

# jsp&el&jstl

---

## 学习目标

---

1. 能够在jsp中编写java代码
2. 能够说出el表达式的作用
3. 能够使用el表达式获取javabean的属性
4. 能够使用jstl标签库的if标签
5. 能够使用jstl标签库的foreach标签
6. 能够说出开发模式的作用
7. 能够使用三层架构模式完成显示用户案例

## 第1章 jsp入门

---

### 1.1 jsp简介

---

#### 1.1.1 jsp的概念

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，本质上就是一个Servlet，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：

- **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
- **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。
- **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。
- **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。

#### 1.1.2 为什么要诞生JSP

我们先来设计一个场景，完成一个需求：往页面输出一个表格。

要实现此需求,我们就需要在servlet中使用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？

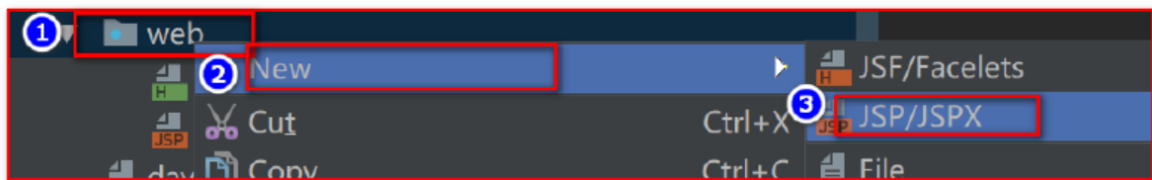
应该有以下两点：

- 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。

### 1.1.3 jsp初体验

#### 1.1.3.1 创建jsp文件



#### 1.1.3.2 编写jsp代码

```
<%@ page import="java.util.Date" %>
<%@ page import="java.text.SimpleDateFormat" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>测试jsp</title>
</head>
<body>
<%
    Date date = new Date();
    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    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>
```

#### 1.1.3.3 访问jsp

在地址栏输入jsp文件名称访问即可。

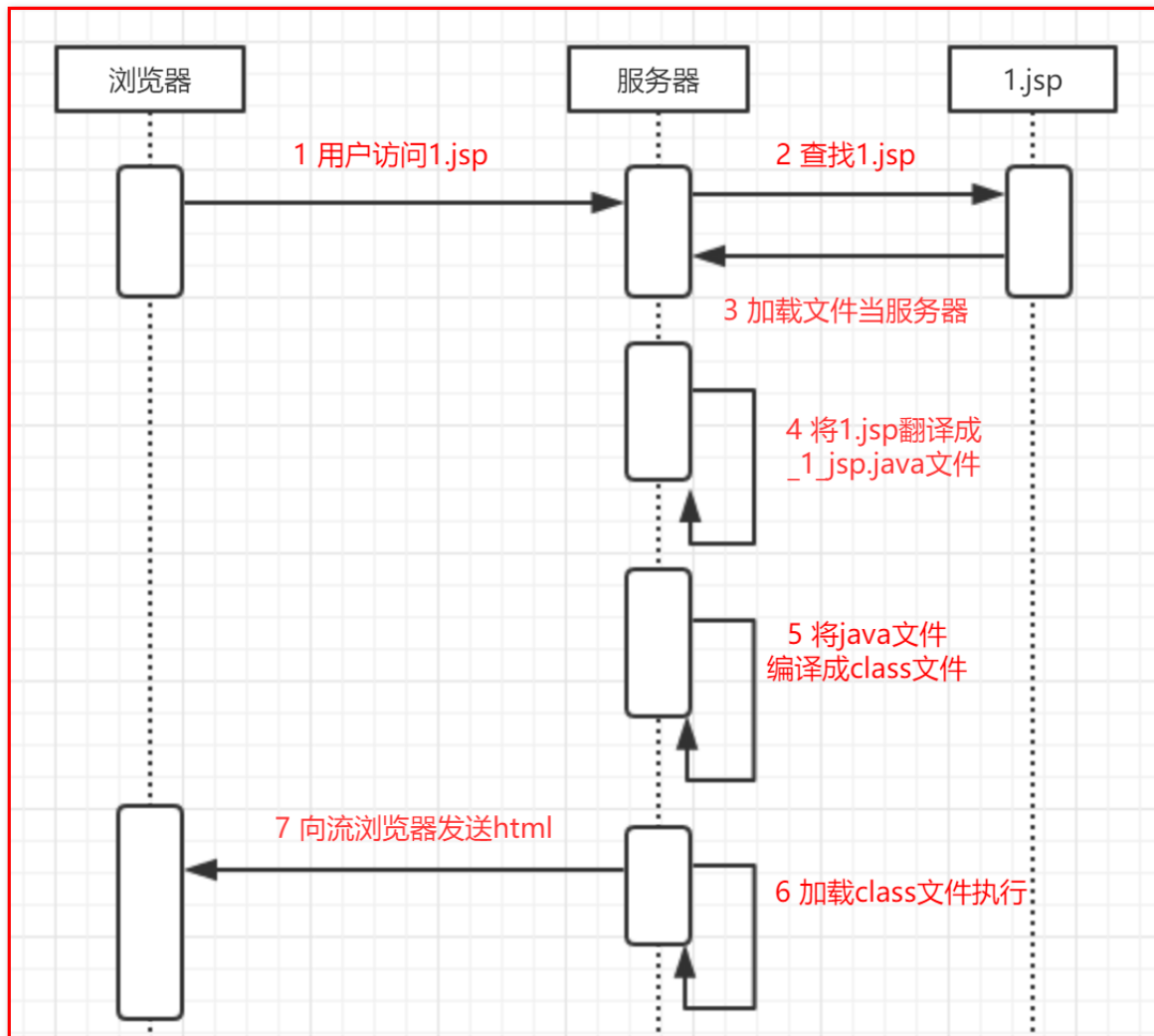
效果：

---

当前系统时间: 2019-12-20 09:04:20

## 1.2 jsp的执行流程

通过上面的jsp初体验, 我们看到jsp中既可以编写java代码也可以直接编写html代码, 相对servlet更加方便, 为啥jsp中可以直接使用request对象呢? 接下来我们来看下它的执行流程



依据上图的流程, 在本地找到了生成的java文件, 其中一部分内容如下图:

```
public final class _1_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent,
               org.apache.jasper.runtime.JspSourceImports {
```

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:

```
/**
 * This is the super class of all JSP-generated servlets.
 *
 * @author Anil K. Vijendran
 */
public abstract class HttpJspBase extends HttpServlet imp
```

