# Day08

* EL表达式（重点）
* JSTL标签库（重点）
* MVC设计模式（重点中的重点）
* Java三层框架（重点中的重点）

## EL（表达式语言）

### EL概述

Expression Language

#### 1.1　EL的作用

JSP2.0要把html和css分离、要把html和javascript分离、要把Java脚本替换成标签。标签的好处是非Java人员都可以使用。

JSP2.0 – 纯标签页面，即：不包含<% … %>、<%! … %>，以及<%= … %>

EL（Expression Language）是一门表达式语言，它对应<%=…%>。我们知道在JSP中，表达式会被输出，所以EL表达式也会被输出。

EL无法处理逻辑，只负责数据显示。等价于<%=%>

逻辑的处理交给jstl处理

#### 1.2　EL的格式

格式：${…}

例如：${1 + 2}

EL作用：输出值 替换 <%=变量名 常量名 常量 表达式 %>

EL只能获取四大域对象中保存的值，底层实际就是pageContext.findAttribute(),获取顺序依次是：page/request/session/appliction

举例：

<%=pageContext.findAttribute(“a”)%>

简化

${“a”}

#### 1.3　关闭EL

如果希望整个JSP忽略EL表达式，需要在page指令中指定isELIgnored=”true”。

如果希望忽略某个EL表达式，可以在EL表达式之前添加“\”，例如：\${1 + 2}。

#### 1.4　EL不显示null

当EL表达式的值为null时，会在页面上显示空白，即什么都不显示。

EL遇到null，””

### 2　EL表达式格式

先来了解一下EL表达式的格式！现在还不能演示它，因为需要学习了EL11个内置对象后才方便显示它。

* 操作List和数组：${list[0]}、${arr[0]}；
* 操作bean的属性：${person.name}、${person[‘name’]}，对应person.getName()方法；
* 操作Map的值：${map.key}、${map[‘key’]}，对应map.get(key)。

.和[]的区别

**[]**用于获取有下标的数据(数组,list集合),

**.**用户获取有属性的对象(map,对象)

如果属性名中有特殊的字符,那么必须使用[“属性值”],属性值必须用引号引起来

<%@page import=*"com.whhp.bean.User"*%>

<%@page import=*"java.util.HashMap"*%>

<%@page import=*"java.util.Map"*%>

<%@page import=*"java.util.ArrayList"*%>

<%@page import=*"java.util.List"*%>

<%@ 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>EL获取数据</h1>

<h3>存取的是存入域中的单值数据</h3>

<!-- 向四个域中存入值 -->

<%

//pageContext.setAttribute("name", "p");

request.setAttribute("name", "r");

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

application.setAttribute("name", "a");

%>

<!-- 在当前页面中获取四个域的值 -->

<%=pageContext.getAttribute("name")%><!-- 如果没有找到值,那么会返回null -->

<%=request.getAttribute("name")%>

<%=session.getAttribute("name")%>

<%=application.getAttribute("name")%>

<hr>

${pageScope.name }

<!-- 如果没找到,会返回"" -->

${requestScope.name } ${sessionScope.name } ${applicationScope.name }

<hr>

${name }

<!-- 类似于使用pageContext的findAttribute方法,依次从page,request,session,application域中搜索指定名称的值 -->

<hr>

<h3>获取数组的数据</h3>

<%

String[] arrs = { "张飞", "关羽", "吕布", "赵云" };

pageContext.setAttribute("arrs", arrs);

%>

${arrs[0] } ${arrs[1] } ${arrs[2] } ${arrs[3] }

<hr>

<h3>获取list集合的数据</h3>

<%

List<String> list = **new** ArrayList<String>();

list.add("马超");

list.add("许褚");

list.add("黄忠");

pageContext.setAttribute("list", list);

%>

${ list[0]} ${ list[1]} ${ list[2]}

<hr>

<h3>获取map集合中的数据</h3>

<%

Map<String, String> map = **new** HashMap<String, String>();

map.put("aaa", "张辽");

map.put("bbb", "徐晃");

map.put("ccc", "甘宁");

map.put("ddd.eee", "陆逊");

pageContext.setAttribute("map", map);

