# 初学servlet&jsp

## web简介

### web应用：

运行在网络上的应用程序：

### 网络应用程序分类：

C/S B/S

C/S 客户端 client/服务器 server 典型应用：QQ、YY、LOL

B/S 浏览器 browser/服务器 server 典型应用：sina baidu 等等

C/S的优点：

个性化更容易实现；

更安全；

占用网络资源少；

B/S的优点：

更新方便；

使用方便，到处可用；

几乎不占本地资源；

### 服务器：从两个方面讲；

硬件服务器：就是一台强大的电脑（CPU、内存、储存）。

软件服务器：通过软件提供某类服务：如：数据库服务器，web服务器。

### 浏览器和服务器之间的对话如何开始的？

1. 浏览器要知道服务器的地址和浏览器想要的内容。（请求--URL）。
2. 浏览器和服务器要说同一种语言（http）。

http：hyper text transport protocal 超文本传输控制协议。

服务器：接受浏览器请求，并对浏览器的请求结果响应。

浏览器：向服务器发送请求，并对服务器响应的结果进行展示。

## Tomcat

### web服务器：

接受浏览器请求，进行结果响应。

### 2、常见的web服务器：

Tomcat（Apache），JBoss（不支持servlet和jsp），weblogic，websphere，Resin，jetty

### 3、Tomcat版本：

1）安装版（需要安装jdk）

2）解压版（绿色版）

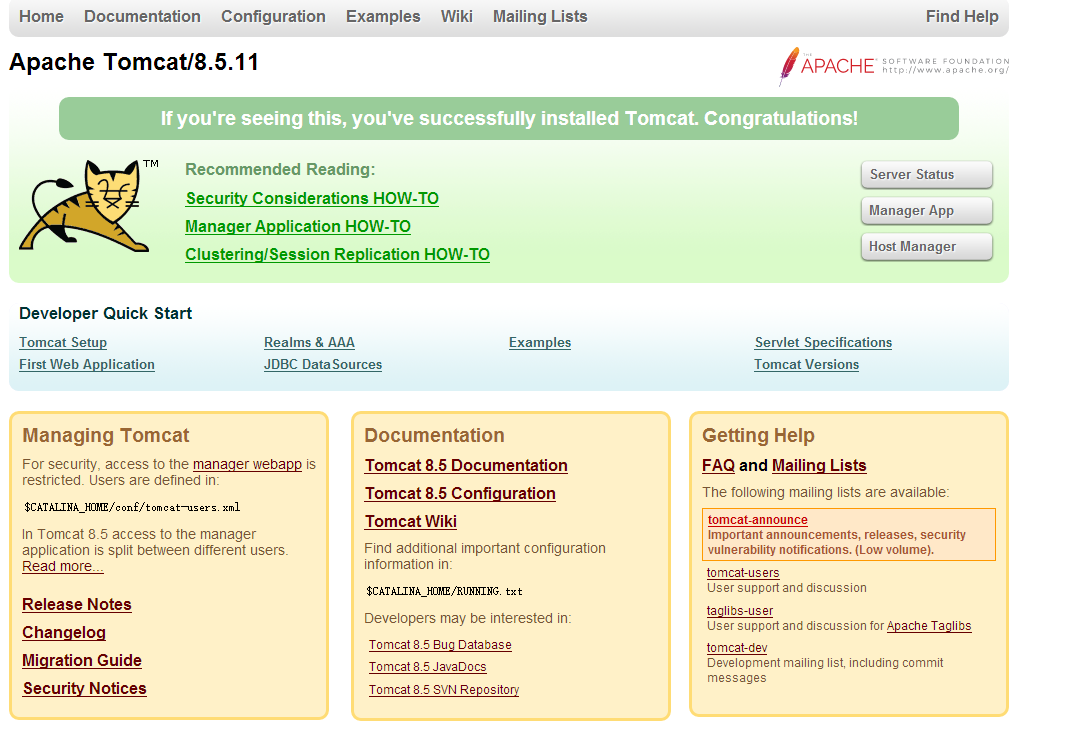
需要配置环境变：JAVA\_HOME

### 4、Tomcat启动：

进入Tomcat\_home\bin\startup.bat

### 验证：在浏览器中输入：<http://localhos:8080>

如果出现如下结果：



则说明安装成功！

### 常见错误：

1. tcnative-1.dll:Can’t load AMD 64-bit.dll on a IA 32-bit plattfore

原因：jdk是32位的。

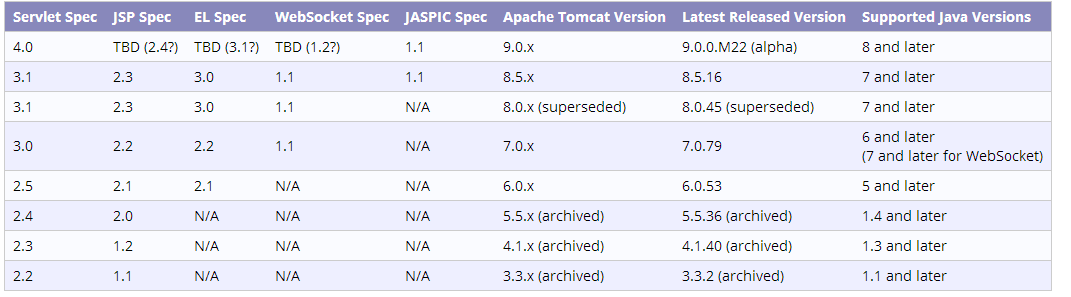
通过java -version检查jdk版本：如果在第三行找不到64说明是32位的。

解决方案：安装64位的jdk，并配置好JAVA-HOME可解决此问题。

1. 执行startup.bat 一闪而过？

原因很多：

检查jdk版本和tomcat版本是否匹配：



如果不匹配可能一闪而过。

如果一闪而过，那么可以在dos命令行去执行startup.bat命令；会将相关信息显示到dos窗口。进而根据dos窗口提示解决问题。

例：jdk8,tomcat8 一闪而过；

Dos窗口提示配置环境变量：CATALINA\_HOME；

配置CATALINA\_HOME：Tomcat\_home（在bin目录的上一级）

1. 正常启动，但是不能访问？

原因：启动后关闭窗口。

解决方案：从新打开，别关闭。

（将Tomcat启动，设为从服务启动。）

## Tomcat部署静态应用

### 1、端口被占用

log中的日志错误：

Failed to initialize end point associated with ProtocolHandler ["ttp-apr-8080"]javalang.Exception: Socketbindfailed: [730048]

原因: 端口号8080 被占用。

解决方案:

