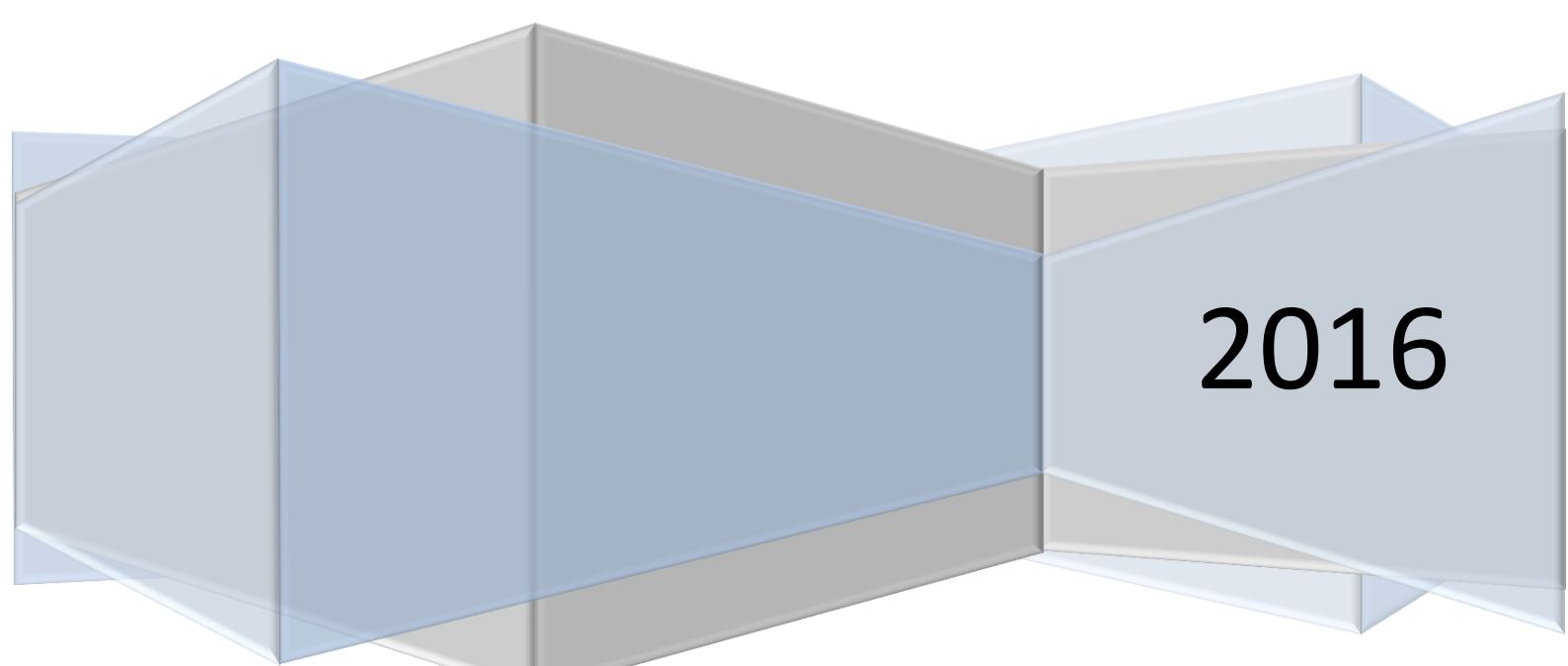


uniquedu

# JavaWeb

JSP+Servlet

何王科



2016

## 目录

1 XML.....	1
2 Java Web 应用服务器——Tomcat 的安装与配置 .....	4
2.0 安装 JDK .....	4
2.1 web 容器 .....	5
2.1.1 下载 Tomcat .....	5
2.1.2 安装 Tomcat .....	6
2.1.3 服务器的配置（手动） .....	8
2.2 eclipse 下载与安装 .....	11
2.2.1 汉化 eclipse.....	12
2.2.2 集成 eclipse 与 tomcat.....	16
2.2.3 完善 Web 项目开发所需配置 .....	20
2.3 第一个 Web 项目 .....	23
3 JSP 基础语法 .....	27
3.1 JSP 注释 .....	27
3.2 Scriptlet.....	28
3.2.1 第一种 Scriptlet: <%%> .....	28
3.2.2 第二种 Scriptlet: <%!%> .....	29
3.2.3 第三种 Scriptlet: <%==%> .....	31
3.3 scriptlet 标签 .....	31
3.4 page 指令.....	32
3.4.1 设置页面的 MIME .....	33
3.4.2 设置文件编码 .....	34
3.4.3 错误页设置.....	34
3.4.4 数据库连接操作 import .....	36
3.5 包含指令 .....	40
3.5.1 静态包含 .....	40
3.5.2 动态包含 .....	42

---

3.6 跳转指令 .....	45
3.7 作业 .....	47
4 JSP 内置对象 .....	55
4.1 JSP 内置对象概览.....	55
4.2 四种属性范围.....	56
4.2.1 page 属性范围 .....	56
4.2.2 request 属性范围.....	60
4.2.3 session 属性范围 .....	65
4.2.4 application 属性范围 .....	69
4.2.5 深入 page 属性范围 .....	72
4.3 request 对象 .....	74
4.3.1 乱码解决.....	75
4.3.2 接受请求参数.....	78
4.3.3 显示全部的头信息 .....	89
4.4 response 对象.....	92
4.4.1 设置头信息.....	92
4.4.2 页面跳转.....	94
4.4.3 操作 Cookie.....	96
4.5 session 对象 .....	103
4.5.1 取得 Session id .....	104
4.5.2 登录及注销 .....	107
4.5.3 判断新用户 .....	112
4.5.4 取得用户的操作时间 .....	112
4.6 application 对象 .....	113
4.6.1 取得虚拟目录对应的绝对路径.....	114
4.6.2 网站计数器 .....	115
4.7 config 对象.....	118

4.7.1 Web 安全性 .....	118
4.7.2 config 对象 .....	121
4.8 out 对象 .....	125
4.9 pageContext 对象 .....	126
5 JavaBean .....	130
5.1 JavaBean 简介 .....	130
5.2 在 JSP 中使用 JavaBean .....	132
5.2.1 使用 JSP 的 page 指令导入所需要的 JavaBean .....	132
5.2.2 使用<jsp:useBean>指令 .....	133
5.3 JavaBean 与表单 .....	134
5.4 设置属性<jsp:setProperty> .....	138
5.5 取得属性<jsp:getProperty> .....	143
5.6 JavaBean 的保存范围 .....	144
5.6.1 page 范围内的 JavaBean .....	146
5.6.2 request 范围的 JavaBean .....	148
5.6.3 session 范围内的 JavaBean .....	151
5.6.4 application 范围的 JavaBean .....	155
5.7 JavaBean 的删除 .....	158
5.8 作业：注册验证 .....	160
6 DAO 设计模式 .....	170
6.1 DAO 设计模式简介 .....	170
6.2 DAO 开发 .....	171
6.3 JSP 调用 DAO .....	187
7 文件上传 .....	194
7.1 上传单个文件 .....	194
7.2 混合表单 .....	196
7.3 为上传文件自动命名 .....	199
7.4 批量上传 .....	203
8 Servlet .....	207
8.1 Servlet 简介 .....	207
8.2 第一个 Servlet 程序（手动创建） .....	207
8.3 Servlet 与表单 .....	210

8.4Servlet 生命周期 .....	213
8.5 取得初始化配置信息 .....	217
8.6 取得其他内置对象 .....	220
8.6.1 取得 HttpSession 实例 .....	220
8.6.2 取得 ServletContext 实例 .....	223
8.7Servlet 跳转 .....	226
8.7.1 客户端跳转 .....	226
8.7.2 服务器端跳转 .....	230
8.8Web 开发模式 .....	233
8.8.1 Model .....	234
8.8.2Modell: Model-View-Controller .....	235
8.9 mvc 设计模式应用 .....	236
9 表达式语言 .....	252
9.1 表达式语言 .....	252
9.2 表达式语言的内置对象 .....	254
9.2.1 访问 4 种属性范围的内容 .....	255
9.2.2 调用内置对象操作 .....	257
9.2.3 接收请求参数 .....	258
9.3 集合操作 .....	262
9.4 在 MVC 中应用 EL .....	264
9.5 运算符 .....	274
预习: JSTL .....	278
10 JSP 标准标签库 .....	279
10.1JSTL 简介 .....	279
10.2 安装 JSTL1.2 .....	279
10.3 核心标签库 .....	281
10.3.1< c:out> .....	282
10.3.2< c:set> .....	284
10.3.3< c:remove>标签 .....	287
10.3.4< c:catch>标签 .....	289

10.3.5<c:if>标签 .....	290
10.3.6<c:choose>、<c:when>、<c:otherwise>标签 .....	292
10.3.7<c:forEach>标签 .....	294
10.3.8<c:forTokens>标签 .....	300
10.3.9<c:import>标签 .....	302
10.3.10<c:url>标签 .....	306
10.3.11<c:redirect>标签 .....	308
11 分页 .....	310
11.1JSP 中分页实现 .....	310
11.2 作业：抽象出分页控件 .....	317
12Session 购物车 .....	318
12.1product 表： .....	318
12.2image.jsp 验证码页面： .....	319
12.3split_page_plugin.jsp 分页页面 .....	324
12.4login.jsp 登录页面： .....	328
12.5check.jsp .....	330
12.6welcome.jsp 欢迎页面 .....	334
12.7logout.jsp 注销页面： .....	336
12.8product_list.jsp 产品列表页 .....	337
12.9add_car.jsp .....	344
12.10product_cars.jsp .....	345
12.11update_car.jsp .....	353
12.12 作业：将上述代码使用 MVC 设计模式实现 .....	354
13 项目实战——购物网站 .....	355
13.1 项目需求概述 .....	355
13.1.1 概述 .....	355
13.1.2 主要功能 .....	355
13.1.3 运行环境 .....	356
13.2 功能需求 .....	357
13.2.1 商品显示模块 .....	357

---

13.2.2 用户模块.....	359
13.2.3 购物车模块.....	362
13.2.4 订单模块.....	365
13.3 引言 .....	368
13.3.1 编写目的.....	368
13.3.2 术语或缩写 .....	368
13.4 总体设计 .....	368
13.4.1 系统说明.....	368
13.4.2 总体架构.....	369
13.5 包结构、类分析.....	370
13.5.1 业务模型层.....	370
13.5.2 控制层.....	371
13.5.3 视图层.....	371
13.6 实现思路 .....	372
13.7 数据库设计 .....	373
13.7.1 表结构.....	373
13.7.2 表关系.....	375

# 1 XML

可扩展的标记性语言

HTML（超文本标记语言）

```
<ul>  
  
<li>name:hunk</li>  
  
<li>phone:15335061230</li>  
  
</ul>
```

XML

```
<?xml version="1.0" encoding="utf-8"?>  
  
<addresslist>  
  
<linkman>  
  
<name>hunk</name>  
  
<phone>15335061230</phone>  
  
</linkman>  
  
</addresslist>
```

XML 由俩个部分组成：

第一个部分：前导区，规定了 XML 的一些属性，一般是有三个属性的：

version：表示的是 XML 的版本，现在是 1.0

encoding：页面中使用的文字编码，如果我们的页面中有中文，就一

定要指定编码。在这里我们就使用 utf-8

**standalone:** 此 XML 是否独立运行，如果需要进行显示可以使用 CSS 或者 XSL 控制。

第二个部分：数据区，所有的数据区必须有一个根元素，一个根元素下面可以存放多个子元素，但是要求每一个元素都必须完结，每一个标记都是区分大小写的。

对比一下 HTML 和 XML 的区别：

比较内容	HTML	XML
可扩展性	不具备	可以定义新的标记语言
侧重点	侧重点在于如何显示	侧重在于如何结构化的描述信息
语法要求	对于标记的嵌套、配对不严格，不要求标记之间的顺序	严格要求嵌套、配对，遵循统一的顺序结构要求
可读性及可维护性	难于阅读以及维护	结构清晰、便于维护和阅读
数据和显示的关系	内容描述和显示融合在一起的	内容描述与显示相分离
保值性	不具备	具有保值性

在网络应用程序开发中，体系结构一般分为 C/S 和 B/S 两种结构。

C/S 结构：客户端/服务器（Client/Server）

B/S 结构：浏览器/服务器（Brower/Server）

在 JavaWeb 开始之前，我们需要几个组件：

JDK

Java Web 应用服务器——Tomcat 的安装与配置

## 2 Java Web 应用服务器——Tomcat 的安装与配置

### 2.0 安装 JDK

下载地址：

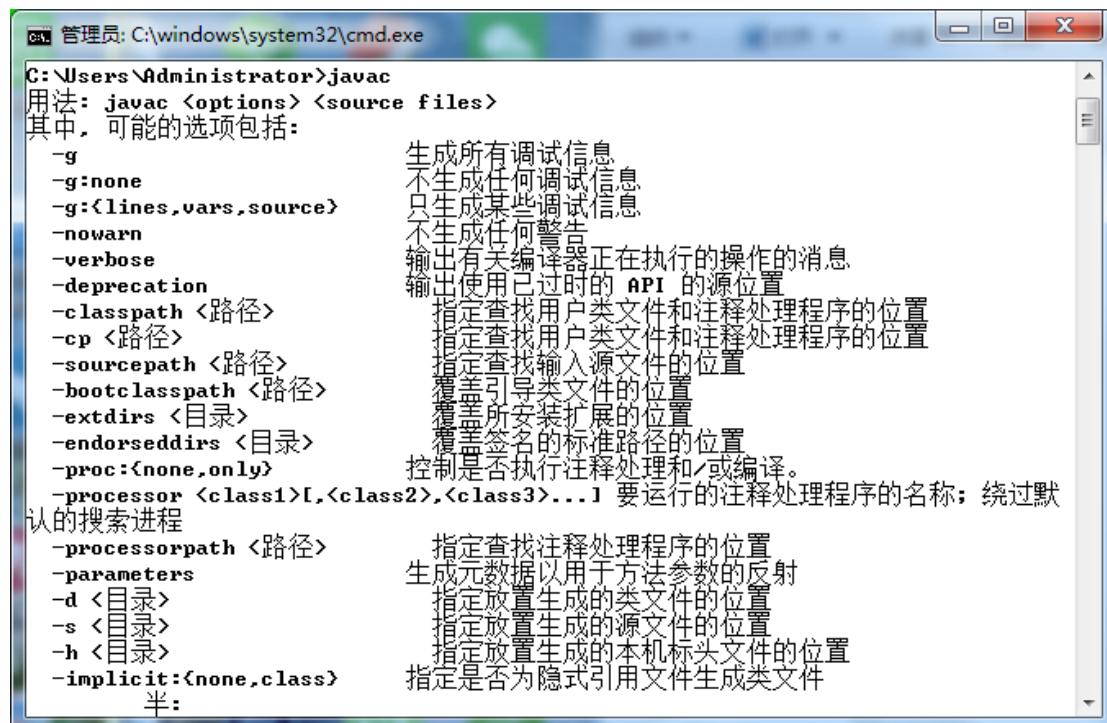
<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

安装详见 java 第一讲。

测试：

打开 cmd，输入 javac

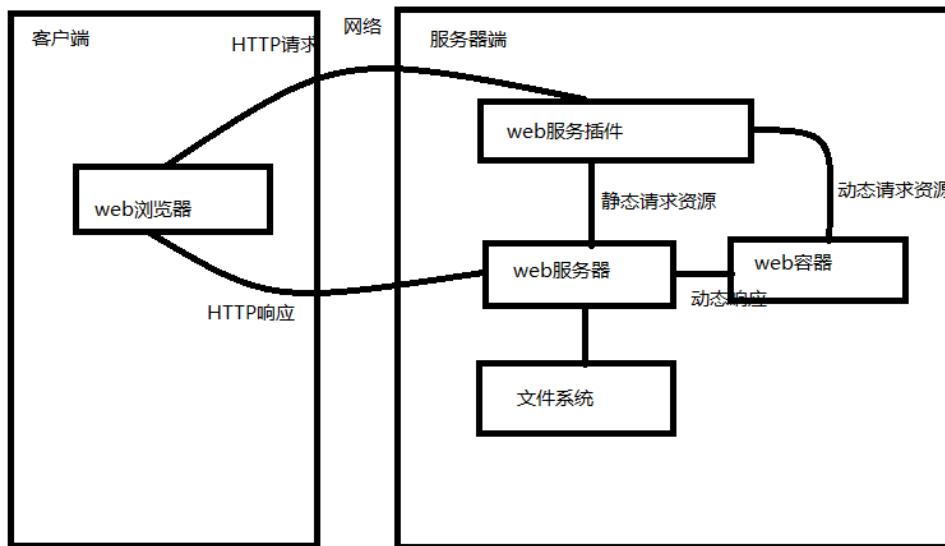
如果见到：



```
C:\Users\Administrator>javac
用法: javac <options> <source files>
其中, 可能的选项包括:
-g           生成所有调试信息
-g:none      不生成任何调试信息
-g:<lines,vars,source> 只生成某些调试信息
-nowarn      不生成任何警告
-verbose     输出有关编译器正在执行的操作的消息
-deprecation 输出使用已过时的 API 的源位置
-classpath <路径> 指定查找用户类文件和注释处理程序的位置
-cp <路径>   指定查找用户类文件和注释处理程序的位置
-sourcepath <路径> 指定查找输入源文件的位置
-bootclasspath <路径> 指定引导类文件的位置
-extdirs <目录> 指定所安装扩展的位置
-endorseddirs <目录> 指定签名的标准路径的位置
-proc:<none,only> 控制是否执行注释处理和/或编译。
-processor <class1>[,<class2>,<class3>...] 要运行的注释处理程序的名称; 绕过默认的搜索进程
-processorpath <路径> 指定查找注释处理程序的位置
-parameters    生成元数据以用于方法参数的反射
-d <目录>      指定放置生成的类文件的位置
-s <目录>      指定放置生成的源文件的位置
-h <目录>      指定放置生成的本机头文件的位置
-implicit:<none,class> 指定是否为隐式引用文件生成类文件
```

## 2.1 web 容器

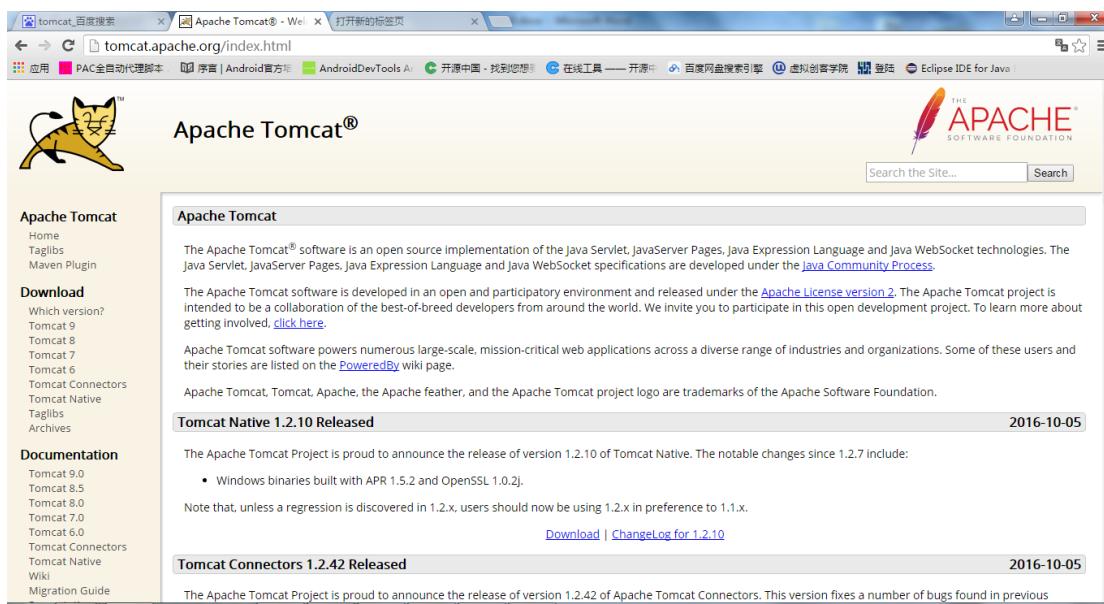
在进行 JavaWeb 开发时必须有 Web 服务器的支持，要运行一个 JavaWeb 程序必须有相应的 web 容器支持。



### 2.1.1 下载 Tomcat

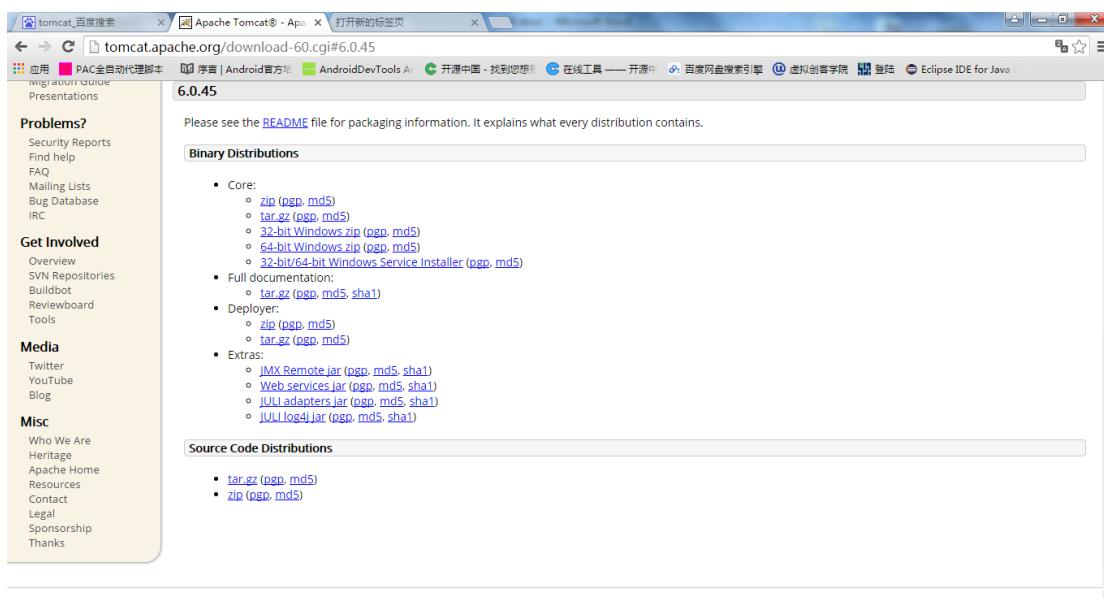
web 容器是我们的 Tomcat

下载地址: <http://tomcat.apache.org/>



The screenshot shows the Apache Tomcat official website. At the top, there's a navigation bar with links like "Home", "Taglibs", "Maven Plugin", "Download", "Documentation", and "Media". The main content area features a large image of a yellow cat sitting on a keyboard, followed by the text "Apache Tomcat®". Below this, there's a search bar and a link to "Search Site...". The page also includes sections for "Apache Tomcat", "Tomcat Native 1.2.10 Released" (date: 2016-10-05), and "Tomcat Connectors 1.2.42 Released" (date: 2016-10-05). The "Tomcat Connectors" section notes that it fixes bugs found in previous versions.

