5）如果找到对应编号的session对象，直接返回该对象

6）如果找不到对应编号的session对象，创建新的session对象，继续走1的流程

结论：通过JSESSION的cookie值在服务器找session对象！！！！！

### Sesson细节

1）java.lang.String getId() ： 得到session编号

2）两个getSession方法：

getSession(true) / getSession() : 创建或得到session对象。没有匹配的session编号，自动创建新的session对象。

getSession(false): 得到session对象。没有匹配的session编号，返回null

3）void setMaxInactiveInterval(int interval) ： 设置session的有效时间

session对象销毁时间：

3.1 默认情况30分服务器自动回收

3.2 修改session回收时间

3.3 全局修改session有效时间

|  |
| --- |
| <!-- 修改session全局有效时间:分钟 -->  <session-config>  <session-timeout>1</session-timeout>  </session-config> |

3.4.手动销毁session对象

void invalidate() ： 销毁session对象

4）如何避免浏览器的JSESSIONID的cookie随着浏览器关闭而丢失的问题

|  |
| --- |
| /\*\*  \* 手动发送一个硬盘保存的cookie给浏览器  \*/  Cookie c = **new** Cookie("JSESSIONID",session.getId());  c.setMaxAge(60\*60);  response.addCookie(c); |

总结：

1）会话管理： 浏览器和服务器会话过程中的产生的会话数据的管理。

2）Cookie技术：

new Cookie("name","value")

response.addCookie(coookie)

request.getCookies()

3）Session技术

request.getSession();

setAttrbute("name","会话数据");

getAttribute("会话数据")

## 自定义缓存

### 定义缓存实体类

|  |
| --- |
| package com.itmayiedu;  public class Cache {  public Cache(String key, Object value, Long timeOut) {  super();  this.key = key;  this.value = value;  this.timeOut = timeOut;  }  public Cache() {  }  /\*\*  \* key  \*/  private String key;  /\*\*  \* 缓存数据  \*/  private Object value;  /\*\*  \* 超时时间  \*/  private Long timeOut;  public String getKey() {  return key;  }  public void setKey(String key) {  this.key = key;  }  public Object getValue() {  return value;  }  public void setValue(Object value) {  this.value = value;  }  public Long getTimeOut() {  return timeOut;  }  public void setTimeOut(Long timeOut) {  this.timeOut = timeOut;  }  } |

### 定义缓存类

|  |
| --- |
| /\*\*  \*  \* @classDesc: 功能描述:(缓存map)  \* @author: YaoSiyuan  \* @createTime: 2017年9月1日 下午3:19:24  \* @version: v1.0  \* @copyright:  \*/  public class CacheManager {  private Map<String, Cache> cacheMap = new HashMap<>();  /\*\*  \*  \* @methodDesc: 功能描述:(往缓存存值)  \* @author: YaoSiyuan  \* @param: @param  \* key  \* @param: @param  \* oj  \* @createTime:2017年9月1日 下午3:20:58  \* @returnType:@param key  \* @returnType:@param oj void  \* @copyright:  \*/  public void put(String key, Object oj) {  put(key, oj, null);  }  /\*\*  \*  \* @methodDesc: 功能描述:(往缓存存值)  \* @author: YaoSiyuan  \* @param: @param  \* key  \* @param: @param  \* oj  \* @createTime:2017年9月1日 下午3:20:58  \* @returnType:@param key  \* @returnType:@param oj void  \* @copyright:  \*/  public synchronized void put(String key, Object oj, Long timeOut) {  if (oj == null) {  return;  }  Cache cache = new Cache();  cache.setKey(key);  if (timeOut != null)  cache.setTimeOut(timeOut + System.*currentTimeMillis*());  cache.setValue(oj);  cacheMap.put(key, cache);  }  /\*\*  \*  \* @methodDesc: 功能描述:(删除)  \* @author: YaoSiyuan  \* @param: @param  \* key  \* @createTime:2017年9月1日 下午3:26:54  \* @returnType:@param key void  \* @copyright:  \*/  public synchronized void deleteCache(String key) {  cacheMap.remove(key);  }  /\*\*  \*  \* @methodDesc: 功能描述:(获取缓存中数据)  \* @author: YaoSiyuan  \* @param: @param  \* key  \* @param: @return  \* @createTime:2017年9月1日 下午3:24:03  \* @returnType:@param key  \* @returnType:@return Cache  \* @copyright:  \*/  public synchronized Object get(String key) {  Cache cache = cacheMap.get(key);  Object oj = null;  if (cache != null) {  oj = cache.getValue();  }  return oj;  }  /\*\*  \*  \* @methodDesc: 功能描述:(检查数据是否在有效期内)  \* @author: YaoSiyuan  \* @param:  \* @createTime:2017年9月1日 下午3:25:21  \* @returnType: void  \* @copyright:  \*/  public synchronized void checkValidityData() {  for (String key : cacheMap.keySet()) {  Cache cache = cacheMap.get(key);  Long timeOut = cache.getTimeOut();  if (timeOut == null) {  return;  }  long currentTime = System.*currentTimeMillis*();  long endTime = timeOut;  long result = (currentTime - endTime);  if (result > 0) {  System.*out*.println("清除:"+key);  cacheMap.remove(key);  }  }  }  public static void main(String[] args) throws InterruptedException {  CacheManager cacheManager = new CacheManager();  // cacheManager.put("lisi", 0);  cacheManager.put("zhangsan", "jj", 5000l);  ScheduledExecutorService scheduledThreadPool = Executors.*newScheduledThreadPool*(5);  scheduledThreadPool.schedule(new Runnable() {  public void run() {  cacheManager.checkValidityData();  }  }, 5000, TimeUnit.*MILLISECONDS*);  Thread.*sleep*(5000);  System.*out*.println(cacheManager.get("zhangsan"));  }  } |