A)将tomcat端口号改变。

Tomcat 安装 目 录 :conf server xml 中COUOIAT

<Connectorport="8080" protocol="HTTP 1.1"

comnectionTimeout="20000”

redirectPort="844 3" />

将8080改为其它端口; 端口号可是0-65535 间，建议大于1024

B) 将占用8080 端口的应用关闭。

Cmd: netstat-ano

查看到占用端口的对应应用的进程编号: pidEhm

进入--任务管理器-进程-查看--选择列-进程pid找到后结束对应应用。

### 关闭服务器建议：

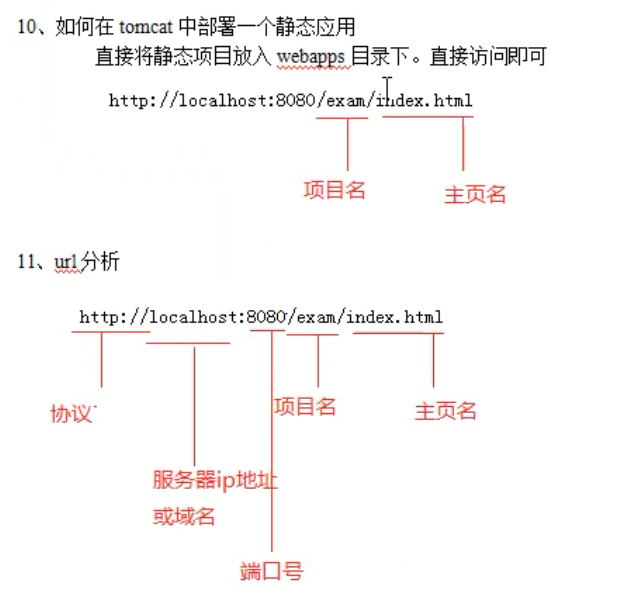
执行bin/shutdown.bat

### 3、分析Tomcat目录结构：



4、Conf里的配置文件:  
Context.xml:配置上下文环境，如JNDI,连接池。  
Server.xml:配置服务器的信息，还有一些应用信息如端口号，虚拟路径  
Web.xml:应用服务的部署文件(查)  
tomcat-users.xml 配置tomcat 用户

### 5、在Tomcat中部署应用



### 6、URL分析

  http协议是在tcp/ip之上。

Localhost域名;

域名就是ip的别名

|  |  |
| --- | --- |
| ip | 域名 |
| 124.192.136.142 | www.jd.com |
| 124.192.136.142 | www.360buy.com |

域名意义就是比里地址，好记。  
  Localhost 的配置:Windows\System32\drivers\etc\hosts 文件  
  端口号: http默认端口是80端口。在开发中一般会指定为其它端口。

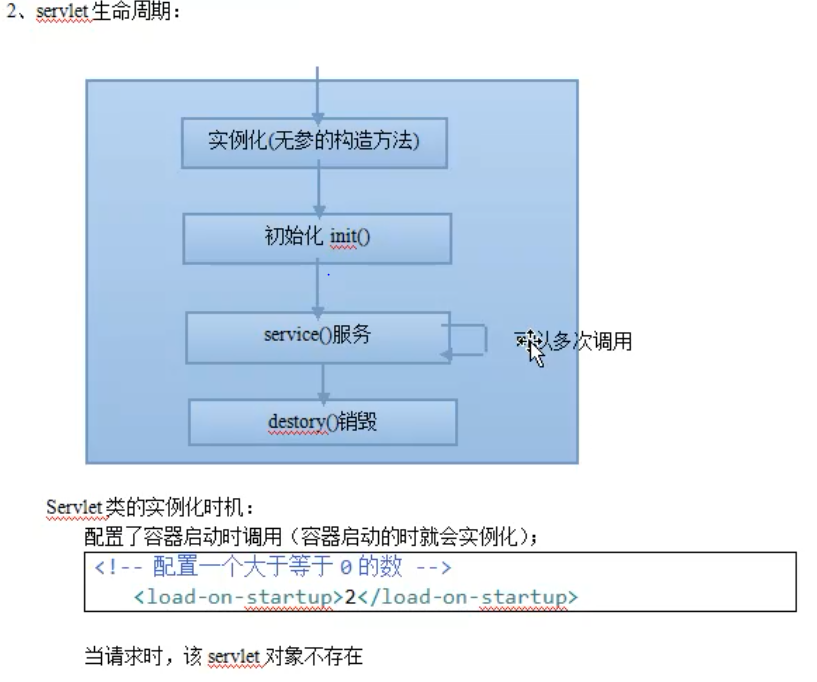
当项目指定为根时，那么不用在url上写项目名称。

主页名(资源名): 可以配置默认资源名:  
   在项目中和加项目部署文件。  
   在项目的根目录添加WEB-INF目录，在WEB-INF目录添加web.xml文件。(web.xm1可以去其他项目拷贝)

### 7、配置虚拟路径

修改: Conf/server.xml  
 在倒数第四行内，在<Host>里配置:  
 <Context path="/" docBase="D:\exam"/>  
Path 指名访问路径，docBase 指名项目存放位置。  
清理浏览器缓存: ctrl+shift+delete

## servlet生命周期



值越小，优先级越高。

## 乱码解决

### 1、提交数据：

Post提交 设置：req.setCharacterEncoding(“utf-8”);

get提交 需要转换：

|  |
| --- |
| String name = req.getParameter(“name”);  Name = new String(name.getBytes(“iso-8859-1”),”utf-8”); |

还可以设置Tomcat：

|  |
| --- |
| <Connector port="8080" protocol="HTTP/1.1"  connectionTimeout="20000"  redirectPort="8443" URIEncoding=”utf-8” /> |

### 2、响应数据：

|  |
| --- |
| //设置的在网络传递编码，默认为iso-8859-1  resp.setCharacterEncoding("utf-8");  //设置浏览器  resp.setContentType("text/html;charset=utf-8"); |

解决不同浏览器编码不一致的问题：

|  |
| --- |
| <Connector port="8080" protocol="HTTP/1.1"  connectionTimeout="20000"  redirectPort="8443" URIEncoding=”utf-8” useBodyEncodingForURI=”ture” /> |

## HTTPServletRequest其他方法

|  |
| --- |
| //得到请求头中的信息  Enumeration<String> en = req.getHeaderNames();  **while**(en.hasMoreElements()) {  String name = en.nextElement();  out.print(name+":"+req.getHeader(name)+"<br/>");  }      System.***out***.println("context:"+req.getContextPath());  System.***out***.println("uri:"+req.getRequestURI());  System.***out***.println("url:"+req.getRequestURL());  System.***out***.println("Host:"+req.getRemoteHost());  System.***out***.println("ip:"+req.getRemoteAddr()); |

## Session（HttpSession）

### Session是浏览器和服务器之间的一次会话，包含多个请求。

(会话结束：关闭浏览器、请求超时)

### Session是服务器为每个客户在服务器端开辟的一块空间。

服务器

A

SessionID：00111

Username:zhangsan

Object:Book

SessionID：00112

Username:lisi

B

SessionID：01212

Username:wangwu

C

### 获取Session：req.getSession();

### 向Session放数值

|  |
| --- |
| Session setAttribute(“key”,”value”); |

### 从Session中取值

|  |
| --- |
| HttpSession session = req.getSession();  String name = (String)sesion.getAttribute(“username”); |