选择你要下载的 Tomcat 版本

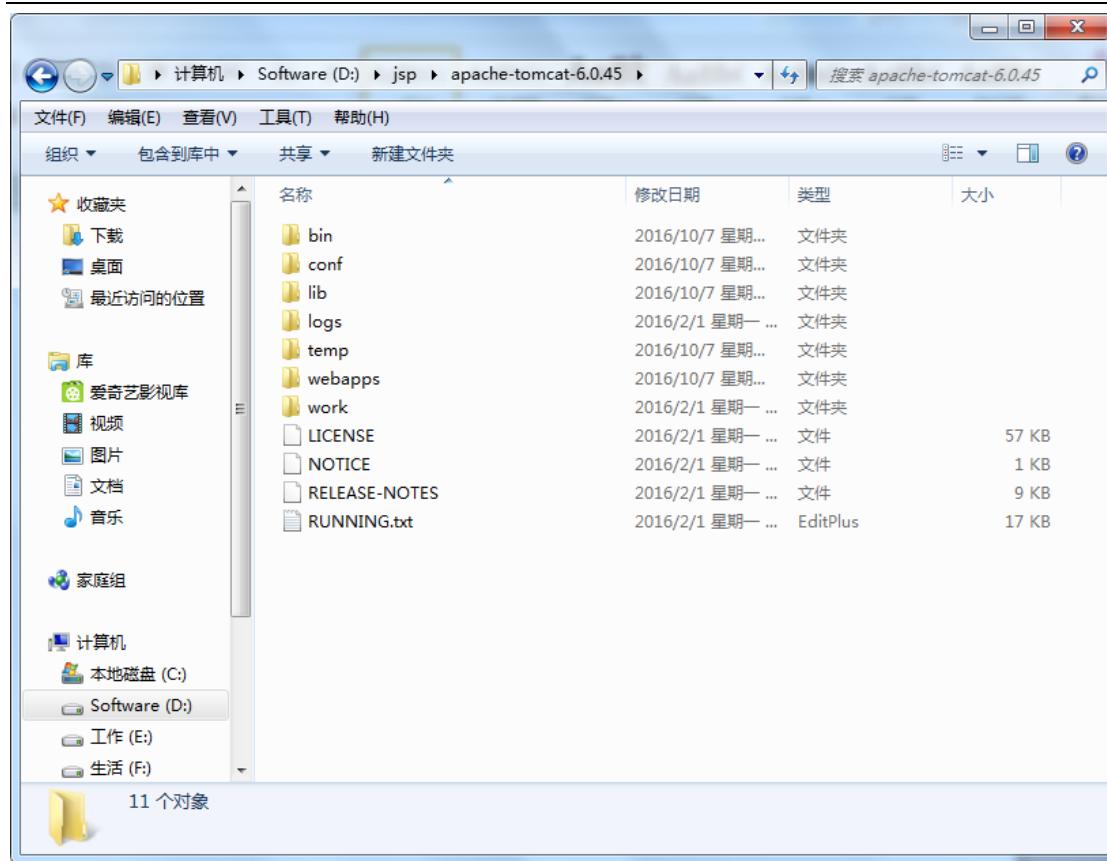


The screenshot shows the Apache Tomcat 6.0.45 download page. On the left, there's a sidebar with links for "Problems?", "Get Involved", "Media", and "Misc". The main content area has a heading "6.0.45" and a sub-section "Binary Distributions" which lists various file formats for download. Below this is a "Source Code Distributions" section with two options: "tar.gz (pgp, md5)" and "zip (pgp, md5)".

选择下载的文件就可以了。

## 2.1.2 安装 Tomcat

直接将下载的 zip 文件解压即可。



## Tomcat 主要目录的作用

**bin:** 所有的可执行命令，启动和关闭服务器的命令就在此文件夹中

**conf:** 服务器的配置文件夹，其中保存的各个配置信息

**lib:** Tomcat 服务器所需要的各个库文件

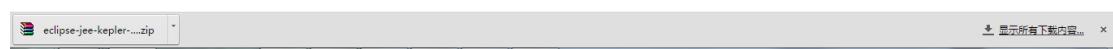
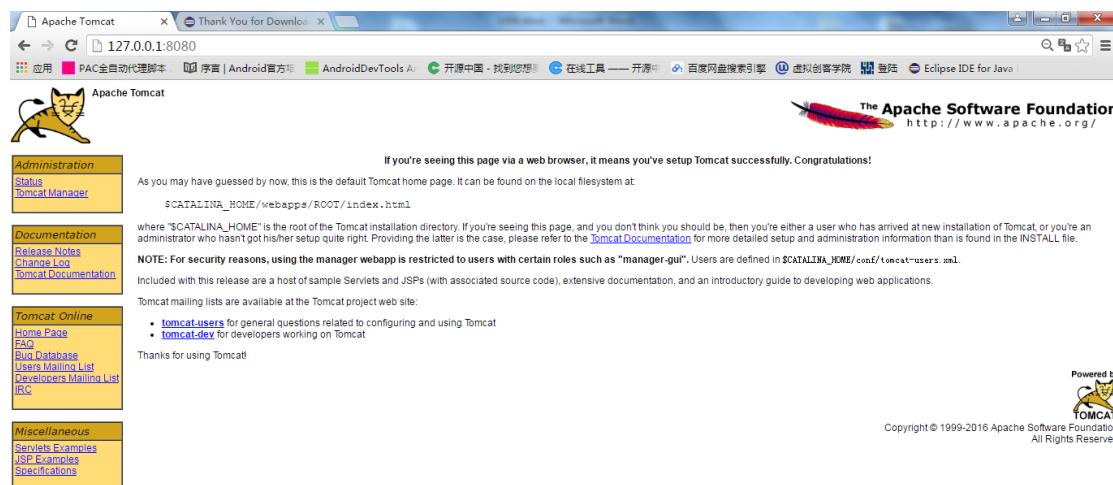
**logs:** 保存的服务器的系统日志

**webapps:** web 应用程序存放目录，文本项目保存在此目录中即可

**work:** 临时文件夹，生成所有的临时文件 (\*.java、\*.class)

在 bin 文件夹中，双击 startup.bat，打开之后不要关闭，这是在浏览器中输入 127.0.0.1:8080

看到：



### 2.1.3 服务器的配置（手动）

#### 1、修改端口号

Tomcat 安装成功之后，默认的端口号是 8080，如果想修改，则打开 conf/server.xml, 找到

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

将 port 修改为 80，此时在浏览器中只需要输入 127.0.0.1 即可访问。

默认采用 8080.

每次修改完服务器配置都需要重新启动服务器。

#### 2、配置虚拟目录

在 Tomcat 的服务器配置中，最重要的就是配置虚拟目录的操作，每一个虚拟目录都保存了一个完整的 Web 项目。

首先，在硬盘上建立一个自己的文件夹，例如在 d 盘中建立一个文件

夹叫做 gzu，并且在此文件夹中建立一个 WB-INF 的子文件夹，同时在 WB-INF 文件夹中建立一个 web.xml 文件，此文件格式如下：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  version="2.5">

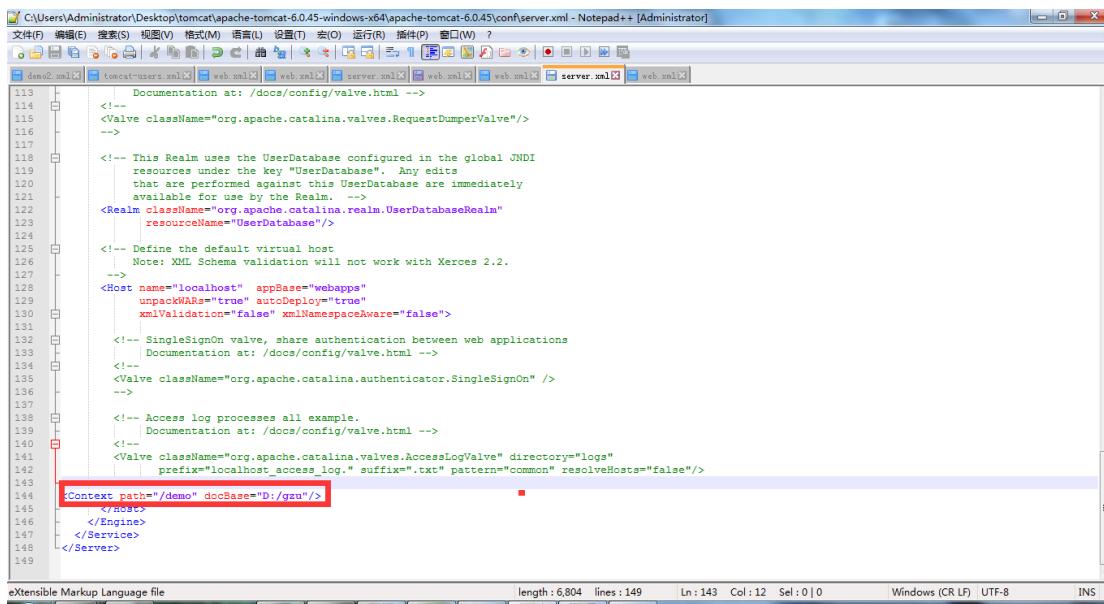
  <display-name>gzu </display-name>
  <description>
    Welcome to Tomcat
  </description>

</web-app>
```

web.xml 是整个 Web 的核心文件。

配置完工作目录之后，下面进行服务器的配置。打开 Tomcat 安装目录的 conf/server.xml 文件，在其中

慧科集团旗下品牌



```

<Context path="/demo" docBase="D:/gzu"/>

```

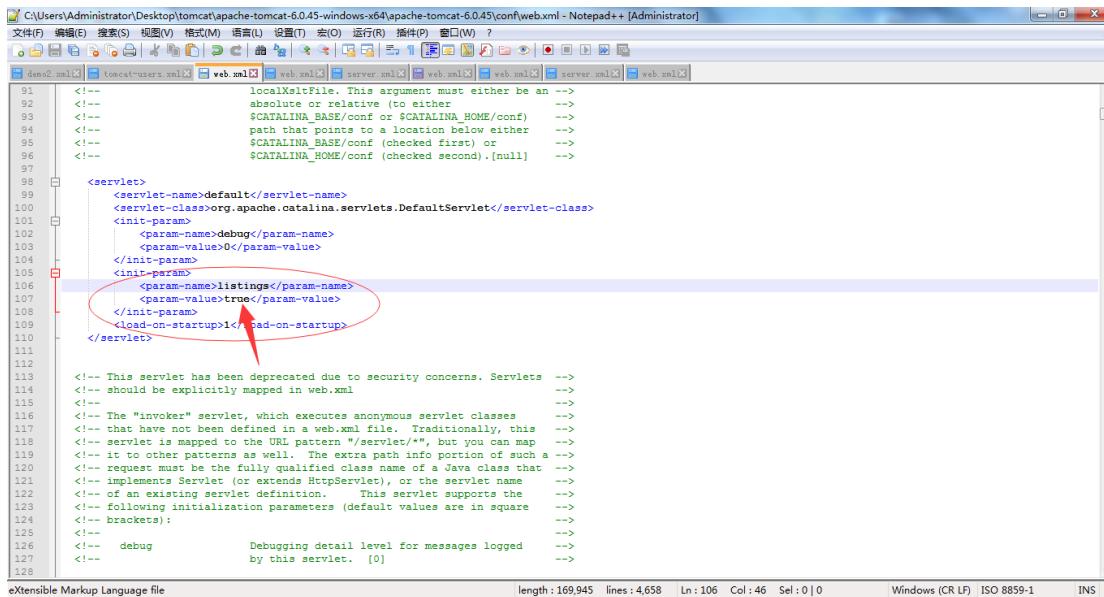
加入

```
<Context path="/demo" docBase="D:/gzu"/>
```

其中：path 表示浏览器上的访问的虚拟路径的名称。

docBase：虚拟路径所代表的真实的路径地址。

在 conf/web.xml 中的



