# jsp&el&jstl&综合案例

# 学习目标

1. 能够说出jsp的优势
2. 能够编写jsp代码片段、声明、脚本表达式
3. 能够说出el表达式的作用
4. 能够使用el表达式获取javabean的属性
5. 能够使用jstl标签库的if标签
6. 能够使用jstl标签库的foreach标签
7. 能够使用jstl标签库的choose标签
8. 能够说出开发模式的作用
9. 能够使用三层架构模式完成显示用户案例

**反馈**

|  |
| --- |
| response.getOutputStream().write("1".getBytes()); PrintWriter p = response.getWriter(); p.print("2"); 只能回应一次请求? or一个页面只能用一个输出流?点解  一次响应只能使用一种类型输出流，要么字符流，要么字节流。这是服务器输出数据的机制。 |
| ResourceBundle可以加载jar包中的配置文件吗  ResourceBundle用于读取配置文件的，根据一个文件名字（不需要扩展名.properties文件）  可以，但是需要jar放到当前classpath路径里面，就可以读取 |
| 在删除cookie的时候如果没有设置setPath（。。。），会出现无法删除cookie的现象，这个是怎么理解呢？  测试过，有删除的 |
| 1. 绝对路径跟相对路径这个知识点一直不大理解.比如: cookie.setPath(request.getContextPath() + "/Servlet"),改为: cookie.setPath("/Servlet") 这样好像也可以,怎么区分的?   可以的，如果设置成这个路径，浏览器就会判断以/servlet开头的携带cookie数据，由于此路径是给浏览器用的，“/”代表的webapps目录下  2. 之前上课的,  <!--路径:  绝对路径:/工程名字/资源路径,/day36/RefererServlet  相对路径: 资源路径,RefererServlet 第一个"/",在动态资源里面代表的当前资源目录day36目录下  好像不是day36目录,而是webapps的目录?  Requst.getRequestDispacter(“/one”).forward(request,response);”/”当前工程目录下  WebServlet(urlPatterns=”/one”),”/”代表当前工程目录下  Response.sendRedirect(“/one”);”/”，代表webapps目录下，  Location:”/one”, ”/”，代表webapps目录下  Js代码，loction.href=”/one”,/ 代表webapps目录下  浏览器执行的路径， “/”代表webapps目录下  服务器执行的路径，“/”代表当前工程目录下 |
| java可以做脚本吗? applet（java的小应用程序，运行在浏览器上。。。。）  可以，只要可以运行jvm，java可以在任何地方运行 |

# jsp的基本入门

## jsp简介

### jsp的概念

JSP(java server Pages),java的服务器页面。

HTML代码与Java代码共同存在，其中，HTML代码用来实现网页中静态内容的显示，Java代码用来实现网页中动态内容的显示。为了与传统HTML有所区别，JSP文件的扩展名为.jsp。

JSP技术所开发的Web应用程序是基于Java的，它可以用一种简捷而快速的方法从Java程序生成Web页面，其使用上具有如下几点特征：

l **跨平台：**由于JSP是基于Java语言的，它可以使用Java API，所以它也是跨平台的，可以应用于不同的系统中，如Windows、Linux等。当从一个平台移植到另一个平台时，JSP和JavaBean的代码并不需要重新编译，这是因为Java的字节码是与平台无关的，这也应验了Java语言“一次编译，到处运行”的特点。

l **业务代码相分离：**在使用JSP技术开发Web应用时，可以将界面的开发与应用程序的开发分离开。开发人员使用HTML来设计界面，使用JSP标签和脚本来动态生成页面上的内容。在服务器端，JSP引擎（或容器，本书中指Tomcat）负责解析JSP标签和脚本程序，生成所请求的内容，并将执行结果以HTML页面的形式返回到浏览器。

l **组件重用：**JSP中可以使用JavaBean编写业务组件，也就是使用一个JavaBean类封装业务处理代码或者作为一个数据存储模型，在JSP页面中，甚至在整个项目中，都可以重复使用这个JavaBean，同时，JavaBean也可以应用到其他Java应用程序中。

l **预编译：**预编译就是在用户第一次通过浏览器访问JSP页面时，服务器将对JSP页面代码进行编译，并且仅执行一次编译。编译好的代码将被保存，在用户下一次访问时，会直接执行编译好的代码。这样不仅节约了服务器的CPU资源，还大大的提升了客户端的访问速度。

### 为什么要诞生JSP

​ 我们先来设计一个场景，完成一个需求：准备一个页面，根据页面的上链接的请求，展示不同古代美女的名称。

#### 准备页面

<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  
<title>Insert title here</title>  
</head>  
<body>  
<!-- 需求：根据不同请求参数，显示不同古代美女名称 -->  
<a href="test?param=1">美女1</a>  
<a href="test?param=2">美女2</a>  
<a href="test?param=3">美女3</a>  
</body>  
</html>

#### 演示servlet

package cn.itcast.web;  
​  
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(name = "TestServlet",urlPatterns = "/test")  
public class TestServlet extends HttpServlet {  
​  
 public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 String parameter = request.getParameter("param");  
 //处理响应的乱码  
 response.setContentType("text/html;charset=utf-8");  
 if("1".equals(parameter)) {  
 response.getWriter().write("<h1><font color='red'>西施</font></h1>");  
 }else if("2".equals(parameter)) {  
 response.getWriter().write("<h1><font color='red'>貂蝉</font></h1>");  
 }else if("3".equals(parameter)) {  
 response.getWriter().write("<h1><font color='red'>王昭君</font></h1>");  
 }else {  
 response.getWriter().write("参数不合法");  
 }  
 }  
​  
 public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 doGet(request, response);  
 }  
​  
}

​ **通过书写servlet代码，我们发现一个问题：** **使用response对象，向页面输出内容，需要拼接html标签，这个操作十分麻烦。**

​ 因此我们需要一个解决方案，那么这个解决方案应该满足什么条件呢？

​ **应该有以下两点：**

​ **1 必须不再拼接html，使用字符串向页面输出代码**

​ **2 需要保留servlet可以通过代码，动态生成网页的功能**

​ **满足这两点的正式我们要学习的JSP技术：它可以简化html书写，同时动态生成页面。**

### 体验jsp

#### 创建JSP文件

<html>  
<head>  
  <title>测试jsp</title>  
</head>  
<body>  
   <%  
     Date date = new Date();  
     SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:dd");  
     String curTime = dateFormat.format(date);  
     request.setAttribute("curTime",curTime);  
   %>  
​  
   当前系统时间：<span style="color: red"><%=request.getAttribute("curTime") %></span>  
</body>  
</html>