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。

```

public void jspService(final javax.servlet.http.HttpServletRequest request, final javax.servlet.http.HttpServletResponse response)
    throws java.io.IOException, javax.servlet.ServletException {

    response.setContentType("text/html;charset=UTF-8");
    pageContext = _jspxFactory.getPageContext(this, request, response,
        null, true, 8192, true);
    _jspx_page_context = pageContext;
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
    out = pageContext.getOut();
    _jspx_out = out;

    Date date = new Date();
    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    String curTime = dateFormat.format(date);
    request.setAttribute("curTime", curTime);

    out.write("\r\n");
    out.write("当前系统时间: <span style='color: red;'>"); 我们自己编写的代码
    out.print(request.getAttribute("curTime"));
    out.write("</span>\r\n");
    out.write("</body>\r\n");
    out.write("</html>\r\n");
}

```

## 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之所以能够直接使用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,response,session,application(ServletContext)等对象了,这些对象成为之jsp内置对象。

## 1.3 jsp的基本语法

### 1.3.1 jsp注释

#### 1.3.1.1 JSP注释格式

<%-- jsp注释 --%>

#### 1.3.1.2 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。

```
<html>
  <head>
    <title>注释</title>
  </head>
  <body>
    <!--这个是html注释-->

  </body>
</html>
```

## 总结

jsp的注释不会显示在源码上，更加安全，所以在jsp页面中推荐使用jsp的注释。

## 1.3.2 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。

### 1.3.2.1 脚本片段

#### ① 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:

```
123 out.write(" ");
124 out.print( "传智播客" );
125 out.write("\r\n");
126 out.write("\r\n");
127 out.write(" ");
128 out.write("\r\n");
129 out.write(" ");
130 int x = 10;
131 out.write("\r\n");
```

## ② 脚本片段作用

在jsp翻译后的\_jspService方法中，嵌入java代码

## ③ 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:

```
131 out.write("\r\n");
132 out.write(" ");
133 for(int j = 0 ; j < 5 ;j++){
134     out.write("\r\n");
135     out.write(" Hello World!!!<br>\r\n");
136     out.write(" ");
137 }
138 out.write("\r\n");
139 out.write("</body>\r\n");
140 out.write("</html>\r\n");
```

### 1.3.2.2 脚本声明

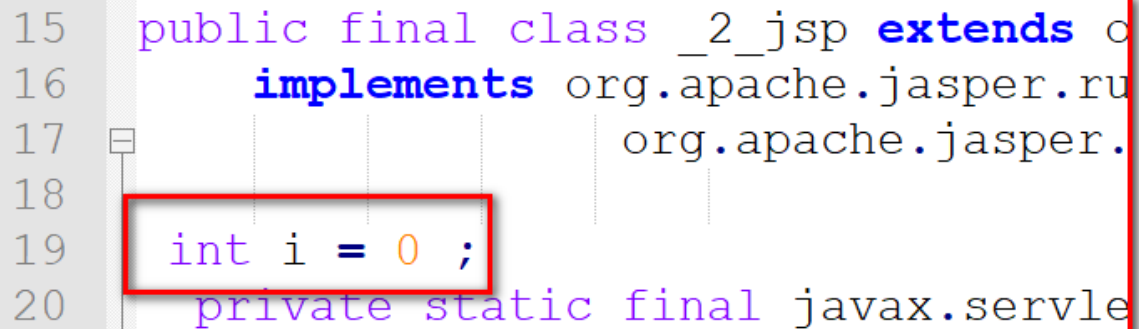
#### ① 脚本声明格式

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：



```
15 public final class _2_jsp extends org.apache.jasper.runtime
16     implements org.apache.jasper.runtime.JspRuntimeLibrary
17 {
18     private int i = 0 ;
19     private static final javax.servlet.ServletException
20     serialVersionUID = "org.apache.jasper.runtime._2_jsp.1";
```

总结：脚本声明书写的java代码会翻译在类的成员位置上。

#### ② 脚本声明作用

在类的成员位置上声明方法和变量

### 1.3.2.3 脚本表达式

#### ① 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>
```

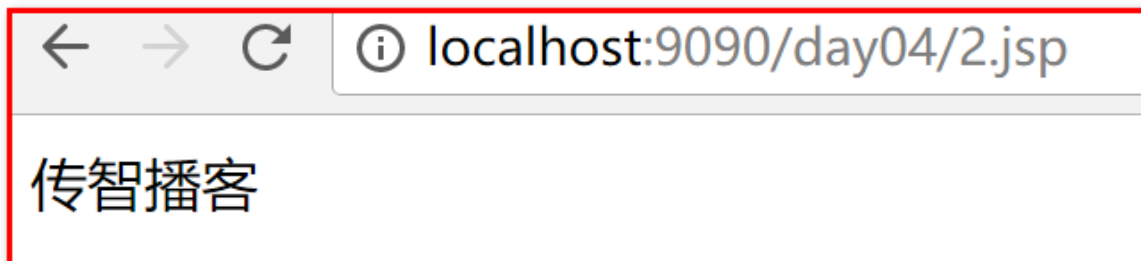
翻译成java文件：在\_jspService方法中，找到了我们书写的内容

```
122 out.write("\r\n");
123 out.write(" ");
124 out.print( "传智播客" );
125 out.write("\r\n");
126 out.write("</body>\r\n");
127 out.write("</html>\r\n");
```

上图的中的out对象是什么？

答：out对象的类型是JspWriter，通过查阅JAVAE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。

## ② 脚本表达式作用

代替response向页面输出内容

## 2.1 jsp指令介绍

### 2.1.1 什么是指令及其作用？

当我们创建一个jsp页面时，在页面第一行有一段代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
```



这个就是指令，对于指令它有以下作用：

- 用于指示JSP执行某些步骤
- 用于指示JSP表现特定行为

### 2.1.2 指令格式

指令的格式 `<%@指令名 attr1="" attr2=""%>`

其中attr1和attr2表示指令中的属性，通常我们将JSP指令写在JSP文件的的最上方，但是这并不是必须的。

### 2.1.3 Jsp中三种指令介绍

JSP中有三大指令，分别是以下三个：

- page
- include
- taglib

其中这三个指令中，page是最复杂的，后面我们会详细讲解；include指令表示静态包含，我们之前学习的请求包含属于动态包含；taglib指令是用来导入标签库，后面会学习一些标签库，在这里只做简单介绍。

## page指令

### ① 重点属性

Ø import

作用是在jsp页面上导包操作。

这个属性可以有多个。

Ø pageEncoding

该属性的作用是：指定当前jsp页面的编码。这个编码是给服务器看的，服务器需要知道当前页面的编码，否则服务器无法正确的把jsp翻译成java文件。

Ø contentType

在jsp文件中有如下指令：

```
<%@page language="java" import="java.util.*" contentType="text/html; charset=ISO-8859-1"%>
```