```

<init-param>
    <param-name>listings</param-name>
    <param-value>true</param-value>
</init-param>
<load-on-startup>1</load-on-startup>

```

然后重启服务器。

HTTP 状态码

2XX: 请求成功

3XX: 重定向

4XX: 客户机出现的错误

403: 禁止

404: 服务器中找不到相应的文件

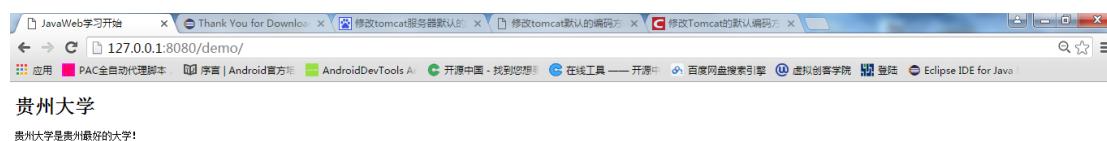
5XX: 服务器中出现的错误

500: 服务器内部的错误。

### 3、配置首页

Tomcat 服务器配置完虚拟目录之后, 可以配置一个 Web 项目的首页,

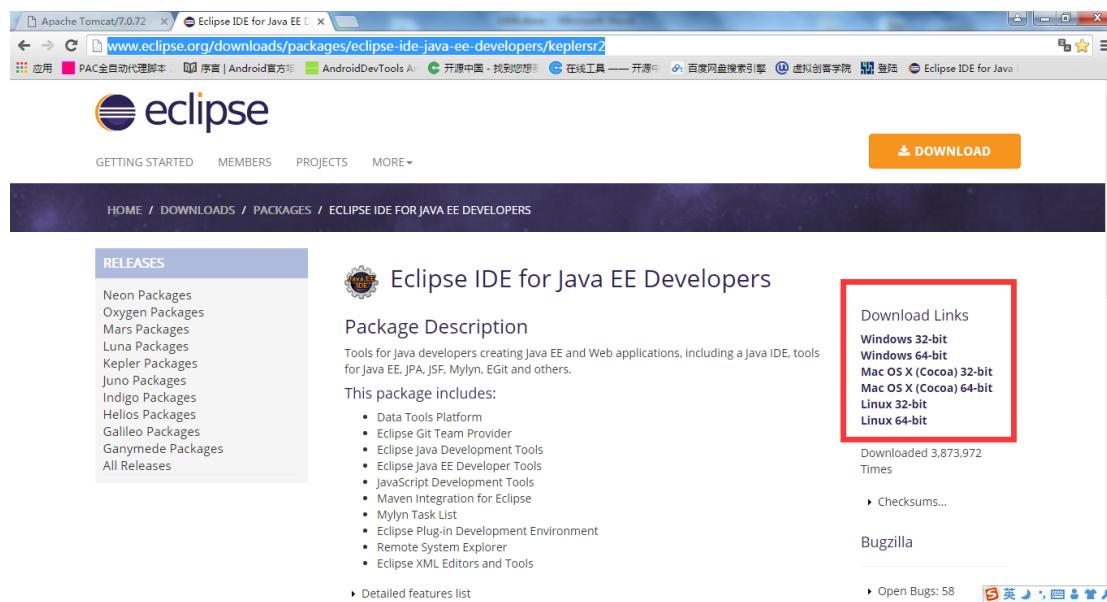
例如在配置好的虚拟目录中建立一个 index.html 文件。



## 2.2 eclipse 下载与安装

下载地址:

<http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/keplersr2>

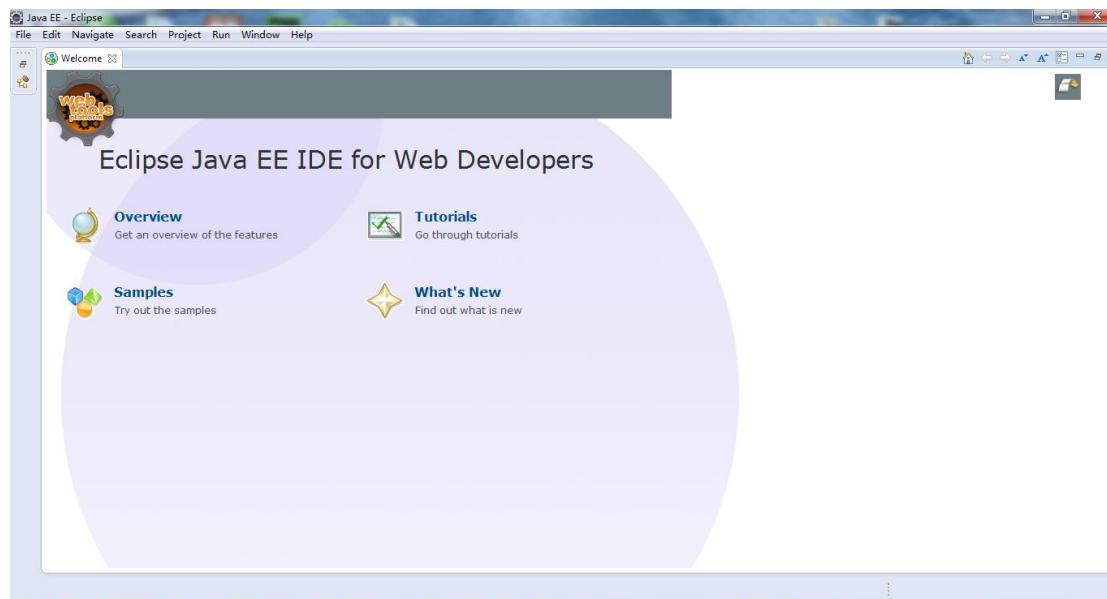


The screenshot shows the Eclipse IDE for Java EE Developers download page. On the left, there's a sidebar with a 'RELEASES' section containing links to various Eclipse versions like Neon, Oxygen, Mars, Luna, Kepler, Juno, Indigo, Helios, Galileo, Ganymede, and All Releases. The main content area features the title 'Eclipse IDE for Java EE Developers' and a 'Package Description' section. Below that is a list of 'This package includes:' with items such as Data Tools Platform, Eclipse Git Team Provider, etc. At the bottom right of the main content area, there's a 'Download Links' section with a red border, listing download links for different operating systems. The page also includes links for Checksums, Bugzilla, and Open Bugs.

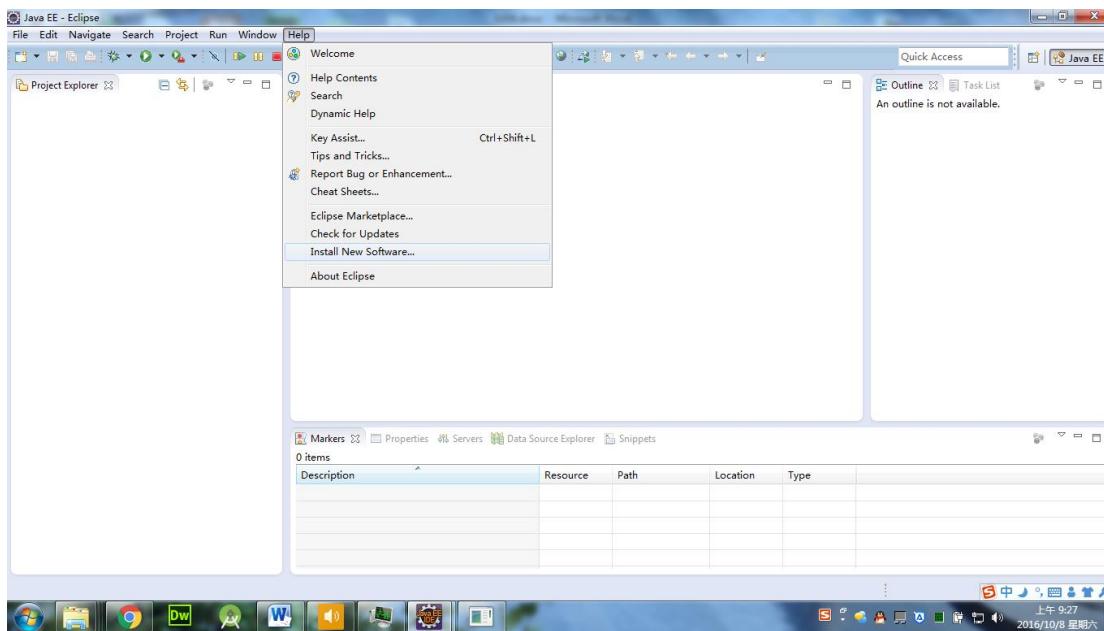
选择自己下载的版本

## 2.2.1 汉化 eclipse

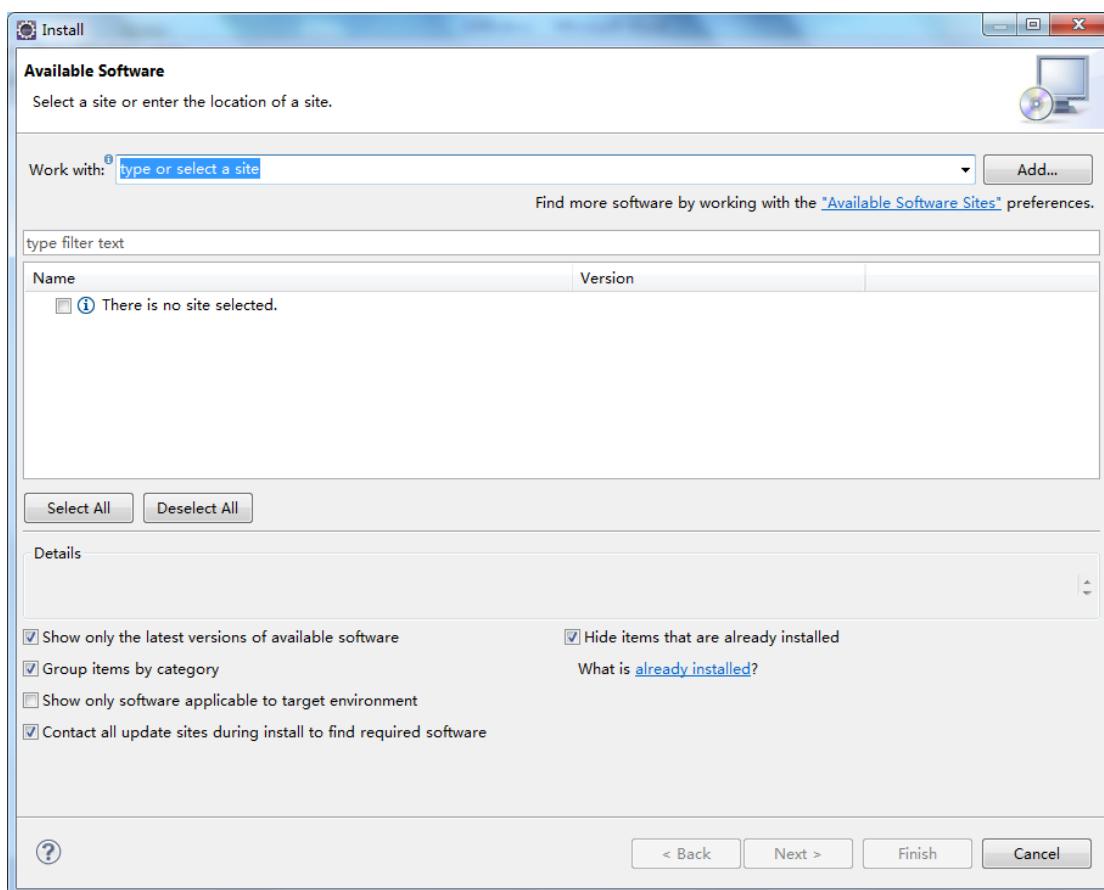
打开 eclipse



点击 help=》 Install New Soft。。。



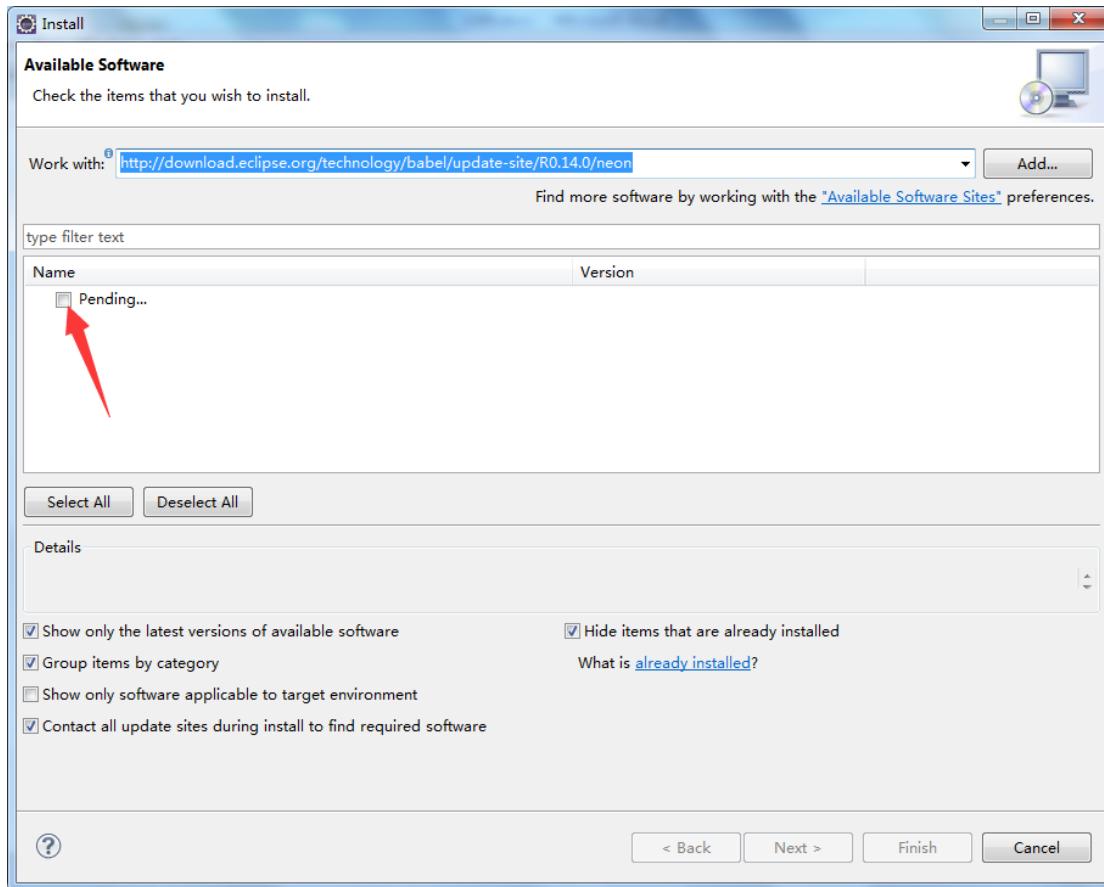
看到如下界面：

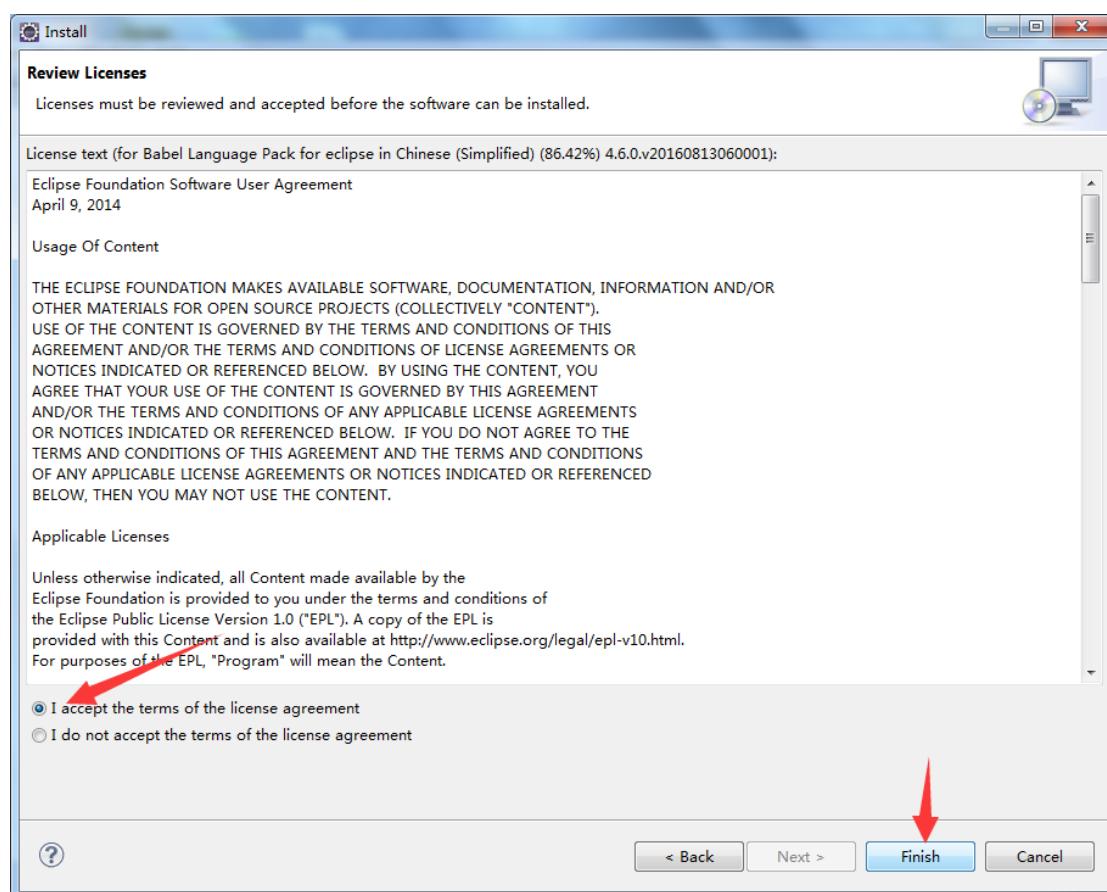
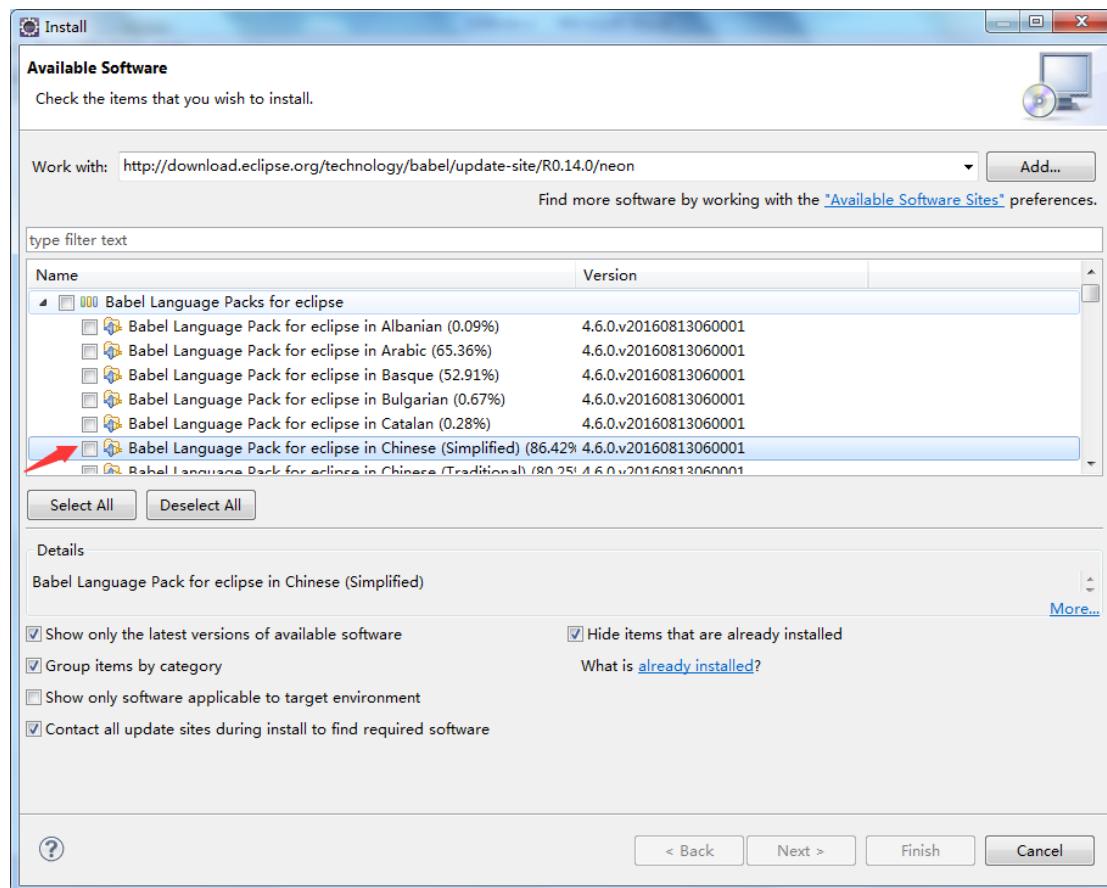


在 work with 里面输入：

<http://download.eclipse.org/technology/babel/update-site/R0.14.0/neo>

n

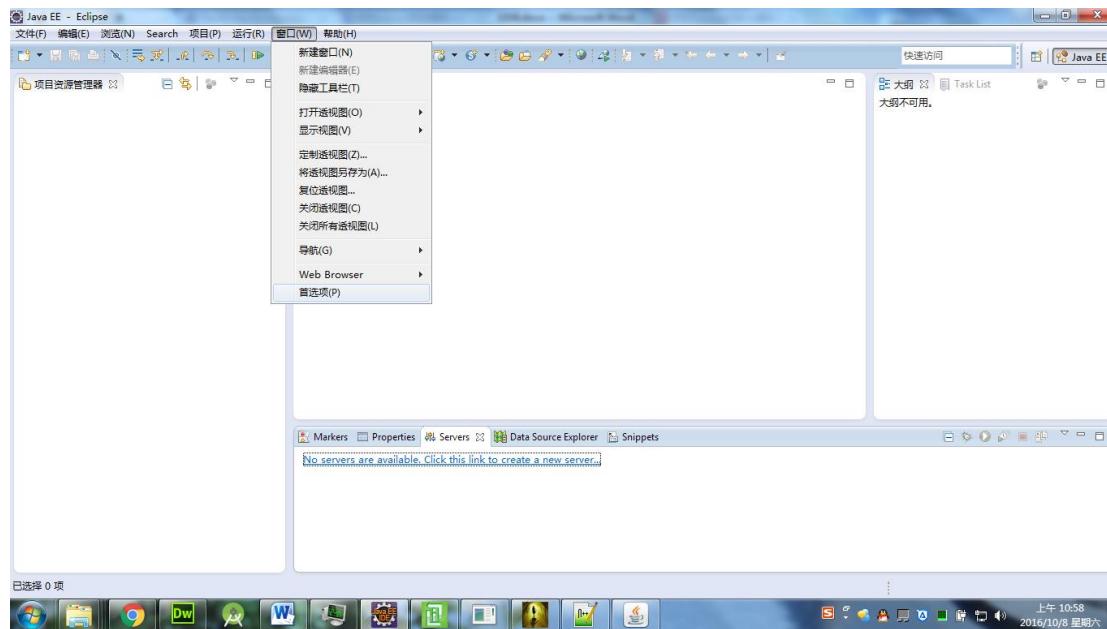




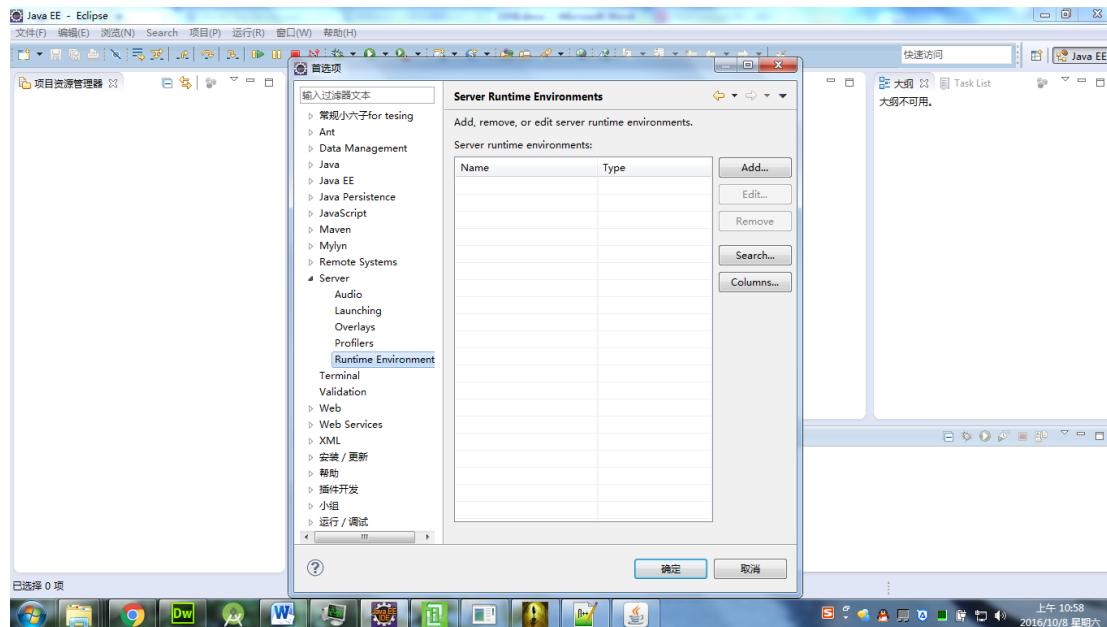
等待安装就好了。

## 2.2.2 集成 eclipse 与 tomcat

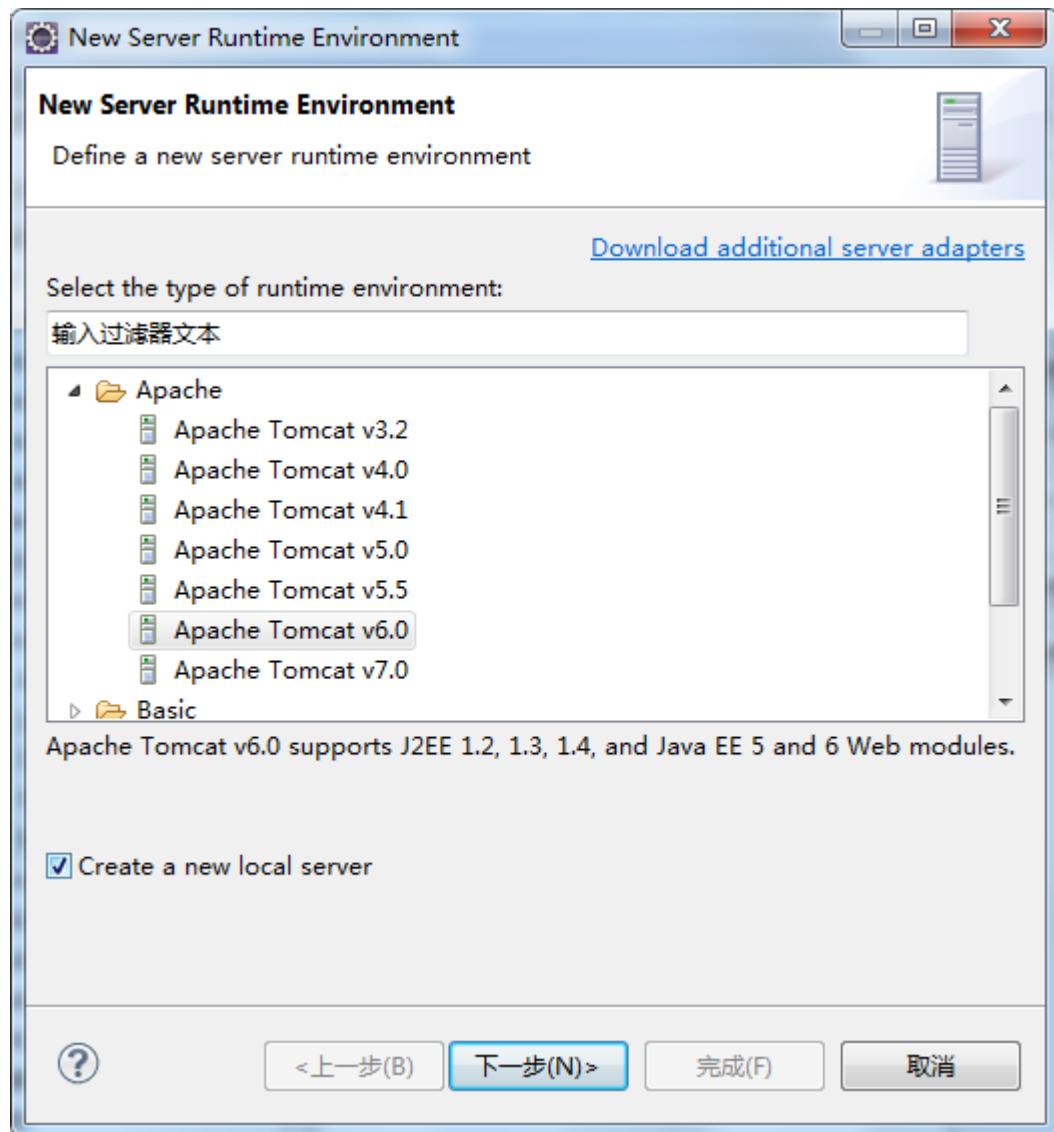
找到窗口=》首选项

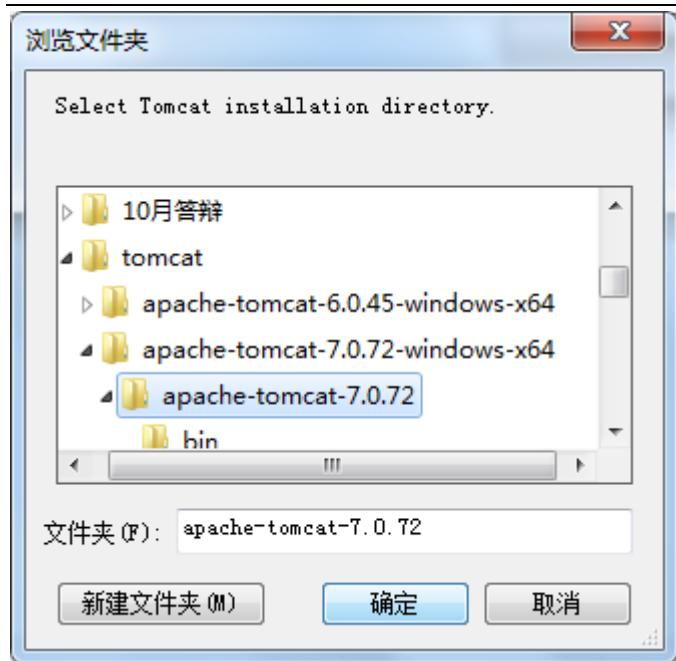


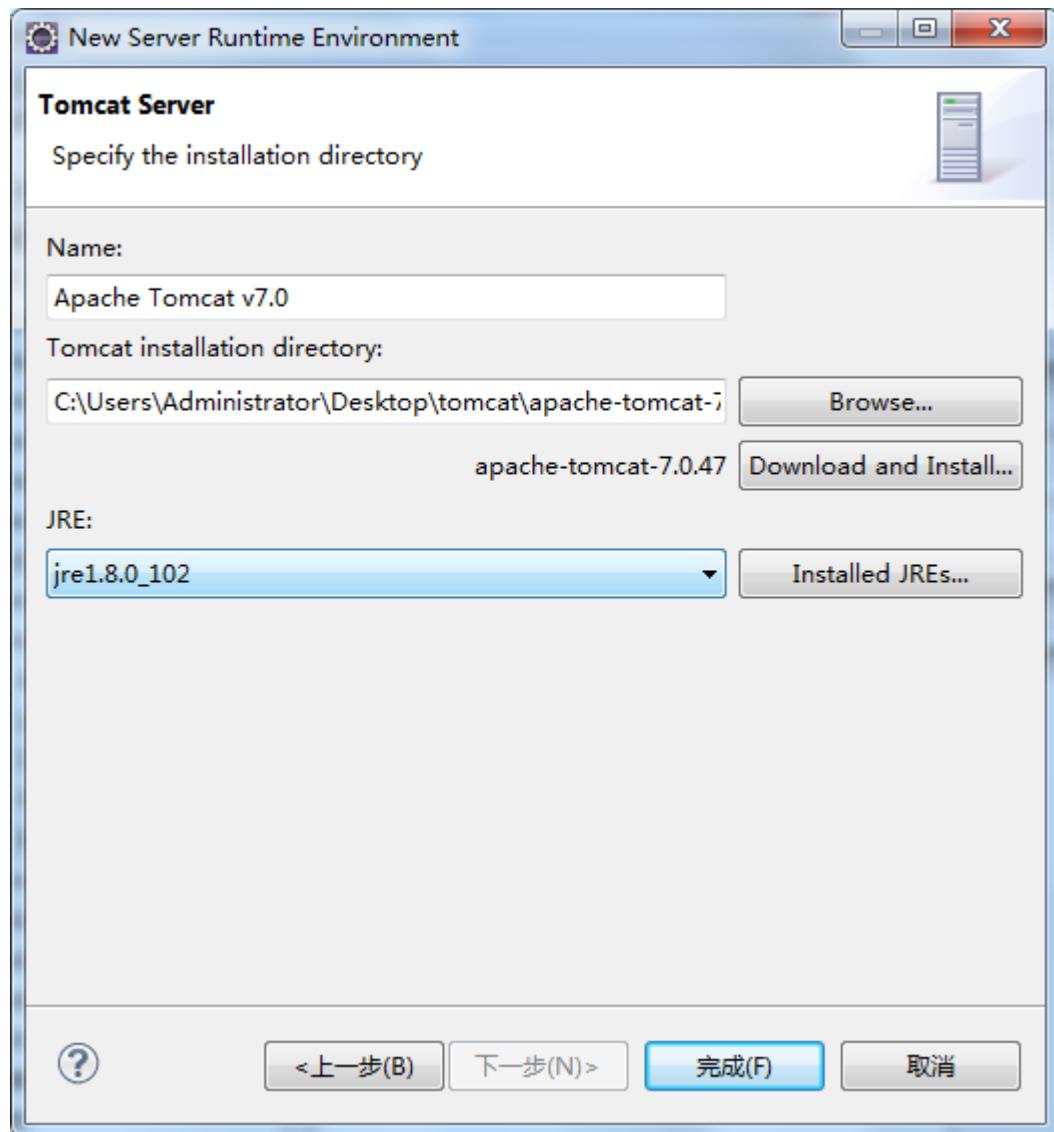
找到

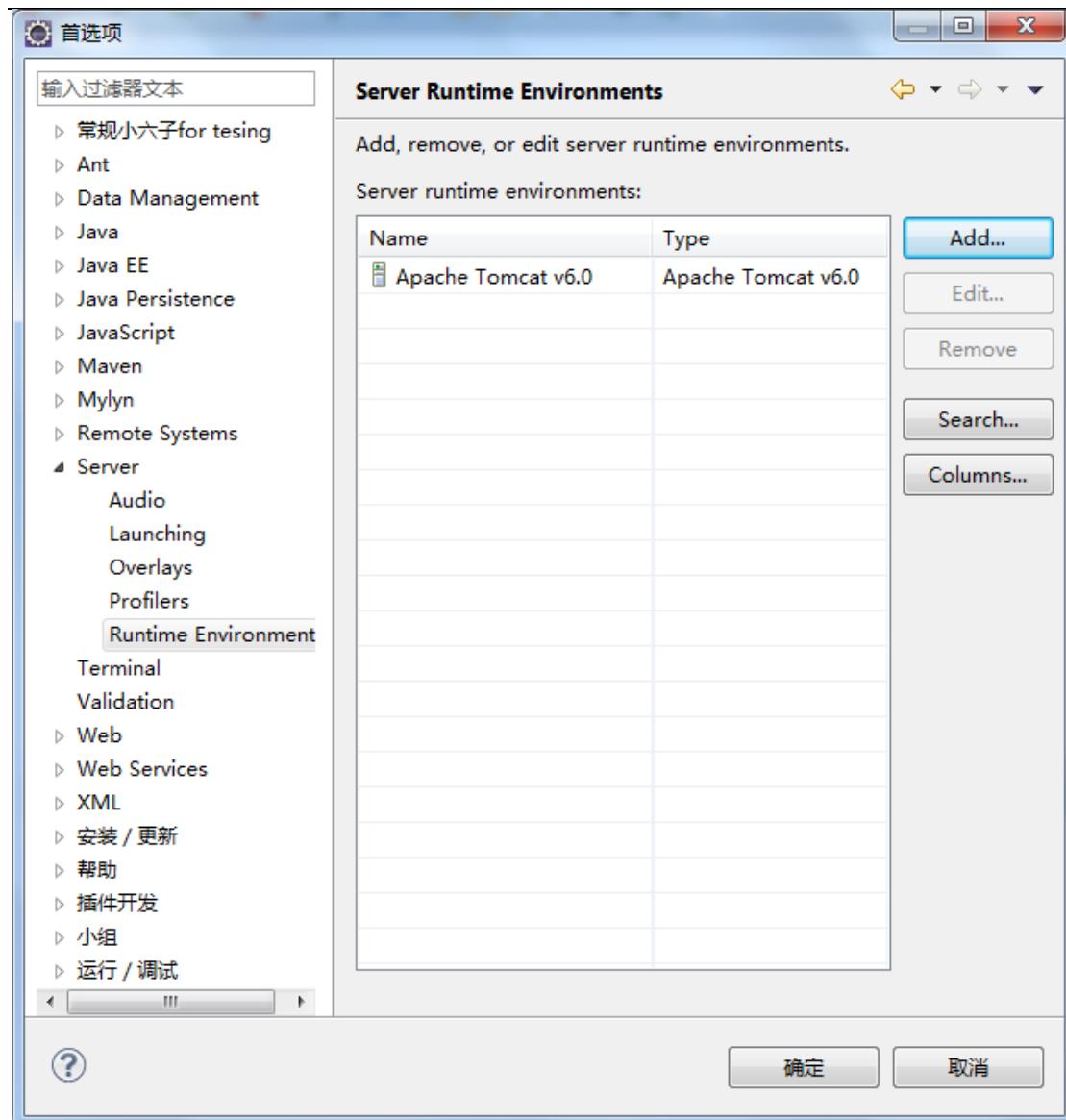


点击 add



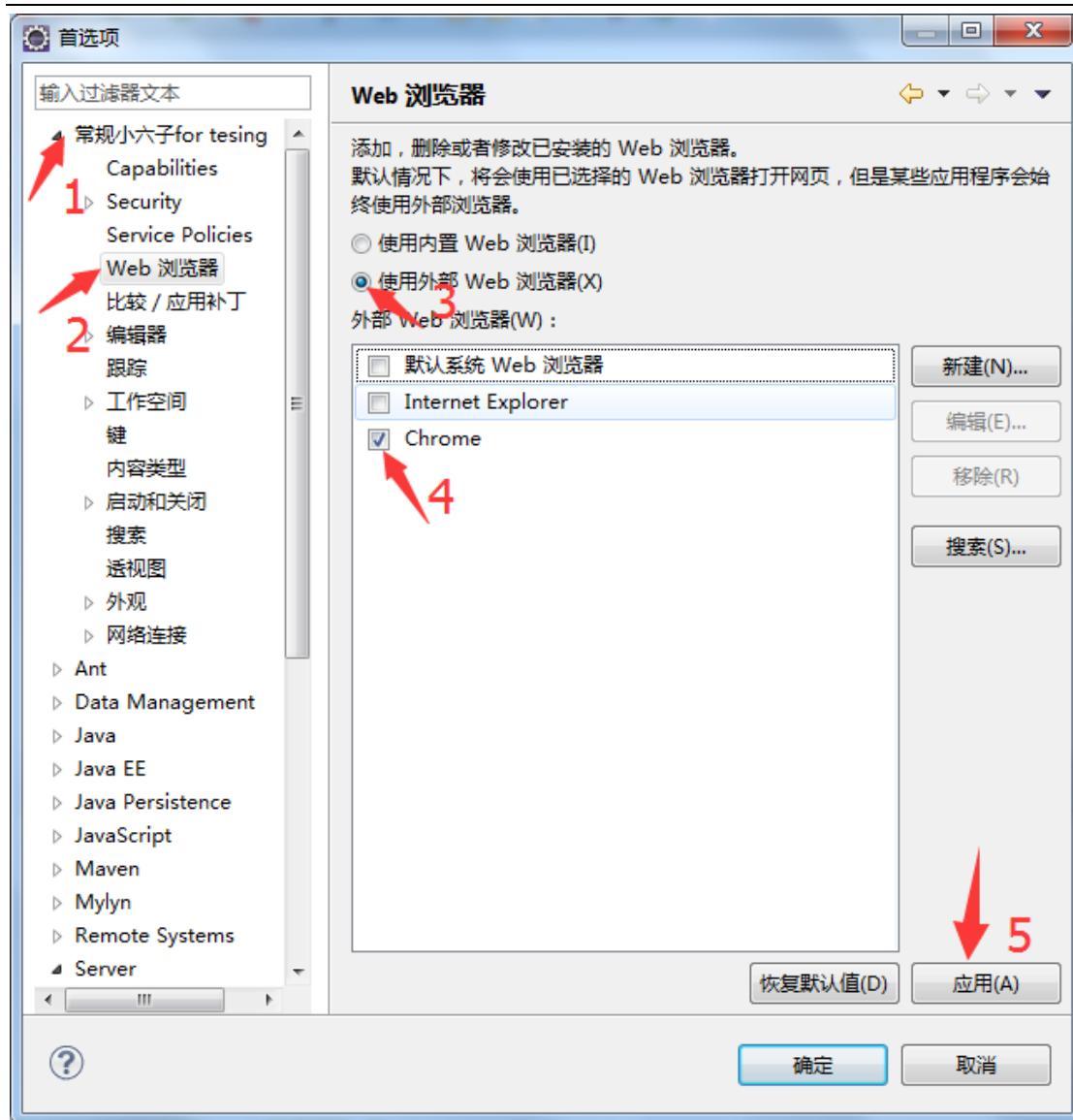




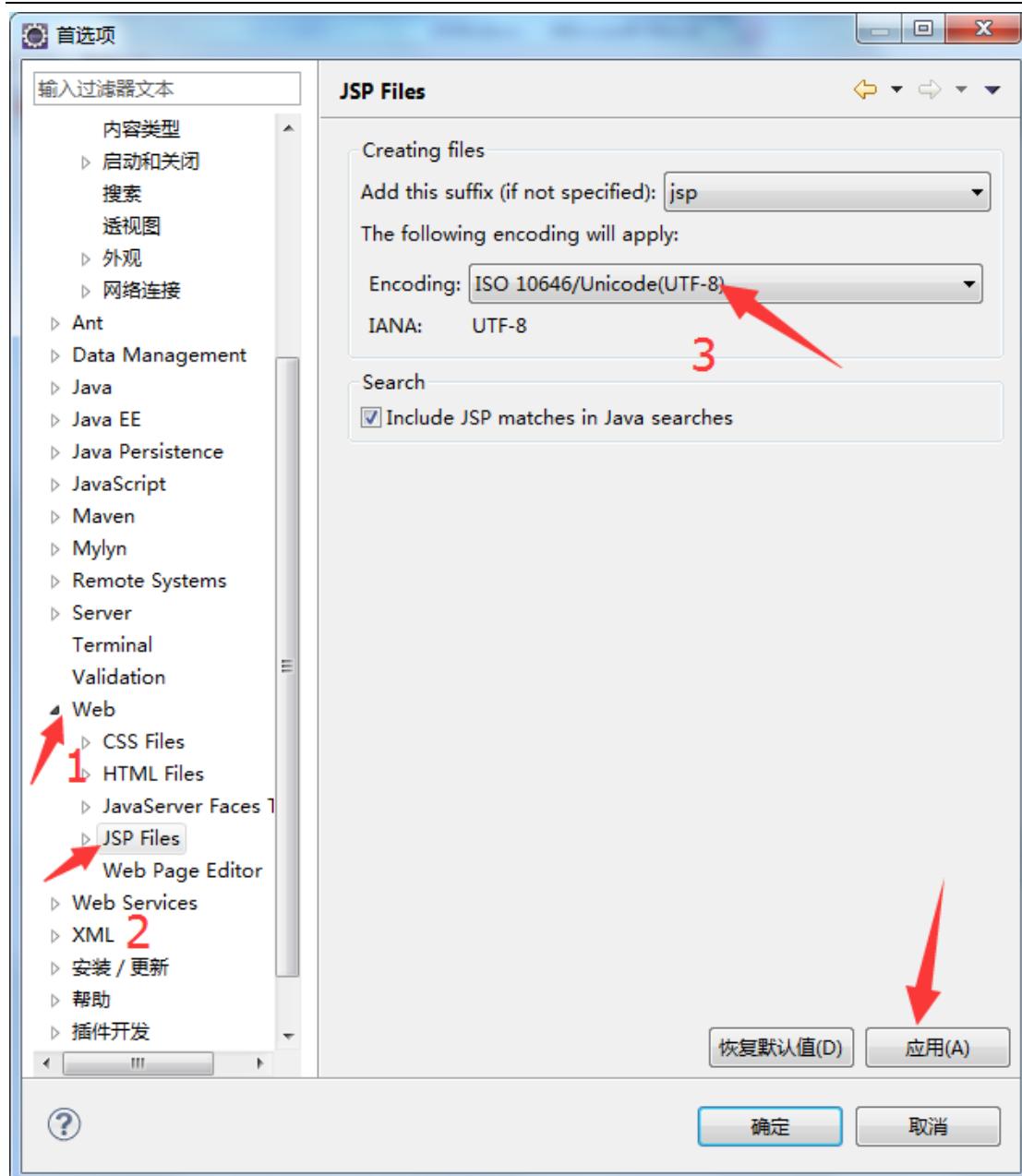


### 2.2.3 完善 Web 项目开发所需配置

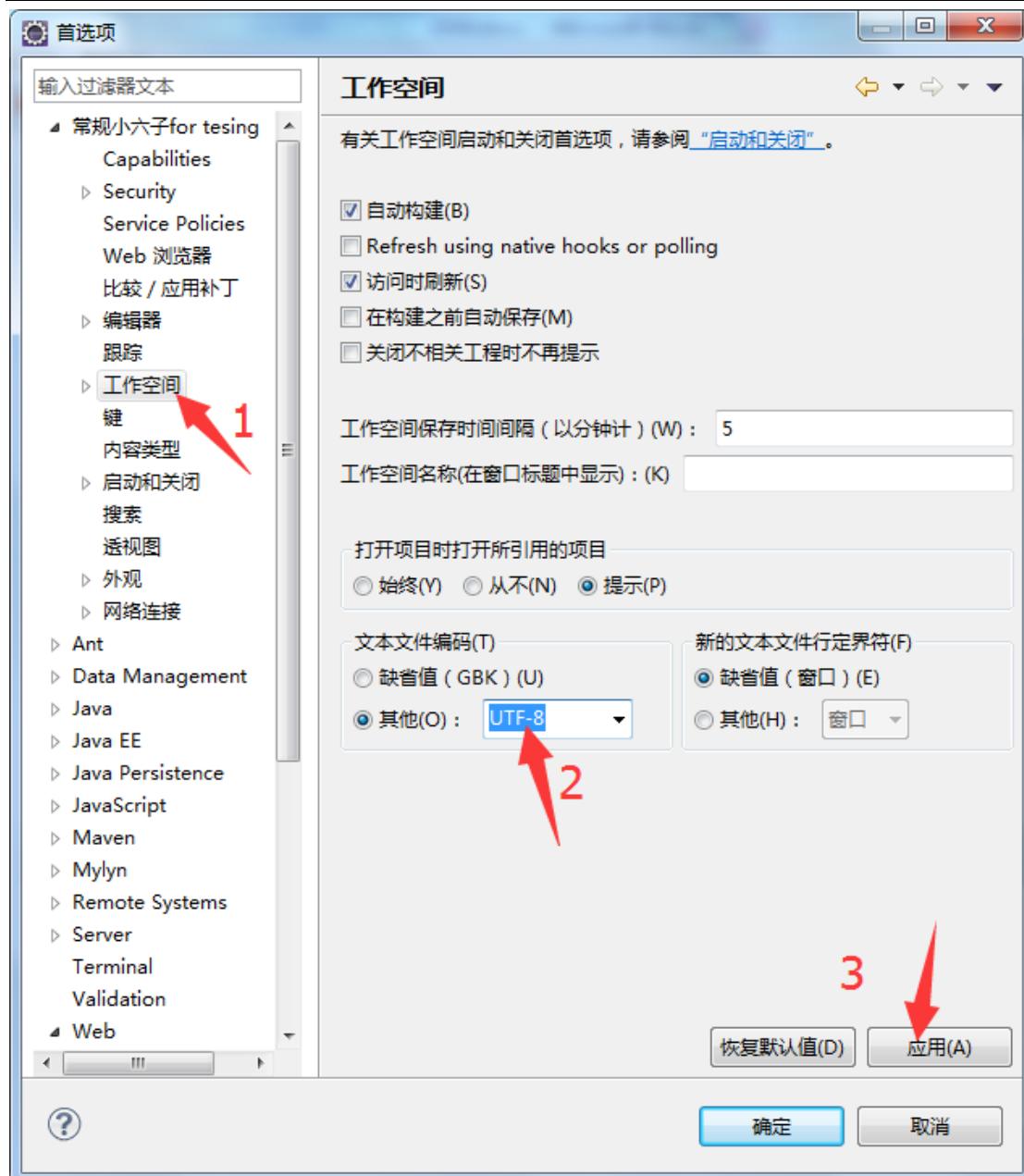
为 eclipse 指定 web 浏览器



指定 JSP 页面的编码格式

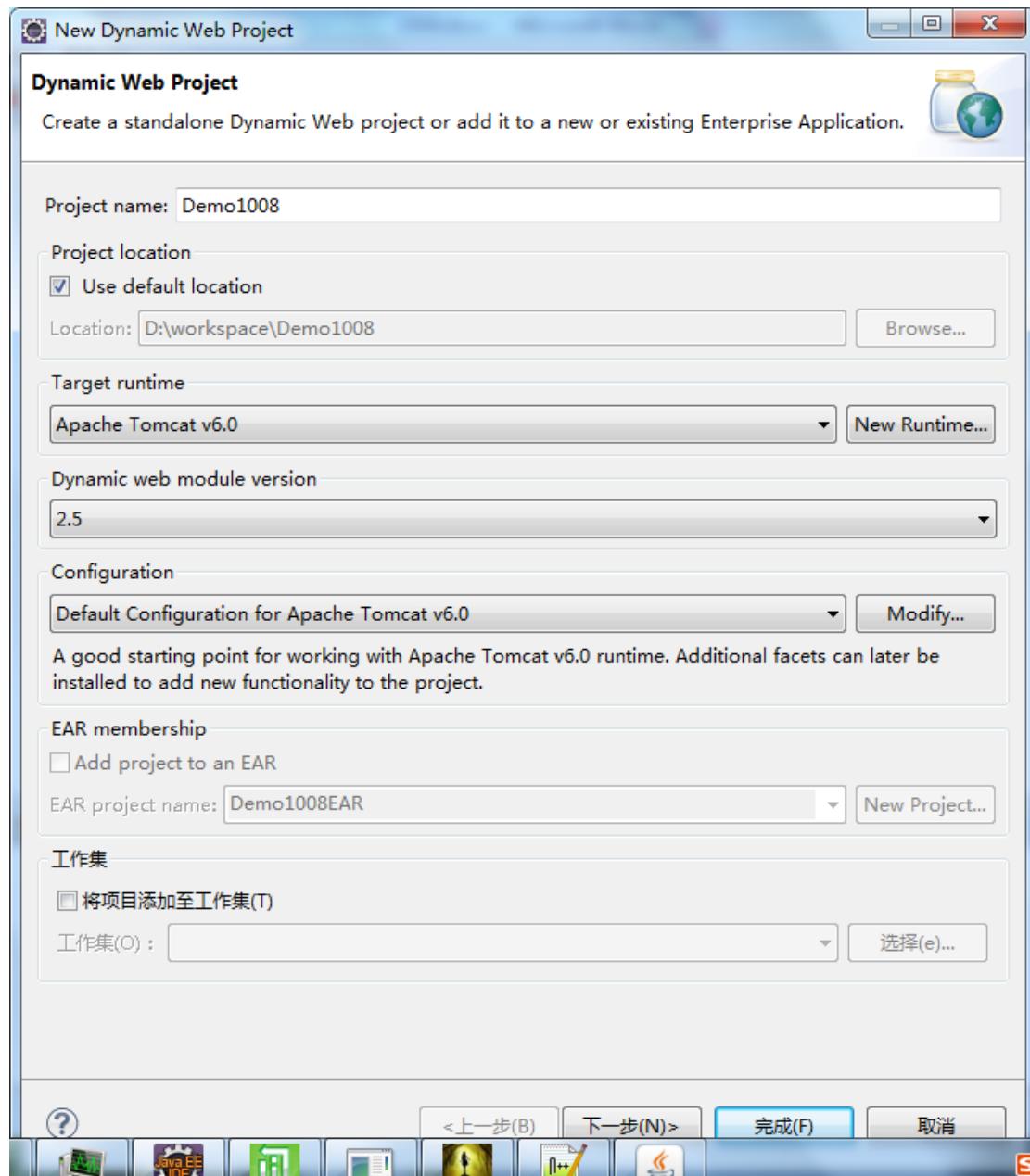


将整个工作空间设置为 UTF-8



## 2.3 第一个 Web 项目

新建



New Dynamic Web Project

**Web Module**

Configure web module settings.

Context root: Demo1008

Content directory: **WebContent**

Generate web.xml deployment descriptor

完成(F) 取消