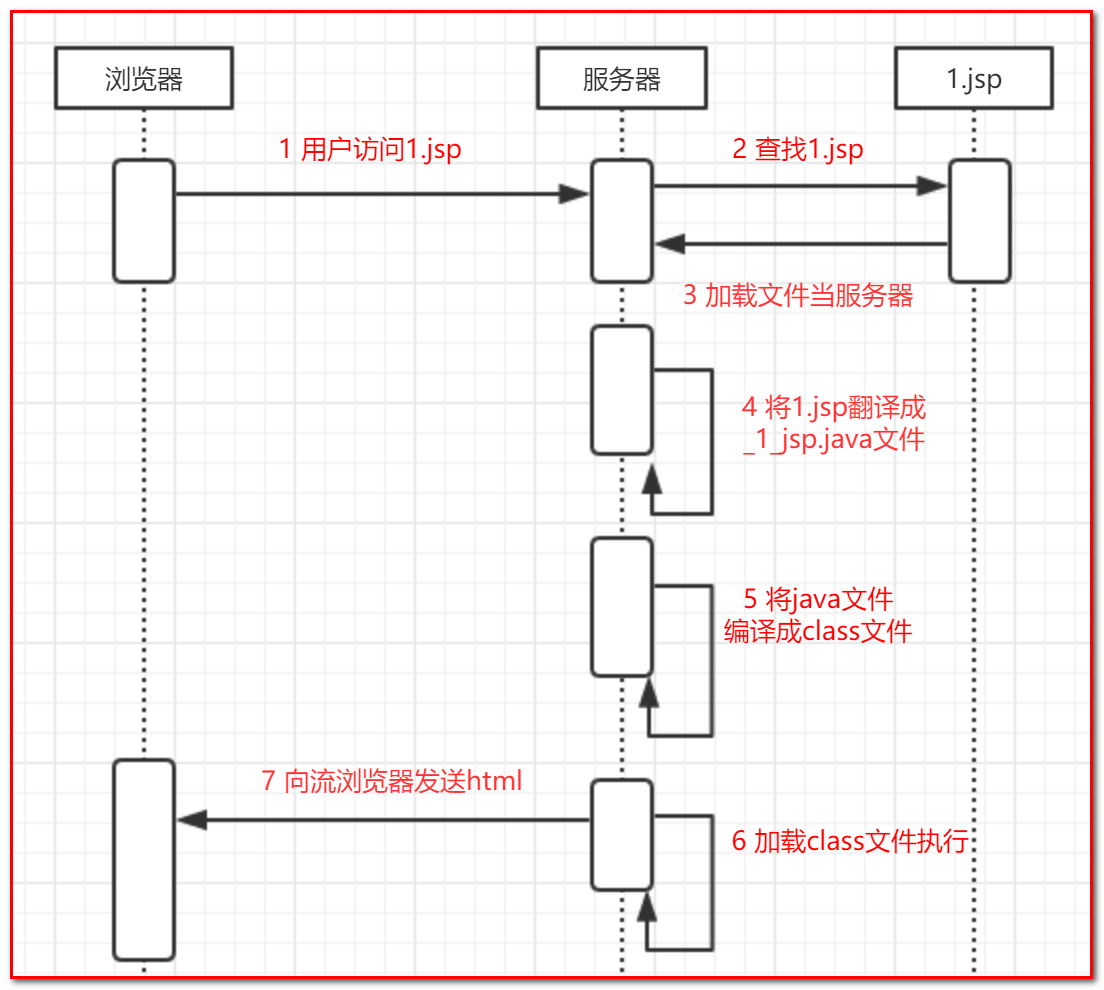
在地址栏输入jsp文件名称访问即可。

效果：

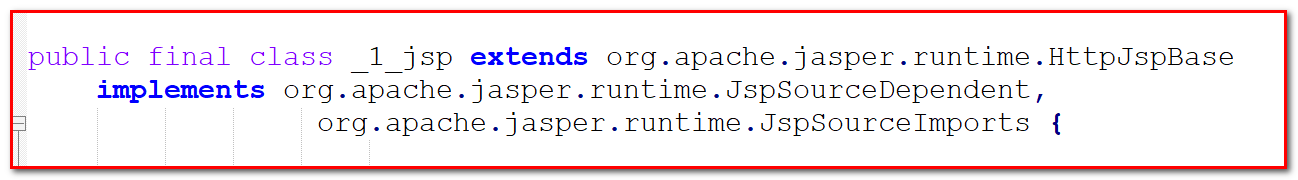
## jsp的执行原理

通过上述的体验jsp之后，我们发现jsp真的非常厉害，既可以编写java代码也可以直接编写html代码，相对servlet更加方便，那么**jsp为什么可以直接使用java代码reqeust 与直接编写html代码呢？**

答案：因为jsp会被翻译成java文件，所以可以写java代码，并且将html代码转换为java代码进行输出。我们一起来看看jsp的执行原理

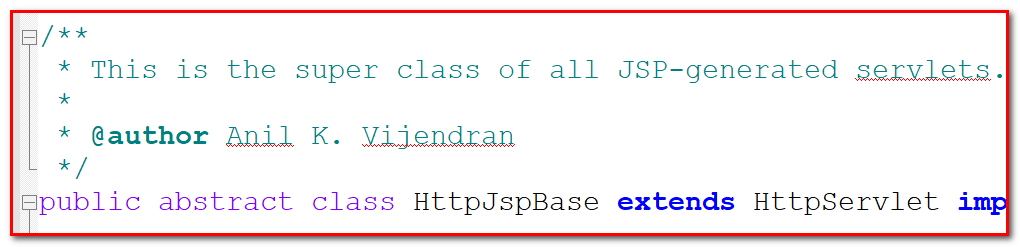


依据上图的流程，C:\Users\ztl.IntelliJIdea2017.3\system\tomcat\Unnamed\_javase\work\Catalina\localhost\webtest\org\apache\jsp找到了生成的java文件，其中一部分内容如下图：



​ 我们可以看到当前的jsp文件被翻译成了一个类，这个类继承HttpJspBase类，那么这个HttpJspBase类又是什么？

​ 注意jsp的翻译有服务器完成，HttpJspBase类一定也是tomcat服务器的内容，顺着org.apache.jasper.runtime.HttpJspBase这个类全名，我们找到这个类的源码：



​ 通过观察源码，**我们发现JSP其实底层就是一个servlet**。通过观察源码，我们发现我们刚刚编写的所有代码都在该Servlet里面的service方法内部。



**总结：**

​ **1.** jsp之所以可以编写html代码，其实本质上也是类似我们使用Servlet直接输出的。

​ **2.**jsp之所以直接使用 直接使用request对象，是因为我们自己编写的代码全部都落入到了service方法内部，在service方法内部一开始就已经声明了request等对象了。

## jsp的基本语法

### jsp注释

#### JSP注释格式

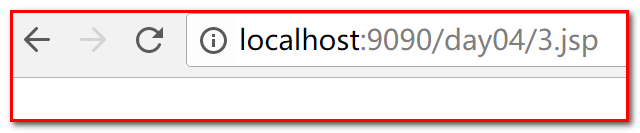
​ <%-- jsp注释 --%>

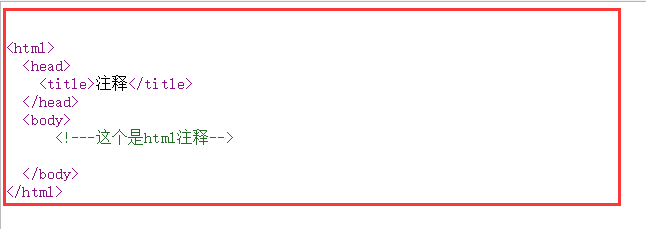
#### JSP注释的使用

**jsp文件**：

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
  <head>  
    <title>注释</title>  
  </head>  
  <body>  
      <!---这个是html注释-->  
      <%-- 这个是jsp的注释--%>  
  </body>  
</html>

访问页面测试：没有任何内容显示，那么也就说html与jsp注释都生效, 但是点击查看源码的时候我们发现我们只能查看到html的注释，jsp的注释根本就看不到。





**总结**

​ **1.** jsp的注释不会显示在源码上，更加安全。

​ **2.**html注释无法注释java代码，html注释java代码，java代码依然会执行，所以推荐使用jsp的注释。

### jsp书写java代码的三种方式

​ 在之前的演示中，我的jsp已经可以向页面输出一个html内容，但是这个还不够，jsp应该还要有像servlet一样可以通过代码，动态生成网页的功能。servlet是使用java代码生成动态网页的，因此，接下来，我们要学习如何在jsp页面使用java代码。

#### 脚本片段

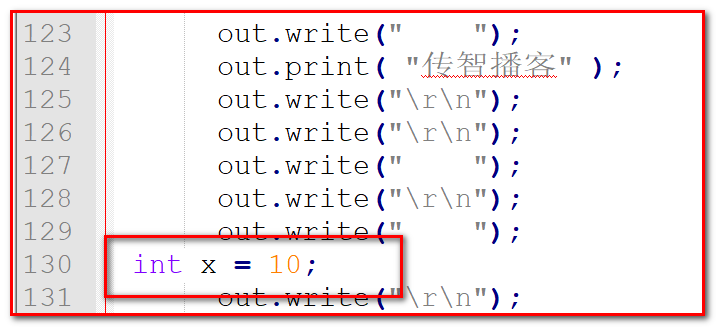
##### 脚本片段格式

​ 格式：<% Java代码片段  %>

​ jsp文件内容：

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>书写java代码的三种方式</title>  
</head>  
<body>  
   <%-- 脚本片段 --%>  
   <% int x = 10; %>  
</body>  
</html>

​ 翻译成java文件：



总结:在脚本片段中书写的Java代码，会翻译到java文件中的\_jspService方法中。

##### 脚本片段作用

在jsp翻译后的\_jspService方法中，嵌入java代码

##### 脚本片段使用注意事项

​ 脚本片段可以分开书写，最终是组合在一起的，示例：

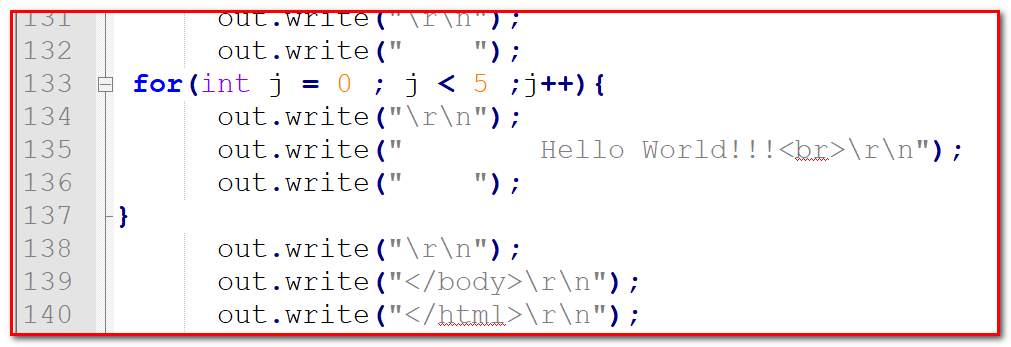
​ 脚本片段内容：

<% for(int j = 0 ; j < 5 ;j++){%>  
       Hello World!!!<br>  
 <%}%>

​ 效果：

Hello World!!!  
Hello World!!!  
Hello World!!!  
Hello World!!!  
Hello World!!!