## 自定义Token

### 5.1 什么是token

token其实就是一个令牌,具有随机性,类似于sessionId。

在对接一些第三方平台的时候,为了能够保证数据安全性,通常会使用一些令牌进行交互

例如: https://mp.weixin.qq.com/wiki?t=resource/res\_main&id=mp1421140183

### 5.2 如何自定义token

token生成规则,只要保证token生成一个不重复的唯一字符串即可。

使用jdk自带的uuid生成规则。

### 5.3 什么是UUID

UUID含义是通用唯一识别码 (Universally Unique Identifier)，这是一个软件建构的标准，也是被开源软件基金会 (Open Software Foundation, OSF)

的组织应用在分布式计算环境 (Distributed Computing Environment, DCE) 领域的一部分。

     UUID 的目的，是让分布式系统中的所有元素，都能有唯一的辨识资讯，而不需要透过中央控制端来做辨识资讯的指定。如此一来，每个人都可以建立不与其它人冲突的 UUID。

在这样的情况下，就不需考虑数据库建立时的名称重复问题。目前最广泛应用的 UUID，即是微软的 Microsoft's Globally Unique Identifiers (GUIDs)，而其他重要的应用，

则有 Linux ext2/ext3 档案系统、LUKS 加密分割区、GNOME、KDE、Mac OS X 等等

### UUID组成

UUID保证对在同一时空中的所有机器都是唯一的。通常平台会提供生成的API。按照开放软件基金会(OSF)制定的标准计算，用到了以太网卡地址、纳秒级时间、芯片ID码和许多可能的数字

UUID由以下几部分的组合：

（1）当前日期和时间，UUID的第一个部分与时间有关，如果你在生成一个UUID之后，过几秒又生成一个UUID，则第一个部分不同，其余相同。

（2）时钟序列。

（3）全局唯一的IEEE机器识别号，如果有网卡，从网卡MAC地址获得，没有网卡以其他方式获得。

UUID的唯一缺陷在于生成的结果串会比较长。关于UUID这个标准使用最普遍的是微软的GUID(Globals Unique Identifiers)。在ColdFusion中可以用CreateUUID()函数很简单地生成UUID，

其格式为：xxxxxxxx-xxxx- xxxx-xxxxxxxxxxxxxxxx(8-4-4-16)，其中每个 x 是 0-9 或 a-f 范围内的一个十六进制的数字。而标准的UUID格式为：xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx (8-4-4-4-12);

### UUID代码

|  |
| --- |
| UUID.randomUUID().toString() |

## 表单重复提交解决方案(防止Http重复提交。)

### 场景模拟

创建一个from.jsp页面

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <title>Form表单</title>  </head>    <body>  <form action="${pageContext.request.contextPath}/DoFormServlet" method="post">  用户名：<input type="text" name="userName">  <input type="submit" value="提交" id="submit">  </form>  </body>  </html> |

DoFormServlet 代码

|  |
| --- |
| import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.annotation.WebServlet;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  @WebServlet("/DoFormServlet")  public class DoFormServlet extends HttpServlet {  @Override  protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  req.setCharacterEncoding("UTF-8");  String userName = req.getParameter("userName");  try {  Thread.*sleep*(300);  } catch (Exception e) {  // TODO: handle exception  }  System.*out*.println("往数据库插入数据...."+userName);  resp.getWriter().write("success");  }  } |

#### 网络延时

在平时开发中，如果网速比较慢的情况下，用户提交表单后，发现服务器半天都没有响应，那么用户可能会以为是自己没有提交表单，就会再点击提交按钮重复提交表单，我们在开发中必须防止表单重复提交。

#### 重新刷新

表单提交后用户点击【刷新】按钮导致表单重复提交

#### 点击浏览器的【后退】按钮回退到表单页面后进行再次提交

用户提交表单后，点击浏览器的【后退】按钮回退到表单页面后进行再次提交

### 解决方案

#### 使用javascript 解决

既然存在上述所说的表单重复提交问题，那么我们就要想办法解决，比较常用的方法是采用JavaScript来防止表单重复提交，具体做法如下：

修改form.jsp页面，添加如下的JavaScript代码来防止表单重复提交

代码:

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <title>Form表单</title>  <script type="text/javascript">  var isFlag = false; //表单是否已经提交标识，默认为false  function submitFlag() {  if (isFlag == false) {  isFlag = true;  return true;  } else {  return false;  }  }  </script>  </head>  <body>  <form action="${pageContext.request.contextPath}/DoFormServlet"  method="post" onsubmit="return submitFlag()">  用户名：<input type="text" name="userName"> <input type="submit"  value="提交" id="submit">  </form>  </body>  </html> |

除了用这种方式之外，经常见的另一种方式就是表单提交之后，将提交按钮设置为不可用，让用户没有机会点击第二次提交按钮，代码如下：

|  |
| --- |
| function dosubmit(){  //获取表单提交按钮  var btnSubmit = document.getElementById("submit");  //将表单提交按钮设置为不可用，这样就可以避免用户再次点击提交按钮  btnSubmit.disabled= "disabled";  //返回true让表单可以正常提交  return true;  } |