```
<%@ 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>第一个JavaWeb页面! </title>
</head>
<body>

<%
    out.print("<h1>Hello,gzu!</h1>");
%>
```

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

## 3 JSP 基础语法

JSP (java server page) 在 JSP 中其核心的基本语法还是 java，所以 java 中各种判断、循环等语句在 JSP 中仍然可以使用。所以在本节中只介绍 JSP 的一些基本语法。

### 3.1 JSP 注释

在 JSP 中有两种注释，一种是显式的，这种注释允许用户看见，另外一种是隐式的，此种注释在客户端是无法看到的。

显示注释：

语法：

```
<!--显式注释-->
```

隐式注释：

有三种，语法：

第一种： //单行注释

第二种： /\*多行注释\*/

第三种： <%--注释--%>JSP 注释

demo1.jsp：

```
<body>  
<!-- 显式注释，表示客户端可以看到的 -->  
<h3>本demo是为了演示注释的基本应用而准备的</h3>  
<%
```

```
//Java提供的单行注释，客户端是看不到的；  
  
/*  
  
这里是java提供的多行注释，按道理  
  
客户端还是看不到  
  
*/  
  
%>  
  
<%-- 这是JSP提供的注释，客户端还是看不到！ --%>  
  
</body>
```

## 3.2 Scriptlet

在 JSP 中，最重要的就是 Scriptlet（脚本小程序），所有嵌入在 HTML 代码中的 Java 程序都必须使用 Scriptlet 标记出来。在 JSP 中，一共有 3 种 Scriptlet 代码。

第一种：<%%>

第二种：<%! %>

第三种：<%= %>

### 3.2.1 第一种 Scriptlet：<%%>

demo2.jsp

```
<body>  
  
<%  
  
int x = 19;
```

```
String str = "www.gzu.edu.cn";  
  
out.print("<h2> x =" + x + "</h2>");  
  
out.print("贵州大学的官网是: "+ str);  
  
%>  
  
</body>
```

### 3.2.2 第二种 Scriptlet: <%!%>

定义全局常量、方法、类

demo3.jsp

```
<body>  
  
<%!  
  
public static final double PAI = 3.1415926;  
  
%>  
  
<%!  
  
public int add(int x,int y){  
  
    return x+y;  
  
}  
  
%>  
  
<%!  
  
class Person{  
  
    private String name;  
  
    private int age;
```

```
//定义了一个含参的构造函数

public Person(String name, int age){

    this.name = name;

    this.age = age;

}

//覆盖了toString方法

public String toString(){

    return "用户名是: " + this.name +",年龄是: " +  
this.age;

}

}

%>

<%

//普通的Scriptlet

out.println("PAI的值是: "+PAI+"<br>");

out.println("2+9="+add(2,9)+"<br>");

out.println(new Person("hunk",30));

%>

</body>
```

在此 Scriptlet 可以定义全局常量，方法和类，但是在<%!%>里面不能出现任何其他的语句，所以又使用第一种 scriptlet 进行输出。

注意：尽量不要在 JSP 中定义类或者方法。

### 3.2.3 第三种 Scriptlet: <%=%>

第三种 Scriptlet 的主要功能是输出一个变量或者是一个具体的值，使用<%= %>的形式完成。也叫表达式输出。

demo4.jsp

```
<body>

<%
int x = 19;

String str = "www.gzu.edu.cn";

%>

<h2>x的值是: <%=x %></h2>

<h1>贵大的官网是: <%=str %></h1>

</body>
```

我们在第一种 scriptlet 中使用了 `out.print()` 进行输出，第三种是直接使用<%= %>来输出。在实际开发中到底用谁？

建议使用<%= %>来进行输出。

## 3.3 scriptlet 标签

```
<jsp:scriptlet>
    java scriptlet 代码
</jsp:scriptlet>
```

这个标签是后来有的，大量的<%%>这种标签在程序中，会是页面看

起来非常混乱，<jsp:scriptlet>就是好看。

### 3.4 page 指令

page 指令在 jsp 页面中定义了相关属性。page 指令常用的属性有：

**autoFlush:** 值有 true 和 false 两个值，如果设置为 true，当缓冲区满是，到客户端的输出被刷新，如果设置为 false，表示缓冲区溢出。默认认为 true。

**buffer:** 指定到客户端输出流的缓冲模式，如果设置为 none，表示不设置缓冲区；如果指定为数值，那么输出时就必须用不小于这个值的缓冲区进行缓冲。此属性要和 autoFlush 一起使用。默认不小于 8KB。

**contentType:** 定义 JSP 字符的编码和页面响应的 MIME 类型，如果是中文，则 `contentType="text/html; charset=UTF-8"`

**errorPage:** 定义了页面出错时要跳转的页面，要与 isErrorPage 属性一起使用

**isErrorPage:** 可以设置为 true 和 false，表示此页面出错时是否处理

**extends:** 主要定义此 JSP 页面产生的 Servlet 是从哪个父类扩展而来的

**import:** 在 JSP 中导包（表示可以出现多次）

**info:** JSP 页面的信息

**language:** 用来定义要使用的脚本语言，目前只有 java

**pageEncoding:** JSP 的页面字符编码

**session:** 值 true 和 false，默认为 true。

isThreadSafe：俩个值 true 和 false，表示的是此页面是否是线程安全的。如果设置为 true 表示一个 JSP 页面可以处理多个用户的请求。

### 3.4.1 设置页面的 MIME

在 page 指令中，使用最多的是就是 contentType 属性，如果想让一个 JSP 文件显示中文，则必须对整个页面指定 MIME 编码。

MIME（多功能 Internet 邮件扩充服务）

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html"  
pageEncoding="UTF-8"%>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
  
<title>第三种Scriptlet</title>  
  
</head>  
  
<body>  
  
<%  
  
int x = 19;  
  
String str = "www.gzu.edu.cn";  
  
%>
```

```
<h2>x的值是: <%=x %></h2>  
  
<h1>贵大的官网是: <%=str %></h1>  
  
</body>  
  
</html>
```

在上面的程序中，page 指定的药使用的开发语言是 java，然后通过 contentType 进行设置，本页面是按照 HTML 文本文件进行显示，页面的编码是 utf-8。

### 3.4.2 设置文件编码

pageEncoding 进行编码的设定。

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

contentType 里面的 charset="utf-8"指的是服务器发送给客户端的内容编码

pageEncoding="UTF-8"当前 JSP 页面文件本身的编码。

### 3.4.3 错误页设置

demo5.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>
```

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

pageEncoding="UTF-8"%>

<%@ page errorPage="error.jsp" %>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>异常页面</title>

</head>

<body>

<%
int x = 10/0;

%>

<h1>页面发生了错误！</h1>

</body>

</html>
```

error.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<html>

<head>

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

```
pageEncoding="UTF-8"%>

<%@ page isErrorPage="true" %>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>错误页面</title>

</head>

<body>

<h1>发生错误了，正在努力查找问题，请稍后。。。。。</h1>

</body>

</html>
```

### 3.4.4 数据库连接操作 import

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<html>

<head>

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

pageEncoding="UTF-8"%>

<%@ page import="java.sql.*"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
```

```
<title>列出数据</title>

</head>

<body>

<%!public static final String DBDRIVER =
"org.gjt.mm.mysql.Driver";>

public static final String DBURL =
"jdbc:mysql://localhost:3306/school";//school是你要操
作的数据库的库名

public static final String DBUSR = "root";
public static final String DBPASS = "123456";%>