其中contentType属性的作用等同于response.setContentType("text/html; charset=ISO-8859-1")的作用。response的setContentType()方法这里我们就不再多说，当我们在jsp文件中提供了contentType属性，那么jsp对应的java文件中就会存在response.setContentType("text/html; charset=ISO-8859-1")这句代码。contentType属性的默认值就是"text/html; charset=ISO-8859-1"，但是我们知道iso-8859-1并不支持中文，所以我们需要将charset的值修改为"utf-8"。

pageEncoding与contentType的区别与联系

pageEncoding与contentType都是page指令的属性，它们都是用来设置编码，有如下联系：

如果这两个属性只提供了其中一个，那么没有提供的那个属性的编码值就是提供的这个属性的编码值，例如：在jsp页面中设置了contentType="text/html; charset=utf-8"，那么没有设置的pageEncoding的值就为utf-8，反之亦然；如果两个属性都没有提供，那么两者的默认编码就是ISO-8859-1。

根据对这两个属性的讲解，它们有如下区别：

pageEncoding是设置当前页面的编码，该编码是给服务器看的，可以让服务器正确的将jsp文件翻译成Java文件；

contentType有两个作用：一是设置响应字符流的编码，二是设置Content-Type响应头，即通知浏览器使用什么编码方式解码响应信息。

## ② 了解属性

Ø language

代表在jsp脚本中可以写的语言 只有一个值 java

Ø extends

它用于设置jsp翻译后的java类的父类. 要求必须是HttpServlet或其子类.

Ø Session

面上是否禁用session。可取值为true/false 如果值为false,在页面上不能使用session。

Ø isElIgnored

用是否忽略el表达式.可取值为true/false 如果值为true,那么页面上的el表达式就不会被解析.

Ø errorPage

设置错误页面，当jsp中如果出现了异常，会自动跳转到指定的错误页面

Ø isErrorPage

指示当前页面是一个错误页面，这时就可以使用一个内置对象 exception，通过这个内置对象就可以获取异常信息。

## include指令

Ø include指令作用

include指令的作用是在JSP页面中静态包含一个文件，同时由JSP解析包含的文件内容

Ø include指令格式

```
<%@ include file="filename" %>
```

Ø include指令功能分析

包含的是目标文件的源码；包含过来，一起翻译

main.jsp

```
<%
```

```
•    String s = "abc";
```

```
%>
```

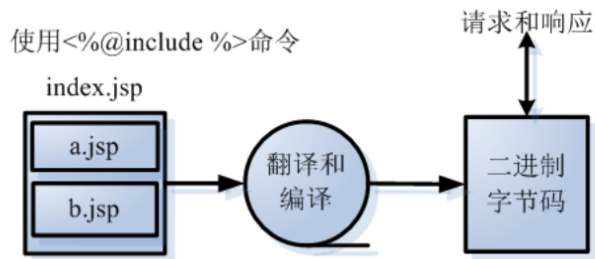
```
<%include file="part.jsp" %>
```

part.jsp

```
<%=s %> 没有定义变量s
```

尽管part.jsp本身会有错误

但是运行main.jsp就可以正确引入part.jsp



## taglib指令

Ø taglib指令作用

作用:是用于导入标签库.

Ø taglib指令格式

```
<%@taglib prefix="" uri="" %>
```

uri 标签文件的URI地址

prefix 标签组的命名空间前缀

## 3.1 jsp内置对象介绍

### 3.1.1 jsp九个内置对象及其类型

内置对象	代表内容	范围
<b>request</b>	触发服务调用的请求	<b>request</b>
<b>response</b>	对请求的应答	<b>page</b>
<b>session</b>	为请求的客户创建的session对象	<b>session</b>
<b>application</b>	从 <b>servlet</b> 配置对象获得的 <b>servlet</b> 上下文（如在 <b>getServletConfig()</b> ， <b>getContext()</b> 的调用中）	<b>application</b>
<b>out</b>	向输出流写入内容的对象	<b>page</b>
<b>pageContext</b>	本 JSP 的页面上下文	<b>page</b>
<b>page</b>	实现处理本页当前请求的类的实例	<b>page</b>
<b>config</b>	本 JSP 的 <b>ServletConfig</b>	<b>page</b>
<b>exception</b>	表示JSP页面运行时产生的异常	<b>Page</b>

变量名	真实类型	作用
pageContext	PageContext	当前页面中共享数据
request	HttpServletRequest	一次请求中共享数据
session	HttpSession	一次会话中共享数据
application	ServletContext	整个web应用共享数据
response	HttpServletResponse	响应对象
page	Object	当前页面(servlet)对象
out	JspWriter	输出对象
config	ServletConfig	servlet配置对象
exception	Throwable	异常对象

### jsp九个内置对象创建过程

我们可以创建一个demo1.jsp页面，启动服务器，在浏览器中访问 demo1.jsp页面，在tomcat下的work目录中查找到它翻译后的demo1\_jsp.java文件，在java文件中我们可以查找到每一个内置对象的创建。

```
request, response这两个对象是_jspService的参数

final javax.servlet.jsp.PageContext pageContext;

javax.servlet.http.HttpSession session = null;

final javax.servlet.ServletContext application;

final javax.servlet.ServletConfig config;

javax.servlet.jsp.JspWriter out = null;

final java.lang.Object page = this;

pageContext

    pageContext = _jspxFactory.getPageContext(this, request, response, null, true,
8192, true);

exception

    java.lang.Throwable exception =
org.apache.jasper.runtime.JspRuntimeLibrary.getThrowable(request);

    if (exception != null) {

        response.setStatus(javax.servlet.http.HttpServletResponse.SC_INTERNAL_SERVER_E
RROR);

    }
}
```

**注意:**exception对象只有在jsp页面上设置了page指令的isErrorPage才会有exception对象。

### 3.1.2 jsp中四个域对象

jsp有四个域对象，分别是：

application：域范围是整个应用；

session：域范围是整个会话，注意一个会话中只有一个用户，如果要统计站点的访问次数，使用的域是ServletContext，因为这个时候统计的是多个用户，而非一个用户；

request：域范围是整个请求链；

pageContext：域范围是一个jsp页面，在一个页面中存数据再取数据没有任何意义，所以这里所说的域指的是在当前jsp页面和当前jsp页面中使用的标签之间共享数据。