#### 使用后端提交解决

对于【场景二】和【场景三】导致表单重复提交的问题，既然客户端无法解决，那么就在服务器端解决，在服务器端解决就需要用到session了。

具体的做法：**在服务器端生成一个唯一的随机标识号，专业术语称为Token(令牌)，同时在当前用户的Session域中保存这个Token。然后将Token发送到客户端的Form表单中，在Form表单中使用隐藏域来存储这个Token，表单提交的时候连同这个Token一起提交到服务器端，然后在服务器端判断客户端提交上来的Token与服务器端生成的Token是否一致，如果不一致，那就是重复提交了，此时服务器端就可以不处理重复提交的表单。如果相同则处理表单提交，处理完后清除当前用户的Session域中存储的标识号。**

在下列情况下，服务器程序将拒绝处理用户提交的表单请求：

1. **存储Session域中的Token(令牌)与表单提交的Token(令牌)不同。**
2. **当前用户的Session中不存在Token(令牌)。**
3. **用户提交的表单数据中没有Token(令牌)。**

转发代码:

|  |
| --- |
| @WebServlet("/ForwardServlet")  public class ForwardServlet extends HttpServlet {  @Override  protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  req.getSession().setAttribute("sesionToken", TokenUtils.*getToken*());  req.getRequestDispatcher("form.jsp").forward(req, resp);  }  } |

转发页面:

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <title>Form表单</title>  </head>  <body>  <form action="${pageContext.request.contextPath}/DoFormServlet"  method="post" onsubmit="return dosubmit()">  <input type="hidden" name="token" value="${sesionToken}"> 用户名：<input type="text"  name="userName"> <input type="submit" value="提交" id="submit">  </form>  </body>  </html> |

后端Java代码:

|  |
| --- |
| /\*\*  \* @description: 功能描述：()  \* @copyright: Copyright (c) 2019  \* @company: 昭阳科技  \* @author:  \* @version: 2.0  \* @date: 2019 2019年3月2日 下午4:16:16  \*/  package com.ysy.servlet;  import java.io.IOException;  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 javax.websocket.Session;  /\*\*  \* @description: TODO (表单servlet)  \* @copyright: Copyright (c) 2019  \* @company: 昭阳科技  \* @author:  \* @version: 2.0  \* @date: 2019 2019年3月2日 下午4:16:16  \*/  @WebServlet("/DoFormServlet")  public class DoFormServlet extends HttpServlet {  @Override  protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  req.setCharacterEncoding("UTF-8");  if (!isSubmit(req)) {  System.out.println("你已经提交了数据，或者tokenu错误");  resp.getWriter().write("你已经提交了数据，或者tokenu错误");  }  String userName = req.getParameter("userName");  try {  Thread.sleep(300);  } catch (Exception e) {  // TODO: handle exception  }  System.out.println("往数据库插入数据...." + userName);  resp.getWriter().write("保存成功");  req.getSession().removeAttribute("sessionToken");  }  public Boolean isSubmit(HttpServletRequest request) {  String parameter = request.getParameter("parameterToken");  Session sessionToken = (Session) request.getSession().getAttribute("sessionToken");  // 判断是否提交  if (sessionToken == null) {  return false;  }  // 判断是否是伪造token  if (!(parameter.equals(sessionToken))) {  return false;  }  return true;  }  } |

|  |
| --- |
| /\*\*  \* @description: 功能描述：()  \* @copyright: Copyright (c) 2019  \* @company: 昭阳科技  \* @author:  \* @version: 2.0  \* @date: 2019 2019年3月2日 下午5:14:12  \*/  package com.ysy.servlet;  import java.io.IOException;  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 javax.servlet.http.HttpSession;  /\*\*  \* @description: 功能描述 ()  \* @copyright: Copyright (c) 2019  \* @company: 昭阳科技  \* @author:  \* @version: 2.0  \* @date: 2019 2019年3月2日 下午5:14:12  \*/  @WebServlet("/local")  public class LocalFromServlet extends HttpServlet {  /\*\*  \* (非 Javadoc) Title: doGet Description: 跳转到from.jsp  \*  \* @param req  \* @param resp  \* @throws ServletException  \* @throws IOException  \* @see javax.servlet.http.HttpServlet#doGet(javax.servlet.http.HttpServletRequest,  \* javax.servlet.http.HttpServletResponse)  \*/  @Override  protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  // 生成Token  String token = TokenUtils.getToken();  HttpSession session = req.getSession();  session.setAttribute("sessionToken", token);  req.getRequestDispatcher("index.jsp").forward(req, resp);  }  } |

# jsp

## EL表达式     ${ }

### 定义：

EL(Expression Language)表达式又称为表达式语言，是JSP中很重要的组成部分。在JSP中使用EL表达式，可以简化对变量和对象的访问。

### 作用：

我们开发人员可以在各种作用域中保存值，然后使用el表达式来访问作用域

### 使用：

* + - EL表达式会根据四种存值空间，从小到大，依次查找，如果都没有找到，则什么也不显示
    - 特定前缀名（用来区分调用哪个）     如果查找特定空间内的值，那么需要加上空间前缀
    - 获取对象的属性值   ${对象名.属性值}  //对象名为存值空间中的key,属性值为对应相应的getXxx方法
    - EL表达式可以方便的调用对象的getXxx()方法，将结果显示到jsp中