<%
Connection conn = null;
PreparedStatement ps = null;
ResultSet rs = null;
%>

<%
try {
    Class.forName(DBDRIVER);//数据库驱动加载
    conn = DriverManager.getConnection(DBURL,
DBUSR, DBPASS);//取得数据库连接
    String sql = "SELECT * from student";//定义查询
语句
}
```

```
ps = conn.prepareStatement(sql);//实例化
prepareStatement对象

rs = ps.executeQuery();//执行查询操作

%>

<center>

<table border="1" width="60%">

<tr>

<td>学号</td>

<td>姓名</td>

</tr>

<%

while (rs.next()) {//循环打印出student表中的
数据

    int sno = rs.getInt(1);//取出学号

    String sname = rs.getString(2);//取出姓
名

%>

<tr>

<td><%=sno%></td>

<td><%=sname%></td>

</tr>

<%
```

```
        }

    %>

    </table>

    </center>

<%

} catch (Exception e) {//异常处理

    System.out.println(e);//在Tomcat中打印异常信息

} finally {//程序的统一出口

    rs.close();//关闭结果集

    ps.close();//关闭操作

    conn.close();//关闭连接

}

%>

</body>

</html>
```

## 3.5 包含指令



### 3.5.1 静态包含

静态包含指令是在 JSP 编译的时候插入一个包含文本或代码的文件，这个过程是静态的，包含的文件可以是 JSP、HTML、文本也可以是一段 java 代码。

**注意：**在一个完整的 HTML 中，对于<html>、<head>、<body>、<title>这些元素只能出现一次。

语法：

```
<%@ include file="要包含的文件的路径"%>
```

先准备三个被包含的文件：

```
<h2>被包含的第一个 HTML 文件</h2>
```

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

<h3>被包含的 JSP 文件</h3>
```

```
<h4>被包含的文件</h4>
```

demo1

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<h1>下面演示的是静态包含</h1>
```

```
<%@ include file="include1.html" %>
<%@ include file="include2.jsp" %>
<%@ include file="include3.inc" %>

</body>
</html>
```

### 3.5.2 动态包含

与静态包含不同，在包含的过程中可以自动区分被包含的页面是动态的还是静态的，如果是静态的，就静态包含一样，如果是动态的，则先进行动态处理，然后在将处理后的结果显示出来。

语法：

```
<jsp:include page="{要包含的文件路径|<%= %>}" flush="true"/>
```

传递参数：

```
<jsp:include page="{要包含的文件路径|<%= %>}" flush="true">
<jsp:param name="参数名" value="参数值"/>
.....
</jsp:include>
```

flush 属性可选的值有 true 和 false。当设置为 false 的时候表示这个网页完全被读进来以后再输出。当设置为 true 时，只有当网页中的 buffer 满了之后才会输出，一般情况都会设置成 true，默认值就是 true。

demo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<h1>下面演示的是动态包含</h1>
<jsp:include page="include1.html" />
<jsp:include page="include2.jsp" />
<jsp:include page="include3.inc" />
</body>
</html>
```

动态包含含参:

定义包含页：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>

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

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<% String username = "hunk"; %>

<h1>动态包含含参</h1>

<jsp:include page="include4.jsp">

<jsp:param value="<%=username %>" name="name"/>

<jsp:param value="30" name="age"/>

</jsp:include>

</body>

</html>
```

被包含页：

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

pageEncoding="UTF-8"%>

<h1>参数1: <%=request.getParameter("name") %></h1>
<h1>参数 2: <%=request.getParameter("age") %></h1>
```

学过了静态包含和动态包含，使用哪个更好？

建议在开发使用动态包含。因为静态包含是先包含再处理，而动态包含如果被包含的页面是动态的。则属于先处理再包含。

## 3.6 跳转指令

在 web 开发的过程中可以使用<jsp:forward>,将一个用户的请求一个页面传递到另外一个页面。

语法：

```
<jsp:forward page="要跳转的文件路径"/>
```

传递参数：

```
<jsp:forward page="要跳转的文件路径">
<jsp:prama name="" value="" />
.....
</jsp:forward>
```

demo4:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
```

```
Transitional//EN">

<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<% String username="hunk"; %>

<jsp:forward page="demo5.jsp">

<jsp:param value="<%=username %>" name="name"/>

<jsp:param value="30" name="age"/>

</jsp:forward>

</body>

</html>
```

demo5:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
```

```
<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<h1>这是跳转之后的页面</h1>

<h2>参数一: <%=request.getParameter("name") %></h2>

<h2>参数二: <%=request.getParameter("age") %></h2>

</body>

</html>
```

## 3.7 作业

用 JSP+JDBC 实现用户登录。

分析：数据库，有一个表 user 表

页面：登录页 login.html，就一个表单

传递给 login\_check.jsp 在这个页面中是不是要连接数据库

在这个 check 页面中要实现跳转，如果成功要跳转至成功页，失败的

话跳转至失败页面。

login.html

```
<!doctype html>

<html>
<head>
<meta charset="utf-8">
<title>登录</title>
</head>
<body>

<h1>请您登录</h1>

<form action="Login_check.jsp" method="post">
<table border="1">
<tr>
<td>用户名: </td>
<td><input type="text" name="userid"></td>
</tr>
<tr>
<td>密 &nbsp;&nbsp;码: </td>
<td><input type="password"
name="password"></td>
</tr>
</table>
</form>

```

```
<tr>

    <td><input type="submit" value="登录"></td>

    <td><input type="reset" value="重置"></td>

</tr>

</table>

</form>

</body>

</html>
```

login\_check.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<html>

<head>

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

pageEncoding="UTF-8"%>

<%@ page import="java.sql.*"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>登录</title>

</head>
```

```
<body>

<%!public static final String DBDRIVER =
"org.gjt.mm.mysql.Driver";

public static final String DBURL =
"jdbc:mysql://127.0.0.1:3306/school";

public static final String DBUSER = "root";
public static final String DBPASS = "123456";%>

<%
    Connection conn = null;//声明数据库连接对象
    PreparedStatement ps = null;//声明数据库操作
    ResultSet rs = null;//声明数据库结果集
    boolean flag = false;
    String name = null;
%>

<%
try {
    Class.forName(DBDRIVER);
    conn = DriverManager.getConnection(DBURL,
DBUSER, DBPASS);

    String sql = "select name from user where userid
= ? and password = ?";
    ps = conn.prepareStatement(sql);
}
```

```
        ps.setString(1,  
request.getParameter("userid"));  
  
        ps.setString(2,  
request.getParameter("password"));  
  
        rs = ps.executeQuery();  
  
        if (rs.next()) {  
  
            name = rs.getString(1);  
  
            flag = true;  
  
        }  
  
    } catch (Exception e) {  
  
        System.out.println(e);  
  
    } finally {  
  
        try {  
  
            rs.close();  
  
            ps.close();  
  
            conn.close();  
  
        } catch (Exception e) {  
  
            System.out.println(e);  
  
        }  
  
    }  
  
%>
```

```
<%

    if (flag) {

%>

<jsp:forward page="Login_sucess.jsp">

    <jsp:param value="<%name%>" name="username" />

</jsp:forward>

<%

    } else {

%>

<jsp:forward page="Login_failure.jsp" />

<%

    }

%>

</body>

</html>
```

*Login\_sucess.jsp*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<html>

<head>

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

```
pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>登录成功</title>

</head>

<body>

<center>

<h1>登录成功</h1>

<h2>欢迎<font
color="red"><%=request.getParameter("username") %></f
ont>光临！ </h2>

</center>

</body>

</html>
```

*Login\_failure.jsp*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<html>

<head>

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

pageEncoding="UTF-8"%>
```

```
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>登录失败</title>

</head>

<body>

<center>

<h1>登录失败了</h1>

<h2>用户名或密码错误, 请返回登录
```

## 4 JSP 内置对象

JSP 内置对象是 Web 程序开发中最重要的。

### 4.1 JSP 内置对象概览

在 JSP 中为了方便用户开发，给我们提供了 9 大内置对象，这些内置对象是由 Web 容器进行实例化的，不需要向我们在 Java 中那样，看见对象就得使用 new 关键字来实例化对象，在 JSP 中，这些内置对象直接调用即可。

<http://tool.oschina.net/apidocs/apidoc?api=javaEE6>

**pageContext** JSP 的页面容器 javax.servlet.jsp.PageContext

**request** 得到用户的请求信息 javax.servlet.http.HttpServletRequest

**response** 服务 器 向 客 户 端 的 响 应 信 息

javax.servlet.http.HttpServletResponse

**session** 用来保存每一个用户的信息 javax.servlet.http.HttpSession

**application** 表示所有用户的共享信息 javax.servlet.ServletContext

**config** 服务器配置，可以获取初始化参数 javax.servlet.ServletConfig

**out** 页面输出 javax.servlet.jsp.JspWriter

**page** 表示从该页面中表示出来的一个 servlet 实例 java.lang.Object

**exception** 表示 JSP 页面所发生的异常，在错误页中才起作用

java.lang.Throwable

## 4.2 四种属性范围

属性范围：就是一个内置对象，可以在对少个页面中保存并继续使用。

在 JSP 中提供的四种属性范围分别是：

`page` 只在一个页面中有效，跳转之后就无效了

`request` 只在一次请求中保存，服务器跳转之后依然有效

`session` 在一次会话范围内，无论何种跳转都可以使用，但是新打开浏览器无法使用

`application` 在整个服务器上保存，所有用户都可以使用。

上面的 4 个内置对象都支持下面三个属性操作方法：

`public void setAttribute (String name , Object o)` 设置属性的名称及内容

`public Object getAttribute (String name)` 根据属性名称取得属性

`public void removeAttribute (String name)` 删除指定的属性

### 4.2.1 page 属性范围

`page` 属性范围（用 `pageContext` 表示），表示将一个属性设置在本页上，跳转之后无法取得。

`demo1.jsp`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
<html>  
<head>
```

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

pageEncoding="UTF-8"%>

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

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>page属性范围</title>

</head>

<body>

<%//设置page属性范围

pageContext.setAttribute("name", "hunk");

pageContext.setAttribute("birthday", new Date());

%>

<%//从page属性范围内取出属性，并执行向下转型

String username =
(String)pageContext.getAttribute("name");

Date userBirthday =
(Date)pageContext.getAttribute("birthday");

%>

<h2>用户名: <%=username %></h2>

<h2>生日: <%=userBirthday %></h2>

</body>
```

```
</html>
```

在上面程序的基础上，我们进一步，采用<jsp:forward>进行跳转，跳转之后看能否取得属性：

在 demo2.jsp 中设置俩个属性，并且实现跳转至 demo3.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ page import="java.util.*" %>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>page属性范围</title>
</head>
<body>
<%//设置page属性范围
pageContext.setAttribute("name", "hunk");
pageContext.setAttribute("birthday", new Date());
%>
<jsp:forward page="demo3.jsp"/>
```

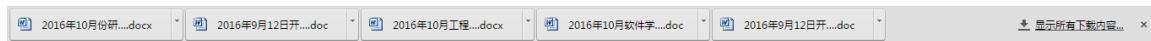
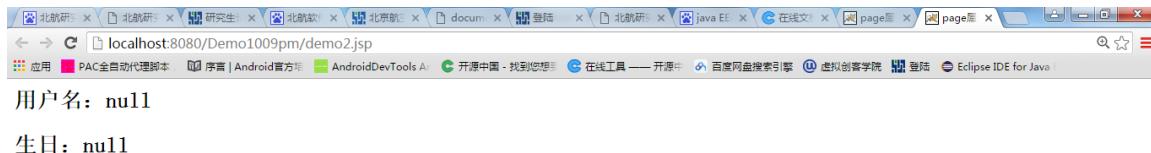
```
</body>  
  
</html>
```

在 demo3 中获取属性：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
  
pageEncoding="UTF-8"%>  
  
<%@ page import="java.util.*" %>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
  
<title>page属性范围</title>  
  
</head>  
  
<body>  
  
<%//从page属性范围中取出属性，并执行向下转型  
  
String username =  
  
(String)pageContext.getAttribute("name");  
  
Date userBirthday =  
  
(Date)pageContext.getAttribute("birthday");  
  
%>
```

```
<h2>用户名: <%=username %></h2>  
  
<h2>生日: <%=userBirthday %></h2>  
  
</body>  
  
</html>
```

结果发现：



获取不到，说 `page` 属性范围只在当前页面有效，跳转之后无效。

### 4.2.2 request 属性范围

上面在将 `page` 属性范围时，发现页面跳转之后，属性失效，要想页面跳转之后，属性仍然保存，则使用 `request` 属性范围，把上面的例子改装一下：

demo2:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>
```

```
<head>

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

pageEncoding="UTF-8"%>

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

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>request属性范围</title>

</head>

<body>

<%//设置page属性范围

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

request.setAttribute("birthday", new Date());%>

<jsp:forward page="demo3.jsp"/>

</body>

</html>
```

demo3

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<html>

<head>
```

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

pageEncoding="UTF-8"%>

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

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>request属性范围</title>

</head>

<body>

<%//从page属性范围内取出属性，并执行向下转型

String username = (String)request.getAttribute("name");

Date userBirthday =
    (Date)request.getAttribute("birthday");

%>

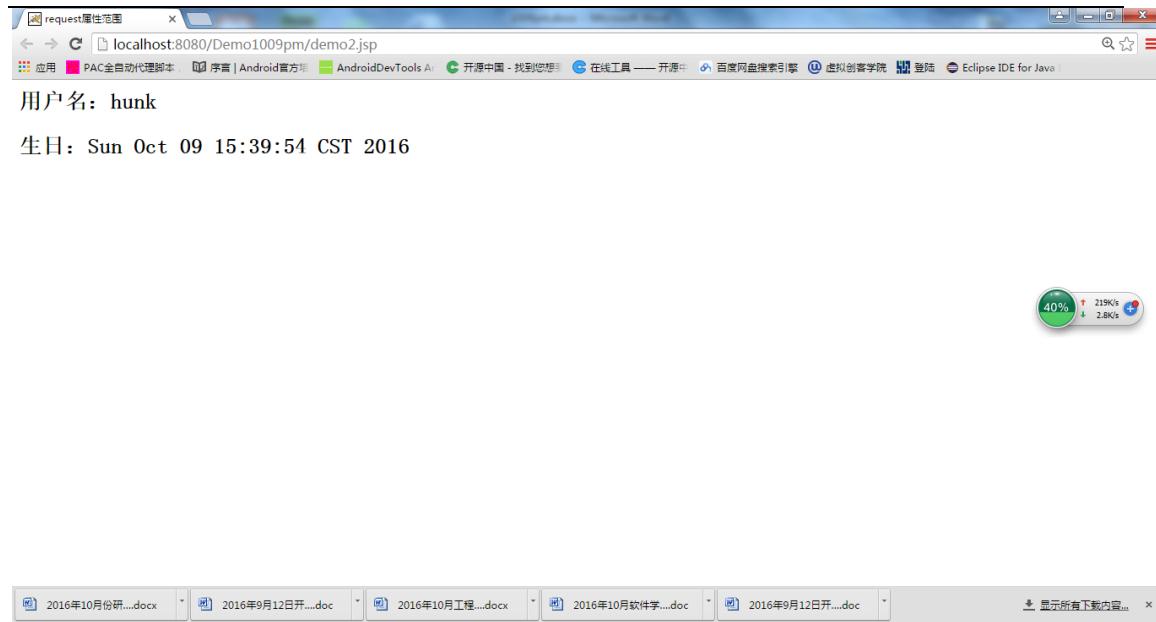
<h2>用户名: <%=username %></h2>

<h2>生日: <%=userBirthday %></h2>

</body>

</html>
```

运行结果:



在上面的基础上，继续改装：在 demo2 中不使用<jsp:forward>跳转了，  
使用超链接跳转。

demo2:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ page import="java.util.*" %>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>request属性范围</title>
</head>
```

```
<body>

<%//设置page属性范围

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

request.setAttribute("birthday", new Date());


%>

<a href="demo3.jsp">跳转一下看看! </a>

</body>

</html>
```

demo3

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<html>

<head>

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

pageEncoding="UTF-8"%>

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

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>request属性范围</title>

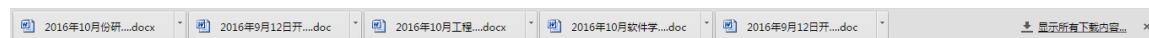
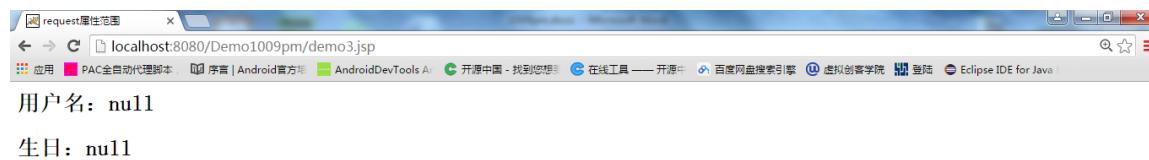
</head>

<body>
```

```
<%//从page属性范围中取出属性，并执行向下转型
```

```
String username = (String)request.getAttribute("name");  
  
Date userBirthday =  
  
(Date)request.getAttribute("birthday");  
  
%>  
  
<h2>用户名: <%=username %></h2>  
  
<h2>生日: <%=userBirthday %></h2>  
  
</body>  
  
</html>
```

运行结果：



在 demo2 中使用的 request 属性范围只针对于服务器端跳转，所以在使用超链接跳转的时候，request 就失效了，此时无法取得属性。

### 4.2.3 session 属性范围

如果希望一个属性设置以后，可以在任何一个页面中都获取到属性，

则可以使用 session 属性范围。继续改装上面的例子：

demo2:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
  
pageEncoding="UTF-8"%>  
  
<%@ page import="java.util.*" %>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
  
<title>request属性范围</title>  
  
</head>  
  
<body>  
  
<%//设置page属性范围  
  
session.setAttribute("name", "hunk");  
  
session.setAttribute("birthday", new Date());  
  
%>  
  
<a href="demo3.jsp">跳转一下看看！ </a>  
  
</body>  
  
</html>
```

## demo3

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ page import="java.util.*" %>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>request属性范围</title>
</head>
<body>
<%//从page属性范围内取出属性，并执行向下转型
String username = (String)session.getAttribute("name");
Date userBirthday =
(Date)session.getAttribute("birthday");
%>
<h2>用户名: <%=username %></h2>
<h2>生日: <%=userBirthday %></h2>
</body>
```

```
</html>
```

运行结果：



但是，现在的问题是，我如果直接打开 demo3，那么情况是？



发现没有取到属性，直接打开取不到，要想取到，就采取 application 属性范围。

#### 4.2.4 application 属性范围

继续改装上面的例子：

demo2:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
  
pageEncoding="UTF-8"%>  
  
<%@ page import="java.util.*" %>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
  
<title>application属性范围</title>  
  
</head>  
  
<body>  
  
<%//设置application属性范围  
  
application.setAttribute("name", "hunk");  
  
application.setAttribute("birthday", new Date());  
  
%>  
  
<a href="demo3.jsp">跳转一下看看! </a>  
  
</body>
```

```
</html>
```

demo3:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ page import="java.util.*" %>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>application属性范围</title>
</head>
<body>
<%//从application属性范围中取出属性，并执行向下转型
String username =
(String)application.getAttribute("name");
Date userBirthday =
(Date)application.getAttribute("birthday");
%>
<h2>用户名: <%=username %></h2>
```

```
<h2>生日: <%=userBirthday %></h2>  
</body>  
</html>
```

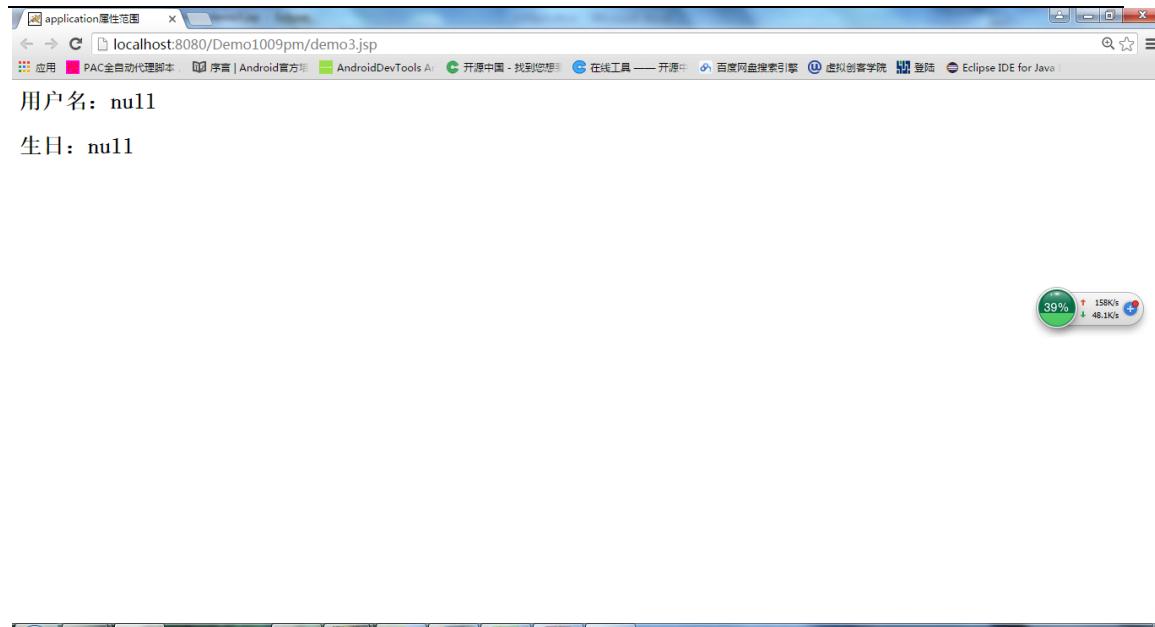
先运行 demo2，跳转之后可以获取到属性，然后关闭浏览器，在浏览器中直接输入：<http://localhost:8080/Demo1009pm/demo3.jsp>

打开发现：



注意：application 属性好，随时打开随时获取，服务器就得付出代价，如果你设置的 application 属性范围过多，那么服务器的负荷就大，必然会影响服务器性能。

我们无论关闭多少次浏览器，发现 application 属性都可以取到，但是，允许你任性，重启一下服务器，然后再输入 <http://localhost:8080/Demo1009pm/demo3.jsp> 你就会发现：



## 4.2.5 深入 page 属性范围

在前面我们通过 `page` 设置属性的时候，是通过 `pageContext` 下面的 `setAttribute (String name, Object value)` ,通过查阅 API 可以发现还有这么一个方法：

```
setAttribute(java.lang.String name,           java.lang.Object value,  
int scope)
```

### **setAttribute**

```
public abstract void setAttribute(java.lang.String name,  
                                java.lang.Object value,  
                                int scope)
```

Register the name and value specified with appropriate scope semantics. If the value passed in is `null`, this has the same effect as calling `removeAttribute( name, scope )`.

#### **Parameters:**

`name` - the name of the attribute to set

value - the object to associate with the name, or null if the attribute is to be removed from the specified scope.

scope - the scope with which to associate the name/object

**Throws:**

java.lang.NullPointerException - if the name is null

IllegalArgumentException - if the scope is invalid

IllegalStateException - if the scope is PageContext.SESSION\_SCOPE but the page that was requested does not participate in a session or the session has been invalidated.

可以看到该方法中多了一个参数，这个整形的参数有四种选择，分别表示四种属性范围，可以通过下面四种整形常量来设置

public static final int PAGE\_SCOPE 表示的 page 属性范围，默认

public static final int REQUEST\_SCOPE 表示的是 request 属性范围

public static final int SESSION\_SCOPE 表示的是 session 属性范围

public static final int APPLICATION\_SCOPE 表示的是 application 属性范围

改装上面的 demo2

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

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

```
pageEncoding="UTF-8"%>

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

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>application属性范围</title>

</head>

<body>

<%//设置application属性范围

pageContext.setAttribute("name", "hunk",
PageContext.APPLICATION_SCOPE);

pageContext.setAttribute("birthday", new
Date(), PageContext.APPLICATION_SCOPE);