### 3.1.3 pageContext对象

#### pageContext对象作用

##### Ø 获取其它内置对象

可以通过pageContext对象获取jsp中其他的内置对象. PageContext类中定义了如下八个方法：  
getPage()、getRequest()、getResponse()、getServletConfig()、getServletContext()、  
getException()、getSession()等方法，其中getOut()是在其父类中定义的，有了这些方法，  
pageContext对象自然能够获取其他八个内置对象了。

##### Ø 操作四大作用域

首先了解pageContext对象的域功能：

```
void setAttribute(String name,Object value)
```

```
Object getAttribute(String name);
```

```
Void removeAttribute(String name);
```

可以利用pageContext对象向request、session、application域中存取数据。

```
Object getAttribute(String name, int scope):
```

该方法的第一个参数是域属性的名称，第二个参数指定是从那个域中取出域属性；在PageContext中已经定义了如下几个静态常量作为scope的可选值：

int APPLICATION\_SCOPE：当参数为这个常量时，表示从application域中获取数据；

int PAGE\_SCOPE：当参数是这个常量时，表示从page域中获取数据；

int REQUEST\_SCOPE：当参数是这个常量时，表示从request域中获取数据；

int SESSION\_SCOPE：当参数是这个常量时，表示从session域中获取数据。

```
void setAttribute(String name,Object value,int scope):
```

该方法的第一个参数指的是域属性的名称，第二参数指的是域属性的值，第三个参数指定域属性保存的域，scope的取值同上；

```
void removeAttribute(String name,int scope):
```

该方法的第一个参数指的是域属性的名称，第二个参数指的是将指定域属性从哪个域中移出，scope的取值同上；

有了以上三个方法，pageContext对象就可以操作其他三个域对象了，例如：

```
pageContext.setAttribute("name","zhangsan",PageContext.SESSION_SCOPE).
```

##### Ø 便捷查询域中数据

PageContext类的父类中定义了如下方法：

Object findAttribute(String name): 该方法是从jsp的四个域page、request、session、application依次查找指定名称的域属性，如果找到就停止，这说明如果这四个域中存在同名的属性，那么返回的数据是从page范围中获取的。

即表示谁的域范围越小，谁的优先级越高。

注意：这里说的page范围和九大内置对象中的page对象不是一个意思，page范围可以理解为是pageContext对象的域范围，即一个jsp页面。

### 3.1.4 jsp动作标签

#### 什么是标签及其作用

JSP标签也称之为Jsp Action(JSP动作)元素，它用于在Jsp页面中提供业务逻辑功能，避免在JSP页面中直接编写java代码，造成jsp页面难以维护。

#### Jsp中常用动作标签

Ø jsp:forward

[jsp:forward](#)标签是用来做请求转发，它与RequestDispatcher接口的forward()方法一样，唯一不同的是它是在jsp页面中使用。

Ø jsp:param

[jsp:param](#)标签作为[jsp:forward](#)标签和[jsp:include](#)标签的子标签，用来给转发或包含的页面传递数据。

## 第2章 EL表达式

### 2.1 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代码在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。**

## 2.2 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**：主要是从域对象中查找指定的数据。

## 2.3 EL表达式的基本使用

### 2.3.1 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中获取数据，那么我们可以获取其他域对象(容器)中的数据吗？我们来测试一下：

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
```

```

<title>Title</title>
</head>
<body>
<%--
    在三个域对象中存入同名的数据（request、session、servletcontext）
    然后再使用EL表达式获取。
--%>
<%
    request.setAttribute("addr", "上海");
    request.getSession().setAttribute("addr", "北京");
    //session.setAttribute("addr", "北京");

    request.getSession().getServletContext().setAttribute("addr", "广州");
    //application.setAttribute("addr", "广州");

%>
    ${addr}
</body>
</html>

```

测试的结果出来后，我们会发现，获取出来的是一——上海。设置三个容器的数据都是同名数据，这造成了我们获取数据的时候，单单使用名称已经无法区分所以数据，那么怎么解决这个问题呢？如何才能从指定的容器中获取数据呢？

我们需要在获取数据的时候指定相关的容器：

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
<%--
    在三个容器中存入数据（request、session、servletcontext）
    然后再使用EL表达式获取。
--%>
<%
    request.setAttribute("addr", "上海");
    request.getSession().setAttribute("addr", "北京");
    //session.setAttribute("addr", "北京");

    request.getSession().getServletContext().setAttribute("addr", "广州");
    //application.setAttribute("addr", "广州");

%>
    指定从request容器获取数据：${requestScope.addr}<br>
    指定从session容器获取数据：${sessionScope.addr}<br>
    指定从servletcontext容器获取数据：${applicationScope.addr}<br>
</body>
</html>

```

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是上海。



## 2.3.2 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。

### 2.3.2.1 获取数组

servlet代码：

```
package cn.itcast.web;

import cn.itcast.domain.Person;

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;

//urlPatterns = "/el1"
public class ELServlet1 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }

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

        //准备需要解析的数据
        //数组
        int[] arr = {11,22,33,44,55};
        request.setAttribute("arr",arr);

        request.getRequestDispatcher("/4.jsp").forward(request,response);
    }
}
```

jsp代码：

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

<!--获取数组数据-->
${arr[4]}<br>

</body>
</html>
```

### 2.3.2.2 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

### 2.3.2.3 获取集合 (list map)

servlet准备数据：

```
package cn.itcast.web;

import cn.itcast.domain.Person;

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.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

//urlPatterns = "/e12"
public class ELServlet2 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doGet(request, response);
    }

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

        //准备需要解析的数据

        //list集合
        List list = new ArrayList();
        list.add("北京传智播客");
        list.add("上海传智播客");
        list.add("广州传智播客");
        request.setAttribute("list",list);
        //map集合
        Map map = new HashMap();
        map.put("language1", "java");
        map.put("language2", "go");
        map.put("language3", "c#");
        map.put("aa.bb.cc", "python");
        request.setAttribute("map",map);

        request.getRequestDispatcher("/5.jsp").forward(request,response);
    }
}
```

jsp代码：

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

```

<html>
<head>
    <title>Title</title>
</head>
<body>

<%--获取list集合数据--%>
获取list集合: ${list}<br>
获取list集合某一个数据: ${list[0]}<br>
获取map集合: ${map}<br>
获取map集合某一数据: ${map.language1}<br>
获取map集合特殊key数据: ${map["aa.bb.cc"]}<br>

</body>
</html>

```

### 2.3.2.4 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

### 2.3.2.5 获取JavaBean数据

```

package cn.itcast.web;

import cn.itcast.domain.Person;

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.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@WebServlet(name = "ELServlet", urlPatterns = {"/el3"})
public class ELServlet3 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }

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

        //准备需要解析的数据

        //JavaBean
        Person p = new Person();
        p.setName("李四");
        p.setAge(18);
    }
}

```