### 从session中移值

|  |
| --- |
| Session.removeAttribute(key); |

### 什么时候用Session？

如果每个用户都有自己不同的数据的时候，就要用session

### 8、登录流程

Login.html

用户名密码错误

LoginServlet{

验证用户名密码是否正确

}

用户名密码正确

获取session将用户信息放入session中

主页

### 9、session和浏览器，服务器都有关，那么session是存放在哪里的？

服务器。

### 10、每个用户都有自己的session，服务器是根据sessionId来区分的。那么服务器是如何知道不同客户的sessionId的呢？

**客户端和服务器“第一次”建立请求是，服务器为给客户分配一个session,并且指派一个sessionId当服务器响应客户端时，会将这个SeesionId以cookie的方式写入浏览器内存中，当给客户再一次发起请求时，该将sessionId传给服务器，服务器根据接收到的sessionId，去查找该客户对应的session。**

### 11、Session建立的时机：

Request.getSession();(一般请求servlet/jsp时)

### 12、浏览器关闭时，session是否被删除？

不会。

### 13、浏览器关闭时，是否能找到原来的session

不能。

### 14、session何时被删除？

1. 超时；
2. 设置session失效时间

|  |
| --- |
| //指定servlet容器使此会话失效之前客户端请求之间的时间间隔，以秒为单位，负数时间指示会话永远不会超时。  session.setMaxInactiveInterval(10\*60); |

b)手动调用Session.invalidate();//使session失效

c)服务器关闭或停止

## Cookie

### cookie小甜饼：

服务器端将**文本**信息（session可以存对象）存储在**客户端**。

模型：  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
 ：Cookie是以name:value存放的，都是String类型

C

B

A

C:sessionId

B:sessionId

A:sessionId

Server

### 服务器如何将数据写入客户端（cookie）

|  |
| --- |
| //新建cookie对象  Cookie ck = new Cookie(“name”,”siggy”);  //通过response对象，将cookie写入客户端  response.addCookie(ck); |

### Cookie不去设置失效时间，默认用完删除。

|  |
| --- |
| setMaxAge(int expiry) |

设置cookie的失效时间，如果等于零，那么是删除。如果小于零，那么不存储。如果大于零，是设置的秒数。

当不设置的时候，储存在本地磁盘。

当setMaxAge(-1);不存储的指的是不存磁盘，但是在浏览器中存在。当关闭浏览器就没有了。

当等于零的时候，cookie会回写到客户端，但是会被立即删除，下一次请求将不会把该cookie携带到请求中。

当大于零的时候，会存在磁盘上，当浏览器关闭，下一次请求时一样会携带到服务器上。

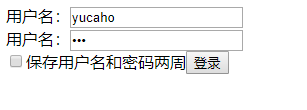
### 取cookie中的值

|  |
| --- |
| //获取cookie值  Cookie[] cks = req.getCookie();  If(cks!=null)  for(Cookie ck:cks){  System.out.println(ck.getName()+”----”+ck.getValue());  } |

Cookie的name和value。会随请求传到服务器端。

Cookie的其他信息有浏览器管理。

### Cookie应用



CookieLogin:获取Cookie值

|  |
| --- |
| **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  req.setCharacterEncoding("utf-8");  resp.setCharacterEncoding("utf-8");  resp.setContentType("text/html;charset=utf-8");  PrintWriter out = resp.getWriter();  String name = **null**;  String password = **null**;  Cookie cks[] = req.getCookies();  **if**(cks != **null**)  **for**(Cookie ck:cks) {  **if**("name".equals(ck.getName()))  name = ck.getValue();  **if**("password".equals(ck.getName()))  password = ck.getValue();  }  out.print("<html>");  out.print("<head>");  out.print("</head>");  out.print("<body>");  out.print("<form action='Welcome' method='post'>");  out.print("用户名：<input type='text' name='name' value='"+(name!=**null**?name:"")+"'>");  out.print("<br/>");  out.print("用户名：<input type='password' name='password' value='"+(password!=**null**?password:"")+"'>");  out.print("<br/>");  out.print("<input type='checkbox' name='remember' value='1'/>保存用户名和密码两周");  out.print("<input type='submit' value='登录'>");  out.print("</form>");  out.print("</body>");  out.print("</html>");  } |

AddCookie:在浏览器中写入Cookie

|  |
| --- |
| **protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  req.setCharacterEncoding("utf-8");  resp.setCharacterEncoding("utf-8");  resp.setContentType("text/html;charset=utf-8");  PrintWriter out = resp.getWriter();  String remember = req.getParameter("remember");  **if**("1".equals(remember))  {  Cookie ck = **new** Cookie("name",req.getParameter("name"));  Cookie ck2 = **new** Cookie("password",req.getParameter("password"));  ck.setMaxAge(2\*7\*24\*3600);  ck.setMaxAge(2\*7\*24\*3600);  resp.addCookie(ck);  resp.addCookie(ck2);  out.print("Cookie写入本地磁盘成功！！！");  } **else**  {  out.print("Cookie写入本地磁盘失败！！！");  }  } |

### Cookie覆盖

cookie没有修改，如果想修改某个cookie的值，只需要新建一个name相同的cookie，即可将原来的覆盖，cookie也不需要删除，当cookie到期由浏览器来维护。

### Cookie是带有domain（域名）的，每个web应用只能读取自己的cookie,cookie和浏览器有关，不同浏览器cookie是不能共享的。

### Cookie是由浏览器管理，cookie是可以被禁用的。

### Cookie vs session

Cookie可以存放大量的文本数据。

安全性低，效率低

Session数据小，安全性高，效率高（可以存对象）。

存student stu = new student(“张安”，”男”,23);

Session.setAttribute(“stu”,stu);

Cookie ck = new(“student”,”1\_张安\_男\_23”);

### cookie的中文处理

Cookie写入浏览器，编码：

|  |
| --- |
| **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  req.setCharacterEncoding("utf-8");  resp.setCharacterEncoding("utf-8");  Cookie ck = **new** Cookie(URLEncoder.*encode*("姓名","utf-8"),URLEncoder.*encode*("喻超","utf-8"));  resp.addCookie(ck);  resp.sendRedirect("index.jsp");  } |

Cookie取出显示，解码：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*,java.net.\*"* 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=ISO-8859-1"*>  <title>Insert title here</title>  </head>  <body>  <%  Cookie[] cks = request.getCookies();  **for**(Cookie ck:cks){  out.print(URLDecoder.decode(ck.getName(),"utf-8")+"-----"+URLDecoder.decode(ck.getValue(),"utf-8")+"<br/>");  }  %>  </body>  </html> |

## ServletContext

### ServletContext是服务器端一块公共的存储区域，可供所有开户存储数据，全局区域。

一个web应用程序只有一个ServletContext。

### 在servlet中获取ServletContext

|  |
| --- |
| ServletContext servletContext = **this**.getServletContext();  ServletContext servletContext1 = req.getSession().getServletContext(); |