%>

<a href="demo3.jsp">跳转一下看看! </a>

</body>

</html>
```

跳转之后可以发现获取到了，并且关闭浏览器后重新打开依然可以获取到。

## 4.3 request 对象

在 web 开发中，request 对象是使用最多的一个对象，其主要作用是接受客户端发送过来的请求。request 是

javax.servlet.http.HttpServletRequest 接口实例化的对象，表示此对象主要是应用在 HTTP 协议上。

Interface HttpServletRequest

All Superinterfaces:

[ServletRequest](#)

HttpServletRequest 接口是 ServletRequest 的子接口，所以在查找 request 对象方法时除了要查询 HttpServletRequest 接口下面方法之外，你还需要查询 ServletRequest 接口下面的方法。

<http://tool.oschina.net/apidocs/apidoc?api=javaEE6>

### 4.3.1 乱码解决

在 web 开发中，使用 request 接受请求的参数是最常见的，但是，在进行参数提交的时候也会存在一些中文乱码的问题。

demo4.html:

```
<!doctype html>

<html>
<head>
<title>登录</title>
</head>
<body>
<form action="demo4.jsp" method="post">
```

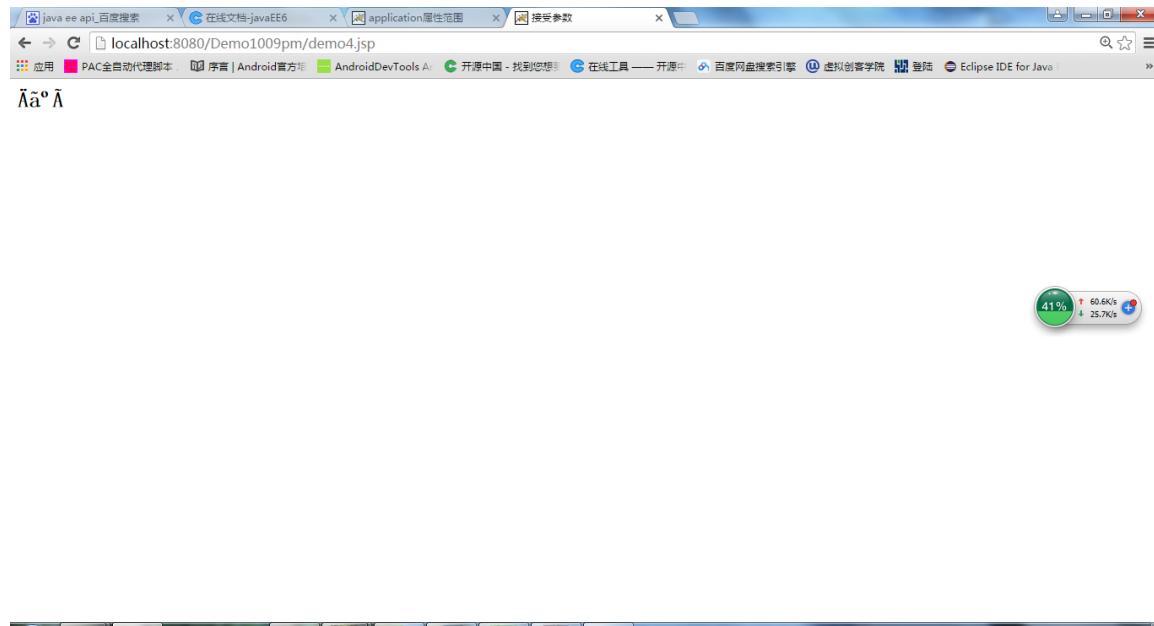
```
username:<input type="text" name="uname">  
<br>  
<input type="submit" value="提交">  
</form>  
</body>  
</html>
```

demo4.jsp:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
<head>  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
pageEncoding="UTF-8"%>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
  
<title>接受参数</title>  
</head>  
  
<%  
  
String username = request.getParameter("uname");  
%>
```

```
<h2><%=username %></h2>  
  
</body>  
  
</html>
```

运行结果：



现在解决乱码问题：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
  
pageEncoding="UTF-8"%>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
  
<title>接受参数</title>
```

```
</head>

<body>

<%
request.setCharacterEncoding("utf-8");

String username = request.getParameter("uname");

%>

<h2><%=username %></h2>

</body>

</html>
```

### 4.3.2 接受请求参数

上面在接受参数的时候是采用 `request.getParameter`, 的方法来接受的，它只能接受一个参数，如果现在是多个参数的话，我们就得采用 `request.getParameterValues` 进行接收。

`demo5.html`

```
<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>登录</title>

</head>

<body>
```

```
<form action="demo5.jsp" method="post">

username:<input type="text" name="uname">

<br>

hobby:<input type="checkbox" name="hobby" value="唱歌">唱歌<br>

<input type="checkbox" name="hobby" value="跳舞">跳舞<br>

<input type="checkbox" name="hobby" value="游泳">游泳<br>

<input type="checkbox" name="hobby" value="足球">足球<br>

<input type="checkbox" name="hobby" value="跑步">跑步<br>

<input type="checkbox" name="hobby" value="睡觉">睡觉<br>

<input type="submit" value="提交">

</form>

</body>

</html>
```

demo5.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

```
<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>接收多个参数</title>

</head>

<body>

<%
request.setCharacterEncoding("utf-8");

String username = request.getParameter("uname");

String hobby[] = request.getParameterValues("hobby");

%>

<h2>姓名: <%=username%></h2>

<h2>兴趣:<br/>

<%
for(int i = 0; i<hobby.length; i++){

%>

<%=hobby[i] %>
```

```
<%  
  
}  
  
%>  
  
</h2>  
  
</body>  
  
</html>
```

运行结果：



姓名：何王科

兴趣：游泳 跑步



为了防止用户不填写任何数据而造成的空指针异常，所以我们在循环输出之前要先判断一下数组是否为空。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"
```

```
pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>接收多个参数</title>

</head>

<body>

<%
    request.setCharacterEncoding("utf-8");

    String id = request.getParameter("id");

    String username = request.getParameter("uname");

    String hobby[] =
request.getParameterValues("hobby");

%>

<h2>id:<%=id %></h2>

<h2>

    姓名: <%=username%></h2>

<%
    if (hobby != null) {
%>

<h2>

    兴趣:

    <%
```

在接收客户端提交的参数的时候，不一定是由表单提交过来的，还有可能是使用地址重写的方式进行提交的。

## URL 重写的格式:

动态的页面地址？参数名称 1=参数值 1&参数名称 2=参数值 2&.....

http://localhost:8080/Demo1009pm/demo5.jsp?uname=%E4%BD%95%E7%8E%8B%E7%A7%91&hobby=%E5%94%B1%E6%AD%8C&hobby=%E8%B7%B3%E8%88%9E&id=3

```
request.getParameterValues
```

```
request.getParameter
```

```
request.getParameter
```

```
request.getParameterValues
```

在 request 内置对象里面 `request.getParameterNames`, 返回的是参数名。返回值类型是枚举。`hasMoreElements()`来判断是否有内容, 取得内容的方法 `nextElement()`;

demo:

```
<!doctype html>

<html>
  <head>
    <meta charset="utf-8">
    <title>getParamateNames</title>
  </head>
  <body>
    <form action="demo1.jsp" method="post">
      姓名: <input type="text" name="uname"><br> 性别:
      <input type="radio" name="sex" value="男" checked>男
      <input type="radio" name="sex" value="女">女<br> 城市:
      <select>
```

```
        name="city">

            <option value="贵阳">贵阳</option>

            <option value="遵义">遵义</option>

            <option value="六盘水">六盘水</option>

        </select> <br> 兴趣: <input type="checkbox"
name="**inst" value="唱歌">唱歌

            <input type="checkbox" name="**inst" value="跳舞"
">跳舞 <input

            type="checkbox" name="**inst" value="足球">足球

<input

            type="checkbox" name="**inst" value="篮球">篮球

<input

            type="checkbox" name="**inst" value="乒乓球">

乒乓球<br> 自我介绍:

            <textarea rows="3" cols="30"

name="note"></textarea>

            <br> <input type="hidden" name="uid" value="1">

<input

            type="reset" value="重置"> <input type="submit"

value="提交">

        </form>

</body>
```

```
</html>
```

demo1.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<%@page import="java.util.Enumeration"%>

<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>getParamateNames</title>

</head>

<body>

<%
    request.setCharacterEncoding("utf-8");//设置统一
编码
%>

<center>

<table border="1">

<tr>
```

```
<td>参数名</td>

<td>参数值</td>

</tr>

<%
Enumeration enu =
request.getParameterNames();

while (enu.hasMoreElements()) {

    String paramName = (String)
enu.nextElement();

%>

<tr>

<td><%=paramName%></td>

<td>

<%
if (paramName.startsWith("**")) {

    String paramValues[] = request

.getParameterValues(paramName);

    for (int i = 0; i <
paramValues.length; i++) {

        %> <%=paramValues[i]%> <%
    }
}
```

```
    } else {

        String paramValue =
request.getParameter(paramName);

%> <%=paramValue%> <%
    }

%>

        </td>

    </tr>

    <%

    %

%>

    <tr>

        <td></td>

        <td></td>

    </tr>

    </table>

</center>

</body>

</html>
```

### 4.3.3 显示全部的头信息

Java 的 web 开发使用的是 HTTP 协议，主要操作的是请求和相应，在请求和相应的同事也会包含一些其他信息，客户端的 IP、Cookie、语言，这些信息就称为头信息。

获取头信息的参数名是 `getHeaderNames()`, 取出其对应的内容采取的方法是: `getHeader()`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<%@page import="java.util.Enumeration"%>

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<%
Enumeration enu = request.getHeaderNames();
```

```
while (enu.hasMoreElements()) {  
  
    String headerName = (String) enu.nextElement();  
  
    String headerValue =  
  
request.getHeader(headerName);  
  
%>  
  
<h2><%=headerName%>-----<%=headerValue%></h2>  
  
<%  
  
}  
  
%>  
  
</body>  
  
</html>
```

demo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
  
pageEncoding="UTF-8"%>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8" />
```

```
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<%

    String method = request.getMethod();

    String ip = request.getRemoteAddr();

    String path = request.getServletPath();

    String contextPath = request.getContextPath();

%>

<h2>

    请求方法: <%=method%></h2>

<h2>

    IP地址: <%=ip%></h2>

<h2>

    访问路径: <%=path%></h2>

<h2>

    上下文名称: <%=contextPath%></h2>

</body>

</html>
```

## 4.4 response 对象

`response.addCookie(arg0);`向客户端增加Cookie

`response.setHeader(arg0, arg1);`设置响应的头信息

`response.sendRedirect(arg0);`页面跳转

### 4.4.1 设置头信息

页面定时刷新

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
  
pageEncoding="UTF-8"%>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
  
<title>在此处插入标题</title>  
  
</head>  
  
<body>  
  
<%!int count = 0;%>  
  
<%  
  
    response.setHeader("refresh", "1");
```

```
%>

<h2>

    已经刷新了 <%=count++%> 次

</h2>

</body>

</html>
```

3 秒钟后自动跳转至 demo1.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<h3>3秒钟后自动跳转至demo1.html, 如果没有跳转, 请点击这里! </h3>

<%
```

```
response.setHeader("refresh", "3;URL = demo1.html");  
%>  
</body>  
</html>
```

注意：该跳转属于客户端跳转。

在 HTML 文件中我们通过

```
<meta http-equiv="refresh" content="3;URL=demo2.html">
```

设置了自动跳转。也是客户端跳转。

#### 4.4.2 页面跳转

demo

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
  
pageEncoding="UTF-8"%>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
  
<title>在此处插入标题</title>  
  
</head>
```

```
<body>

<%
    response.sendRedirect("demo1.html"); //客户端直接跳转
%>

</body>

</html>
```

那么问题来了：

<jsp:forward>服务器端跳转

```
<%
System.out.println("=====跳转之前的代码
=====");
%>

<jsp:forward page="demo2.html"/>

<%
System.out.println("-----跳转之后的代码
-----");
%>
```

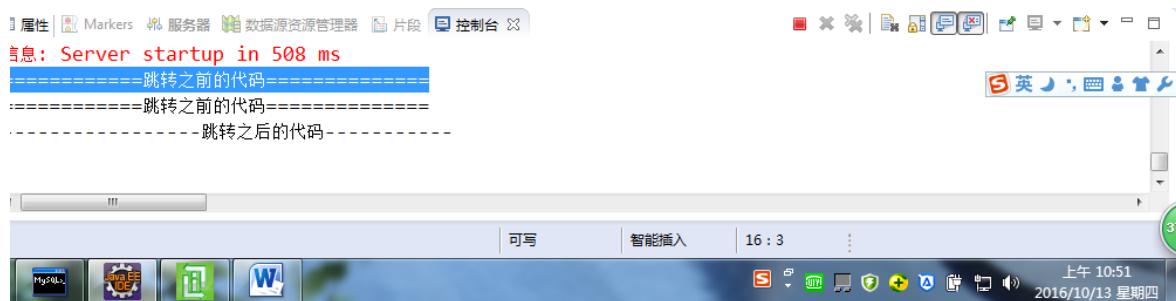
response.sendRedirect("demo1.html"); 客户端跳转

```
<%
System.out.println("=====跳转之前的代码
=====");
%>
```

```
%>
<% response.sendRedirect("demo2.html"); %>

<%
System.out.println("-----跳转之后的代码
-----");

%>
```



比如你在页面执行是数据库连接，按照一贯的方法，在执行完数据库的一系列操作之后，我们要关闭一堆连接信息。如果你采取的是服务器端跳转，那么你必须在跳转之前把连接关闭。

#### 4.4.3 操作 Cookie

Cookie 是浏览器提供的一种技术，Cookie 里面的值是服务器端保存的。所以安全性就很差。

在 JSP 中提供了 `javax.servlet.http.Cookie` 操作类，此类的构造方法是：

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

常用的普通方法有：

<code>java.lang.String</code> <code>getName()</code>	Returns the name of the cookie.
<code>java.lang.String</code> <code>getValue()</code>	

	Gets the current value of this Cookie.
void <a href="#">setMaxAge</a> (int expiry)	Sets the maximum age in seconds for this Cookie.

所有的 cookie 都是有服务器端设置到客户端上的。所以要想客户端增加 cookie，必须调用 response 对象下面的 addCookie ()；

demo：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<%
Cookie c1 = new Cookie("name", "hunk");
Cookie c2 = new Cookie("password", "123456");

```

```
response.addCookie(c1);  
  
response.addCookie(c2);  
  
%>  
  
</body>  
  
</html>
```

上述代码是向客户端设置了两个 cookie 对象，如果想取得客户端设置的 cookie 对象，可以通过 request 对象完成。方法：

Cookie[] getCookies()

Returns an array containing all of the Cookie objects the client sent with this request.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
  
pageEncoding="UTF-8"%>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
  
<title>在此处插入标题</title>
```

```
</head>

<body>

<%

Cookie c[] = request.getCookies(); //取得了服务端的所有
cookie

for(int i = 0; i < c.length; i++){

%>

<h2><%=c[i].getName() %>----<%=c[i].getValue() %></
h2>

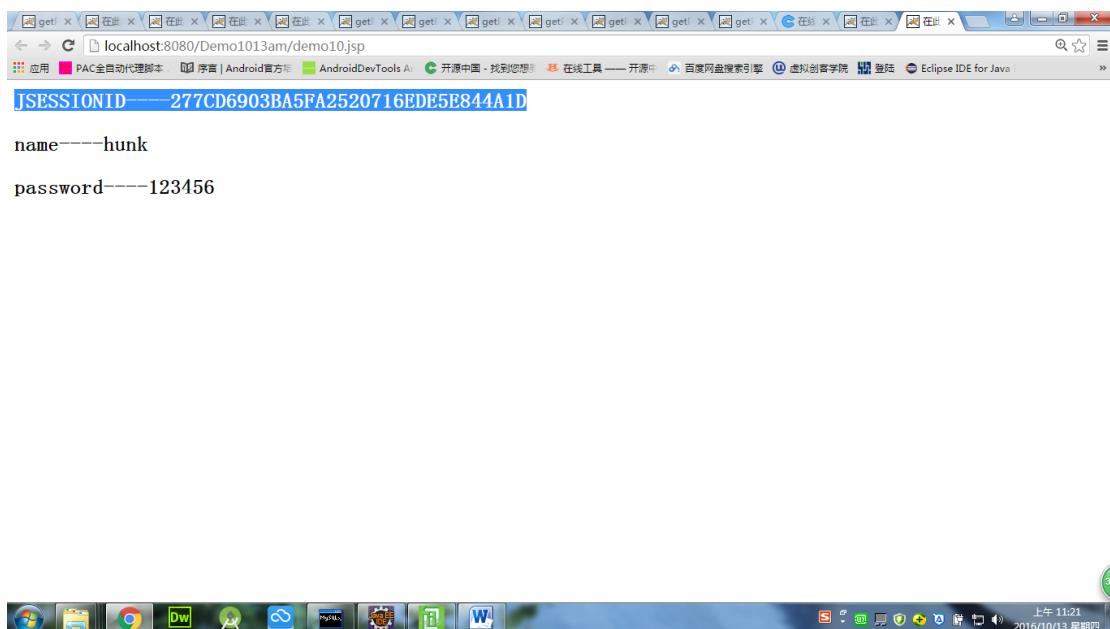
<%
}

%>

</body>

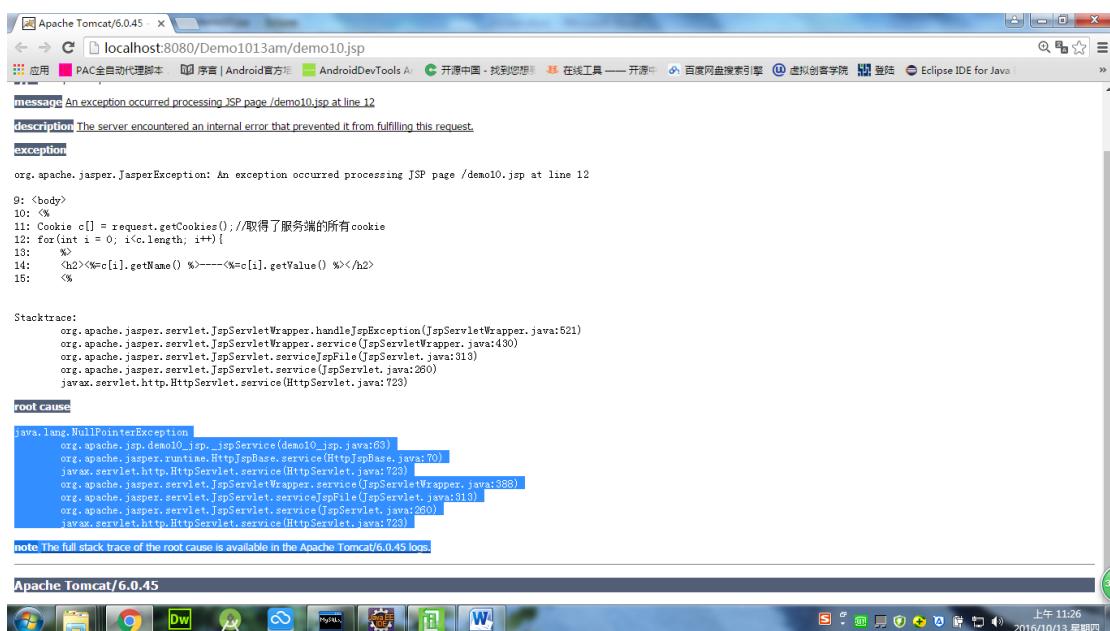
</html>
```

运行结果：



图中的 JSESSIONID 是什么？

系统自动设置的。在每一个用户访问服务器的时候，服务器为了区分每一个客户端，都会自动设置一个 JSESSIONID 的 Cookie，表示用户的唯一标示。



理论上讲，我们给客户端设置了一个 cookie 之后，即使关闭了浏览器，

我们也应该能够取得我们设置的 cookie。现在关闭了浏览器，却发现没有取得 cookie，而是出现了空指针异常，说明，在之前设置的两个 cookie 并没有真正保存在客户端上。而只是保存在了客户端的浏览器上，当关闭浏览器后，cookie 就不在了，所以说你取到的是 null。此时如果你想真正的将 cookie 保存在客户端上，就必须设置 cookie 的保存时间，使用的是 setMaxAge().

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
  
pageEncoding="UTF-8"%>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
  
<title>在此处插入标题</title>  
  
</head>  
  
<body>  
  
<%  
  
Cookie c1 = new Cookie("name", "hunk");  
  
Cookie c2 = new Cookie("password", "123456");  
  
c1.setMaxAge(60);
```

```
c2.setMaxAge(60);  
  
response.addCookie(c1);  
response.addCookie(c2);  
  
%>  
</body>  
</html>
```

关闭浏览器，在 60 秒之内打开，发现 cookie 还在，60 秒之后 cookie 就不在。

虽然 Cookie 中可以保存信息，但是并不能无限制的保存，一般一个客户端最多只能保存 300 个 Cookie，所以数据量太大的时候将无法使用 Cookie。

也可也通过设置头信息的方式为客户端增加 cookie，在开发中并不常见，了解即可。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"
```

```

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<%
response.setHeader("Set-Cookie", "name=hunk");

%>

</body>

</html>

```

## 4.5 session 对象

在实际开发中 `session` 对象主要的作用是完成用户的登录、注册等常见的功能，每一个 `session` 对象都表示不同的用户访问。

java.lang.String <code>getId()</code>	Returns a string containing the unique identifier assigned to this session.
long <code>getCreationTime()</code>	Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
long <code>getLastAccessedTime()</code>	Returns the last time the client sent a request

		associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT, and marked by the time the container received the request.
	boolean	<a href="#">isNew()</a> Returns true if the client does not yet know about the session or if the client chooses not to join the session.
	void	<a href="#">invalidate()</a> Invalidates this session then unbinds any objects bound to it.
java.util.Enumeration<java.lang.String>		<a href="#">getAttributeNames()</a> Returns an Enumeration of String objects containing the names of all the objects bound to this session.