```

        request.setAttribute("p",p);
        request.getRequestDispatcher("/6.jsp").forward(request,response);
    }
}

```

jsp:

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>

获取JavaBean数据: ${p}<br>
获取JavaBean指定属性的数据: ${p.name} ${p.age}<br>
使用中括号, 获取JavaBean指定属性的数据: ${p["name"]} ${p["age"]}<br>

</body>
</html>

```

### 2.3.2.6 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。

## 2.3.3 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

### 2.3.3.1 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, 默认, 将所有数据转换成Long类型
    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>
```

### 2.3.3.2 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, 默认, 将所有数据转换成Long类型
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。

### 2.3.3.3 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

逻辑运算符	说 明	范 例	结 果
&& 或 and	交集	<code>\${ A &amp;&amp; B }</code> 或 <code>\${ A and B }</code>	true / false
或 or	并集	<code>\${ A    B }</code> 或 <code>\${ A or B }</code>	true / false
! 或 not	非	<code>\${ !A }</code> 或 <code>\${ not A }</code>	true / false

```
<%@ 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>
```

### 2.3.3.4 逻辑运算符注意事项

注意: 逻辑运算中的异或"^"EL不支持。

### 2.3.3.5 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

关系运算符	说 明	范 例	结 果
= 或 eq	等于	<code>\${ 5 == 5 }</code> 或 <code>\${ 5 eq 5 }</code>	true
!= 或 ne	不等于	<code>\${ 5 != 5 }</code> 或 <code>\${ 5 ne 5 }</code>	false
< 或 lt	小于	<code>\${ 3 &lt; 5 }</code> 或 <code>\${ 3 lt 5 }</code>	true
> 或 gt	大于	<code>\${ 3 &gt; 5 }</code> 或 <code>\${ 3 gt 5 }</code>	false
<= 或 le	小于等于	<code>\${ 3 &lt;= 5 }</code> 或 <code>\${ 3 le 5 }</code>	true
>= 或 ge	大于等于	<code>\${ 3 &gt;= 5 }</code> 或 <code>\${ 3 ge 5 }</code>	false

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>
```

### 2.3.3.6 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。

### 2.3.3.7 empty运算符和三元运算符

empty运算符是用来判断当前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：

```
<%--再当前jsp导入ArrayList--%>
<%@ page import="java.util.ArrayList" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
<%
    request.setAttribute("str",null);
    request.setAttribute("list",new ArrayList<>());
%>
${empty str}<br>
${empty list}<br>
${str == null? "数据为空":str}<br>
```

```
</body>
</html>
```