### 设值、取值、删除

|  |
| --- |
| servletContext.setAttribute("name", "books");  servletContext.getAttribute("name");  servletContext.removeAttribute("name"); |

### 用处：

* 1. 可以读取全局配置文件

|  |
| --- |
| <context-param>  <param-name>name</param-name>  <param-value>yuchao</param-value>  </context-param> |

Servlet中代码：

|  |
| --- |
| ServletContext ct = **this**.getServletContext();  String value = ct.getInitParameter("name"); |

b) 读取资源文件

资源文件放于根目录下 和WEB-INF同级

|  |
| --- |
| InputStream in = ct.getResourceAsStream("db.properties");  Properties pro = **new** Properties();  pro.load(in);  out.print(pro.getProperty("driver")+"<br/>");  out.print(pro.getProperty("url")+"<br/>"); |

如果资源在src下：

|  |
| --- |
| //类加载器  InputStream in = MyServletContext.**class**.getClassLoader().getResourceAsStream("db.properties");  Properties pro = **new** Properties();  pro.load(in);  out.print(pro.getProperty("driver")+"<br/>");  out.print(pro.getProperty("url")+"<br/>"); |

c) 作为全局变量的运用；网站计数器，网络聊天室

### servletContext生命周期，和服务器同生共死。

### Request、Session、ServletContext/aoolication

存放时间长短：

ServletContext最长，和服务器一样

Session次子，在一个会话范围（失效时间）

Request 请求范围

## JSP

### jsp:java servlet page，动态页面：指数据是动态改变的。

### 动态页面技术：

Jsp应用广

PHP(php+apache+mysql+linux(全免费))

asp.net（微软，学习成本低，容易入门）

asp(过时)

### jsp发展：

Applet --> java se --> servlet --> jsp --> frmework(框架ssh)

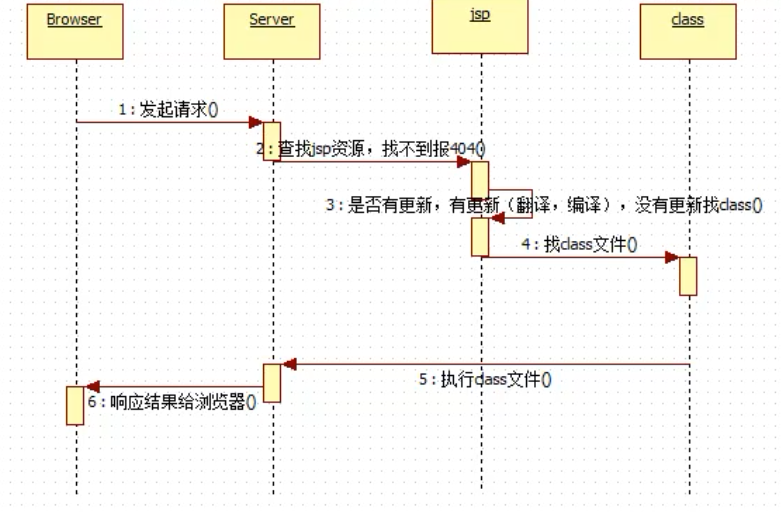
### hello jsp：

|  |
| --- |
| <%@ 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>  <h1>hello jsp</h1>  </body>  </html> |

### 分析：

Jsp本质上是一个servlet。

执行流程：



1. jsp包含的部分：

三大指令，六大动作，九大对象

表达式 <%= %>

注释：

<%--注释内容--%>

Java注释 单行 双行

HTML注释 <!-- -->

html

javascript

css

Jsp

标签

方法声明 <%! %>

java小脚本 <% %>

指令 <%@ %>

### 案例：

|  |
| --- |
| <body>  <%  String name = "张三";  %>  <%!  String getName(){  **return** "welcome!!!";  }  %>  <h1>hello:<%=name %></h1>  <h2><%=getName() %></h2>  </body> |

### 是否可以不用servlet就可以实现功能？

可以。

### Jsp九大内置对象：

request 请求

response 响应对象

session 会话

application 相当于servletContext

out 输出

config 取配置信息 servletconfig

pageContext

page 页面对象

exception 异常对象

### 静态导入指令：

<%@include file=”validate.jsp”%>

将被导入页面和导入页面，合在一起进行翻译，编译，最后产生一个servlet，那么两个页面的变量名不能重复。

### jsp动态导入

<jsp:include page=”validate.jsp”%><jsp:include

动态导入，被导入页面和导入页面分别翻译，编译，产生两个servlet，所以两个页面的变量名可以重复。都会被执行。

### 静态导入和动态导入：

静态导入只会执行一次，动态导入执行两次。那么如果导入页面需要用被导入页面的结果来判断是否执行，那么应该用静态导入。

## 软件架构模式

### 模式：经典的模型。

### Javabean Java类。

分为两类情况：实体bean，业务bean

实体bean：实体类，特指属性是private修饰，然后提供get/set方法。不提供业务方法。

Pojo:Plain Ordinary Java Object

业务bean：除实体bean以外，都是业务bean。

### jsp-javabean----modelⅠ模式

特点：效率高，逻辑混乱，适合小项目

### ModelⅡ模式----jsp+servlet+javabean

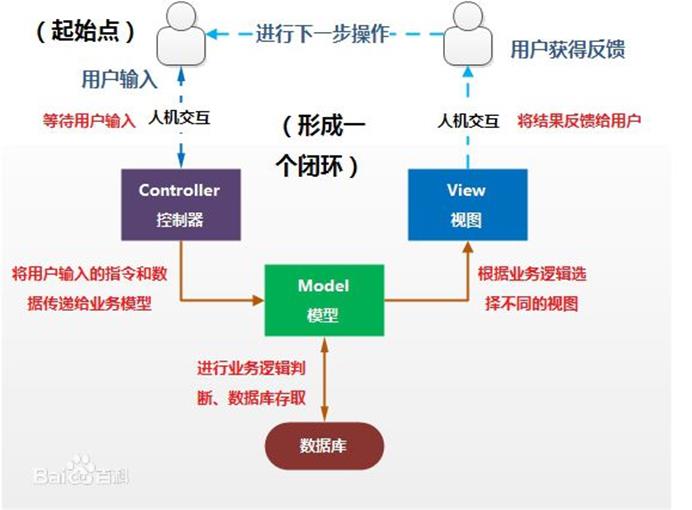
Jsp----数据展示

Servlet----逻辑控制

Javabean----业务处理

MVC设计模式：Model（模型），View（视图）,Controller（控制器）

好处：便于分工，适合大项目，益于维护和扩展



### 使用mvc设计模式以后的开发要点：

1. jsp只做数据的展示，尽量不写java小脚本。
2. Servlet：对用户输入数据的一个封装（request.getParameter()），对业务结果的设置（request.setArrribute()）,控制页面的流向。
3. Javabean：做相关的业务处理。