在 `HttpSession` 接口中最重要的还是属性的操作，主要是可以完成用户登录的合法性验证。

#### 4.5.1 取得 Session id

当一个用户连接到服务器之后，服务器就自动为此 session 分配一个不会重复的 session id，服务器依靠这些不同的 session id 来区分每一个不同的用户，在 Web 中我依靠是 `getId()` 来取得 id。

demo

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<html>
<head>
```

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<%
String id = session.getId();

%>

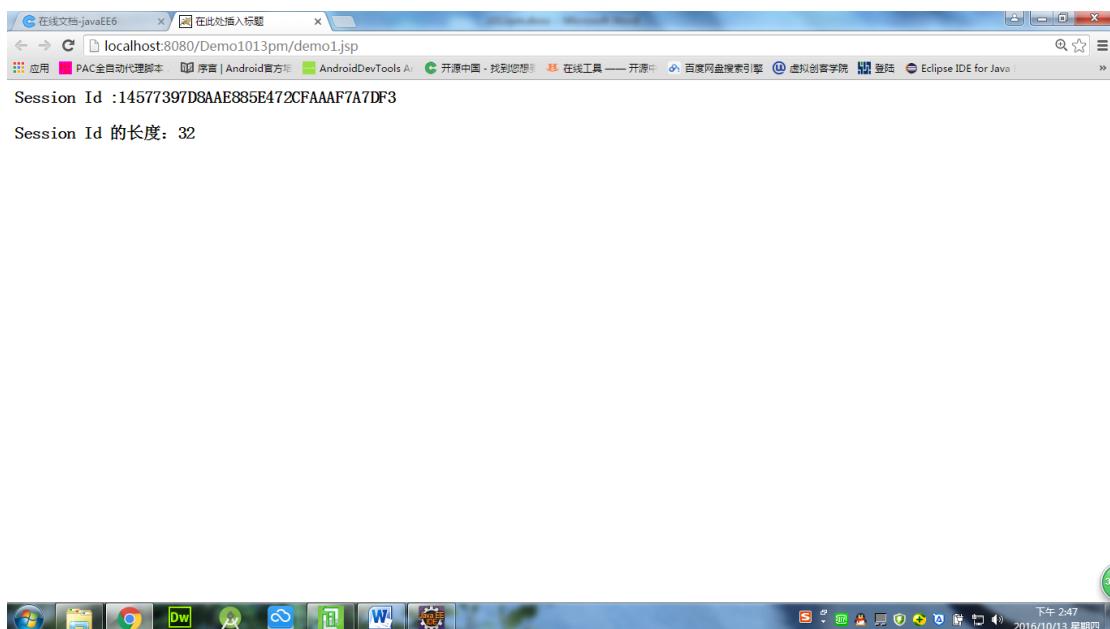
<h3>Session Id :<%=id %></h3>

<h3>Session Id 的长度: <%=id.length() %></h3>

</body>

</html>
```

运行结果：



问题：这个 SessionID 的长度和我们 Cookie 的很相似？

在前面操作 cookie 的时候，有一个 JSESSIONID 的 cookie，它的值是否和我们这里的 Session ID 一样？

两者是一样的。Session 使用的是 cookie 机制。

在使用 Session 的时候注意，对已每一个已经连接到服务器上的用户，如果重新启动服务器，则这些用户再次发出请求实际上表示的是一个新连接的用户，服务器会为每一个用户重新分配新的 session id。

可以实现，需要配置 tomcat 下面的 server.xml，在服务器关闭的时候，把 session id 序列化存储在数据库、文件中，当服务器重新启动的时候，执行反序列化操作。

```
<Context path="/gzu" docBase="D:\jsp">

<Manager className="org.apache.catalina.session.PersistentManager">
    debug=0
    saveOnRestart="true"
```

```
maxActiveSession="-1"  
  
minIdleSwap="-1"  
  
maxIdleSwap="-1"  
  
maxIdleBackup="-1"  
  
<Store           className="org.apache.catalina.session.FileStore"  
directory="d:\temp"/>  
  
</Manager>  
  
</Context>
```

#### 4.5.2 登录及注销

实现思路：当用户登录成功之后，设置一个 session 范围的属性，然后在其他需要验证的页面中判断此 session 是否存在，如果存在，则是合法的已登录用户。如果不存在，则给出提示，并且跳转至用户登录页面，用户登录成功之后可以注销。

分析：`login.jsp`：完成登录的表单显示，同时向页面本身提交数据，以完成登录的验证。如果登录成功，则保存属性，如果登录失败，则显示登录失败信息。

`welcome.jsp`：要求在用户登录后才可以显示登录成功的信息，如果没有登录，则要给出未登录的提示，同时给出一个登录的链接地址。

`logout.jsp`：完成登录的注销，注销后，页面要跳转至 `login.jsp` 页面，等待用户继续登录。

`login:`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>登录</title>
</head>
<body>
<form action="Login.jsp" method="post">
username:<input type="text" name="uname"><br>
password:<input type="password" name="psw"><br>
<input type="reset" value="重置">
<input type="submit" value="登录">
</form>
<%
String name = request.getParameter("uname");
String psw = request.getParameter("psw");
if(!(name==null||"".equals(name)||psw==null||"".equal
```

```
s(PSW)){\n\n    if("hunk".equals(name)&&"123456".equals(PSW)){\n\n        response.setHeader("refresh",\n"3;URL=welcome.jsp");\n\n        session.setAttribute("username", name);\n\n    %>\n\n        <h2>用户登录成功，三秒种之后跳转至欢迎页！</h2>\n\n        <h3>如果没有跳转，请点击<a href="welcome.jsp">这里\n</a></h3>\n\n    <%\n\n    }else{\n\n        %>\n\n        <h2>用户名或密码错误！登录失败！</h2>\n\n        <%\n\n    }\n\n    %>\n\n</body>\n</html>
```

welcome

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
```

```
Transitional//EN">

<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>欢迎页</title>

</head>

<body>

<%
if(session.getAttribute("username")!=null){

%>

<h2>欢迎<%=session.getAttribute("username") %>光临本
系统, <a href="Logout.jsp">注销</a></h2>

<%
}else{

%>

<h3>请先进行<a href="Login.jsp">登录</a></h3>

<%
}
```

```
%>

</body>

</html>
```

logout

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>注销</title>
</head>
<body>
<%
response.setHeader("refresh", "2;URL=login.jsp");
session.invalidate(); //是session失效;
%>
<h2>您已经退出本系统，两秒种后将自动转回登录页！</h2>
<h3>如果没有跳转，请点击<a href="Login.jsp">这里</a></h3>
```

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

#### 4.5.3 判断新用户

```
<%  
  
if(session.isNew()){  
  
%>  
  
<h3>欢迎新用户！ </h3>  
  
<%  
  
}else{  
  
%>  
  
<h2>欢迎您再次光临本系统！ </h2>  
  
<%  
  
}  
  
%>
```

#### 4.5.4 取得用户的操作时间

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html";
```

```
charset=UTF-8"

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>取得用户操作时间</title>

</head>

<body>

<%
long start = session.getCreationTime();

long end = session.getLastAccessedTime();

long time = (end-start)/1000;

%>

<h2>您已经在本网站停留了<%=time %>秒！</h2>

</body>

</html>
```

## 4.6 application 对象

实例化的是 javax.servlet.ServletContext 的接口。

java.lang.String	<a href="#">getRealPath</a> (java.lang.String path) Gets the <i>real</i> path corresponding to the given <i>virtual</i> path.
java.util.Enumeration<java.lang.Str	<a href="#">getAttributeNames</a> ()

ing>	Returns an Enumeration containing the attribute names available within this ServletContext.
java.lang.String	<a href="#">getContextPath()</a> Returns the context path of the web application.

#### 4.6.1 取得虚拟目录对应的绝对路径

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>取得虚拟路径的绝对目录</title>
</head>
<body>
<%
//String path = application.getRealPath("/");
String path =
this.getServletContext().getRealPath("/");
```

```
%>

<h3>真实路径是: <%=path %></h3>

</body>

</html>
```

## 4.6.2 网站计数器

分析:

网站的访问人数，很大

用户在每一次访问时都得进行判断是不是新用户，只有新用户才计数  
在进行更改、保存的需要同步操作。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

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

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
```

```
charset=UTF-8">

<title>计数器</title>

</head>

<body>

<%!
    BigInteger count = null ;
%>

<%! // 为了开发简便，将所有的操作定义在方法之中，所有的异常直接加入完整的try...catch处理

public BigInteger load(File file){
    BigInteger count = null ; // 接收数据
    try{
        if(file.exists()){
            Scanner scan = new Scanner(new
FileInputStream(file)) ;
            if(scan.hasNext()){
                count = new BigInteger(scan.next()) ;
            }
            scan.close() ;
        } else { // 应该保存一个新的，从0开始
            count = new BigInteger("0") ;
            save(file,count) ; // 保存一个新的文件
        }
    }
}
```

```
        }

    }catch(Exception e){

        e.printStackTrace() ;

    }

    return count ;

}

public void save(File file,BigInteger count){

    try{

        PrintStream ps = null ;

        ps = new PrintStream(new

FileOutputStream(file)) ;

        ps.println(count) ;

        ps.close() ;

    }catch(Exception e){

        e.printStackTrace() ;

    }

}

%>

<%

String fileName =

this.getServletContext().getRealPath("/") +

"count.txt"; // 这里面保存所有的计数的结果
```

```
File file = new File(fileName) ;  
  
if(session.isNew()){  
  
    synchronized(this){  
  
        count = load(file) ; // 读取  
  
        //count++;  
  
        count = count.add(new BigInteger("1")) ; // 在  
原本的基础上增加1。  
  
        save(file,count) ;  
  
    }  
  
}  
  
%>  
  
<h2>您是第<%=count==null?0:count%>位访客！</h2>  
  
</body>  
  
</html>
```

## 4.7 config 对象

### 4.7.1 Web 安全性

在 WEB-INF 文件夹下边新建一个 jsp 文件

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>
```

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

呵呵， 在安全， 我也进来了！

</body>

</html>
```

然后修改一下 web.xml 配置文件

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"

id="WebApp_ID" version="2.5">

<display-name>Demo1013pm</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>
```

```
<welcome-file>index.htm</welcome-file>

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

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

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

</welcome-file-list>

<servlet>

    <servlet-name>he</servlet-name>

    <jsp-file>/WEB-INF/NewFile.jsp</jsp-file>

</servlet>

<servlet-mapping>

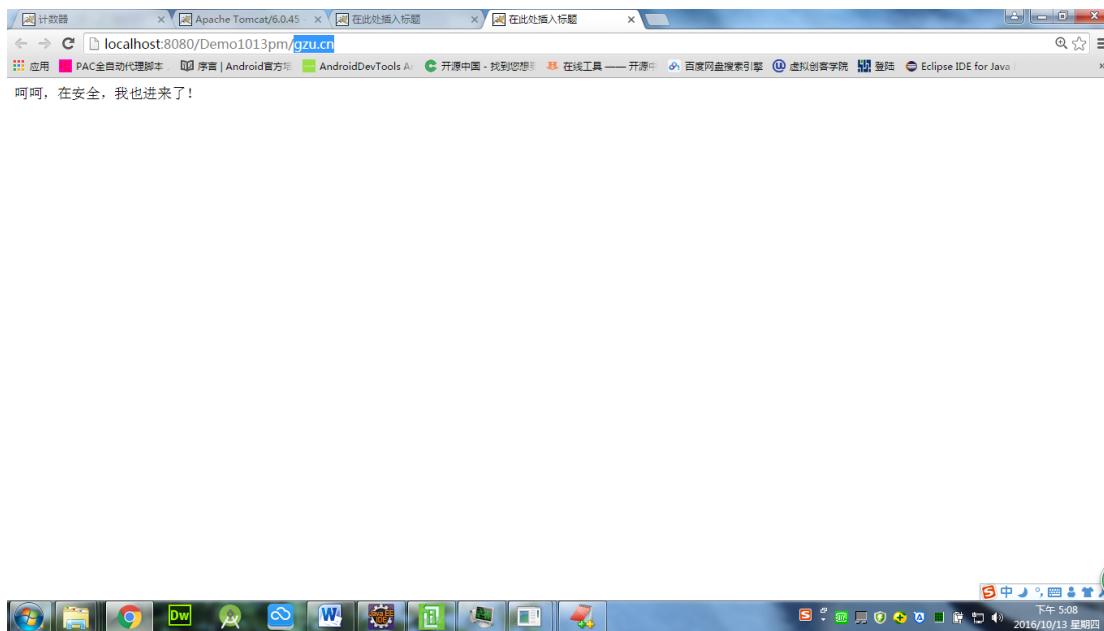
    <servlet-name>he</servlet-name>

    <url-pattern>/gzu.cn</url-pattern>

</servlet-mapping>

</web-app>
```

运行结果：



## 4.7.2 config 对象

java.lang.String <u><a href="#">getInitParameter</a></u> (java.lang.String name)	Gets the value of the initialization parameter with the given name.
java.util.Enumeration<java.lang.String>	<u><a href="#">getInitParameterNames</a></u> () Returns the names of the servlet's initialization parameters as an Enumerationof String objects, or an empty Enumeration if the servlet has no initialization parameters

所有的初始化参数必须在 web.xml 中进行配置。

在 WEB-INF 中新建一个 demo6: demo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

```
<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<%
String dbDriver = config.getInitParameter("driver");

%>

<h3>驱动程序: <%=dbDriver %></h3>

</body>

</html>
```

然后修改 web.xml 配置文件

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/java
```

```
ee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"

    id="WebApp_ID" version="2.5">

        <display-name>Demo1013pm</display-name>

        <welcome-file-list>

            <welcome-file>index.html</welcome-file>

            <welcome-file>index.htm</welcome-file>

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

            <welcome-file>default.html</welcome-file>

            <welcome-file>default.htm</welcome-file>

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

        </welcome-file-list>

        <servlet>

            <servlet-name>demo6</servlet-name>

            <jsp-file>/WEB-INF/demo6.jsp</jsp-file>

            <init-param>

                <param-name>driver</param-name>

                <param-value>org.gjt.mm.mysql.Driver</param-value>

            </init-param>

        </servlet>

        <servlet-mapping>
```

```
<servlet-name>demo6</servlet-name>

<url-pattern>/gzu.driver</url-pattern>

</servlet-mapping>

<servlet>

    <servlet-name>he</servlet-name>

    <jsp-file>/WEB-INF/NewFile.jsp</jsp-file>

</servlet>

<servlet-mapping>

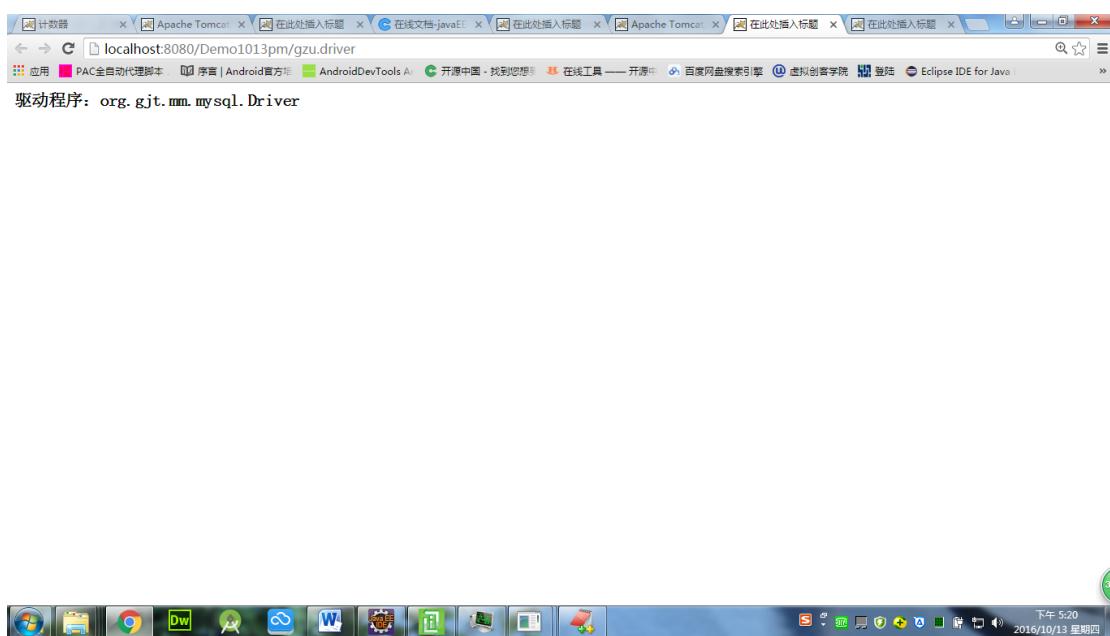
    <servlet-name>he</servlet-name>

    <url-pattern>/gzu.cn</url-pattern>

</servlet-mapping>

</web-app>
```

运行结果：



## 4.8 out 对象

javax.servlet.jsp.JspWriter 接口的实例化对象。

int	<a href="#">getBufferSize()</a>
-----	---------------------------------

This method returns the size of the buffer used by the JspWriter.

返回 JSP 中缓冲区的大小

abstract int	<a href="#">getRemaining()</a>
--------------	--------------------------------

This method returns the number of unused bytes in the buffer.

返回 JSP 中未使用的缓冲区大小

demo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<%
int buffer = out.getBufferSize();
```

```

int available = out.getRemaining();

int use = buffer-available;

%>

<h3>使用中的缓冲区大小是: <%=use %></h3>

</body>

</html>

```

## 4.9 pageContext 对象

javax. servlet. jsp. PageContext 接口的实例。

abstract void	<a href="#">forward</a> (java.lang.String relativeUrlPath)
	<p>This method is used to re-direct, or "forward" the current ServletRequest and ServletResponse to another active component in the application.</p>
abstract void	<a href="#">include</a> (java.lang.String relativeUrlPath)
	<p>Causes the resource specified to be processed as part of the current ServletRequest and ServletResponse being processed by the calling Thread.</p>
abstract <a href="#">ServletConfig</a>	<a href="#">getServletConfig</a> () The ServletConfig instance.
abstract <a href="#">ServletContext</a>	<a href="#">getServletContext</a> () The ServletContext instance.
abstract <a href="#">HttpSession</a>	<a href="#">getSession</a> () The current value of the session object (an HttpSession).
abstract <a href="#">ServletRequest</a>	<a href="#">getRequest</a> () The current value of the request object (a ServletRequest).

abstract <a href="#">ServletResponse</a>	<a href="#">getResponse()</a>
	<a href="#">e</a>
	The current value of the response object (a <a href="#">ServletResponse</a> ).

demo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<%
pageContext.forward("demo8.jsp?name=hunk");
%>
</body>
</html>
```

demo8.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<%
String name =
pageContext.getRequest().getParameter("name");//相当于
是调用了request. getParameter("name")
%>
<%=name %>
<%=pageContext.getServletContext().getRealPath("/") %
>//相当于是我们的application对象
</body>
</html>
```

作业：使用 JSP 完成一个购物车程序的开发。（session）

JavaBean

## 5 JavaBean

### 5.1 JavaBean 简介

JavaBean 是 java 语言开发的一个可以重复使用的组件，在 JSP 中可以使用 JavaBean 减少重复代码。JSP+JavaBean 开发优点：

- 1、使 Java 代码与 HTML 代码实现了分离。日后维护方便，代码看起来舒服。
- 2、减少了代码的重复度。

如果说你在 JSP 中要使用 JavaBean，那你 JavaBean 必须满足以下几点：

- 1、类必须打包。
- 2、类必须声明为 public class
- 3、类中所有的属性必须封装， private
- 4、类中被封装的属性必须提供 getter、setter 方法
- 5、一个 javaBean 中至少应该还有一个无参构造。

demo:

```
package edu.gzu.vo;

public class SimpleBean {
    private String name;
    private int age;

    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;  
}  
  
}
```

如果一个类中只包含了属性以及相应的 getter、setter 方法，那这种类就叫做 JavaBean

此外还有别称

POJO(Plain Ordinary Java Objects) 简单 Java 对象

VO (Value Object)

TO (Transfers Object) 传输对象。

## 5.2 在 JSP 中使用 JavaBean

### 5.2.1 使用 JSP 的 page 指令导入所需要的 JavaBean

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<%@page import="edu.gzu.vo.*"%>

<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<%
SimpleBean sb = new SimpleBean();
sb.setName("hunk");
sb.setAge(30);

%>

<h2>name:<%=sb.getName() %></h2>
```

```
<h2>age:<%=sb.getAge() %></h2>  
</body>  
</html>
```

## 5.2.2 使用<jsp:useBean>指令

语法：

```
<jsp:useBean id="实例化对象的名称" scope="保存范围" class="包.类名"/>
```

这里采取的就是反射机制。

demo

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<jsp:useBean id="sb" scope="page"  
class="edu.gzu.vo.SimpleBean"/>  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
  
pageEncoding="UTF-8"%>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
  
<title>在此处插入标题</title>
```

```
</head>

<body>

<%

sb.setName("hunk");

sb.setAge(30);

%>

<h2>name:<%=sb.getName() %></h2>

<h2>age:<%=sb.getAge() %></h2>

</body>

</html>
```

可以发现使用`<jsp:useBean>`不需要我们手动的去实例化对象，直接根据 `id` 值来调用对象里面的方法。

每次修改 `JavaBean` 都需要重新启动服务器，如果你不想一直重新启动服务器，可以这样做：使用自动加载（自己下去查）

用到的一个属性是 `reloadable=“true”`

## 5.3 JavaBean 与表单

定义表单：

```
<!doctype html>

<html>

<head>

<meta charset=UTF-8/>
```

```
<title>表单</title>

</head>

<body>

<form action="demo4.jsp" method="post">

name:<input type="text" name="uname"><br>

age:<input type="text" name="age"><br>

<input type="submit" value="submit">

</form>

</body>

</html>
```

接受数据：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<%@page import="edu.gzu.vo.SimpleBean"%>

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
```

```
<title>接受表单数据</title>

</head>

<body>

<%

request.setCharacterEncoding("utf-8");

SimpleBean sb = new SimpleBean();

sb.setName(request.getParameter("uname"));

sb.setAge(Integer.parseInt(request.getParameter("age"

)));

%>

<h2>name:<%=sb.getName() %></h2>

<h2>age:<%=sb.getAge() %></h2>

</body>

</html>
```

上面使用了 `request.getParameter` 来接受表单提交过来的参数，

如果说现在提交的参数有 100 个，你怎么办？

把上面的代码进行一下修改：

```
<!doctype html>

<html>

<head>

<meta charset=UTF-8/>

<title>表单</title>
```

```
</head>

<body>

<form action="demo5.jsp" method="post">

    name:<input type="text" name="name"><br>//javaBean中的
    属性

    age:<input type="text" name="age"><br>

    <input type="submit" value="submit">

</form>

</body>

</html>
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<%
    request.setCharacterEncoding("utf-8");
%>

<jsp:useBean id="sb" scope="page"
class="edu.gzu.vo.SimpleBean" />
<jsp:setProperty property="*" name="sb"/>//name是我们实
例化对象的名字

<html>

<head>
```

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>接受表单数据</title>

</head>

<body>

    <h2>

        name:<%=sb.getName()%></h2>

    <h2>

        age:<%=sb.getAge()%></h2>

</body>

</html>
```

## 5.4 设置属性<jsp:setProperty>

上面发现使用<jsp:setProperty>操作属性的时候，通过“\*”的形式来完成属性自动设置。<jsp:setProperty>一共有 4 中使用方法：

- 1、 自动匹配，
- 2、 指定属性，

```
<jsp:setProperty name="(id)" property="属性名称">
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<%
    request.setCharacterEncoding("utf-8");
%>

<jsp:useBean id="sb" scope="page"
class="edu.gzu.vo.SimpleBean" />
<jsp:setProperty property="name" name="sb"/>

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type"
content="text/html; charset=UTF-8">
<title>接受表单数据</title>
</head>
<body>

<h2>
    name:<%=sb.getName()%></h2>
```

```
<h2>  
    age:<%=sb.getAge()%></h2>  
</body>  
</html>
```

### 3、指定参数

```
<jsp:setProperty name="(id)" property="属性名称" param="参数  
名称">  
  
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<%  
    request.setCharacterEncoding("utf-8");  
%>  
  
<jsp:useBean id="sb" scope="page"  
class="edu.gzu.vo.SimpleBean" />  
  
<jsp:setProperty property="name" name="sb"  
param="age"/>  
  
<jsp:setProperty property="age" name="sb"  
param="name"/>  
  
<html>  
<head>  
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
```

```
charset=UTF-8"

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type"
content="text/html; charset=UTF-8">

<title>接受表单数据</title>

</head>

<body>

<h2>

    name:<%=sb.getName()%></h2>

    <h2>

        age:<%=sb.getAge()%></h2>

    </body>

</html>
```

#### 4、指定内容

```
<jsp:setProperty name="(id)" property="属性名称" value="内
容">

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<%
    request.setCharacterEncoding("utf-8");
    int age = 21;
```

```
%>

<jsp:useBean id="sb" scope="page"
  class="edu.gzu.vo.SimpleBean" />

<jsp:setProperty property="name" name="sb"
  value="hunk"/>

<jsp:setProperty property="age" name="sb"
  value="<%=@age %>" />

<html>
<head>
<%@ page language="java" contentType="text/html;
  charset=UTF-8"
  pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type"
  content="text/html; charset=UTF-8">
<title>接受表单数据</title>
</head>
<body>

<h2>
  name:<%=sb.getName()%></h2>
<h2>
```

```
age:<%=sb.getAge()%></h2>  
</body>  
</html>
```

上面四种设置属性的方法，第一种自动匹配使用的最多，务必掌握。  
其他了解。

## 5.5 取得属性<jsp:getProperty>

使用<jsp:getProperty>会自动调用 JavaBean 中的 getter 方法。

语法：

```
<jsp:getProperty name="(id)" property="属性名称">
```

demo：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
<%  
    request.setCharacterEncoding("utf-8");  
%>  
<jsp:useBean id="sb" scope="page"  
class="edu.gzu.vo.SimpleBean" />  
<jsp:setProperty property="*" name="sb" />  
<html>  
<head>  
<%@ page language="java" contentType="text/html";
```

```
charset=UTF-8"

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>接受表单数据</title>

</head>

<body>

<h2>

    name:<jsp:getProperty property="name"
name="sb"/></h2>

<h2>

    age:<jsp:getProperty property="age"
name="sb"/></h2>

</body>

</html>
```