### 2.3.3.8 empty运算符注意事项

注意：以上的empty运算符案例中，empty运算符可以和逻辑运算符组合使用。

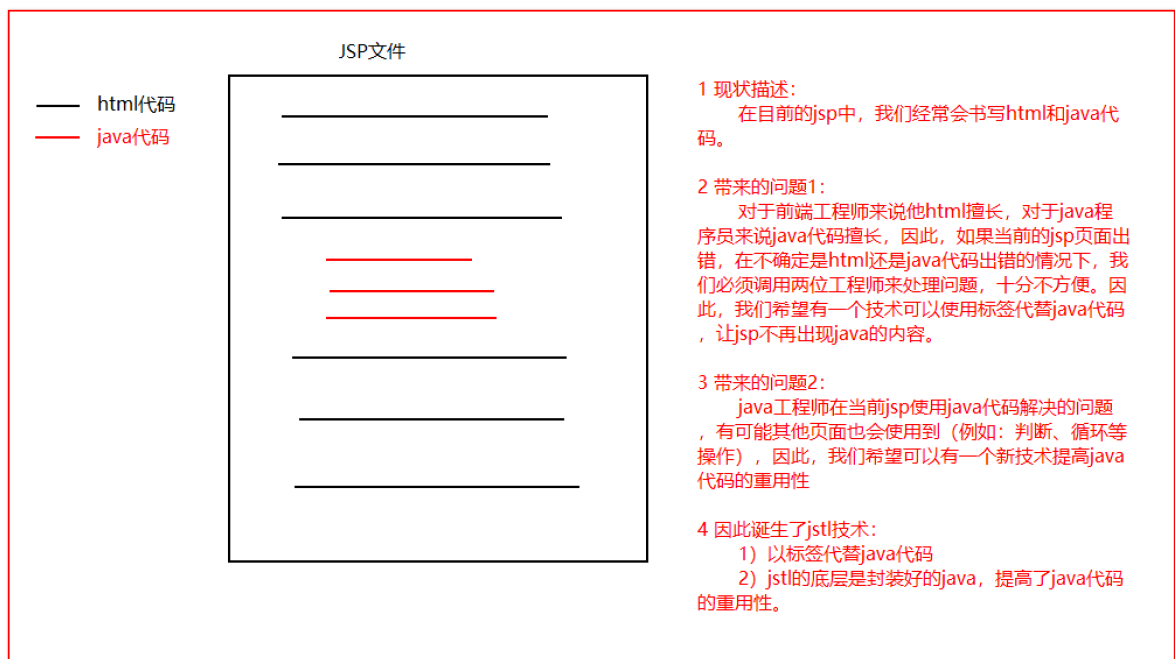
例如：\${not empty str}表示str不为空，返回true。

## 第3章 JSTL的核心标签库使用

### 3.1 jstl标签的基本概述

#### 3.1.1 jstl的由来

接下来我们要学习一个新的知识点，那么首先，我们要知道他的由来：

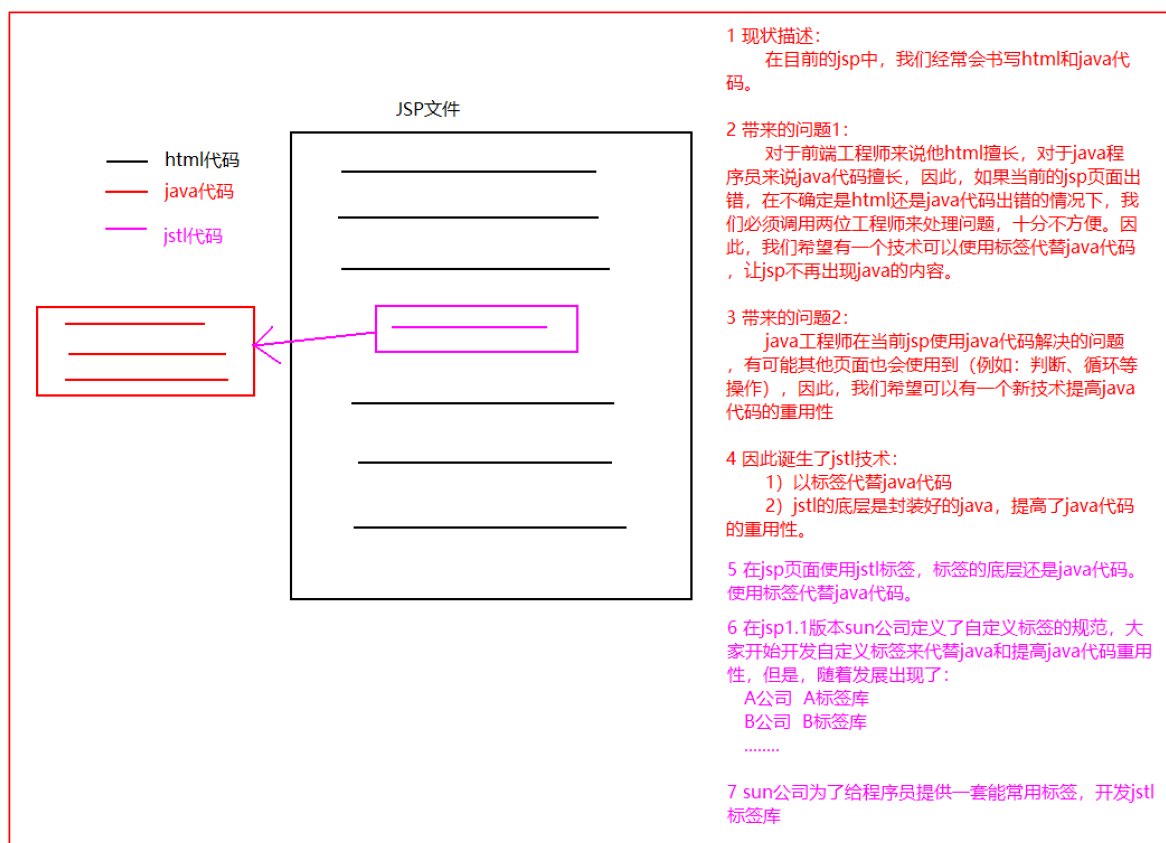


#### 3.1.2 jstl的概述

从JSP 1.1规范开始JSP就支持使用自定义标签，使用自定义标签大大降低了JSP页面的复杂度，同时增强了代码的重用性，因此自定义标签在WEB应用中被广泛使用。许多WEB应用厂商都开发出了自己的一套标签库提供给用户使用，这导致出现了许多功能相同的标签，令网页制作者无所适从，不知道选择哪一家的好。

为了解决这个问题，Apache Jakarta小组归纳汇总了那些网页设计人员经常遇到的问题，开发了一套用于解决这些常用问题的自定义标签库，这套标签库被SUN公司定义为标准标签库（The JavaServer Pages Standard Tag Library），简称JSTL。

使用JSTL可以解决用户选用不同WEB厂商的自定义标签时的困惑，JSP规范同时也允许WEB容器厂商按JSTL标签库的标准提供自己的实现，以获取最佳性能。



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, 目前已经不再 (不允许) 使用 (jsp开发模式二, 这个开发模式不允许, 在页面操作数据库)
5. xml: jstl操作xml文件的。目前已经不再使用 (页面传递数据, 页面解析数据, 使用json格式字符串代替xml)

因此我们需要知道的只有jstl的核心标签库。

### 3.1.2 jstl核心标签库列表



标签名称	作用
<c:if>	用户java代码if(){}语句功能 重点掌握
<c:forEach>	用户代替java代码for循环语句 重点掌握
<c:choose>	用于指定多个条件选择的组合边界，它必须与 <a href="#">c:when</a> 和 <a href="#">c:otherwise</a> 标签一起使用
<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: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

注意：jsp本质上也是servlet,以前也会在jsp中处理业务逻辑,现在jsp的主要作用就是数据的展示,所以我们需要掌握在页面上遍历和判断的操作。

## 3.2 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。

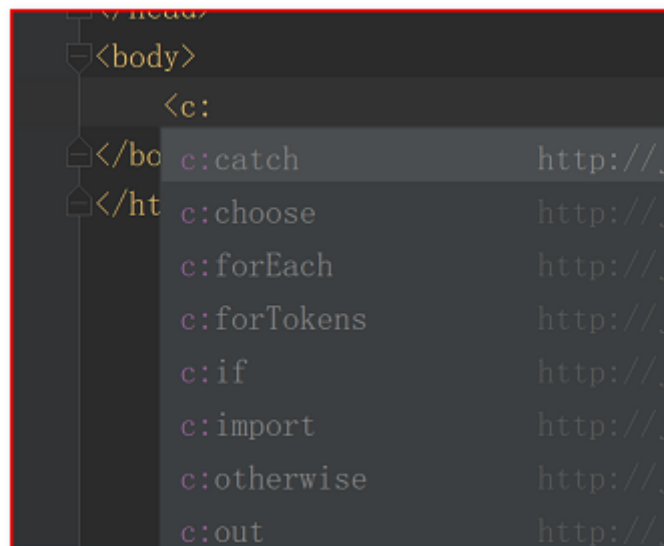
### 3.2.1 导入jar包

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

### 3.2.2 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：



## 3.3 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、foreach标签，接下来我们学习下。

### 3.3.1 if标签

#### 3.3.1.1 if标签作用

起到java代码的判断的作用

#### 3.3.1.2 if标签属性介绍

表 8.5 <c:if>标签的属性			
属性名	是否支持 EL	属性类型	属性描述
test	true	boolean	决定是否处理标签体中的内容的条件表达式
var	false	String	用于指定将 test 属性的执行结果保存到某个 Web 域中的某个属性的名称
scope	false	String	指定将 test 属性的执行结果保存到哪个 Web 域中

**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。

**注:** 是否支持EL表达式指的是 是否可以在属性中书写EL表达式。

#### 3.3.1.3 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

#### 3.3.1.4 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>
    <!-- 获取test属性的值, 将其以flag为属性名, test中计算的值为属性, 放入session中 -->
    <c:if test="${ 3>4 }" scope="session" var="flag">
        测试test标签
    </c:if>
    <br/>
    在页面上判断: 3是否大于4, 若页面上能显示"测试test标签", 证明test的返回值为true; test中表达
    式的运算结果为: ${session.flag}
</body>
</html>

```

## 3.3.2 foreach标签

### 3.3.2.1 foreach标签作用

起到java代码的for循环作用

### 3.3.2.2 foreach标签属性介绍

表 8.6 <c:forEach>标签的属性

属性名	是否支持 EL	属性类型	属性描述
var	false	String	指定将当前迭代到的元素保存到 page 这个 Web 域中的属性名称
items	true	任何支持的类型	将要迭代的集合对象
varStatus	false	String	指定将代表当前迭代状态信息的对象保存到 page 这个 Web 域中的属性名称
begin	true	int	如果指定 items 属性, 就从集合中的第 begin 个元素开始进行迭代, begin 的索引值从 0 开始编号; 如果没有指定 items 属性, 就从 begin 指定的值开始迭代, 直到 end 值时结束迭代
end	true	int	参看 begin 属性的描述
step	true	int	指定迭代的步长, 即迭代因子的迭代增量

**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

### 3.3.2.3 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：

#### ① foreach不循环对象

begin、end、step 三属性的演示：设置循环开始、结束和步长。