%>

${map.aaa} ${map.bbb} ${map.ccc} ${map['ddd.eee']}

<hr>

<h3>获取对象中的数据</h3>

<%

User user = **new** User("1", "aaa", "123");

pageContext.setAttribute("user", user);

%>

${user.id } ${user.username } ${user.password }

<hr>

<h3>获取集合中对象的数据</h3>

<%

List<User> userList = **new** ArrayList<User>();

User user1 = **new** User("1", "aaa", "123");

User user2 = **new** User("2", "bbb", "456");

User user3 = **new** User("3", "ccc", "789");

userList.add(user1);

userList.add(user2);

userList.add(user3);

pageContext.setAttribute("userList", userList);

%>

${userList[0].id } - ${userList[0].username } - ${userList[0].password }<br>

${userList[1].id } - ${userList[1].username } - ${userList[1].password }<br>

${userList[2].id } - ${userList[2].username } - ${userList[2].password }<br>

</body>

</html>

### 3　EL内置对象

EL一共11个内置对象，无需创建即可以使用。这11个内置对象中有10个是Map类型的，最后一个是pageContext对象。

* pageScope
* requestScope
* sessionScope
* applicationScope
* param；
* paramValues；
* header；
* headerValues；
* initParam；
* cookie；
* pageContext；

#### 3.1　域相关内置对象（重点）

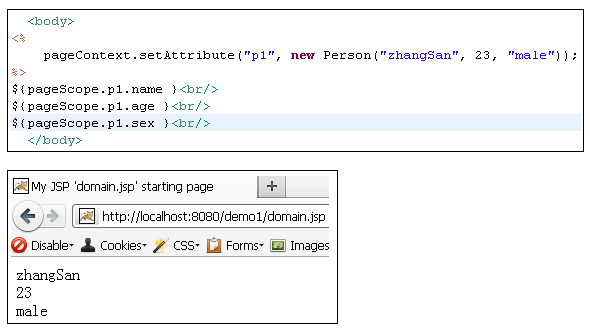
域内置对象一共有四个：

* pageScope：${pageScope.name}等同与pageContext.getAttribute(“name”)；
* requestScope：${requestScope.name}等同与request.getAttribute(“name”)；
* sessionScoep： ${sessionScope.name}等同与session.getAttribute(“name”)；
* applicationScope：${applicationScope.name}等同与application.getAttribute(“name”)；

如果在域中保存的是JavaBean对象，那么可以使用EL来访问JavaBean属性。因为EL只做读取操作，所以JavaBean一定要提供get方法，而set方法没有要求。

Person.java

|  |
| --- |
| **public** **class** Person {  **private** String name;  **private** **int** age;  **private** String sex;  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **int** getAge() {  **return** age;  }  **public** **void** setAge(**int** age) {  **this**.age = age;  }  **public** String getSex() {  **return** sex;  }  **public** **void** setSex(String sex) {  **this**.sex = sex;  }  } |



　　全域查找：${person}表示依次在pageScope、requesScopet、sessionScope、appliationScope四个域中查找名字为person的属性。