## el表达式

### el:expression language 表示式语言

### 语法：

${scope.object.attr} ${scope.object[“attr”]}

### 如果不填scope,那么从

pageScope--->requestScope--->sessionScope--->applicationScope

### param

表示获取请求对象中的表单数据（单个的）

### paramValues

获取请求对象中的表单数据（多个的 CheckBox select multiple）

### el中的关系运算符

|  |  |  |  |
| --- | --- | --- | --- |
| 关系运算符 | 说明 | 范例 | 结果 |
| == 或eq | 等于 | ${5==5}或${5eq5} | true |
| != 或 ne | 不等于 | ${5!=5}或${5ne5} | false |
| < 或 lt | 小于 | ${3<5}或${3lt5} | true |
| > 或 gt | 大于 | ${3>5}或${3gt5} | false |
| <= 或 le | 小于等于 | ${3<=5}或${3le5} | true |
| >= 或 ge | 大于等于 | ${3>=5}或${3ge5} | false |

表达式语言不仅在数字与数字之间比较，还可以在字符与字符之间比较，字符串的比较是根据其对应Unicode值来比较大小的。

### 逻辑运算符

|  |  |  |
| --- | --- | --- |
| 逻辑运算符 | 范例 | 结果 |
| &&或and | 交集${A&&B}或${A and B} | True/false |
| ||或or | 并集${A||B}或${A or B} | True/false |
| !或not | 非${!A}或${not A} | True/false |

### empty判断值是否为空

${not empty param.name}

1. 条件运算${条件?1:2}

### 运用：

${not empty mes?mes:””}

### 11、.属性是调用对象的get方法

## JSTL

### jstl:jsp standard tag library jsp标准标记库

### 使用步骤：

1. 导入相关jar包
2. 导入标签库

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

1. Jstl标签分类：

迭代标签，条件标签，通用标签

### 迭代标签的使用（varstatus属性）

|  |
| --- |
| <!-- 迭代标签，items里面填el表达式，var迭代变量 -->  <c:forEach items=*"*${list }*"* var=*"bean"*>  <tr>  <td>${bean.id }</td>  <td>${bean.name }</td>  <td>${bean.author }</td>  <td>${bean.price }</td>  <td>${bean.pubDate }</td>  </tr>  </c:forEach> |

### jstl的条件判断

|  |
| --- |
| <!-- 条件标签 -->  <c:if test=*"*${1 == 1 }*"*>  1 == 1  </c:if> |

### 多个条件，当when中的条件为真，执行when里面的代码，当when中的条件为false时，执行otherwise中的代码。

|  |
| --- |
| <!-- 多条件标签 -->  <c:choose>  <c:when test=*"*${1==2 }*"*>  1==2  </c:when>  <c:otherwise>  1!=2  </c:otherwise>  </c:choose> |

### 通用标签

|  |
| --- |
| <!-- 通用标签 -->  <c:set var=*"t"* value=*"123"* scope=*"session"*></c:set>  <c:out value=*"*${t }*"*></c:out>  <br/>  <hr/>  <c:remove var=*"t"*/>  <c:out value=*"*${t }*"*></c:out> |

### 日期格式化

需导入标签库

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

格式化代码

|  |
| --- |
| <fmt:formatDate value=*"*${bean.pubDate }*"* pattern=*"yyyy-MM-dd"*/> |

### 数字格式化

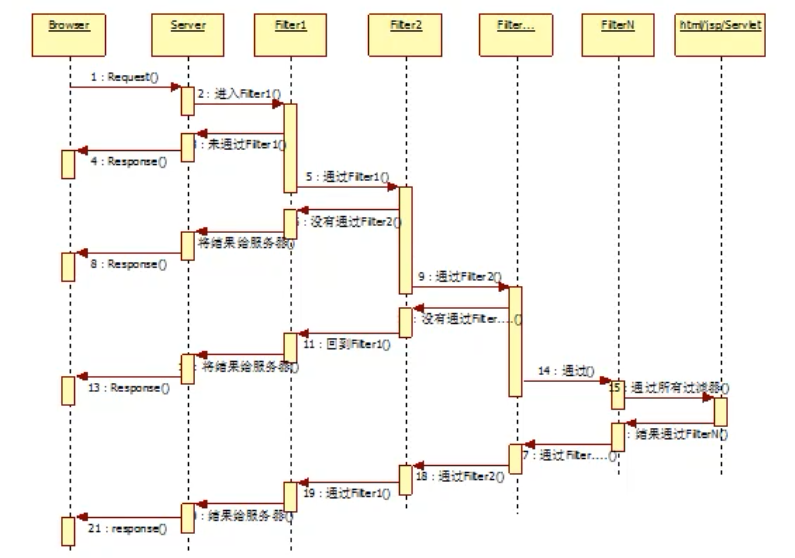
|  |
| --- |
| <fmt:formatNumber value=*"*${bean.price }*"* type=*"currency"*> |

## Filter

### filter过滤器：

对处理用户请求进行预处理。对web应用进行资源控制。

### Filter工作流程



### 实现HelloFilter

步骤：

实现filter接口

|  |
| --- |
| **public** **class** HelloFilter **implements** Filter{  **public** **void** destroy() {  }  **public** **void** init(FilterConfig arg0) **throws** ServletException {  }  **public** **void** doFilter(ServletRequest request, ServletResponse response, FilterChain chain)  **throws** IOException, ServletException {    System.***out***.println("访问servlet前");  System.***out***.println("HelloFilter");  chain.doFilter(request, response);  System.***out***.println("访问servlet后");  }  } |

配置filter

|  |
| --- |
| <filter>  <filter-name>HelloFilter</filter-name>  <filter-class>cn.yuchao.filter.HelloFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>HelloFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

### filter生命周期

Filter是在服务器启动就已经完成实例化和初始化。由服务器来管理。

### 权限管理

过滤权限

|  |
| --- |
| /\*\*  \* 权限管理  \*判断有没有登录：  \*未登录跳转到登录界面，并且无论如何允许访问Login.jsp和Login servlet 两个资源  \*登录则可以访问资源  \*  \*分析总结：只有当没登录且访问的资源不是Login.jsp和Login servlet 这两个资源时，拦截，不许与访问。  \*/  **public** **class** AuthorizationFilter **implements** Filter{  **public** **void** destroy() {  }  **public** **void** init(FilterConfig arg0) **throws** ServletException {  }  **public** **void** doFilter(ServletRequest request, ServletResponse response, FilterChain chain)  **throws** IOException, ServletException {  //ServletRequest没有getSession()方法，强制转换  HttpServletRequest req = (HttpServletRequest)request;  HttpServletResponse resp = (HttpServletResponse)response;    Object obj = req.getSession().getAttribute("user");  **if**(obj == **null**) {  String url = req.getRequestURI().substring(req.getRequestURI().lastIndexOf("/")+1);  **if**(!("Login.jsp".equals(url)||"Login".equals(url))) {  resp.sendRedirect("Login.jsp");  **return** ;  }  }    //登录后或访问的是Login.jsp和Login servlet 这两个资源时，可以访问。  chain.doFilter(req, resp);  }  } |