* **使用 el 表达式获得 user.name 的实质是什么？**
  + 调用了 user 对象的 getName()方法。

### 算数运算：

            12 + 15 = ${12+15}<br>

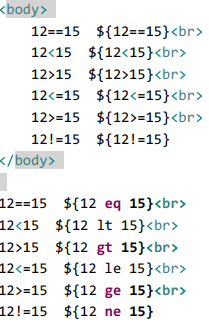
            12 \* 15 = ${12\*15}<br>

           12 - 15 = ${12-15}<br>

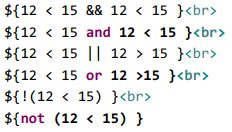
           12 / 15 = ${12/15}<br>

           12 % 15 = ${12%15}<br>

### 关系运算：

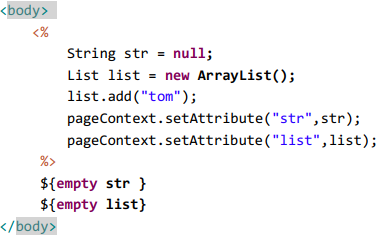


### 逻辑运算：



### empty运算符

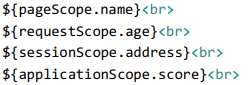
* 判断数组，集合是否为空           not empty 不为空



### 访问对象：



### 对特定空间的访问：



## JSTL   标签库

定义：

* JSTL全称为（ JavaServer Page Standard Tag Libray Java标准标签库），是由SUN公司提供的简化JSP页面设计的标签。 JSTL是由Core、 I18N、 SQL、 XML、 Functions五个核心标签库组成，其中**最重要的是Core标签库**。 **JSTL和EL表达式通常会一起使用**。

使用：引入jar包  jstl-1.2.jar   包含EL

* taglib指令：  使用该指令引入jstl core标签库
  + <%@ taglib prefix="c" uri="<http://java.sun.com/jsp/jstl/core>" %>    //prefix前缀
* 使用：内容需写在花括号内
  + if判断：<c:if test="${条件}"></c:if>
  + if/else判断：

<c:choose><c:when test="${条件}"></c:when>

   <c:otherwise></c:otherwise></c:choose>

* **<c:choose>、 <c:when>和<c:otherwise>一起实现Java语言中的if/else if/else功能。**
* **<c:choose>标签是<c:when>和<c:otherwise>标签的父标签，在<c:choose>标签中只能出现这两个子**标签。
* 迭代list:   <c:forEach items="${迭代对象}"  var="变量名">${n} </c:forEach>



            对数组的迭代： <br/>

           <c:forEach var="str" items="${strs}" varStatus="status">

${status.count}

${str}<br/>

           </c:forEach>

           对List集合的迭代： <br/>

           <c:forEach var="listItem" items="${list}" begin="0" end="1">

${listItem}<br/>

           </c:forEach>

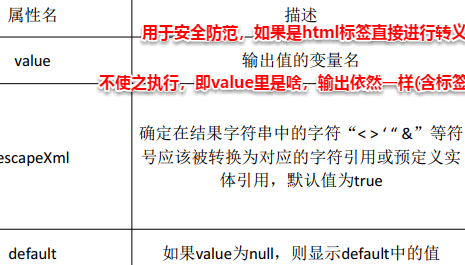
           对Map集合的迭代： <br/>

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

${mapItem.key} : ${mapItem.value}<br/>

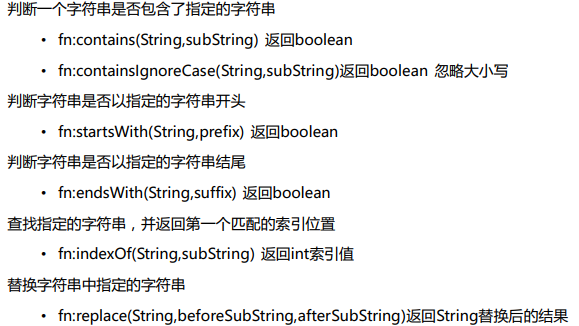
           </c:forEach>

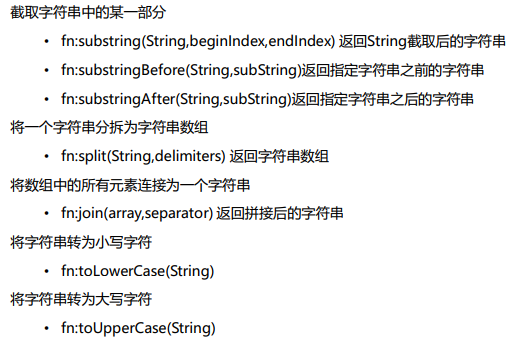
* <c:out>标签用于将空间内的变量进行输出

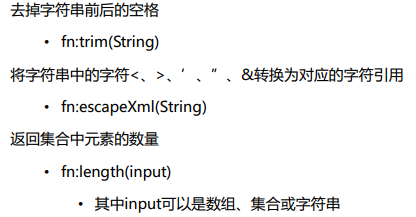


**使用Function标签库**

* 导入jar包
* 使用taglib指令引入
  + <%@ taglib prefix="fn" uri="[http://java.sun.com/jsp/jstl/](http://java.sun.com/jsp/jstl/core)functions" %>
* 使用：







## jsp九大内置对象：

    request  response  pageContext   session  application  out  config  page   exception

## jsp指令；

    jsp指令用于提供整个jsp页面的相关信息以及用于jsp页面与容器之间的通信

    分为：page指令：用于设定整个jsp页面的属性和相关功能  有11个属性

             include指令：表示在jsp编译时插入一个包含文件或者代码的文件，是一种静态包含

              taglib指令：声明jsp文件使用了标签库

## jsp和servlet的区别

        1.jsp在本质上就是servlet，但是俩者的创建方式不一样，servlet完全是java程序代码构成擅长于流程控制和事务处理而通过servlet来生成动态网页；jsp由Html代码和jsp标签构成，可以方便的编写动态网页，因此在实际应用中采用servlet来控制业务流程，而采用jsp来生成动态网页。在struts框架中，jsp位于MVC设计模式的视图层，而servlet位于控制层