#### 3.2　请求参数相关内置对象

JSP三个jsp动作标签：

<jsp:include>

<jsp:forward>

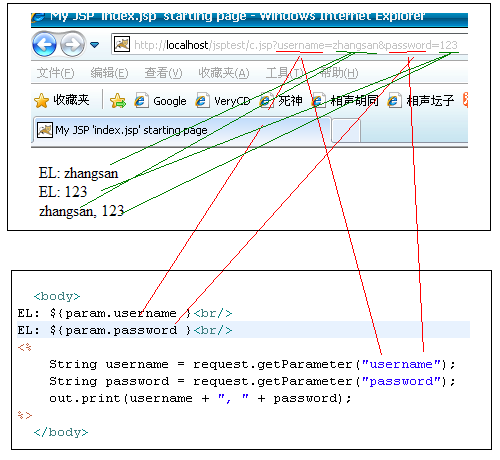
<jsp:param> 包含或者转发向其他web资源传递参数，

Get用url?key=value

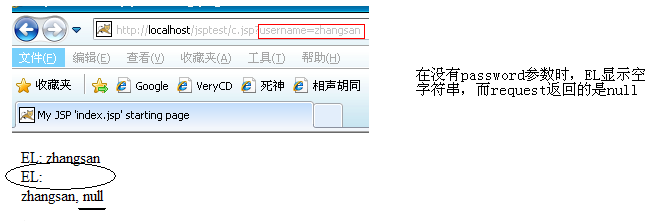
POST 请求体 key=value

　　param和paramValues这两个内置对象是用来获取请求参数的。

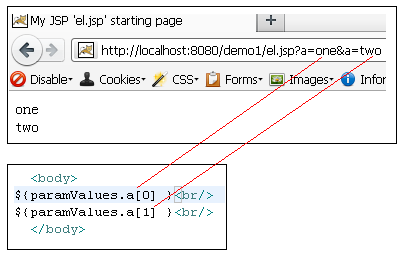
* param：Map<String,String>类型，param对象可以用来获取参数，与request.getParameter()方法相同。



注意，在使用EL获取参数时，如果参数不存在，返回的是空字符串，而不是null。这一点与使用request.getParameter()方法是不同的。



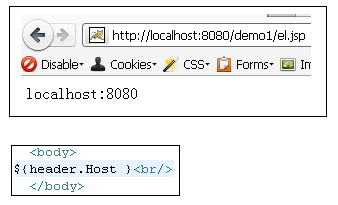
* paramValues：paramValues是Map<String, String[]>类型，当一个参数名，对应多个参数值时可以使用它。



#### 3.3　请求头相关内置对象

header和headerValues是与请求头相关的内置对象：

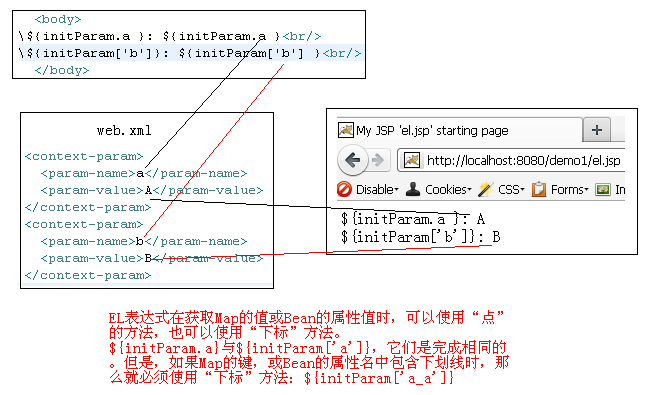
* header： Map<String,String>类型，用来获取请求头。



* headerValues：headerValues是Map<String,String[]>类型。当一个请求头名称，对应多个值时，使用该对象，这里就不在赘述。

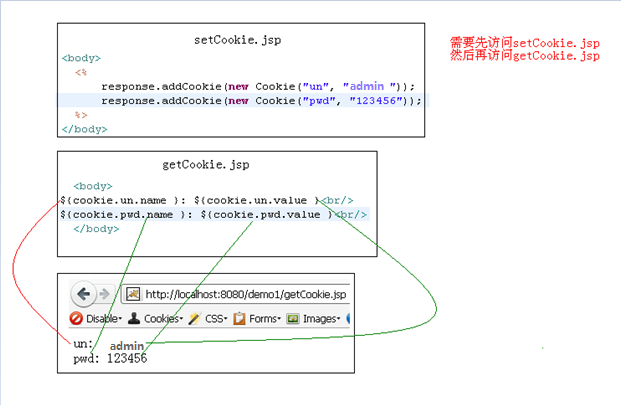
#### 3.4　应用初始化参数相关内置对象

* initParam：initParam是Map<String,String>类型。它对应web.xml文件中的<context-param>参数。



#### 3.5　Cookie相关内置对象

* cookie：cookie是Map<String,Cookie>类型，其中key是Cookie的名字，而值是Cookie对象本身。



#### 3.6　pageContext对象

pageContext：pageContext是PageContext类型！可以使用pageContext对象调用getXXX()方法，例如pageContext.getRequest()，可以${pageContext.request}。也就是读取JavaBean属性！！！