​ java源码:



#### 脚本声明

​ 脚本片段虽然可以嵌入java代码，但是如果，我们要给当前的jsp中定义一些方法或者成员变量，就行需要一个新的技术——脚本声明。

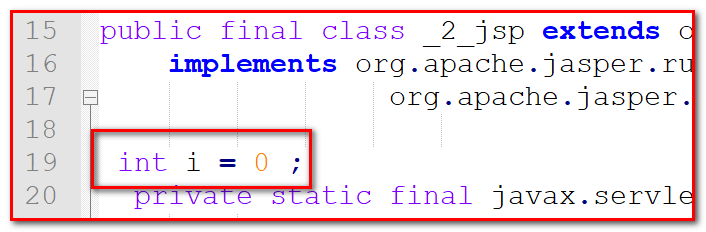
##### 脚本声明格式

​ 格式：<%!  书写Java代码  %>

​ jsp文件内容：

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>书写java代码的三种方式</title>  
</head>  
<body>  
   <%! int i = 0 ;%>  
</body>  
</html>  
​

​ 翻译成java文件：



总结：脚本声明书写的java代码会翻译在类的成员位置上。

##### 脚本声明作用

在类的成员位置上声明方法和变量

##### 脚本声明使用注意事项

声明方法的时候，不要声明与jsp翻译的java文件中默认的一些方法或者变量同名的方法或者变量。

#### 脚本表达式

​ 虽然脚本声明和脚本片段已经可以书写Java代码了，但是如果我们要使用java代码向页面输出一些内容，还是需要使用原来的response对象，比较繁琐，因此，我们需要一个更加简便的方式，可以代替response向页面输出内容——这个就是脚本表达式。

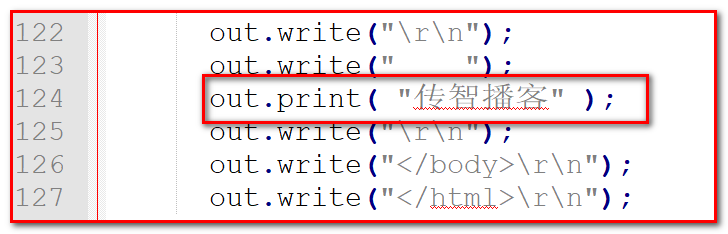
##### 脚本表达式格式

​ 格式：<%= 表达式 %>

​ jsp文件内容：

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>书写java代码的三种方式</title>  
</head>  
<body>  
      
   <%-- 脚本表达式 --%>  
   <%= "传智播客" %>  
</body>  
</html>

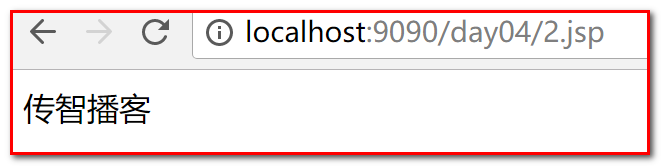
​ 翻译成java文件：在\_jspService方法中，找到了我们书写的内容



​ 上图的中的out对象是什么？

​ 答：out对象的类型是JspWriter，通过查阅JAVAEE文档发现其父类是java.io.Writer，是一个Writer 字符流。

​ 页面输出：

​ 

总结：脚本表达式书写的java代码，会翻译到java文件中的\_jspService方法内，被out.print输出到页面。

##### 脚本表达式作用

代替response向页面输出内容

##### 脚本表达式使用注意事项

使用表达式向页面输出内容的时候，不要同时使用response对象向页面输出内容，会出现页面内容输出顺序和代码顺序不一致。

## 企业动态资源开发实践

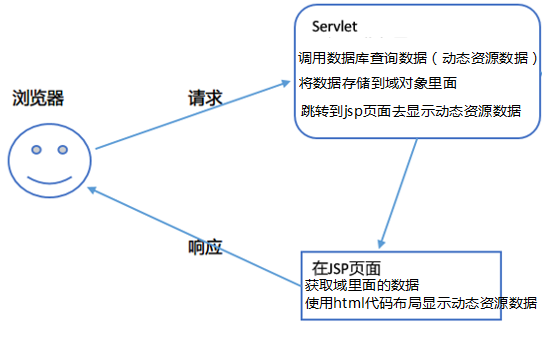
jsp本质是servlet，也就是jsp和servlet都可以开发动态资源，处理请求与响应。那么以后到底使用jsp呢还是servlet呢？

jsp：编写html布局代码方便，所以主要负责显示布局动态资源数据

servlet:写的代码全部是java代码，所主要主要负责产生动态资源数据

jsp与servlet配合使用，用户请求servlet负责从数据库获取数据，存储到域对象里面，之后跳转到jsp页面显示布局动态资源数据。

架构设计



jsp页面显示动态资源数据，页面会充斥着<% %>等元素，导致页面写代码或观看都非常乱，我们有一个目标将页面上所有<% %>等全部使用非常简洁的技术代替，让jsp代码写起来简洁和统一，做如下代替

页面上的脚本表达式<%=java代码%>=====================🡺使用EL技术代替

页面上的脚本代码片段<%java代码;%>的循环、条件判断等=======🡺使用JSTL技术代替

# EL表达式

## EL表达式的基本概述

​ 想要知道什么是EL表达式，它为了解决什么问题而诞生，我们先通过一个场景来了解一下：

​ 现在有一个需求：在jsp使用java代码再request中设置四个数据（10 20 30 40 ）的向页面输出（10+20+（30-40））计算结果，以我们现在的技术去实现会是这样实现：

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>Title</title>  
</head>  
<body>  
<%-- 在jsp设置(request)四个数据（10 20 30 40 ）的向页面输出（10+20+（30-40））计算结果 --%>  
<%  
    request.setAttribute("num1", 10);  
    request.setAttribute("num2", 20);  
    request.setAttribute("num3", 30);  
    request.setAttribute("num4", 40);  
%>  
java代码输出：<%= (Integer)request.getAttribute("num1") +  
       (Integer)request.getAttribute("num2") +  
       ( (Integer)request.getAttribute("num3") - (Integer)request.getAttribute("num4"))%>  
</body>  
</html>  
​

​ 从上面的代码中我们发现，使用之前的脚本片段来完成实在太过复杂和繁琐，一个简单的算术计算不应该如此艰难的完成。

​ 因此我们需要一个新的技术，来简化java代码的一些操作，这个就是我们需要学习的EL表达式技术。

​ **EL全称：Expression Language**

​ **作用：代替jsp中脚本表达式的功能，简化对java代码的操作。**

## EL表达式的格式和作用

1. **EL表达式的格式**：${表达式内容}
2. **EL表达式的作用**： 从域对象中查找指定的数据。

## EL表达式的基本使用

### EL获得容器（域对象）的数据

​ 再上面的案例中，我们使用原来的方式获取数据十分麻烦，接下来我们使用EL表达式的方式完成上面的需求：

jsp演示代码：

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>Title</title>  
</head>  
<body>  
<%-- 在jsp设置(request)四个数据（10 20 30 40 ）的向页面输出（10+20+（30-40））计算结果 --%>  
<%  
    request.setAttribute("num1", 10);  
    request.setAttribute("num2", 20);  
    request.setAttribute("num3", 30);  
    request.setAttribute("num4", 40);  
%>  
使用EL表达式输出：${num1 + num2 + (num3 - num4)}  
​  
</body>  
</html>  
​

​ 显而易见的使用EL表达式我们获取数字，并执行运算都方便了许多。在上面这个案例中我们是从request域对象获取数据，那么我们可以获取其域对象中的数据吗？我们来测试一下：

|  |
| --- |
| <**h2**>EL获得容器（域对象）的数据</**h2**> *<%--el作用：获取域里面的数据--%>* **<%** request.setAttribute(**"name"**,**"admin1"**);  request.getSession().setAttribute(**"name"**,**"admin2"**);  request.getSession().getServletContext().setAttribute(**"name"**,**"admin3"**); **%>** <**p**>获取域里面的name:**${**name**}**</**p**>*<%--el通过直接写域里面的name，会依次从request，session,servletCOntext里面获取，找到为止--%> <%--如果想通过el获取指定域里面的数据，必须指定域的范围  el的对象 servlet对象或jsp对象  requestScope====>request域对象  sessionSoope====>session域对象  applicationScope==>servletContext域对象 --%>* <**p**>获取request域里面的name:**${**requestScope.name**}**</**p**> <**p**>获取session域里面的name:**${**sessionScope.name**}**</**p**> <**p**>获取servletContext域里面的name:**${**applicationScope.name**}**</**p**> <**p**>获取servletContext域里面的name2:**${**applicationScope.name2**}**</**p**>*<%--el获取的数据如果域里面没有，el可以达到有就显示，没有就不显示--%>* |

​ 那么我们之前没有指定容器是如何获取数据的呢？其实${addr}在获取容器的时候，默认按request、session、servletcontext顺序从获取数据，只要获取到就不再往下找了，所以上一个案例一直获取到的上海。