        2.jsp是Servlet技术的扩展，本质上就是Servlet的简易方式。jsp编译后是“类Servlet”。Servlet和jsp最主要的不通电在于，servlet的应用逻辑时在java文件中并且完全从表示层中的HTML里分离开来。而jsp的情况是java和HTML可以组合成一个扩展名为.jsp的文件。jsp侧重于视图，servlet主要用于控制逻辑

## 基础

### map遍历

首先需要引入c标签

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

1.集合中为基本的数据类型遍历代码：

<c:forEach items="${maps }" var="item" >

<span>${ item.key}</span>

<span>${ item.value}</span>

</c:forEach>

2.集合中的value为对象，对象中包含id和name两个属性

<c:forEach items="${maps.value }" var="item" >

<span>id:${ item.id}</span>

<span>name:${ item.name}</span>

</c:forEach>

private HashMap<String,CartItem> cartItems=new HashMap<String,CartItem>();

private double total;//总计

<tbody>

<tr>

<th>商品名</th>

<th>个数</th>

<th>小计</th>

</tr>

<c:forEach items="${cart.cartItems}" var="map">

<tr>

<th><c:out value="${map.key}"></c:out></th>

<th><c:out value="${map.value.buyNum}"></c:out></th>

<th><c:out value="${map.value.subTotal}"></c:out></th>

</c:forEach>

</tbody>

### 循环

<c:forEach>标签有如下属性：

|  |  |  |  |
| --- | --- | --- | --- |
| **属性** | **描述** | **是否必要** | **默认值** |
| items | 要被循环的信息 | 否 | 无 |
| begin | 开始的元素（0=第一个元素，1=第二个元素） | 否 | 0 |
| end | 最后一个元素（0=第一个元素，1=第二个元素） | 否 | Last element |
| step | 每一次迭代的步长 | 否 | 1 |
| var | 代表当前条目的变量名称 | 否 | 无 |
| varStatus | 代表循环状态的变量名称 | 否 | 无 |

一般情况下使用items,var即可，varStatus在想获取当前对象下标时使用

#### 后台

public ModelAndView toaddSmallBoard() {

/\*需要获取板块信息\*/

ModelAndView modelAndView=new ModelAndView();

List boards=boardService.getBoards();

modelAndView.addObject("boards",boards);

modelAndView.setViewName("admin/addSmallBoard");

return modelAndView;

}

#### 前台

我这里使用了select下拉框，具体使用情况示情形定

<select class="form-control" name="board.boFaterId">

<c:forEach items="${boards}" var="board" varStatus="id">

<option value="${board.boardId}">${board.boardName} </option>

</c:forEach>

</select>

#### 日期转换

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

<fmt:formatDate value="${msproduct.startTime }" pattern="yyyy-MM-dd HH:mm:ss"/>

## 技巧

### 未登录实现请登录

|  |
| --- |
| <c:choose>  <c:when test="${logUserEmail == null || logUserEmail==''}">  <!-- 顶部未登录 -->  您当前使用的是默认账号  </c:when>  <c:otherwise>  <!-- 顶部已登录 -->  ${logUserEmail }  </c:otherwise>  </c:choose> |

|  |
| --- |
| <c:choose>  <c:when test="${user == null }">  <!-- 顶部未登录 -->  请 <a href="${loginURL }">登陆</a>  </c:when>  <c:otherwise>  <!-- 顶部已登录 -->  欢迎你！，<a href="${user.email }" target="\_blank"><strong>${user.email }</strong></a>  <c:if test="${admin }">  | <a href="NODAY.htm" target="\_blank">管理</a>  </c:if>  | <a href="logout">退出</a>  </c:otherwise>  </c:choose> |

## 面试

### 两种jsp引用的区别

## 常见问题

### Property 'XXX' not found on type java.lang.String解决方案

## 标签指令错误。

原指令标签：

```

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

```改正后的指令标签：

```

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

```

## foreach标签错误

原标签

```

<c:forEach items="XXX" var="xx">

...

</c:forEach>

```

改正后指令

```

<c:forEach items="${XXX}" var="xx">

...

</c:forEach>

```

## 属性值错误

比如${xxx }存在空格也是有可能出错的，总之认真检查。

# 数据库连接池

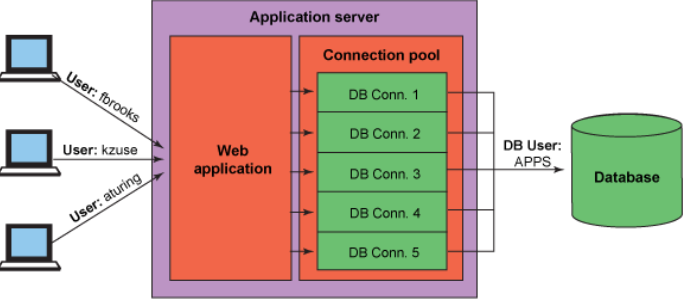
## 数据库连接池   不是javaEE特有，javaSE也可以用

### 使用背景：

池化：在需要的时候去池里面获取，用完之后再放回到池中

数据库连接池   实现共享的原则，需要时从池中获取连接去完成crud,用完之后放回

### 概念图：



### 第三方插件(连接池组件)：

#### DBCP连接池组件：

<http://commons.apache.org/proper/commons-dbcp/>

下载：进入页面之后，点击download按钮  根据jdk版本进行选择下载

例：commons-dbcp2-2.2.0-bin.zip进行下载    2.2~jdk7

下载好之后，打开文件夹，选择commons-dbcp2-2.2.0.jar拷贝进项目中的lib文件夹中

在开发时需要依赖一些jar包，这时需要将其他的也导入才可以

例：dbcp需要依赖

pool的jar包(与dbcp的jar下载过程类似，下载好之后导入即可)

logging的jar包  下载过程与上类似

 创建数据库连接池( 创建一个即可)：BasicDataSource对象    总共需要3个jar包

创建BasicDataSource对象(即为数据库连接池)并设置一些属性   使静态代码块(**确保每次只运行一次，且在获得连接之前创建)**

获取连接  从数据库连接池中获取连接

conn = dataSource.getConnection();