|  |  |
| --- | --- |
| **EL表达式** | **说明** |
| ${pageContext.request.queryString} | pageContext.getRequest().getQueryString(); |
| ${pageContext.request.requestURL} | pageContext.getRequest().getRequestURL(); |
| ${pageContext.request.contextPath} | pageContext.getRequest().getContextPath(); |
| ${pageContext.request.method} | pageContext.getRequest().getMethod(); |
| ${pageContext.request.protocol} | pageContext.getRequest().getProtocol(); |
| ${pageContext.request.remoteUser} | pageContext.getRequest().getRemoteUser(); |
| ${pageContext.request.remoteAddr} | pageContext.getRequest().getRemoteAddr(); |
| ${pageContext.session.new} | pageContext.getSession().isNew(); |
| ${pageContext.session.id} | pageContext.getSession().getId(); |
| ${pageContext.servletContext.serverInfo} | pageContext.getServletContext().getServerInfo(); |

### 4 EL运算符

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 说明 | 范例 | 结果 |
| + | 加 | ${17+5} | 22 |
| - | 减 | ${17-5} | 12 |
| \* | 乘 | ${17\*5} | 85 |
| /或div | 除 | ${17/5}或${17 div 5} | 3 |
| %或mod | 取余 | ${17%5}或${17 mod 5} | 2 |
| ==或eq | 等于 | ${5==5}或${5 eq 5} | true |
| !=或ne | 不等于 | ${5!=5}或${5 ne 5} | false |
| <或lt | 小于 | ${3<5}或${3 lt 5} | true |
| >或gt | 大于 | ${3>5}或${3 gt 5} | false |
| <=或le | 小于等于 | ${3<=5}或${3 le 5} | true |
| >=或ge | 大于等于 | ${3>=5}或${3 ge 5} | false |
| &&或and | 并且 | ${true&&false}或${true and false} | false |
| !或not | 非 | ${!true}或${not true} | false |
| ||或or | 或者 | ${true||false}或${true or false} | true |
| empty | 是否为空 | ${empty “”}，可以判断字符串、数据、集合的长度是否为0，为0返回true。empty还可以与not或!一起使用。${not empty “”} | true |

<%@ 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>EL执行运算</h1>

<h3>EL执行算数运算</h3>

<%

pageContext.setAttribute("a1", "10");

pageContext.setAttribute("a2", "20");

pageContext.setAttribute("a3", "30");

pageContext.setAttribute("a4", "40");

%>

${a1 + a2 + a3}

<h3>EL执行逻辑运算</h3>

${a1 < a2 } - ${ a1 **lt** a2 } <br> <!-- less than -->

${a1 > a2 } - ${ a1 **gt** a2 } <br> <!-- great than -->

${a1 <= a2 } - ${ a1 **le** a2 } <br> <!-- less eqaul -->

${a1 >= a2 } - ${ a1 **ge** a2 } <br> <!-- great eqaul -->

${a1 == a2 } - ${ a1 **eq** a2 } <br> <!-- eqaul -->

<h3>EL执行关系运算</h3>

${a1<a2 && a3<a4 } - ${a1<a2 **and** a3<a4 } <br>

${a1<a2 || a3<a4 } - ${a1<a2 **or** a3<a4 } <br>

${!(a1<a2) } - ${**not**(a1<a2)} <br>

<h3>EL执行三元运算</h3>

${a1<a3 ?"正确":"错误" }

<h3>empty运算</h3>

${user == null } - ${**empty** user }<br>

${user != null } - ${**not empty** user }<br>

</body>

</html>

## JSTL标签库

### 什么是JSTL

标签：<p></p>

JSTL是apache对EL表达式的扩展（也就是说JSTL依赖EL），JSTL是标签语言！JSTL标签使用以来非常方便，它与JSP动作标签一定，只不过它不是JSP内置的标签，需要我们自己导包，以及指定标签库而已！