### EL获取和解析复杂数据

​ 上面的案例我们在获取数据的时候，都是简单的字符串，接下来我们来获取一些复杂数据，复杂数据特指：数组，集合（List   Map）和JavaBean。

#### 获取数组

|  |
| --- |
| <**h2**>EL获取和解析域里面的复杂数据——数组</**h2**> **<%** *//往域里面写数组数据* User[] users = { **new** User(20,**"张三"**), **new** User(22,**"李四"**)};  request.setAttribute(**"users"**,users);  **%>** <**p**>获取request域中对象数组里面第一个元素name的值：**${**users[0].name**}**</**p**>*<%--el获取数组，是通过下标[索引]方式获取元素--%>* <**p**>获取request域中对象数组里面第一个元素age的值：**${**users[0].age**}**</**p**> <**p**>获取request域中对象数组里面第二个元素name的值：**${**users[1].name**}**</**p**> <**p**>获取request域中对象数组里面第二个元素age的值：**${**users[1].age**}**</**p**> |

#### 获取数组注意事项

​ **获取数组中某一数据，使用中括号添加角标即可**

#### 获取集合（list map）

|  |
| --- |
| <**h2**>EL获取和解析域里面的复杂数据——List</**h2**> **<%** *//往域里面写List数据* List<User> userList = **new** ArrayList<User>();  userList.add( **new** User(20,**"张三"**));  userList.add( **new** User(22,**"李四"**));  request.setAttribute(**"userList"**,userList);  **%>** <**p**>获取request域中List集合里面第一个元素name的值：**${**userList[0].name**}**</**p**>*<%--el获取集合，也是是通过下标[索引]方式获取元素，与数组一样--%>* <**p**>获取request域中List集合里面第一个元素age的值：**${**userList[0].age**}**</**p**> <**p**>获取request域中List集合里面第二个元素name的值：**${**userList[1].name**}**</**p**> <**p**>获取request域中List集合里面第二个元素age的值：**${**userList[1].age**}**</**p**> <**h2**>EL获取和解析域里面的复杂数据——Map</**h2**> **<%** *//往域里面写Map数据* Map<String,User> map = **new** HashMap<String,User>();  map.put(**"one"**, **new** User(20,**"张三"**));  map.put(**"two"**, **new** User(22,**"李四"**));  map.put(**"three.four"**,**new** User(24,**"王五"**));  request.setAttribute(**"map"**,map);  **%>** <**p**>获取request域中map集合里面one元素name的值：**${**map.one.name**}**</**p**>*<%--el获取集合，也是是通过下标[索引]方式获取元素，与数组一样--%>* <**p**>获取request域中map集合里面one元素age的值：**${**map.one.age**}**</**p**> <**p**>获取request域中map集合里面two元素name的值：**${**map.two.name**}**</**p**> <**p**>获取request域中map集合里面two元素age的值：**${**map.two.age**}**</**p**> <**p**>获取request域中map集合里面three.four元素name的值：**${**map.three.four.name**}**</**p**>*<%--如果el获取的属性名中有特殊字符，不可以使用.的方式获取数据--%>* <**p**>获取request域中map集合里面three.four元素name的值：**${**map[**"three.four"**].name**}**</**p**>*<%--el获取数据属性名有特殊符号，需要使用${["属性名"]}进行获取--%>* |

#### 获取集合注意事项

1. 设置map集合数据的key，尽量不要出现”.“
2. 凡是在EL表达式中使用”.“可以获取的数据，使用"[]"也可以获取

#### 获取JavaBean数据

|  |
| --- |
| <**h2**>EL获取和解析域里面的复杂数据——javaBean对象</**h2**> **<%** *//往域里面写javaBean对象* User user = **new** User(20,**"张三"**);  request.setAttribute(**"user"**,user);  **%>** <**p**>获取request域中user对象里面name的值：**${**user.name**}**</**p**>*<%--el获取javaBean对象直接.属性名就可以，el会自动调用该属性getXXX封装方法--%>* <**p**>获取request域中user对象里面age的值：**${**user.age**}**</**p**> |

​

#### 获取JavaBean数据注意事项

1. JavaBean数据获取类似获取map集合的方式，可以使用”.“获取数据的地方，都可以使用"[]"获取数据。
2. 如果el表达式获取不到数据，页面没有显示内容，不是显示”null“
3. el在获取JavaBean的数据时候，底层调用的是getXXX方法。

### EL执行运算

​ EL不仅可以用来获取数据，之前的案例我们还看到了可以执行运算，因此，接下来我们要学习EL执行运算相关的知识点，它包括了算术运算、逻辑运算、比较运算、empty运算符、三元运算

#### 支持算术运算符：+  -  \*  /  %

jsp演示：

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>Title</title>  
</head>  
<body>  
<%  
    request.setAttribute("x",3);  
    request.setAttribute("y",4);  
    request.setAttribute("z","5");  
%>  
${x + y}<br>  
${x - y}<br>  
${x \* y}<br>  
${x / y}<br>  
${x % y}<br>  
<%--注意事项:  
    1 在EL中，只要是数字就能执行运算，EL在执行计算的时候，会将数字除尽，最多保留16位小数   
    2 在EL中，如果在一个算式中有数据不存在，那么这个数据不参与运算，不报错继续执行。  
--%>  
${x+y+z}<br>  
${x+y+z+a}<br>  
</body>  
</html>  
​

课堂代码

|  |
| --- |
| <**h2**>el的运算符</**h2**> <**p**>5/2=**${**5/2**}**</**p**>*<%--el与java不同，可以除尽，最多保留小数16位--%>* <**p**>**${**2+**"3"**+5+a**}**</**p**>*<%--el中如果与数字相加的字符串，会转换为数字之后进行计算，注意如果数据没有也不影响计算--%>* |

#### 算术运算注意事项

1. 在EL中，只要是数字就能执行运算，EL在执行计算的时候，会将数字除尽，最多保留16位小数
2. 在EL中，如果在一个算式中有数据不存在，那么这个数据不参与运算，不报错继续执行。

#### 逻辑运算符

下图展示了EL可以支持的逻辑运算，注意：英文和符号效果一致，推荐使用符号



<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>Title</title>  
</head>  
<body>  
<%  
    request.setAttribute("flag", true);  
    request.setAttribute("info", false);  
​  
%>  
${flag && info }  
${flag || info }  
${!info }  
</body>  
</html>  
​

#### 逻辑运算符注意事项

​ 注意：逻辑运算中的异或"^"EL不支持。

#### 比较运算

下图展示了EL支持的比较运算符，注意：英文和符号效果一致，推荐使用符号



jsp代码演示：

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>Title</title>  
</head>  
<body>  
<%  
    request.setAttribute("x",3);  
    request.setAttribute("y",4);  
%>  
${x < y }  
${x <= y }  
${x > y }  
${x >= y }  
${x == y }  
${x != y }  
</body>  
</html>  
​

#### 比较运算注意事项

​ 注意：使用比较运算符要保证数据是存在的并且是可比较的。例如：不要出现设置数据为request.setAttribute("y",null);然后再进行比较。

#### empty运算符和三目运算符

​ empty运算符是用来判断当获取的数据是否为空或者当前获取的集合是否没有任何数据，三元运算符和java的三元运算符功能一致。

jsp：

|  |
| --- |
| <**h2**>使用empty判断空</**h2**> *<%--  ${empty 域里面的数据}  1.判断数据是否为null,为null 返回true  2.判断数据是否为空字符串，为空字符串，返回true  3.判断数据是集合的话，如果集合元素的个数为0，返回true --%>* <**p**>空字符串：**${empty ""}**</**p**> <**p**>空对象：**${empty null}**</**p**> **<%** request.setAttribute(**"list"**,**new** ArrayList<String>()); **%>** <**p**>集合元素个数为0：**${empty** list**}**</**p**> **<%** List<String> list = **new** ArrayList<String>();  list.add(**"abc"**);  request.setAttribute(**"list"**,list);  **%>** <**p**>集合元素个数为1：**${empty** list**}**</**p**>  *<%-- ${not empty 域里面的数据} 判空取反 --%>* |

#### empty运算符注意事项

​ 注意：以上的empty运算符案例中，empty运算符可以和比较运算符组合使用。

​ 例如：${not empty str}表示str不为空，返回true。

# JSTL的核心标签库使用

## jstl标签的基本概述

### jstl的概述

​jsp的标准标签库（Jsp Standard Tag Library），简称JSTL。此技术由Apache Jakarta小组研发。jsp标准标签库，由很多标签组成，这些标签称为服务器端标签。服务器端标签的作用是封装了具有特定的功能的java代码，以标签形式编写，代替特定功能的java代码。

**JSTL标签库提供5大功能（了解）：**

1. core：jstl的核心标签库。（目前还在使用）
2. fmt：格式化（国际化）的标签（使用较少，对页面显示数据，格式化，现在都交给前端去做）
3. functions：jstl中提供对字符串操作的函数库(不再使用，建议在数据显示在页面之前，在后台程序中，先格式化好字符串，然后直接显示，不再页面做处理，如果有前端，交给前端处理（javascript  解析json格式数据）)
4. sql：jstl提供的在jsp页面上书写sql，操作数据库，（jsp只负责布局显示动态资源数据，不做数据库操作，交给dao层操作数据库）
5. xml：jstl操作xml文件的。目前已经不再使用（jsp只负责布局显示动态资源数据，解析xml在java源代码src下面解析）