### 过滤乱码

|  |
| --- |
| **public** **class** CharacterEncoding **implements** Filter{  String encoding = **null**;  **public** **void** destroy() {  }  **public** **void** init(FilterConfig config) **throws** ServletException {  encoding = config.getInitParameter("encoding");  }  **public** **void** doFilter(ServletRequest request, ServletResponse response, FilterChain chain)  **throws** IOException, ServletException {  //解决post提交和响应乱码；  request.setCharacterEncoding(encoding);  response.setCharacterEncoding(encoding);  //解决get提交乱码；  chain.doFilter(**new** HttpServletRequestWrapper((HttpServletRequest)request) {  **public** String getParameter(String name) {  **if**("get".equals(**this**.getMethod().toLowerCase())) {  String value = **this**.getParameter(name);  **try** {  **return** **new** String(value.getBytes("iso-8859-1"),encoding);  } **catch** (UnsupportedEncodingException e) {  e.printStackTrace();  }  }  **return** **super**.getParameter(name);  }  }, response);  }  } |

两个过滤：web.xml

|  |
| --- |
| <filter>  <filter-name>CharacterEncoding</filter-name>  <filter-class>cn.yuchao.filter.CharacterEncoding</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>utf-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>CharacterEncoding</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>    <filter>  <filter-name>AuthorizationFilter</filter-name>  <filter-class>cn.yuchao.filter.AuthorizationFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>AuthorizationFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

## 分页处理

### 分析要点：

每页显示的记录数——pageSize

总共记录数——totalCount

总共页数——totalPage

totalPage = totalCount%pageSize == 0?otalCount/pageSize:otalCount/pageSize+1

当前页数——currentPage

### 将上面的要点封装：PageUtil类

|  |
| --- |
| **package** cn.yuchao.entity;  **public** **class** Page {  **private** **int** pageSize;  **private** **int** totalPage;  **private** **int** totalCount;  **private** **int** currentPage;  **public** **int** getPageSize() {  **return** pageSize;  }  **public** **void** setPageSize(**int** pageSize) {  **this**.pageSize = pageSize;  }  **public** **int** getTotalPage() {  **return** totalPage = (totalCount%pageSize==0)?(totalCount/pageSize):(totalCount/pageSize+1);  }  **public** **void** setTotalPage(**int** totalPage) {  **this**.totalPage = totalPage;  }  **public** **int** getTotalCount() {  **return** totalCount;  }  **public** **void** setTotalCount(**int** totalCount) {  **this**.totalCount = totalCount;  }  **public** **int** getCurrentPage() {  **return** currentPage;  }  **public** **void** setCurrentPage(**int** currentPage) {  **this**.currentPage = currentPage;  }    } |

### mysql数据库分页语句

通过limit来实现

Limit index size：从下标index开始查询，查询size条记录

如：每页显示3条记录，显示第二页的数据 那么sql语句应该为 limit 3,3;

语句传参写法 limit (currentPage-1)\*pageSize,pageSize

### 修改查询语句

|  |
| --- |
| **public** **class** NovelDao **extends** BaseDao{  //查询一页书籍  **public** List<Novel> getAll(Page page) {  List<Novel> novels = **new** ArrayList<Novel>();  String sql = "select \* from novel limit ?,?";  ResultSet res = **this**.executeQuery(sql,(page.getCurrentPage()-1)\*page.getPageSize(),page.getPageSize());  **try** {  **if**(res != **null**) {  **while**(res.next()) {  novels.add(**new** Novel(res.getInt(1),res.getString(2),res.getString(3),res.getInt(4),res.getDouble(5),res.getDate(6)));  }  }  **return** novels;  } **catch** (SQLException e) {  e.printStackTrace();  } **finally** {  **this**.close();  }  **return** **null**;  } |

### Servlet处理

|  |
| --- |
| **protected** **void** list(HttpServletRequest request, HttpServletResponse response) {  //设置page属性  page.setPageSize(10);  //默认为第一页  **int** currentPage = 1;  **if**(request.getParameter("currentPage")!=**null**)  currentPage = Integer.*parseInt*(request.getParameter("currentPage"));  page.setCurrentPage(currentPage);  //获得总记录数  **int** totalCount = noveldao.count();  page.setTotalCount(totalCount);  request.setAttribute("page", page);    //查询全部书籍存入request中  List<Novel> novellist = noveldao.getAll(page);  List<Category> categorylist = categorydao.getAll();  **if**(novellist != **null**)  request.setAttribute("novellist", novellist);  **if**(categorylist != **null**)  request.setAttribute("categorylist", categorylist);  **try** {  request.getRequestDispatcher("list.jsp").forward(request, response);  } **catch** (ServletException | IOException e) {  e.printStackTrace();  }  } |

### 页面处理

|  |
| --- |
| <body>  <h1 align=*"center"*>欢迎：${sessionScope.user }</h1>  <table align=*"center"*>  <tr align=*"right"*>  <td colspan=*"6"* style="text-align:*right*;"><a href=*"add.jsp"*>添加书籍</a></td>  </tr>  <tr>  <td>编号</td>  <td>书名</td>  <td>作者</td>  <td>类型</td>  <td>价格</td>  <td>出版日期</td>  </tr>  <c:forEach items=*"*${requestScope.novellist }*"* var=*"bean"* varStatus=*"status"*>  <tr <c:if test="${status.index%2 == 0 }">style="background:#999999"</c:if>>  <td>${bean.id }</td>  <td>${bean.name }</td>  <td>${bean.author }</td>  <c:forEach items=*"*${requestScope.categorylist }*"* var=*"cate"*>  <c:if test=*"*${cate.id == bean.category }*"*>  <td>${cate.category }</td>  </c:if>  </c:forEach>  <td><fmt:formatNumber value=*"*${bean.price }*"* type=*"currency"*/></td>  <td><fmt:formatDate value=*"*${bean.pubDate }*"* pattern=*"yyyy-MM-dd"*/></td>  <td><a href=*"novel?option=query&id=*${bean.id }*"*>修改</a></td>  <td><a href=*"novel?option=delete&id=*${bean.id }*"*>删除</a></td>  </tr>  </c:forEach>  <tr>  <td colspan=*"8"* style="text-align:*right*;">  <a href=*"novel?pageSize=*${page.currentPage-1 }*"*  <c:if test="${page.currentPage == 1 }">style="color:#ccc;" onclick="JavaScript:return false;"</c:if>  >上一页</a>    <c:forEach begin=*"1"* step=*"1"* end=*"*${page.totalPage }*"* var =*"pa"*>  <a href=*"novel?pageSize=*${pa}*"*>${pa}</a>  </c:forEach>    <a href=*"novel?pageSize=*${page.currentPage+1 }*"*  <c:if test="${page.currentPage == page.totalPage }">style="color:#ccc;" onclick="JavaScript:return false;"</c:if>  >下一页</a>    总共${page.totalPage }页    <input id=*"fin"* type=*"number"* max=*"*${page.totalPage }*"* min=*"1"* name=*"pageSize"* style="width:*50px*"/>  <input type=*"button"* value=*"跳转"* onclick="find()"/>  </td>  </tr>  </table>    <script type=*"text/javascript"*>  **function** find(){  **var** b = document.getElementById("fin").value;  location.href = "novel?pageSize="+b;  }  </script>  </body> |