配置文件：解决硬编码问题另一种解决方法  配置文件一般放在resource文件夹中

新建一个xxx.properties文件(后缀名必须是properties)

以键值对的形式存入配置文件中     都是字符串，但不加双引号   等号左右不能有空格

注释以#开头

Properties对象具有map相关的属性和特性，兼具读取properties配置文件的功能

load(类.class.getClassLoader().getResourceAsStream("配置文件的文件名"))需要传进去文件的二进制流，相对路径

优化数据库连接池的代码如下：

Connectionmanager.java

import java.io.IOException;

import java.sql.Connection;

import java.sql.SQLException;

import java.util.Properties;

import javax.sql.DataSource;

import [org.apache.commons.dbcp2.BasicDataSource](http://org.apache.commons.dbcp2.basicdatasource/);

import com.kaishengit.exception.DataAccessException;

public class ConnectionManager {

  //定义变量，其值从配置文件中获取

    private static String DRIVER;

    private static String URL;

    private static String NAME;

    private static String PASSWORD;

  //创建数据库连接池   将BasicDataSource对象提高作用域，以便可以获取连接

    private static BasicDataSource dataSource = new BasicDataSource();

   //创建获取配置文件的Properties对象

     private static Properties prop = new Properties();

    //使用静态代码块，创建数据库连接池，确保数据库连接池只创建一次，且在获取数据库连接前创建

    static {

        try {

            //获取字节流  加载配置文件，以获取一些常量，解决硬编码问题

            prop.load(ConnectionManager.class.getClassLoader().getResourceAsStream("config.properties"));

            //根据配置文件中key值获取value值  getProperty("key值"，"获取不到时的默认值")

            DRIVER = prop.getProperty("jdbc.driver");

            URL = prop.getProperty("jdbc.url");

            NAME = prop.getProperty("jdbc.username","root");

            PASSWORD = prop.getProperty("jdbc.password","rootroot");

        } catch (IOException e) {

            throw new DataAccessException("获取数据库连接配置异常",e);

        }

        //数据库连接池的一些配置

        dataSource.setDriverClassName(DRIVER);

        dataSource.setUrl(URL);

        dataSource.setUsername(NAME);

        dataSource.setPassword(PASSWORD);

        //数据库连接池的其他配置

        dataSource.setInitialSize(5);//初始连接数

        dataSource.setMaxIdle(20);//最大连接数

        dataSource.setMinIdle(5);//最小连接数

        dataSource.setMaxWaitMillis(5000);//连接等待，超过这个连接时间自动断开

    }

       //将数据库连接池看成一个属性，并为其创建get方法，以便于其他类去使用

    public static DataSource getDataSource(){

        return dataSource;

    }

    //获取数据库连接

    public static Connection getConnection() {

        Connection conn = null;

        try {

            //连接从数据库连接池中获取

            conn = dataSource.getConnection();

        } catch (SQLException e) {

            throw new DataAccessException("获取数据库连接异常",e);

        }

        return conn;

    }

}

* DbHelp.java

import java.sql.SQLException;

import org.apache.commons.dbutils.QueryRunner;

import org.apache.commons.dbutils.ResultSetHandler;

import com.kaishengit.exception.DataAccessException;

public class DbHelp {

//获取数据库连接池，将数据库连接池拿过来获取连接，就不需要一个一个去获取连接

//这样在使用该类的update()，以及query()时，就不需要传conn参数了，会自动获取连接

//如果连接是从连接池获得的，close是归还到连接池，而不是断开

private static QueryRunner runner = new QueryRunner(ConnectionManager.getDataSource());

public static void executeUpdate(String sql, Object... params) {

try {

System.out.println("SQL:" + sql);

//update()底层实现：自定义一个布尔值，如果是单个连接，就需要自己销毁，值为false,否则为true

runner.update(sql, params);

} catch (SQLException e) {

throw new DataAccessException("Ö´ÐÐ" + sql + "Òì³£", e);

}

}

public static <T> T query(String sql, ResultSetHandler<T> rsh, Object... params) {

try {

System.out.println("SQL:" + sql);

return runner.query(sql, rsh, params);

} catch (SQLException e) {

throw new DataAccessException("Ö´ÐÐ" + sql + "Òì³£", e);

}

}

}

#### c3p0

<https://sourceforge.net/projects/c3p0/>

##### 代码：

             BasicDataSource dataSource = new BasicDataSource();

       dataSource.setDriverClassName(DRIVER);

       dataSource.setUrl(URL);

       dataSource.setUsername(NAME);

       dataSource.setPassword(PWD);

       dataSource.setInitialSize(5);