```
<%@ 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 --%>
<%
    for (int i = 1; i <= 5; i++) {

    }
%>
<!--
begin="1" int i = 1;
end="5"    i<=5;
step="6" 步长，控制循环，间隔几次循环，执行一次循环体中的内容

step="1" 1 2 3 4 5  M M M M M
step="2" 12 34 5    M M M
step="3" 123 45     M M
step="4" 1234 5     M M
step="5" 12345      M

--%>
<c:forEach begin="1" end="5" step="1" >
    M
</c:forEach>
</body>
</html>
```

var属性演示：在不循环对象的时候，相当于将for循环中的循环变量i，每次都记录下来

```
<%@ 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 --%>
<%
    for (int i = 1; i <= 5; i++) {

    }
%>
```

```

%>
<%--
var 属性在不循环对象的时候，相当于将for循环中的循环变量i，每次都记录下来。
--%>
<c:forEach begin="11" end="15" step="1" var="info">
    ${info}
</c:forEach>
</body>
</html>

```

varStatus属性演示：保存了当前循环过程中的信息

```

<%@ 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

## ② 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>

```

## 第4章 MVC模式和三层架构

### 4.1 MVC设计模式

MVC设计模式：Model-View-Controller简写。

MVC是软件工程中的一种软件架构模式，它是一种**分离业务逻辑与显示界面**的设计方法。它把软件系统分为三个基本部分：模型（Model）、视图（View）和控制器（Controller）。

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

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。

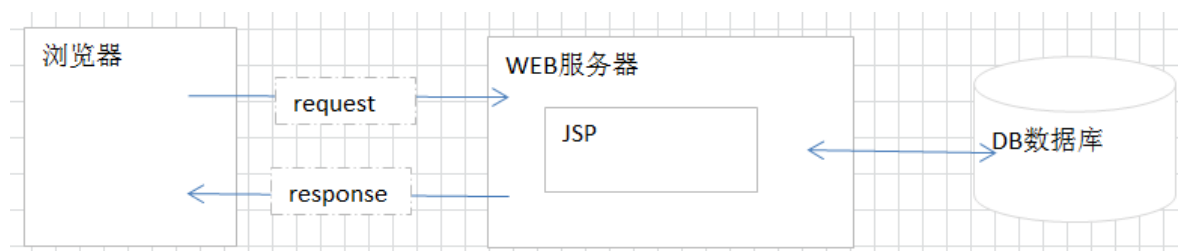
注意，MVC不是Java的特有的，几乎现在所有B/S结构的软件都采用了MVC设计模式。

## 4.2 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，JSP Model2 第二代，JSP Model 3 三个时期。

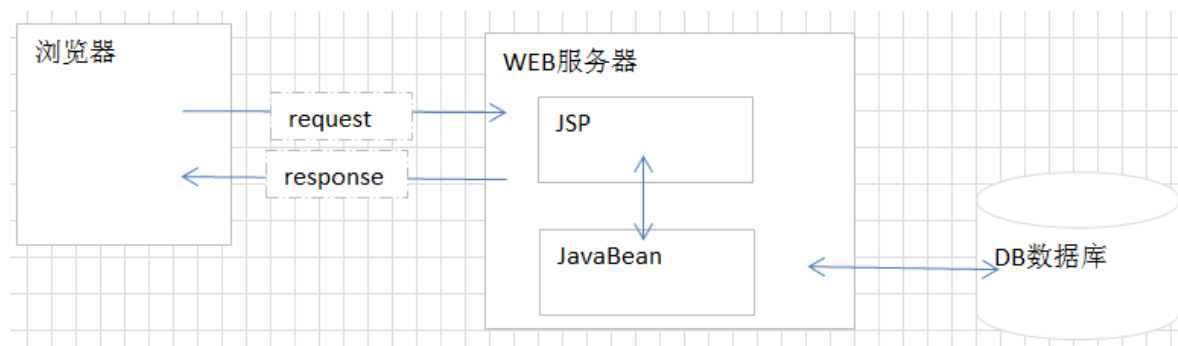
### 4.2.1 JSP Model1 第一代

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。



### 4.2.2 JSP Model1 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。

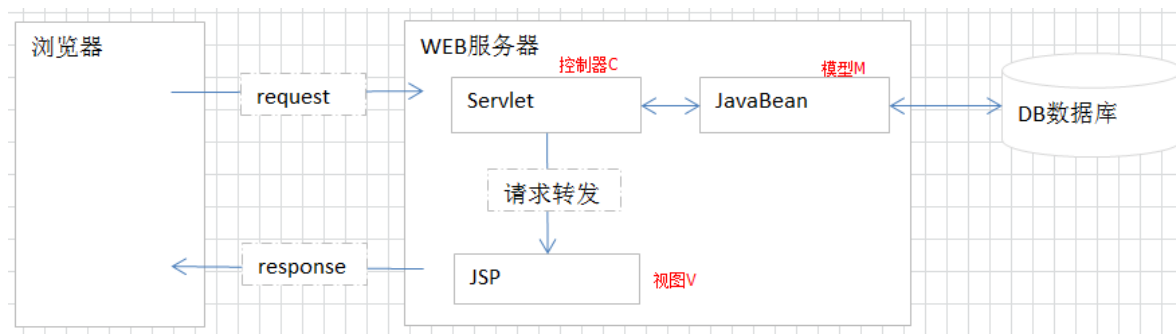


### 4.2.3 JSP Model 2

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。

- JSP：视图层，用来与用户打交道。负责接收传来的数据，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；

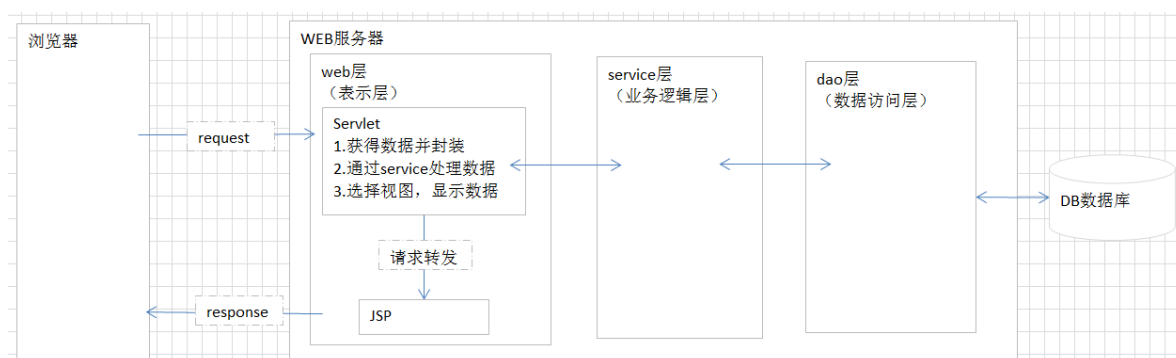
- JavaBean：模型层，完成具体的业务工作，例如：转账等。



## 4.3 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。

- 表示层：又称为 web层，与浏览器进行数据交互的。
- 业务逻辑层：又称为service层，专门用于处理业务数据的。
- 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。



在我们开发的时候一般是通过包结果来体现三层架构的,三层架构包命名一般如下：

- 简单版

cn.itcast 公司域名倒写

cn.itcast.dao dao层

cn.itcast.service service层

cn.itcast.web.servlet web层

cn.itcast.domain JavaBean

cn.itcast.util 工具

- 完整版

cn.itcast 公司域名倒写

cn.itcast.xxx 项目名称

cn.itcast.xxx.yyy 子模块

cn.itcast.xxx.yyy.dao 子模块dao层接口

cn.itcast.xxx.yyy.dao.impl 子模块dao层实现类

cn.itcast.xxx.yyy.service 子模块service层接口



cn.itcast.xxx.yyy.service.impl 子模块service层实现类