如果你使用MyEclipse开发JavaWeb，那么在把项目发布到Tomcat时，你会发现，MyEclipse会在lib目录下存放jstl的Jar包！如果你没有使用MyEclipse开发那么需要自己来导入这个JSTL的Jar包：jstl-1.2.jar。

### 2　JSTL标签库

JSTL一共包含四大标签库：

* core：核心标签库，我们学习的重点；==================使用最频繁
* fmt：格式化标签库，只需要学习两个标签即可；===============日期格式化，货币
* sql：数据库标签库，不需要学习了，它过时了；
* xml：xml标签库，不需要学习了，它过时了。

### 3　使用taglib指令导入标签库

除了JSP动作标签外，使用其他第三方的标签库都需要：

* 导包；
* 在使用标签的JSP页面中使用taglib指令导入标签库；

下面是导入JSTL的core标签库：

标签表达形式：<前缀:动作></>

URI URL

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

* prefix="c"：指定标签库的前缀，这个前缀可以随便给值，但大家都会在使用core标签库时指定前缀为c；
* uri="http://java.sun.com/jstl/core"：指定标签库的uri，它不一定是真实存在的网址，但它可以让JSP找到标签库的描述文件；

### 4　core标签库常用标签

#### 4.1 out和set

${}取值，如果值不存在，不会返回null。直接输出’\u0000’

　out

|  |  |
| --- | --- |
| <c:out value=”aaa”/> | 输出aaa字符串常量 |
| <c:out value=”${aaa}”/> | 与${aaa}相同 |
| <c:out value=”${aaa}” default=”xxx”/> | 当${aaa}不存在时，输出xxx字符串 |
| <%  request.setAttribute("a","<script>alert('hello');</script>");  %>  <c:out value="${a }" default="xxx" escapeXml="false" /> | 当escapeXml为false，不会转换“<”、“>”。这可能会受到JavaScript攻击。 |

set

|  |  |
| --- | --- |
| <c:set var=”a” value=”hello”/> | 在pageContext中添加name为a，value为hello的数据。 |
| <c:set var=”a” value=”hello” scope=”session”/> | 在session中添加name为a，value为hello的数据。 |

#### 4.2 remove

|  |  |
| --- | --- |
| <%  pageContext.setAttribute("a", "pageContext");  request.setAttribute("a", "session");  session.setAttribute("a", "session");  application.setAttribute("a", "application");  %>  <c:remove var="a"/>  <c:out value="${a }" default="none"/> | 删除所有域中name为a的数据！ |
| <c:remove var="a" scope=”page”/> | 删除pageContext中name为a的数据！ |

<c:out> ${} 取值

<c:set> pageContext.setAttribute(‘key’,’value’,scope); 四大域对象存值

<c:remove> pageContext.removeAttribute(‘key”,scope);移除四大域对象存储的对象

#### 4.3 url

${pageContext.request.contextPath}==获取web应用名称

　　url标签会在需要URL重写时添加sessionId。

|  |  |
| --- | --- |
| <c:url value="/"/> | 输出上下文路径：/day08\_01/ |
| <c:url value="/" var="a" scope="request"/> | 把本该输出的结果赋给变量a。范围为request |
| <c:url value="/AServlet"/> | 输出：/day08\_01/AServlet |
| <c:url value="/AServlet">  <c:param name="username" value="abc"/>  <c:param name="password" value="123"/>  </c:url> | 输出：/day08\_01/AServlet?username=abc&password=123  如果参数中包含中文，那么会自动使用URL编码！ |

#### 4.4　if

练习：

保存一个学生年龄判断学生是否成年了。

Int age=24;

If(age>=18){

Sysout(“成年了”);

}

　　if标签的test属性必须是一个boolean类型的值，如果test的值为true，那么执行if标签的内容，否则不执行。

|  |
| --- |
| <c:set var=*"a"* value=*"hello"*/>  <c:if test="${**not empty** a }">  <c:out value="${a }"/>  </c:if> |