## 文件上传

### 注意要点：

1. 表单提交一定是post
2. 在表单中药设置enctype=”multipart/form-data”以二进制流的方式传数据给服务器。

### IE上传的信息

|  |
| --- |
| ------WebKitFormBoundaryLbE9vGVhsqQDsaaN  Content-Disposition: form-data; name="name"  wrfw  ------WebKitFormBoundaryLbE9vGVhsqQDsaaN  Content-Disposition: form-data; name="text"; filename="a.txt"  Content-Type: text/plain  I am yuchao!!!  ------WebKitFormBoundaryLbE9vGVhsqQDsaaN-- |

### Servlet的处理代码：

|  |
| --- |
| **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  req.setCharacterEncoding("utf-8");    InputStream in = req.getInputStream();  **byte**[] bys = **new** **byte**[1000];  **int** len = in.read(bys);  String path = req.~~getRealPath~~("/upload");  File file = **new** File("D:\\apache-tomcat-8.5.28\\wtpwebapps\\fileupload\\upload\\a.txt");  OutputStream out = **new** FileOutputStream(file);  BufferedOutputStream bout = **new** BufferedOutputStream(out);  bout.write(bys, 0, len);  in.close();  bout.close();  out.close();    } |

总结：如果直接提交写入文件，得到的不是上传文件的本身，需要进行单独的解析处理。有公共类库来处理这个事情。

Smartupload(过时)，Apache下的Commons-fileupload来处理

### 使用Commons-fileupload来实现文件上传

导入Commons-fileupload.jar和Commons-io.jar

Servlet处理代码

|  |
| --- |
| **protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  req.setCharacterEncoding("utf-8");  //创建磁盘文件工厂  FileItemFactory fif = **new** DiskFileItemFactory();  //文件上传处理  ServletFileUpload sf = **new** ServletFileUpload(fif);  **try** {  List<FileItem> list = sf.parseRequest(req);  String path = req.~~getRealPath~~("/upload");  **for**(FileItem fil:list) {  **if**(fil.isFormField()) {  System.***out***.println(fil.getFieldName()+"-----"+fil.getName()+"-----"+fil.getString("utf-8"));  } **else** {  String f = UUID.*randomUUID*()+fil.getName().substring(fil.getName().lastIndexOf("."));  fil.write(**new** File(path,f));  }  }  } **catch** (FileUploadException e) {  e.printStackTrace();  } **catch** (Exception e) {  e.printStackTrace();  }  } |

## 文件下载

### 通过超链接下载

优点：实现简单

缺点：暴露服务器的文件目录造成安全隐患

|  |
| --- |
| <body>  <a href=*"download/t.txt"* >t.txt下载</a>  <br/>  <a href=*"download/17AJAX.docx"*>17AJAX.docx下载</a>  </body> |

### 2、通过文件流实现下载

1. 设置媒体文件类型（MIME）

可以在Tomcat-->conf-->web.xml中查找，查询“bin”、“com”

所以可以设置为：application/octet-stream / application/x-msdownload

实例：

Servlet：

|  |
| --- |
| **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  //设置媒体文件类型  resp.setContentType("application/octet-steam");  resp.setHeader("Content-Disposition", "attachment;filename=t.txt");    String path = req.~~getRealPath~~("/download");  File file = **new** File(path,"t.txt");    resp.setContentLength((**int**)file.length());    InputStream io = **new** FileInputStream(file);  **byte** by[] = **new** **byte**[400];  **int** len = 0;  OutputStream out = **null**;  **while**((len=io.read(by)) != -1) {  out = resp.getOutputStream();  out.write(by,0,len);  }  io.close();  out.close();  } |

### 多文件下载

* 1. 前台传文件名

### 文件下载的中文名处理

|  |
| --- |
| resp.setHeader("Content-Disposition", "attachment;filename="+URLEncoder.*encode*(filename,"utf-8")+""); |

## AJAX（局部刷新数据）

1. AJAX 指异步 JavaScript 及 XML（Asynchronous JavaScript And XML**）**

### 特点：

**页面没有改变**

**异步发送请求，获取数据**

**数据响应时间不确定**

**不能与同步一起使用（同步的业务数据不能通过异步来得到）**

**打电话——同步**

**发短信——异步**

### 应用

**——注册用户名验证**

**——自动补全**

**——网络电视下面的广告**

**——crud**

**Ajax——x xml——数据交换——json**

### javaScript——XMLHttpRequest对象

### 使用原始的xmlhttprequest对象 三个步骤：

1. **创建爱你xmlhttprequest对象**
2. **发布异步请求open()，send()**

**例：open(“GET”,”a.jsp”,”true”);**

**send(null);**

1. **编写回调函数onreadstatechange**

### ajax实例

**Jsp:**

|  |
| --- |
| <script type=*"text/javascript"* src=*"js/jquery-3.2.1.js"*></script>  <script type=*"text/javascript"*>  $(**function**(){  $("input").click(**function**(){  $.post("ajax",{name:"yuye"},**function**(date){  $("div").html(date);  });  });  });  </script>  </head>  <body>  <p>静夜思</p>  <input type=*"button"* value=*"显示"* />  <div></div>  </body> |

**Servlet:**

|  |
| --- |
| @WebServlet("/ajax")  **public** **class** AjaxServlet **extends** HttpServlet{  **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  req.setCharacterEncoding("utf-8");  System.***out***.println(req.getParameter("name"));  resp.setCharacterEncoding("utf-8");  //resp.setContentType("text/html");  resp.getWriter().println("窗前明月光，疑似地上霜。\n举头望明月，低头思故乡。");  }  **protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  **this**.doGet(req, resp);  }  } |

## 十九、ajax获取显示list数据

### servlet中处理：

|  |
| --- |
| //转回为可以用eval方法运行的字符串格式  String html = "[";  **for**(**int** i = 0;i<list.size();i++) {  **if**(i>0)  html += ",";  html += "{id:'"+list.get(i).getId()  +"',name:'"+list.get(i).getName()  +"',author:'"+list.get(i).getAuthor()  +"',price:'"+list.get(i).getPrice()+"'}";  }  html += "]";  resp.getWriter().write(html); |