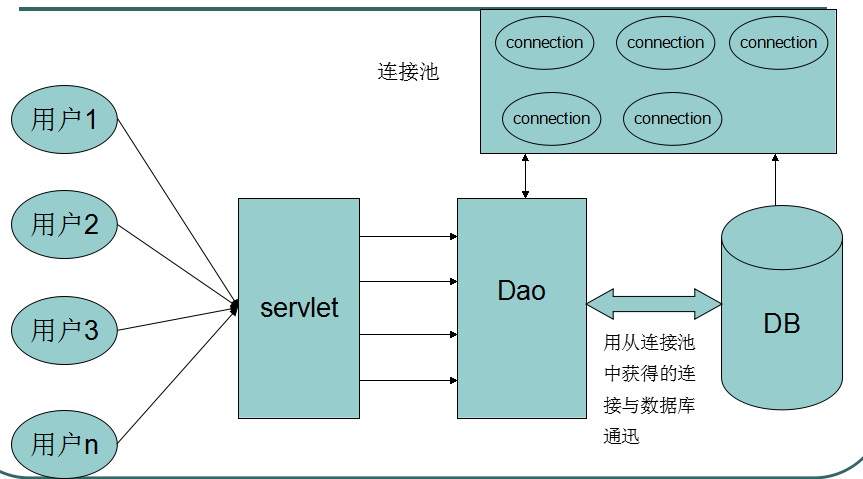
       dataSource.setMaxWait(5000);

       dataSource.setMaxActive(20);

       dataSource.setMinIdle(10);

       conn = dataSource.getConnection();