#### 4.5　choose==多重if

choose标签对应Java中的if/else if/else结构。when标签的test为true时，会执行这个when的内容。当所有when标签的test都为false时，才会执行otherwise标签的内容。

|  |
| --- |
| <c:set var=*"score"* value="${param.score }"/>  <c:choose>  <c:when test="${score > 100 || score < 0}">错误的分数：${score }</c:when>  <c:when test="${score >= 90 }">A级</c:when>  <c:when test="${score >= 80 }">B级</c:when>  <c:when test="${score >= 70 }">C级</c:when>  <c:when test="${score >= 60 }">D级</c:when>  <c:otherwise>E级</c:otherwise>  </c:choose> |

#### 4.6　forEach

forEach当前就是循环标签了，forEach标签有多种两种使用方式：

* 使用循环变量，指定开始和结束值，类似for(int i = 1; i <= 10; i++) {}；
* 循环遍历集合，类似for(Object o : 集合)；

循环变量方式：

|  |
| --- |
| <c:set var=*"sum"* value=*"0"* />  <c:forEach var=*"i"* begin=*"1"* end=*"10"*>  <c:set var=*"sum"* value="${sum + i}" />  </c:forEach>  <c:out value="*sum =* ${sum }"/> |
| <c:set var=*"sum"* value=*"0"* />  <c:forEach var=*"i"* begin=*"1"* end=*"10"* step=*"2"*>  <c:set var=*"sum"* value="${sum + i}" />  </c:forEach>  <c:out value="*sum =* ${sum }"/> |

遍历集合或数组方式：

|  |
| --- |
| <%  String[] names = {"zhangSan", "liSi", "wangWu", "zhaoLiu"};  pageContext.setAttribute("ns", names);  %>  <c:forEach var=*"item"* items="${ns }">  <c:out value="*name:* ${item }"/><br/>  </c:forEach> |

遍历List

|  |
| --- |
| <%  List<String> names = **new** ArrayList<String>();  names.add("zhangSan");  names.add("liSi");  names.add("wangWu");  names.add("zhaoLiu");  pageContext.setAttribute("ns", names);  %>  <c:forEach var=*"item"* items="${ns }">  <c:out value="*name:* ${item }"/><br/>  </c:forEach> |

遍历Map

|  |
| --- |
| <%  Map<String,String> stu = **new** LinkedHashMap<String,String>();  stu.put("number", "N\_1001");  stu.put("name", "zhangSan");  stu.put("age", "23");  stu.put("sex", "male");  pageContext.setAttribute("stu", stu);  %>  <c:forEach var=*"item"* items="${stu }">  <c:out value="${item.key }*:* ${item.value }"/><br/>  </c:forEach> |

forEach标签还有一个属性：varStatus，这个属性用来指定接收“循环状态”的变量名，例如：<forEach varStatus=”vs” …/>，这时就可以使用vs这个变量来获取循环的状态了。

* count：int类型，当前以遍历元素的个数；
* index：int类型，当前元素的下标；
* first：boolean类型，是否为第一个元素；
* last：boolean类型，是否为最后一个元素；
* current：Object类型，表示当前项目。

|  |
| --- |
| <c:forEach var=*"item"* items="${ns }" varStatus=*"vs"*>  <c:if test="${vs.first }">第一行：</c:if>  <c:if test="${vs.last }">最后一行：</c:if>  <c:out value="*第*${vs.count }*行:* "/>  <c:out value="*[*${vs.index }*]:* "/>  <c:out value="*name:* ${vs.current }"/><br/>  </c:forEach> |

### 5　fmt标签库常用标签

fmt标签库是用来格式化输出的，通常需要格式化的有时间和数字。

格式化时间：

|  |
| --- |
| <%@ taglib prefix=*"fmt"* uri=*"http://java.sun.com/jsp/jstl/fmt"* %>  ......  <%  Date date = **new** Date();  pageContext.setAttribute("d", date);  %>  <fmt:formatDate value="${d }" pattern=*"yyyy-MM-dd HH:mm:ss"*/> |

格式化数字：

|  |
| --- |
| <%  **double** d1 = 3.5;  **double** d2 = 4.444;  pageContext.setAttribute("d1", d1);  pageContext.setAttribute("d2", d2);  %>  <fmt:formatNumber value="${d1 }" pattern=*"0.00"*/><br/>  <fmt:formatNumber value="${d2 }" pattern=*"#.##"*/> |