**因此我们需要了解的只由jstl的核心标签库。**

### jstl核心标签库列表

| **标签名称** | **作用** |
| --- | --- |
| <c:out> | 通常用于输出一段文本内容到客户端浏览器 |
| <c:set> | 用于设置各种Web域中的属性 |
| <c:remove> | 用于删除各种Web域中的属性 |
| <c:catch> | 用于捕获嵌套在标签体中的内容抛出的异常 |
| <c:if> | 用户java代码if(){}语句功能 |
| <c:choose> | 用于指定多个条件选择的组合边界，它必须与<c:when>和<c:otherwise>标签一起使用 |
| <c:forEach> | 用户代替java代码for循环语句 |
| <c:forTokens> | 用户迭代操作String字符 |
| <c:param> | 给请求路径添加参数 |
| <c:url> | **重写url，在请求路径添加sessionid** |
| <c:import> | **用于在JSP页面中导入一个URL地址指向的资源内容** |
| <c:redirect> | **用于将当前的访问请求转发或重定向到其他资源** |

## jstl标签的安装

​ 我们知道了jstl可以帮助我们解决jsp页面出现java和提高java在页面的重用性问题，那么接下来，我们需要的是学习使用jstl。

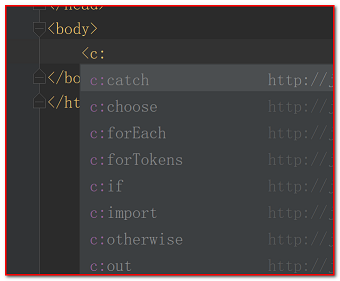
### 导入jar包

javax.servlet.jsp.jstl.jar  
jstl-impl.jar

### 使用taglib指令在jsp页面导入要使用的jstl标签库

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

在jsp书写"<c:"，看到如下提示，说明安装成功：



## 常用的jstl标签

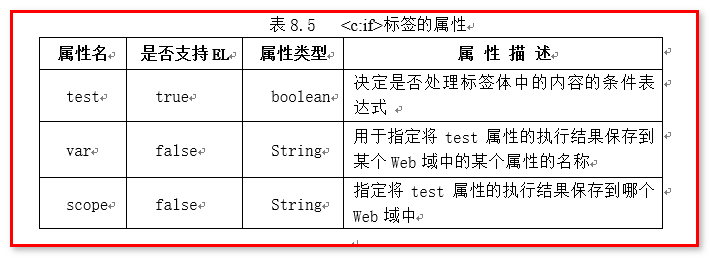
​ jstl的核心标签内容有很多，现在目前还常用的标签只有if、choose、foreach标签，接下来我们一个一个学习。

### if标签

#### if标签作用

​ **起到java代码的判断的作用**

#### if标签属性介绍



​ **test：判断是否执行标签内的内容（true——执行标签中的内容，false，不执行）。**

​ **var：用来保存test属性的结果（使用var属性给他取个名字），这个结果可以保存到指定的容器中。**

​ **scope：指定保存数据的容器域对象。**

​ **注：是否支持EL表达式——是否可以书写EL表达式在属性中。**

#### if标签注意事项

​ **if标签，相当于java中的if(){}语句，而不是if(){}else{}语句**

​ **按照属性的数据类型传入数据，否则报错**

#### if标签演示

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<html>  
<head>  
    <title>Title</title>  
</head>  
<body>  
​  
    <c:if test="${true}" scope="session" var="flag">  
       测试if标签，设置test为true  
    </c:if>  
    <br>测试if标签,获取test属性的值，从知道的session容器中获取：${sessionScope.flag}  
</body>  
</html>  
​

​ 课堂代码

|  |
| --- |
| <%@ **taglib prefix**="**c**" **uri**="**http://java.sun.com/jsp/jstl/core**" %> <%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %> <**html**> <**head**>  <**title**>if标签</**title**> </**head**> <**body**> *<%--  if标签  格式：<c:if test="条件表达式"> //条件成立时运行的数据 </c:if>  作用，相当于if(){}  常用属性  test，用于编写逻辑条件表达式  scope,用于将表达式的值存储到指定域里面，request/session/application  var,用于设置域里面的name --%>* <**c:if test="${false}" scope="request" var="flag"**>  <**p**>进入了if条件句</**p**> </**c:if**>  表达式里面的值：**${**flag**}**，外部的语句 </**body**> </**html**> |

### choose标签

#### choose标签作用

​ <c:choose>标签用于指定多个条件选择的组合边界，它必须与<c:when>和<c:otherwise>标签一起使用。三个标签组合发挥java代码if(){}else if(){} else{}语句的作用。

#### choose标签子标签介绍

​ <c:when>，相当于else if(){}。<c:when>标签含有test属性，作用与if相同

​ <c:otherwise>，相当于else{}。

#### choose标签注意事项

​ 三个标签必须组合使用，一组标签中不能出现两个<c:otherwise>。

#### choose标签演示

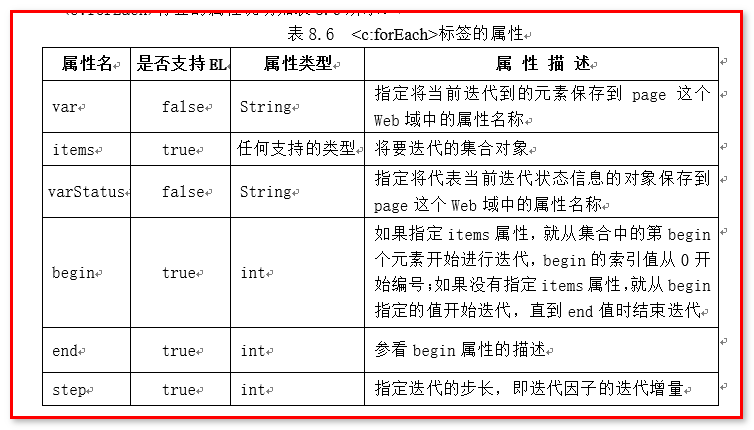
|  |
| --- |
| <%@ **taglib prefix**="**c**" **uri**="**http://java.sun.com/jsp/jstl/core**" %> <%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %> <**html**> <**head**>  <**title**>if标签</**title**> </**head**> <**body**> *<%--  choose标签  格式：  <c:choose>  <c:when test=""></c:when>======>if  <c:when test=""></c:when>======>else if  <c:otherwise></c:otherwise>====>else  </c:choose>  作用：  相当于 if-elseif-esle结构  常用属性  test，用于编写逻辑条件表达式  注意：  不能在服务器端标签里面写html注释，不支持  --%>* **<%** request.getSession().setAttribute(**"loginUser"**,**null**); **%>** <**c:choose**>  <**c:when test="${not empty** loginUser**}"**>  欢迎，**${**loginUser**}**用户，登录成功  </**c:when**>  <**c:otherwise**>  您，还没有登录  </**c:otherwise**> </**c:choose**>  </**body**> </**html**> |

### foreach标签

#### foreach标签作用

​ **起到java代码的for循环作用**

#### foreach标签属性介绍



**var：在不循环对象的时候，保存的是控制循环的变量；在循环对象的时候，保存的是被循环对象中的元素**

**items：指定要循环的对象**

**varStatus：保存了当前循环过程中的信息（循环的开始、结束、步长、次数等）**

**begin：设置循环的开始**

**end：设置循环的结束**

**step：设置步长——间隔几次循环，执行一次循环体中的内容**

#### foreach标签演示

​ 演示foreach循环标签的时候我们分开两种情况：不循环对象和循环对象。

​ 我们先来看不循环对象的时候：

##### foreach演示

|  |
| --- |
| <%@ **page import**="**com.itheima.model.User**" %> <%@ **page import**="**java.util.List**" %> <%@ **page import**="**java.util.ArrayList**" %> <%@ **taglib prefix**="**c**" **uri**="**http://java.sun.com/jsp/jstl/core**" %> <%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %> <**html**> <**head**>  <**title**>forEach标签</**title**> </**head**> <**body**> *<%--  forEach标签  格式：  普通有次数循环格式：  <c:forEach begin="1" end="5" step="1">  <p>hello world</p>  </c:forEach>  增强for循环格式：  <c:forEach items="${userList}" var="user" varStatus="i">  <p>${i.index},${i.count},${user.name},${user.age}</p>  </c:forEach>  作用：  循环遍历，相当于for循环  常用属性  begin,循环开始值  end,循环结束值  step="1",叫步长，循环递增量，默认是1  items，循环遍历的集合，一般从域里面获取  var,循环变量，存储循环元素对象的，jstl会将当前的名字和对象存储到域里面，所以，可以通过el获取数据  varStatus="i",循环过程信息对象  forEach标签在循环过程中，产生了很多循环过程信息（count循环的序号，index循环的索引），并且将  这个属性设置的值对象存储的域里面，我们可以通过${i.count}获取循环序号，${i.index}获取循环索引     注意：  不能在服务器端标签里面写html注释，不支持  --%> <%--要求输出5遍hello world--%>* <**c:forEach begin="1" end="5" step="1"**>  <**p**>hello world</**p**> </**c:forEach**> *<%--增强for循环，遍历用户列表集合--%>* **<%** *//往域里面写List数据* List<User> userList = **new** ArrayList<User>();  userList.add( **new** User(20,**"张三"**));  userList.add( **new** User(22,**"李四"**));  request.setAttribute(**"userList"**,userList);  **%>** <**c:forEach items="${**userList**}" var="user" varStatus="i"**> <**p**>**${**i.index**}**,**${**i.count**}**,**${**user.name**}**,**${**user.age**}**</**p**> </**c:forEach**>  </**body**> </**html**> |