cn.itcast.xxx.yyy.domain 子模块JavaBean (子模块yyy可省略)

cn.itcast.xxx.yyy.web.servlet 子模块web层, servlet

cn.itcast.xxx.yyy.web.filter 子模块web层, filter

cn.itcast.xxx.util 工具类

cn.itcast.xxx.exception 自定义异常

cn.itcast.xxx.constant 常量

## 第5章 使用三层架构和MVC模式完成用户显示列表案例

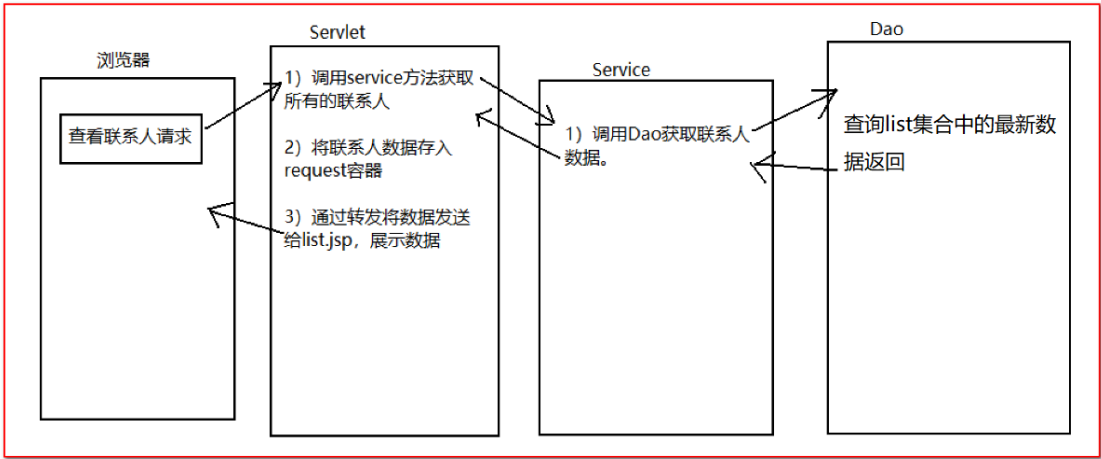
### 5.1 案例需求

使用三层架构和MVC模式开发代码，完成用户显示列表功能。

### 5.2 案例效果

显示所有联系人							
编号	姓名	性别	年龄	籍贯	QQ	邮箱	操作
1	张三	男	11	广州	766335435	766335435@qq.com	<div>修改</div> <div>删除</div>
2	李四	男	12	上海	243424242	243424242@qq.com	<div>修改</div> <div>删除</div>
3	王五	女	13	广州	474574574	474574574@qq.com	<div>修改</div> <div>删除</div>
4	赵六	男	14	北京	987069697	987069697@qq.com	<div>修改</div> <div>删除</div>
5	钱七	女	15	广州	412132145	412132145@qq.com	<div>修改</div> <div>删除</div>
<div>添加联系人</div>							

### 5.3 案例分析



### 5.4 实现步骤

#### 5.4.1 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>

```

## 5.4.2 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

### 5.4.3 编写servlet代码

```
//urlPatterns = "/queryAll"  
public class QueryAllServlet extends HttpServlet {  
    protected void doPost(HttpServletRequest request, HttpServletResponse  
response) throws ServletException, IOException {  
        doGet(request,response);  
    }  
  
    protected void doGet(HttpServletRequest request, HttpServletResponse  
response) throws ServletException, IOException {  
        //获取联系人数据  
        ContactService contactService = new ContactService();  
        List<Contact> list = contactService.queryAll();  
        //将联系人数据转发到页面展示  
        request.setAttribute("list",list);  
        request.getRequestDispatcher("/list.jsp").forward(request,response);  
    }  
}
```

### 5.4.4 编写service代码

```
public class ContactService {  
  
    private ContactDao contactDao = new ContactDao();  
  
    public List<Contact> queryAll() {  
        return contactDao.queryAll();  
    }  
}
```

### 5.4.5 编写dao代码

```
public class DaoDemo {  
  
    private static List<Contact> list;  
  
    static {  
        list=new ArrayList();  
        Contact contact1 = new Contact("1", "张三", "男", 21, "北京", "14444222",  
"zs@qq.com");  
        Contact contact2 = new Contact("2", "李四", "男", 22, "北京", "24444222",  
"ls@qq.com");  
        Contact contact3 = new Contact("3", "王五", "男", 23, "北京", "34444222",  
"ww@qq.com");  
        list.add(contact1);  
        list.add(contact2);  
    }  
}
```

```
        list.add(contact3);  
    }  
  
    public List<Contact> queryAll() {  
        return list;  
    }  
}
```