## MVC

1. 早期实现数据表的显示怎么做？
   1. Jdbc查询List
   2. Servlet负责循环集合并通过html+css+javascript格式美化以后响应浏览器
2. JSP=DHTML+JAVA

### 1　MVC设计模式



MVC设计模式

MVC模式（Model-View-Controller）是[软件工程](http://zh.wikipedia.org/wiki/%E8%BD%AF%E4%BB%B6%E5%B7%A5%E7%A8%8B" \o "软件工程)中的一种[软件架构](http://zh.wikipedia.org/wiki/%E8%BD%AF%E4%BB%B6%E6%9E%B6%E6%9E%84" \o "软件架构)模式，把软件系统分为三个基本部分：模型（Model）、视图（View）和控制器（Controller）。

MVC模式最早为[Trygve Reenskaug](http://zh.wikipedia.org/w/index.php?title=Trygve_Reenskaug&action=edit&redlink=1" \o "Trygve Reenskaug（页面不存在）)提出，为[施乐帕罗奥多研究中心](http://zh.wikipedia.org/wiki/%E5%B8%95%E7%BE%85%E5%A5%A7%E5%A4%9A%E7%A0%94%E7%A9%B6%E4%B8%AD%E5%BF%83" \o "帕罗奥多研究中心)（Xerox PARC）的[Smalltalk](http://zh.wikipedia.org/wiki/Smalltalk" \o "Smalltalk)语言发明的一种软件设计模式。

MVC可对程序的后期维护和扩展提供了方便，并且使程序某些部分的重用提供了方便。而且MVC也使程序简化，更加直观。

* 控制器Controller：对请求进行处理，负责请求转发；
  + Servlet
* 视图View：界面设计人员进行图形界面设计；
  + JSP HTML
* 模型Model：程序编写程序应用的功能（实现算法等等）、数据库管理；
  + DAO

注意，MVC不是Java的东西，几乎现在所有B/S结构的软件都采用了MVC设计模式。但是要注意，MVC在B/S结构软件并没有完全实现，例如在我们今后的B/S软件中并不会有事件驱动！

### JavaWeb与MVC

M-model 模型 javabean泛指所有与数据库操作相关的部分 SERVICE+DAO

V-view 视图 图形化界面jsp

C-controller 控制器 负责页面的重定向和转发的部分 servlet

　　JavaWeb的经历了JSP Model1、JSP Model1二代、JSP Model2三个时期。

#### JSP Model1第一代

JSP产生之后，有一段时间，放弃servlet html+java+jsp标签

JSP Model1是JavaWeb早期的模型，它适合小型Web项目，开发成本低！Model1第一代时期，服务器端只有JSP页面，所有的操作都在JSP页面中，连访问数据库的API也在JSP页面中完成。也就是说，所有的东西都耦合在一起，对后期的维护和扩展极为不利。

Model1体现：

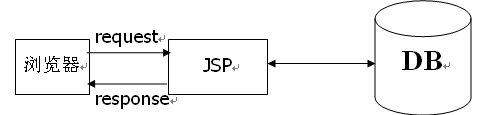
Jsp:负责1 连接数据库 2.jdbc 3.封装响应 4.控制页面 5.响应客户端请求

设置场景：

用户功能：

1. 注册 reg.jsp
2. 登录 login.jsp

公共的代码：连接数据库，执行sql



举例：显示图书列表

JSP要做哪些事情？1.调用数据库代码获取集合列表 2.循环集合+html显示

#### 2.2　JSP Model1第二代 又称model2第一代

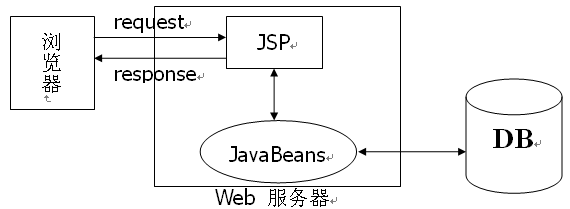
　　JSP Model1第二代有所改进，把业务逻辑的内容放到了JavaBean中，而JSP页面负责显示以及请求调度的工作。虽然第二代比第一代好了些，但还让JSP做了过多的工作，JSP中把视图工作和请求调度（控制器）的工作耦合在一起了。