### JavaScript中处理：

eval()方法中的字符串加对括号，以免出错

|  |
| --- |
| <script src=*"js/jquery-3.2.1.js"*></script>  <script>  $(**function**(){  $("input").click(**function**(){  $.post("list",**function**(date){  //把后台传来的字符串转回为对象数组(json)  alert(date);  **var** da = eval("("+date+")");    **var** htm="<tr><td>编号</td><td>书名</td><td>作者</td><td>价格</td></tr>";  **for**(**var** i = 0; i<da.length; i++){  htm+="<tr><td>"+da[i].id+"</td><td>"+da[i].name+"</td><td>"+da[i].author+"</td><td>"+da[i].price+"</td></tr>"  }    $("#mylist").html(htm);  });  });  });  </script> |

### 用gson插件

A)导入gson.jar

B)JavaScript中$.post(“glist”,fuciont(date)(),’json’)

|  |
| --- |
| **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  //处理乱码  resp.setCharacterEncoding("utf-8");    //假装从数据库获得数据  List<Book> list = **new** ArrayList<Book>();  list.add(**new** Book(1,"java","sun",12.1));  list.add(**new** Book(2,"java编译原理","sun",13.2));  list.add(**new** Book(3,"易语言","不知道",14.2));    Gson gson = **new** Gson();  resp.getWriter().print(gson.toJson(list));  } |

## 二十、监听器

### 监听器：

监听事件，在事件的发生前或者后执行一些代码处理

### 监听操作的上下文：

事件源（事件发生的位置）——》事件

1

2

监听器

3

事件处理

### 监听模型：是一个观察者模式

### Servlet有六个事件和8个监听器，主要提供了对ServletContext、HttpSession、ServletRequest的监听。

|  |  |
| --- | --- |
| Listener接口 | Event类 |
| ServletContextListener | ServletContextEvent |
| ServletContextAttributeListener | ServletContextAttributeEvent |
| HttpSessionListener | HttpSessionEvent |
| HttpSessionActivationListener |
| HttpSessionAttributeListener | HttpSessionBindingEvent |
| HttpSessionBindingListener |
| ServletRequestListener | ServletRequestEvent |
| ServletRequestAttributeListener | ServletRequestAttributeEvent |

### 实现ServletContextListener

步骤：实现ServletContextListener接口

编写配置文件

监听器代码：

|  |
| --- |
| **import** javax.servlet.ServletContextEvent;  **import** javax.servlet.ServletContextListener;  **public** **class** HelloContextListener **implements** ServletContextListener{  //服务器停止调用  **public** **void** contextDestroyed(ServletContextEvent arg0) {  System.***out***.println("Listener销毁！！！");  }  //服务器启动调用  **public** **void** contextInitialized(ServletContextEvent arg0) {  System.***out***.println("Listener初始化！！！");  }  } |

配置文件：

|  |
| --- |
| <listener>  <listener-class>cn.yuchao.listener.HelloContextListener</listener-class>  </listener> |

使用注解配置listener：

|  |
| --- |
| @WebListener |

### 监听器的应用——在线人数

人数——放在何处？——存在ServletContext中

上线、下线和HttpSession有关

代码：

|  |
| --- |
| @WebListener  **public** **class** IndexListener **implements** ServletContextListener,HttpSessionListener{  **private** **int** count = 0;  **private** ServletContext sc = **null**;  **public** **void** contextDestroyed(ServletContextEvent se) {    }  **public** **void** contextInitialized(ServletContextEvent se) {  sc = se.getServletContext();  }  **public** **void** sessionCreated(HttpSessionEvent see) {  sc.setAttribute("count", ++count);  }  **public** **void** sessionDestroyed(HttpSessionEvent see) {  sc.setAttribute("count", --count);  }  } |

Servlet：

|  |
| --- |
| @WebServlet("/index")  **public** **class** IndexServlet **extends** HttpServlet{  **protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  HttpSession session = req.getSession();  session.setAttribute("user", "1");  session.setMaxInactiveInterval(120);  resp.sendRedirect("index.jsp");  }  **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  **this**.doPost(req,resp);  }  } |

jsp：

|  |
| --- |
| <p>现在在线人数：${count }</p> |

### 注意：初始化顺序：Listener——filter——servlet

## 二十一、总结

### 1、Servlet简介

Servlet其实就是一个java类，是一个服务器端的小程序，用来处理用户的请求。

Servlet做的事情

1. 接受请求
2. 获取请求数据
3. 调用业务处理——将处理结果封装到req/sessoin.context
4. 响应

Servlet实现方式：

1、实现Servlet接口

2、继承GenericServlet类

3、继承HttpServlet类

servlet生命周期

容器实例化——》init初始化——》service——》销毁

Servlet配置详解

### 2、HttpServletRequest对象

四种请求类型（浏览器地址填写url，a标签，form表单，location.href）

请求方法：post/get

Post数据——表单提交

Get数据是追加到URL后使用？追加的，多个数据用&隔开

注意：如果form表单用get提交action后面不能用？追加数据，用了也会被覆盖

获取请求数据req.getParameter(“”);

设置属性req.setAttribute(“”);

转发req.getRequestDispather(“”).forward(req,resp);

获取Session:req.getSession();

获取ServletContext:req.getServletContext();

解决乱码req.setCharacterEncoding(“utf-8”);

获取URL:req.getRequestURL()

URI:req.getRequestURI()

获取目录路径：req.getRealPath()

获取IP：req.getRemoteAddr()

获取Cookies:req.getCookies()

................

### 3、HttpServletResponse对象

响应数据到浏览器

想浏览器输出数据resp.getWriter.print();——互斥——resp.getOutputStaream();

解决乱码：resp.setCharacterEncoding();

设置类型：resp.setContent();

重定向：resp.sendRedirect();

添加Cookie:resp.addCookie();

Resp.setHeader();

### 4、HttpSession对象

原理——为每一个客户在服务器开辟的一块空间用于存放数据。

Session.setAttribute();

Session.setMaxInactiveInterval();

Session.getAttribute();

Session.removeAttribute();

### 5、Cookie对象

原理——服务器在客户端存放的文本信息

Cookie ck = new Cookie(name,value);

ck.getName();

ck.getValue();

ck.setMaxAge();

### 6、ServletContext对象

Sc.setAttribute()......

Req——》session——》servletContext

### 7、Filter

原理

实现——实现Filter接口——doFilter

配置

生命周期

### 8、Listener

原理

实现

配置

Servlet——filter——listener

### 9、Jsp

基本语法

Model1

Model2——mvc

三大指令：page、include、taglib

六大动作：useBean、setProperty、getProperty、include、forward、param

九大对象:out、request、response、session、application、page、pageContext、config、exception

### 10、EL表达式

### 12、Jstl

通用标签set、out、remove

条件标签if、choose

迭代标签for

### 13、文件上传下载

Jspsmartupload、commons-fileupload

### 14、Ajax——servlet中ajax的处理

Json——gson

jQuery——ajax