循环过程中具体的过程信息如下：

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<html>  
<head>  
    <title>Title</title>  
</head>  
<body>  
<%-- 演示foreach标签 --%>  
<%--  
varStatus :保存了当前循环过程中的信息，信息包括以下内容：  
  1 public java.lang.Integer getBegin()  
 返回为标签设置的begin属性的值，如果没有设置begin属性则返回null  
  2 public int getCount()  
 返回当前已循环迭代的次数  
  3 public java.lang.Object getCurrent()  
 返回当前迭代到的元素对象  
  4 public java.lang.Integer getEnd()  
  返回为标签设置的end属性的值，如果没有设置end属性则返回null  
  5 public int getIndex()  
 返回当前迭代的索引号  
  6 public java.lang.Integer getStep()  
 返回为标签设置的step属性的值，如果没有设置step属性则返回null  
  7 public boolean isFirst()  
 返回当前是否是第一次迭代操作  
  8 public boolean isLast()  
 返回当前是否是最后一次迭代操作  
--%>  
<c:forEach begin="11" end="15" step="1" var="info" varStatus="sta">  
    <td>${sta.index}</td>  
    <td>${sta.count}</td>  
    <td>${sta.first}</td>  
    <td>${sta.last}</td><br>  
</c:forEach>  
</body>  
</html>  
​

接下来我们在来看foreach如何循环对象

##### foreach循环对象（数组、list、map）

<%--import="java.util.\*"导入java.util下的内容，给当前jsp使用--%>  
<%@ page contentType="text/html;charset=UTF-8" import="java.util.\*" language="java" %>  
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  
<html>  
<head>  
    <title>Title</title>  
</head>  
<body>  
<%--演示循环数组--%>  
<%  
    int[] arr = {666,888,999,1024};  
    request.setAttribute("arr", arr);  
​  
%>  
<%-- var在循环对象的时候，临时保存被循环到元素 --%>  
<c:forEach items="${arr }" var="num">  
${num }  
</c:forEach>  
<hr>  
<%  
    List list = new ArrayList();  
    list.add("卡奴");  
    list.add("兰恩");  
    list.add("云娜");  
​  
    request.setAttribute("list", list);  
​  
%>  
<c:forEach items="${list }" var="wind">  
   ${wind }  
</c:forEach>  
<hr>  
<%  
    Map map = new HashMap();  
    map.put("ms1", "简历");  
    map.put("ms2", "身份证");  
    map.put("ms3", "学历证明");  
    map.put("ms4", "体检报告");  
​  
    request.setAttribute("map", map);  
​  
%>  
<c:forEach items="${map }" var="entry">  
​  
   ${entry.key }  
   ${entry.value }  
</c:forEach>  
</body>  
</html>  
​

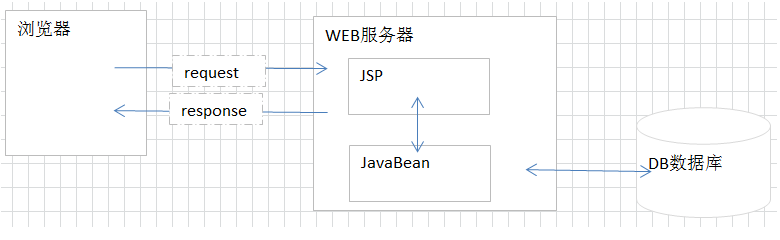
# 三层架构和MVC模式

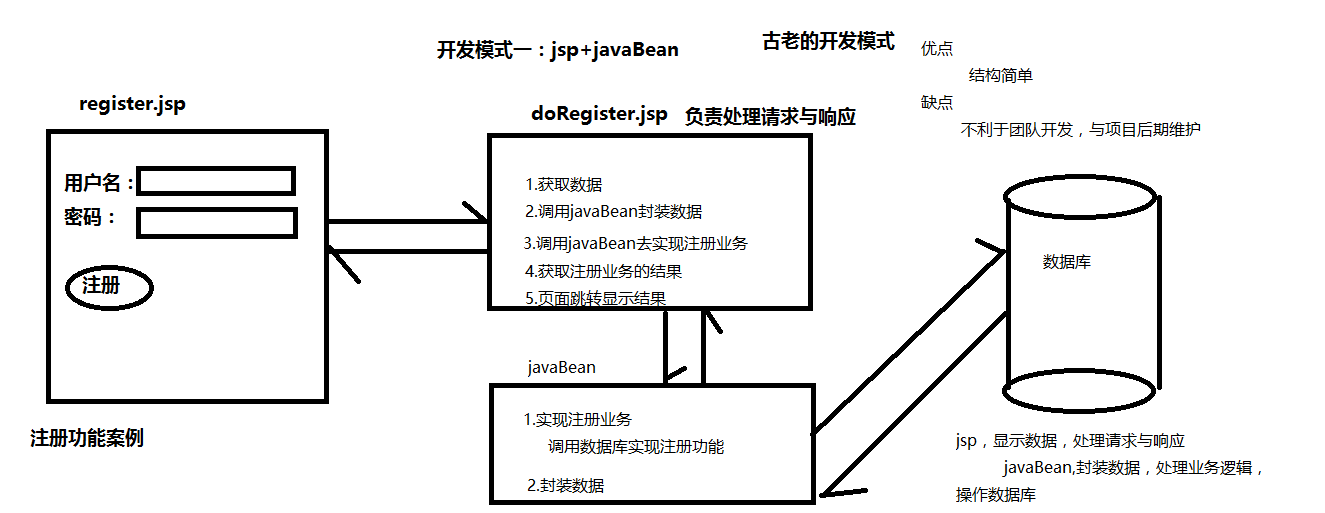
## JSP开发模式

当SUN公司推出JSP后，同时也提供相应的开发模式，JavaWeb经历了JSP Model1 第一代，JSPModel2第二代，JSP Model 3 三个时期。

### JSP Model1 第一代

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，所有的东西都在一起，对后期的维护和扩展极为不利。

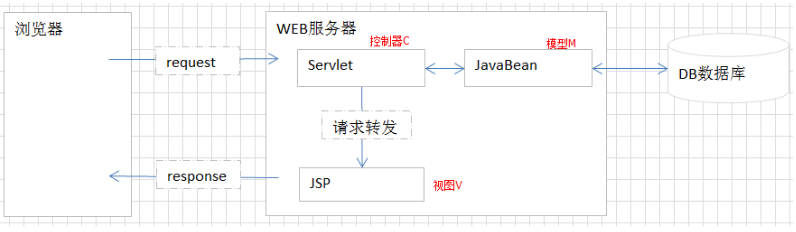


​ 

### JSP Model2 第二代

JSP Model1第二代有所改进，把业务逻辑的内容放到了JavaBean中，而JSP页面只负责显示，servlet负责处理请求与调度的工作。虽然第二代比第一代好了些，但还让JavaBean做了过多的工作，java Bean依然要封装数据、处理业务和操作数据库等混合在一起了。​

JSP Model 2 Model2使用到的技术有：Servlet、JSP、JavaBean。Model2 是MVC设计模式在Java语言的具体体现。λ JSP：视图层，用来与用户打交道。负责接收用来的数据，以及显示数据给用户；λ 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；λ JavaBean：模型层，完成具体的业务工作，例如：转账等。​

​ 

#### MVC开发模式

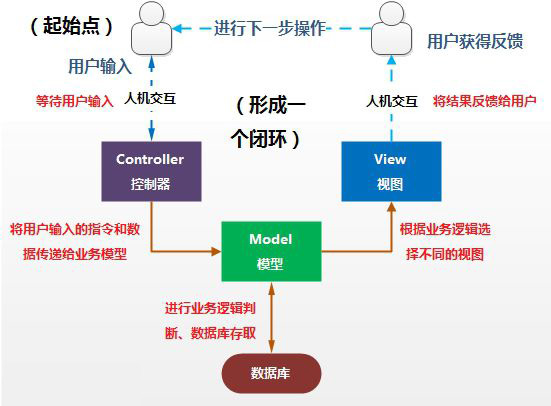
##### 什么是MVC(模型-视图-控制器)