JSP：通过调用javabean提供数据库操作方法，实现数据的管理

JSP数据操作完毕之后：

1. 成功
2. 失败

扮演两个人角色：负责数据的收集和显示 控制器实现页面的转发和重定向 Service调用



#### JSP Model2

M:MODEL 模型

V：VIEW 视图 JSP+HTML

C：CONTROLLER 控制器 Servlet

JSP Model2模式已经可以清晰的看到MVC完整的结构了。

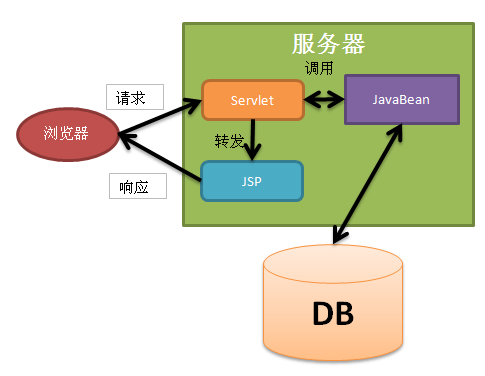
* JSP：视图层，用来与用户打交道，类似Scanner。负责接收用来的数据，以及显示数据给用户；
* Servlet：控制层，负责找到合适的模型对象来处理业务逻辑，转发到合适的视图；
* JavaBean：模型层（理解层dao+service），完成具体的业务工作，例如：开启、转账等。

Jsp响应客户端请求：servlet处理的数据结果以一种较为美好的格式显示，便于阅读

学生管理系统：javase

界面不太友好，但是CRUD功能实现

javaWeb技术：crud main===》Servlet+jsp 提升程序扩展性和维护性



<%=%>===========🡺MVC剔除掉

JSP Model2适合多人合作开发大型的Web项目，各司其职，互不干涉，有利于开发中的分工，有利于组件的重用。但是，Web项目的开发难度加大，同时对开发人员的技术要求也提高了。

## JavaWeb经典三层框架

MVC就是三层架构，不对！！

JDBC整合基于三层框架

domain:存放都是数据库转换出来的实体类。

Pojo:数据库中所有的表转换出来的java类 也称 javaBean

Dao:数据库sql语句执行的代码insert update

Service:业务逻辑层（需求）调用DAO功能获取数据库的数据，然后执行相应需求

控制器：Servlet获取客户端提交的数据，调用Service方法得到处理的结果，存储域对象，转发或者重定向JSP

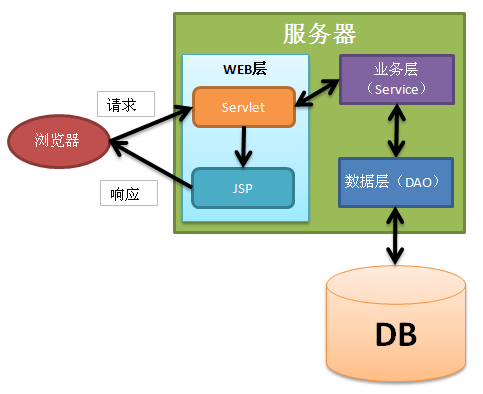
界面层：jsp 获取用户输入的数据 EL显示数据

我们常说的三层框架是由JavaWeb提出的，也就是说这是JavaWeb独有的！

所谓三层是表述层（WEB层）、业务逻辑层（Business Logic），以及数据访问层（Data Access）。

* WEB层：包含JSP和Servlet等与WEB相关的内容；
* 业务层：业务层中不包含JavaWeb API，它只关心业务逻辑；
* 数据层：封装了对数据库的访问细节；

　　注意，在业务层中不能出现JavaWeb API，例如request、response等。也就是说，业务层代码是可重用的，甚至可以应用到非Web环境中。业务层的每个方法可以理解成一个万能，例如转账业务方法。业务层依赖数据层，而Web层依赖业务层！



预习：分页