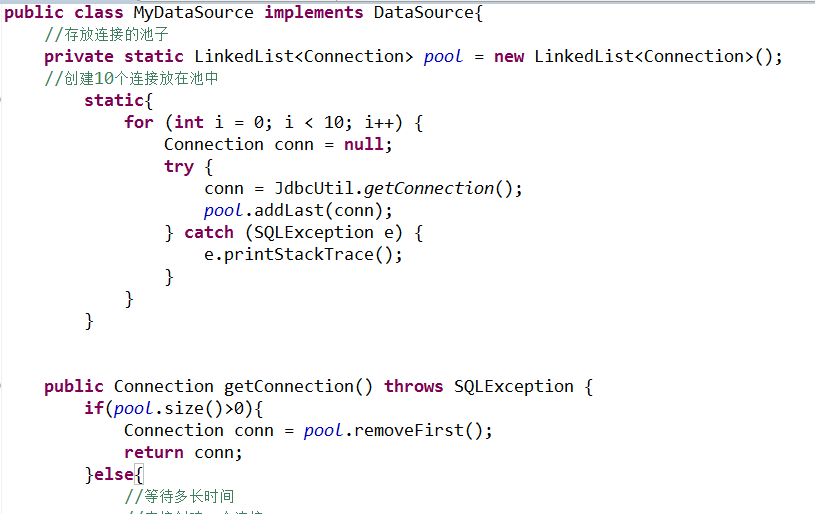
### 连接池原理：（面试）



目的：解决建立数据库连接耗费资源和时间很多的问题，提高性能。

### 编写标准的数据源

自定义数据库连接池要实现javax.sql.DataSource接口，一般都叫数据源。



### 编写数据源时遇到的问题及解决办法

#### 

#### a、装饰设计模式：使用频率很高

目的：改写已存在的类的某个方法或某些方法，装饰设计模式（包装模式）

口诀：

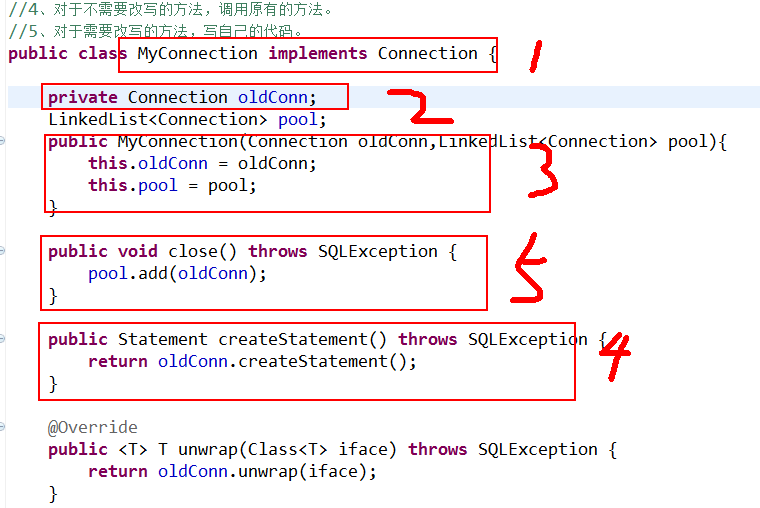
1、编写一个类，实现与被包装类相同的接口。（具备相同的行为）

2、定义一个被包装类类型的变量。

3、定义构造方法，把被包装类的对象注入，给被包装类变量赋值。

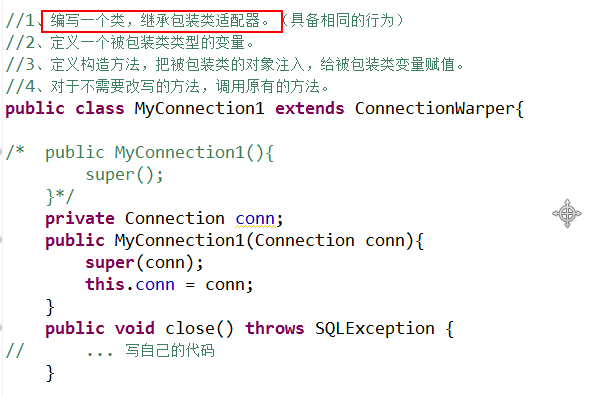
4、对于不需要改写的方法，调用原有的方法。

5、对于需要改写的方法，写自己的代码。



#### b、默认适配器：装饰设计模式一个变体





### 常用的数据源配置

#### 3.1、DBCP

DBCP：Apache推出的Database Connection Pool

使用步骤：

> 添加jar包 commons-dbcp-1.4.jar commons-pool-1.5.6.jar

> 添加属性资源文件

> 编写数据源工具类



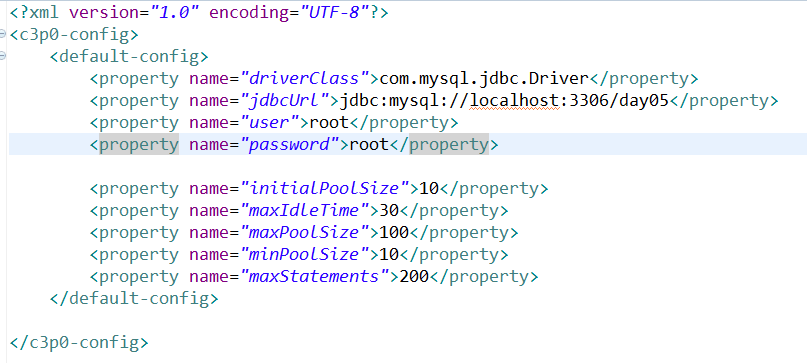
#### 3.2、C3P0

使用步骤：

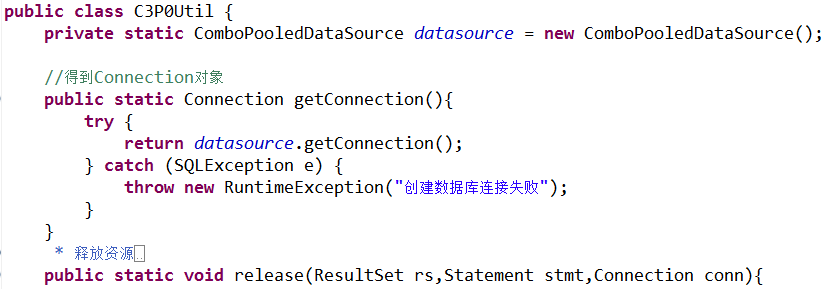
1、添加jar包

2、编写配置文件

c3p0-config.xml，放在classpath中，或classes目录中



3、编写工具类：



### 5、用JavaWeb服务器管理数据源：Tomcat

开发JavaWeb应用，必须使用一个JavaWeb服务器，JavaWeb服务器都内置数据源。

Tomcat：（DBCP）

数据源只需要配置服务器即可。

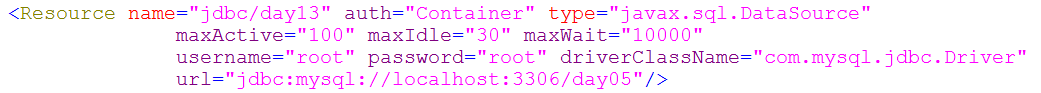
配置数据源的步骤：

1、拷贝数据库连接的jar到tomcatlib目录下

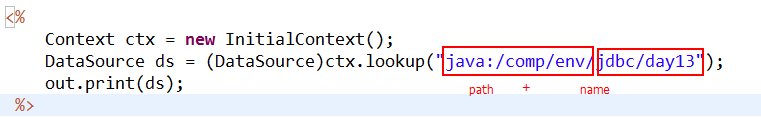
2、配置数据源XML文件

a)如果把配置信息写在tomcat下的conf目录的context.xml中，那么所有应用都能使用此数据源。

b)如果是在当前应用的META-INF中创建context.xml, 编写数据源，那么只有当前应用可以使用。



3、使用连接池



JNDI：java nameing directory interface

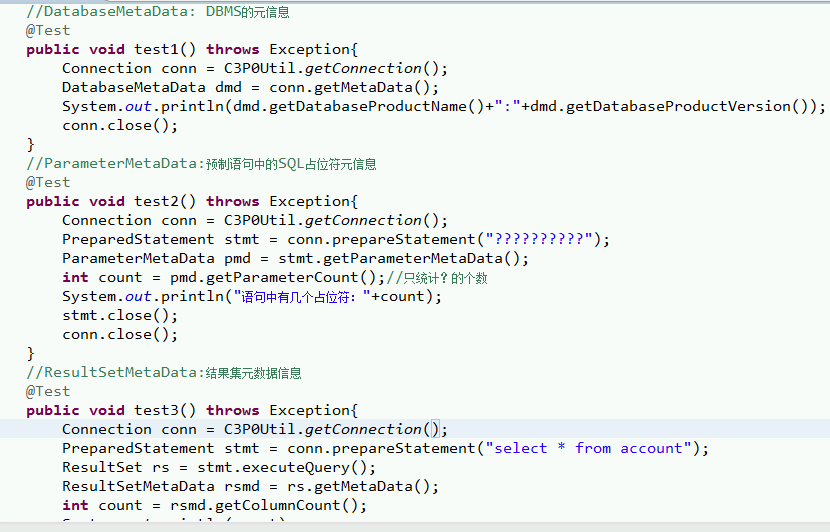
JNDI容器就是一个Map

|  |  |
| --- | --- |
| key（String） | value（Object） |
| path+name | 对象 |
| path+"jdbc/day16" | DataSource对象 |

## 自定义JDBC框架（练技术）

### 数据库元信息的获取（很简单、很无聊、很重要）

a、元信息：（Meta Data）指数据库或表等的定义信息。



### 2、自定义JDBC框架

反射；策略设计模式；