M:Model 模型层：由业务逻辑和数据访问层组成，由JavaBean来实现

C:Controller 控制器，由Servlet实现

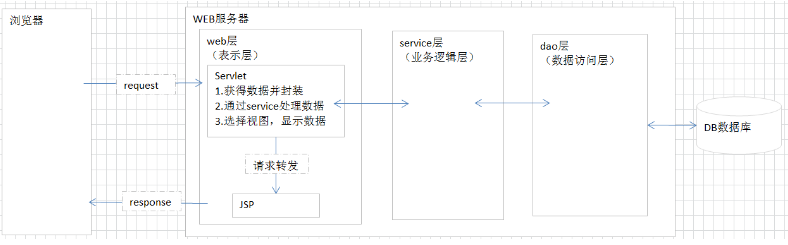
V:View 表示层，也称为Web层，由JSP实现

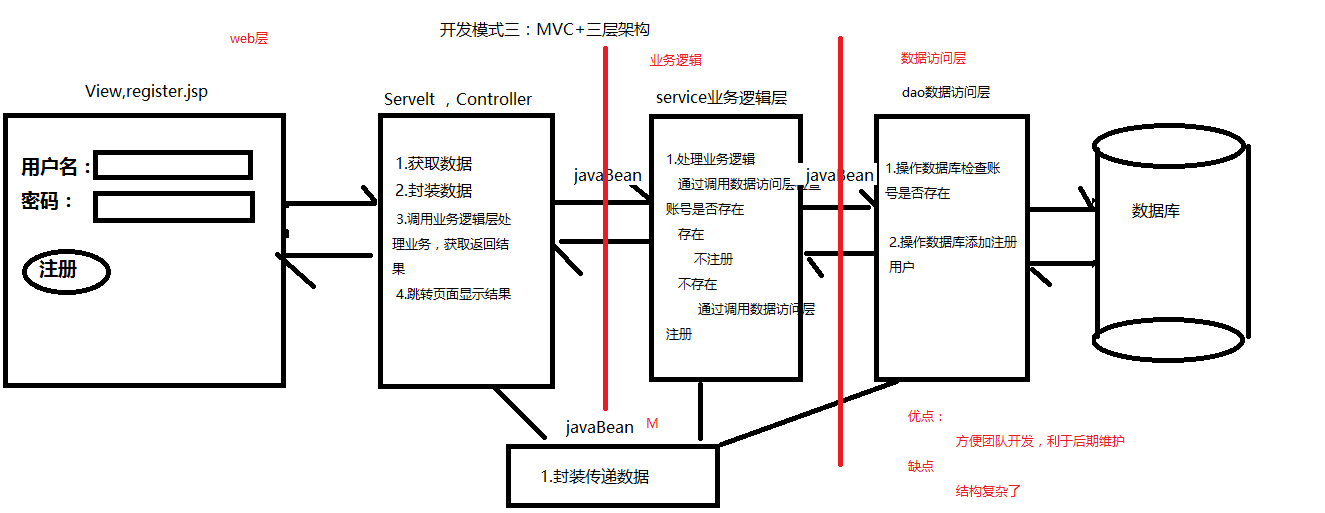
在JavaWeb中模式2就是MVC的一种实现



### MVC+三层架构

JSP模式是理论基础，但实际开发中，我们常将服务器端程序，根据逻辑进行分层。一般比较常见的是分三层，我们称为：经典三层体系架构。三层分别是：表示层、业务逻辑层、数据访问层。λ 表示层：又称为 web层，与浏览器进行数据交互的。λ 业务逻辑层：又称为service层，专门用于处理业务数据的。λ 数据访问层：又称为dao层，与数据库进行数据交换的。将数据库的一条记录与JavaBean进行对应。





# 使用三层架构和MCV模式完成用户显示列表案例

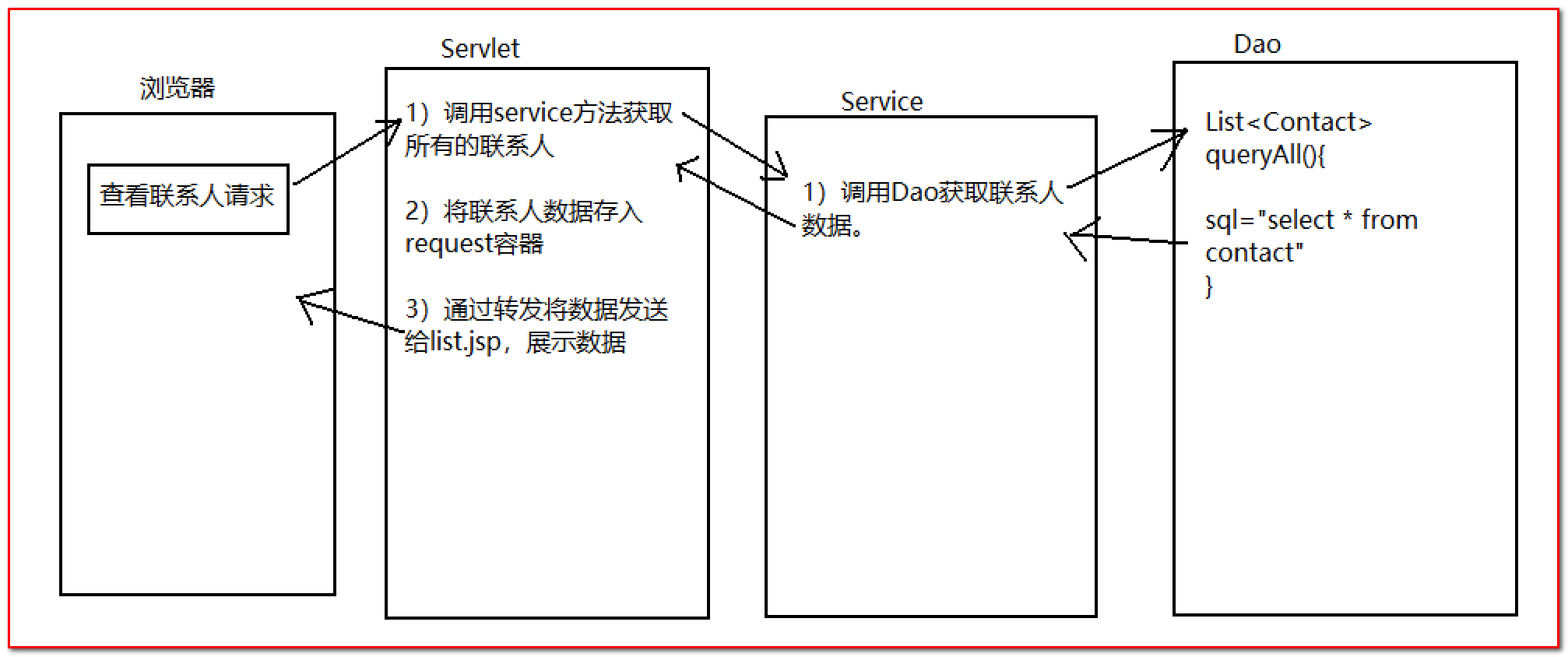
## 案例需求

​ 使用三层架构和MVC模式开发代码，完成用户显示列表功能。

## 案例效果



## 案例分析



## 实现步骤

### 导入页面

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<!DOCTYPE html>  
<!-- 网页使用的语言 -->  
<html lang="zh-CN">  
<head>  
    <!-- 指定字符集 -->  
    <meta charset="utf-8">  
    <!-- 使用Edge最新的浏览器的渲染方式 -->  
    <meta http-equiv="X-UA-Compatible" content="IE=edge">  
    <!-- viewport视口：网页可以根据设置的宽度自动进行适配，在浏览器的内部虚拟一个容器，容器的宽度与设备的宽度相同。  
    width: 默认宽度与设备的宽度相同  
    initial-scale: 初始的缩放比，为1:1 -->  
    <meta name="viewport" content="width=device-width, initial-scale=1">  
    <!-- 上述3个meta标签\*必须\*放在最前面，任何其他内容都\*必须\*跟随其后！ -->  
    <title>Bootstrap模板</title>  
​  
    <!-- 1. 导入CSS的全局样式 -->  
    <link href="css/bootstrap.min.css" rel="stylesheet">  
    <!-- 2. jQuery导入，建议使用1.9以上的版本 -->  
    <script src="js/jquery-2.1.0.min.js"></script>  
    <!-- 3. 导入bootstrap的js文件 -->  
    <script src="js/bootstrap.min.js"></script>  
    <style type="text/css">  
        td, th {  
            text-align: center;  
       }  
    </style>  
</head>  
<body>  
<div class="container">  
    <h3 style="text-align: center">显示所有联系人</h3>  
    <table border="1" class="table table-bordered table-hover">  
        <tr class="success">  
            <th>编号</th>  
            <th>姓名</th>  
            <th>性别</th>  
            <th>年龄</th>  
            <th>籍贯</th>  
            <th>QQ</th>  
            <th>邮箱</th>  
            <th>操作</th>  
        </tr>  
        <tr>  
            <td>1</td>  
            <td>张三</td>  
            <td>男</td>  
            <td>20</td>  
            <td>广东</td>  
            <td>44444222</td>  
            <td>zs@qq.com</td>  
            <td><a class="btn btn-default btn-sm" href="修改联系人.html">修改</a>&nbsp;<a class="btn btn-default btn-sm" href="修改联系人.html">删除</a></td>  
        </tr>  
        <tr>  
            <td>2</td>  
            <td>张三</td>  
            <td>男</td>  
            <td>20</td>  
            <td>广东</td>  
            <td>44444222</td>  
            <td>zs@qq.com</td>  
            <td><a class="btn btn-default btn-sm" href="修改联系人.html">修改</a>&nbsp;<a class="btn btn-default btn-sm" href="修改联系人.html">删除</a></td>  
        </tr>  
        <tr>  
            <td>3</td>  
            <td>张三</td>  
            <td>男</td>  
            <td>20</td>  
            <td>广东</td>  
            <td>44444222</td>  
            <td>zs@qq.com</td>  
            <td><a class="btn btn-default btn-sm" href="修改联系人.html">修改</a>&nbsp;<a class="btn btn-default btn-sm" href="修改联系人.html">删除</a></td>  
        </tr>  
        <tr>  
            <td>4</td>  
            <td>张三</td>  
            <td>男</td>  
            <td>20</td>  
            <td>广东</td>  
            <td>44444222</td>  
            <td>zs@qq.com</td>  
            <td><a class="btn btn-default btn-sm" href="修改联系人.html">修改</a>&nbsp;<a class="btn btn-default btn-sm" href="修改联系人.html">删除</a></td>  
        </tr>  
        <tr>  
            <td>5</td>  
            <td>张三</td>  
            <td>男</td>  
            <td>20</td>  
            <td>广东</td>  
            <td>44444222</td>  
            <td>zs@qq.com</td>  
            <td><a class="btn btn-default btn-sm" href="修改联系人.html">修改</a>&nbsp;<a class="btn btn-default btn-sm" href="修改联系人.html">删除</a></td>  
        </tr>  
        <tr>  
            <td colspan="8" align="center"><a class="btn btn-primary" href="添加联系人.html">添加联系人</a></td>  
        </tr>  
    </table>  
</div>  
</body>  
</html>  
​

### 导入页面相关的资源文件

复制今天资料文件夹/案例原型下的三个文件夹到web根路径：

css

fonts

js

### 导入jar包、配置文件、实体类和工具类

1. 导入jar包
2. commons-beanutils-1.8.3.jar
3. commons-logging-1.1.1.jar
4. druid-1.0.9.jar
5. javax.servlet.jsp.jstl.jar
6. jstl-impl.jar
7. mysql-connector-java-5.1.18-bin.jar
8. spring-beans-4.2.4.RELEASE.jar
9. spring-core-4.2.4.RELEASE.jar
10. spring-jdbc-4.2.4.RELEASE.jar
11. spring-tx-4.2.4.RELEASE.jar

​

1. 配置文件:druid-config.properties
2. 实体类

package cn.itcast.domain;  
​  
public class Contact {  
​  
    private int id;  
    private String name;  
    private String sex;  
    private int age;  
    private String address;  
    private String qq;  
    private String email;  
​  
    public int getId() {  
        return id;  
   }  
​  
    public void setId(int id) {  
        this.id = id;  
   }  
​  
    public String getName() {  
        return name;  
   }  
​  
    public void setName(String name) {  
        this.name = name;  
   }  
​  
    public String getSex() {  
        return sex;  
   }  
​  
    public void setSex(String sex) {  
        this.sex = sex;  
   }  
​  
    public int getAge() {  
        return age;  
   }  
​  
    public void setAge(int age) {  
        this.age = age;  
   }  
​  
    public String getAddress() {  
        return address;  
   }  
​  
    public void setAddress(String address) {  
        this.address = address;  
   }  
​  
    public String getQq() {  
        return qq;  
   }  
​  
    public void setQq(String qq) {  
        this.qq = qq;  
   }  
​  
    public String getEmail() {  
        return email;  
   }  
​  
    public void setEmail(String email) {  
        this.email = email;  
   }  
​  
    @Override  
    public String toString() {  
        return "Contact{" +  
                "id=" + id +  
                ", name='" + name + '\'' +  
                ", sex='" + sex + '\'' +  
                ", age=" + age +  
                ", address='" + address + '\'' +  
                ", qq='" + qq + '\'' +  
                ", email='" + email + '\'' +  
                '}';  
   }  
}

​

1. 导入工具类JDBCUtils​
2. 执行sql脚本

|  |
| --- |
| CREATE TABLE IF NOT EXISTS `contact` (  `id` int(11) NOT NULL AUTO\_INCREMENT COMMENT '编号',  `name` varchar(50) DEFAULT NULL COMMENT '姓名',  `sex` varchar(50) DEFAULT NULL COMMENT '性别',  `age` int(11) DEFAULT NULL COMMENT '年龄',  `address` varchar(50) DEFAULT NULL COMMENT '籍贯',  `qq` varchar(50) DEFAULT NULL COMMENT 'QQ号',  `email` varchar(50) DEFAULT NULL COMMENT '邮箱',  PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=11 DEFAULT CHARSET=utf8 COMMENT='联系人表';  INSERT INTO `contact` (`id`, `name`, `sex`, `age`, `address`, `qq`, `email`) VALUES  (1, '张三', '男', 11, '广东', '766335435', '766335435@qq.com'),  (2, '李四', '男', 12, '广东', '243424242', '243424242@qq.com'),  (3, '王五', '女', 13, '广东', '474574574', '474574574@qq.com'),  (4, '赵六', '女', 18, '广东', '77777777', '77777777@qq.com'),  (5, '钱七', '女', 15, '湖南', '412132145', '412132145@qq.com'); |

### 编写servlet代码

|  |
| --- |
| **package** com.itheima.web;  **import** com.itheima.model.Concat; **import** com.itheima.service.ConcatService;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException; **import** java.util.List;  @WebServlet(name = **"ConcatListServlet"**, urlPatterns = **"/ConcatListServlet"**) **public class** ConcatListServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   *//实例业务类* **private** ConcatService **concatService** = **new** ConcatService();  **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {     **try** {  *//查询联系人列表  //调用业务逻辑类查询用户列表* List<Concat> concatList = **concatService**.findAll();  *//将列表存储到域里面* request.setAttribute(**"concatList"**,concatList);  *//跳转页面list.jsp显示数据* request.getRequestDispatcher(**"/list.jsp"**).forward(request,response);  } **catch** (Exception e) {  e.printStackTrace();  response.setContentType(**"text/html;charset=utf-8"**);  response.getWriter().write(**"服务器正忙。。。请求稍后再试。。。"**);  }  } } |

### 编写service代码

|  |
| --- |
| **package** com.itheima.service;  **import** com.itheima.dao.ConcatDao; **import** com.itheima.model.Concat;  **import** java.util.List;  */\*\*  \* 联系人业务类  \*/* **public class** ConcatService {   *//实例数据访问类* **private** ConcatDao **concatDao** = **new** ConcatDao();   **public** List<Concat> findAll()**throws** Exception {  *//调用dao查询数据库用户列表* **return concatDao**.findAll();  } } |

### 编写dao代码

|  |
| --- |
| **package** com.itheima.dao;  **import** com.itheima.model.Concat; **import** com.itheima.util.JdbcUtils; **import** org.springframework.dao.DataAccessException; **import** org.springframework.jdbc.core.BeanPropertyRowMapper; **import** org.springframework.jdbc.core.JdbcTemplate;  **import** java.sql.SQLException; **import** java.util.List;  */\*\*  \* 联系人数据访问类  \*/* **public class** ConcatDao {  *//核心类* **private** JdbcTemplate **jdbcTemplate** = **new** JdbcTemplate(JdbcUtils.*getDataSource*());   */\*\*  \* 查询数据用户列表  \** ***@return*** *List*<*Concat*>  *\** ***@throws*** *SQLException  \*/* **public** List<Concat> findAll()**throws** SQLException {  **try** {  *//定义sql* String sql=**"SELECT** *\** **FROM contact"**;  *//执行sql* **return jdbcTemplate**.query(sql,**new** BeanPropertyRowMapper<Concat>(Concat.**class**));  } **catch** (DataAccessException e) {  e.printStackTrace();  **return null**;  }  } } |

### list.jsp

|  |
| --- |
| <**div class="container"**>  <**h3 style="text-align**: **center"**>显示所有联系人</**h3**>  <**table border="1" class="table table-bordered table-hover"**>  <**tr class="success"**>  <**th**>编号</**th**>  <**th**>姓名</**th**>  <**th**>性别</**th**>  <**th**>年龄</**th**>  <**th**>籍贯</**th**>  <**th**>QQ</**th**>  <**th**>邮箱</**th**>  <**th**>操作</**th**>  </**tr**>  <**c:forEach items="${**concatList**}" var="concat"**>  <**tr**>  <**td**>**${**concat.id**}**</**td**>  <**td**>**${**concat.name**}**</**td**>  <**td**>**${**concat.sex**}**</**td**>  <**td**>**${**concat.age**}**</**td**>  <**td**>**${**concat.address**}**</**td**>  <**td**>**${**concat.qq**}**</**td**>  <**td**>**${**concat.email**}**</**td**>  <**td**><**a class="btn btn-default btn-sm" href="修改联系人.html"**>修改</**a**>**&nbsp;**<**a class="btn btn-default btn-sm" href="修改联系人.html"**>删除</**a**></**td**>  </**tr**>  </**c:forEach**>  <**tr**>  <**td colspan="8" align="center"**><**a class="btn btn-primary" href="添加联系人.html"**>添加联系人</**a**></**td**>  </**tr**>  </**table**> </**div**> |