## 5.6 JavaBean 的保存范围

<jsp:useBean>注意到有一个 scope 属性，表示的是一个 JavaBean 的保存范围，保存范围还是四种：

page

request

session

application

下面写一个 JavaBean

demo:

```
package edu.gzu.vo;

public class Count {
    private int count =0;

    public Count(){
        System.out.println("=====一个新的对象
产生=====");
    }

    public int getCount() {
        return ++this.count;
    }

    public void setCount(int count) {
        this.count = count;
    }
}
```

{}

### 5.6.1 page 范围内的 JavaBean

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<jsp:useBean id="count" scope="page"
class="edu.gzu.vo.Count"/>

<html>
<head>

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

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>
<body>

<h2>第<jsp:getProperty property="count" name="count"/>
次访问</h2>

<jsp:forward page="demo7.jsp"/>
```

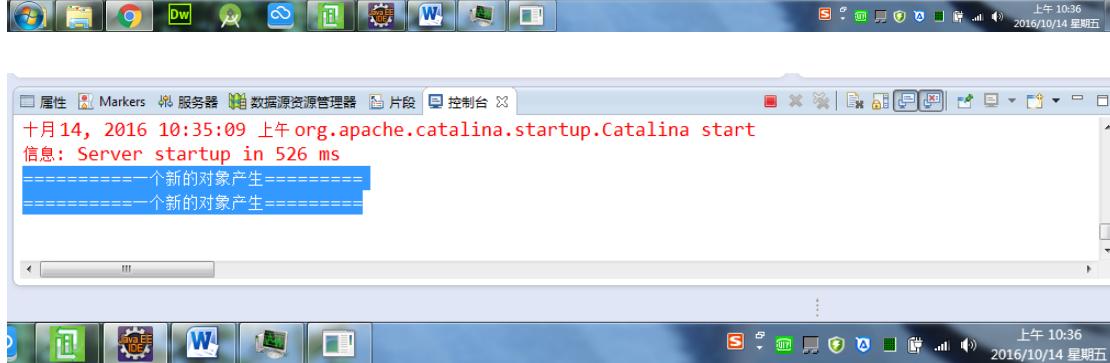
```
</body>  
  
</html>
```

demo7

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<jsp:useBean id="count" scope="page"  
class="edu.gzu.vo.Count"/>  
  
  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
  
pageEncoding="UTF-8"%>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
  
<title>在此处插入标题</title>  
  
</head>  
  
<body>  
  
<h2>第<jsp:getProperty property="count" name="count"/>  
次访问</h2>
```

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

运行结果：



## 5.6.2 request 范围的 JavaBean

demo6:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<jsp:useBean id="count" scope="request"
```

```
class="edu.gzu.vo.Count"/>

<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<h2>第<jsp:getProperty property="count" name="count"/>
次访问</h2>

<jsp:forward page="demo7.jsp"/>

</body>

</html>
```

demo7:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<jsp:useBean id="count" scope="request"
class="edu.gzu.vo.Count"/>
```

```
<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

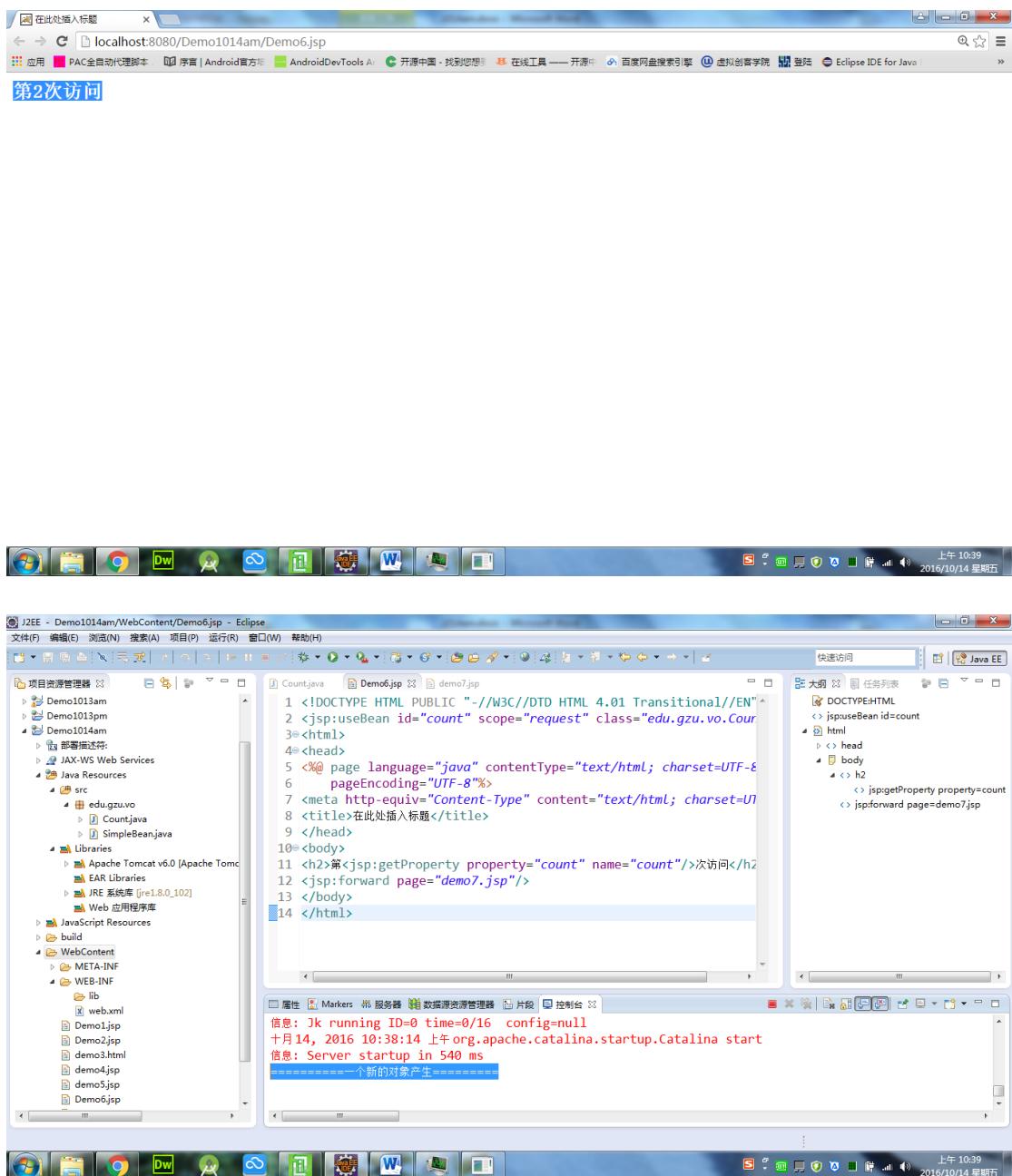
</head>

<body>

<h2>第<jsp:getProperty property="count" name="count"/>
次访问</h2>

</body>

</html>
```



## 5.6.3 session 范围内的 JavaBean

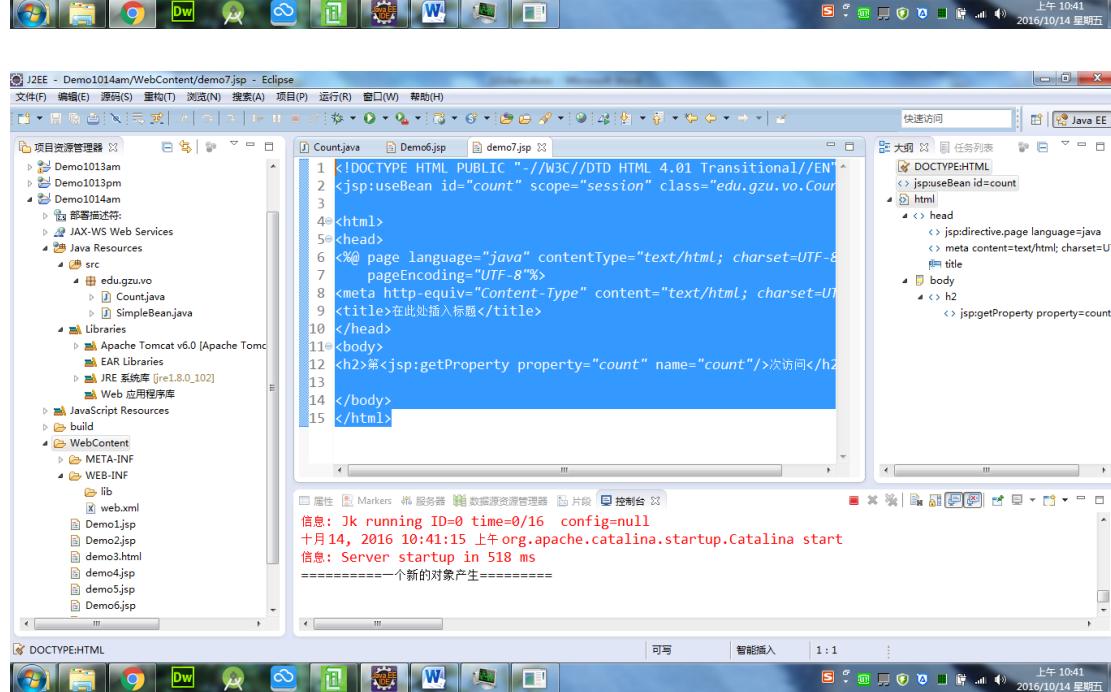
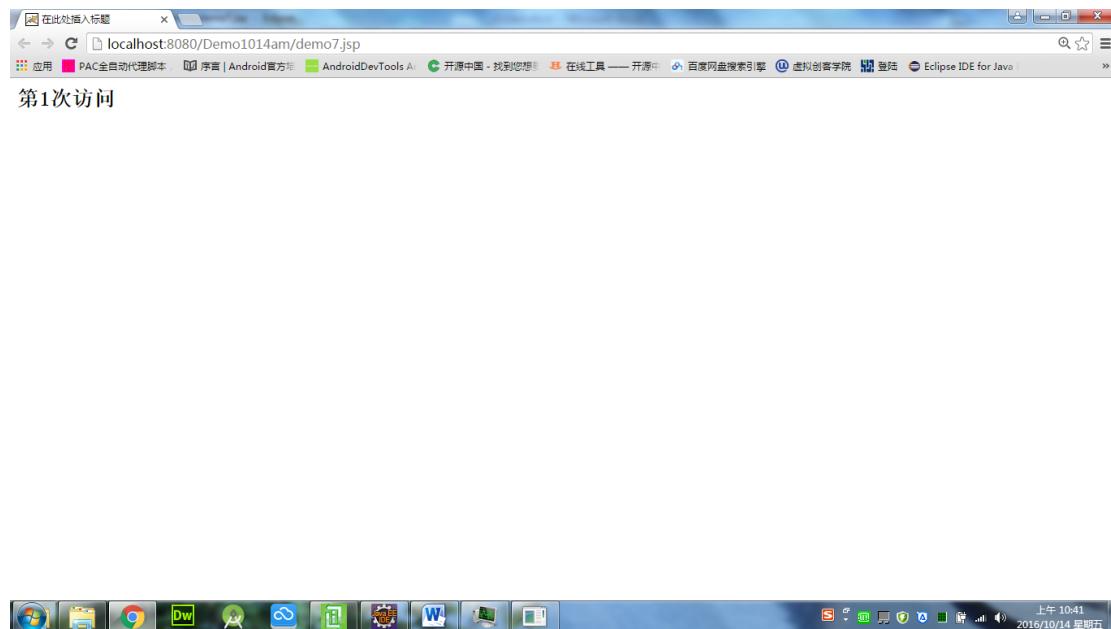
demo7

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

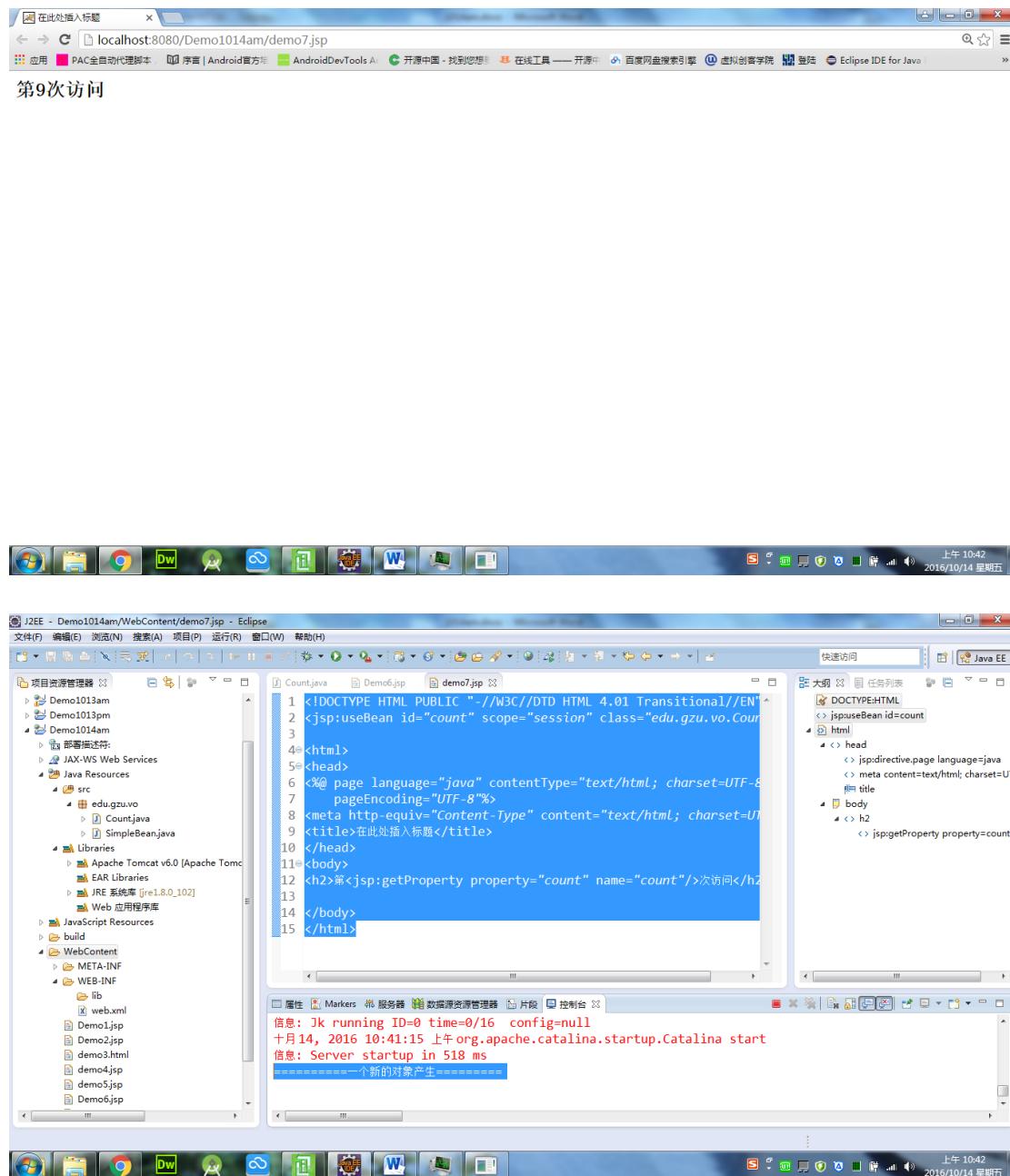
<jsp:useBean id="count" scope="session"
```

```
class="edu.gzu.vo.Count"/>

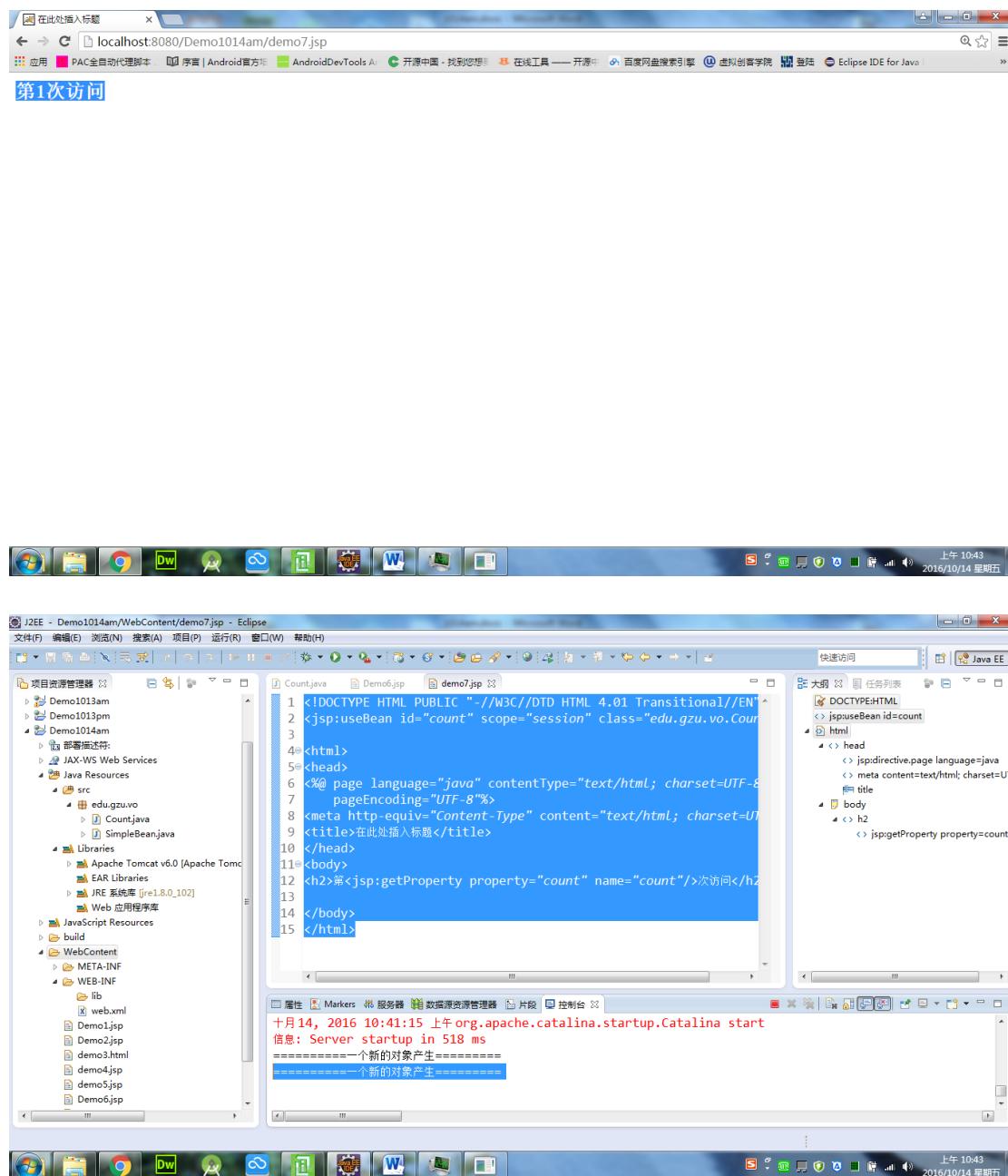
<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<h2>第<jsp:getProperty property="count" name="count"/>
次访问</h2>
</body>
</html>
```



刷新页面后：



关闭浏览器：



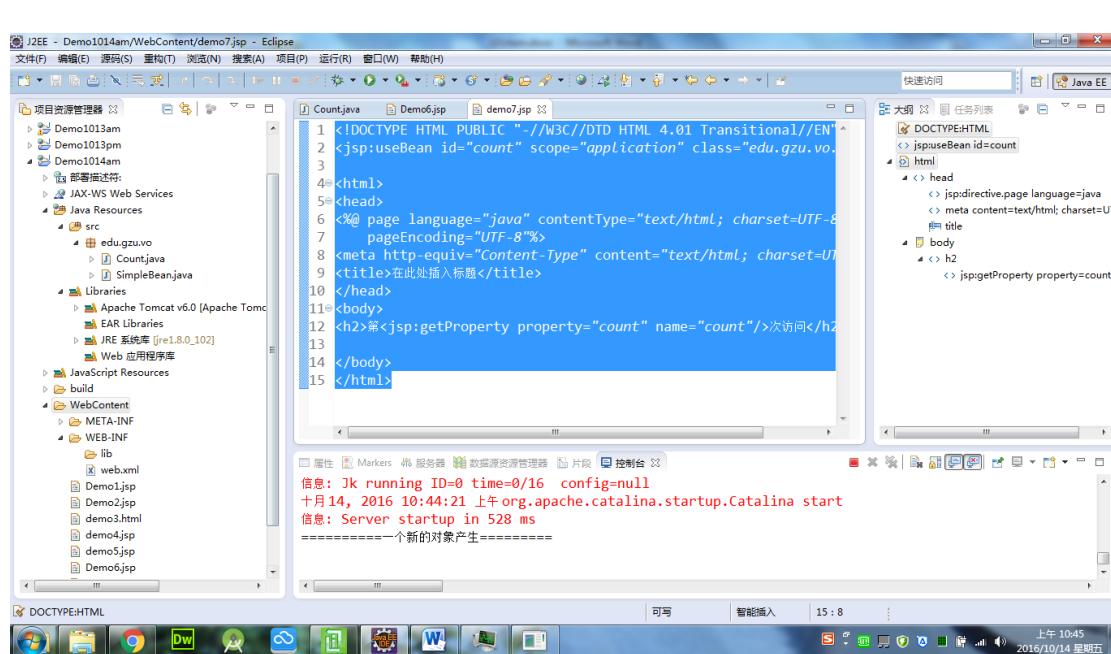
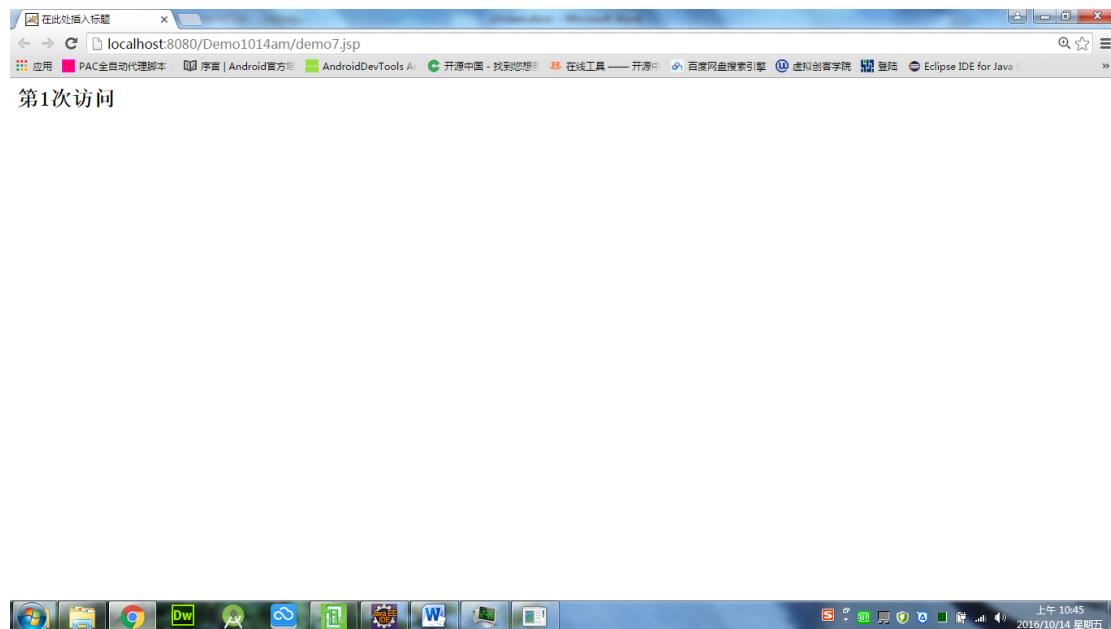
## 5.6.4 application 范围的 JavaBean

demo7

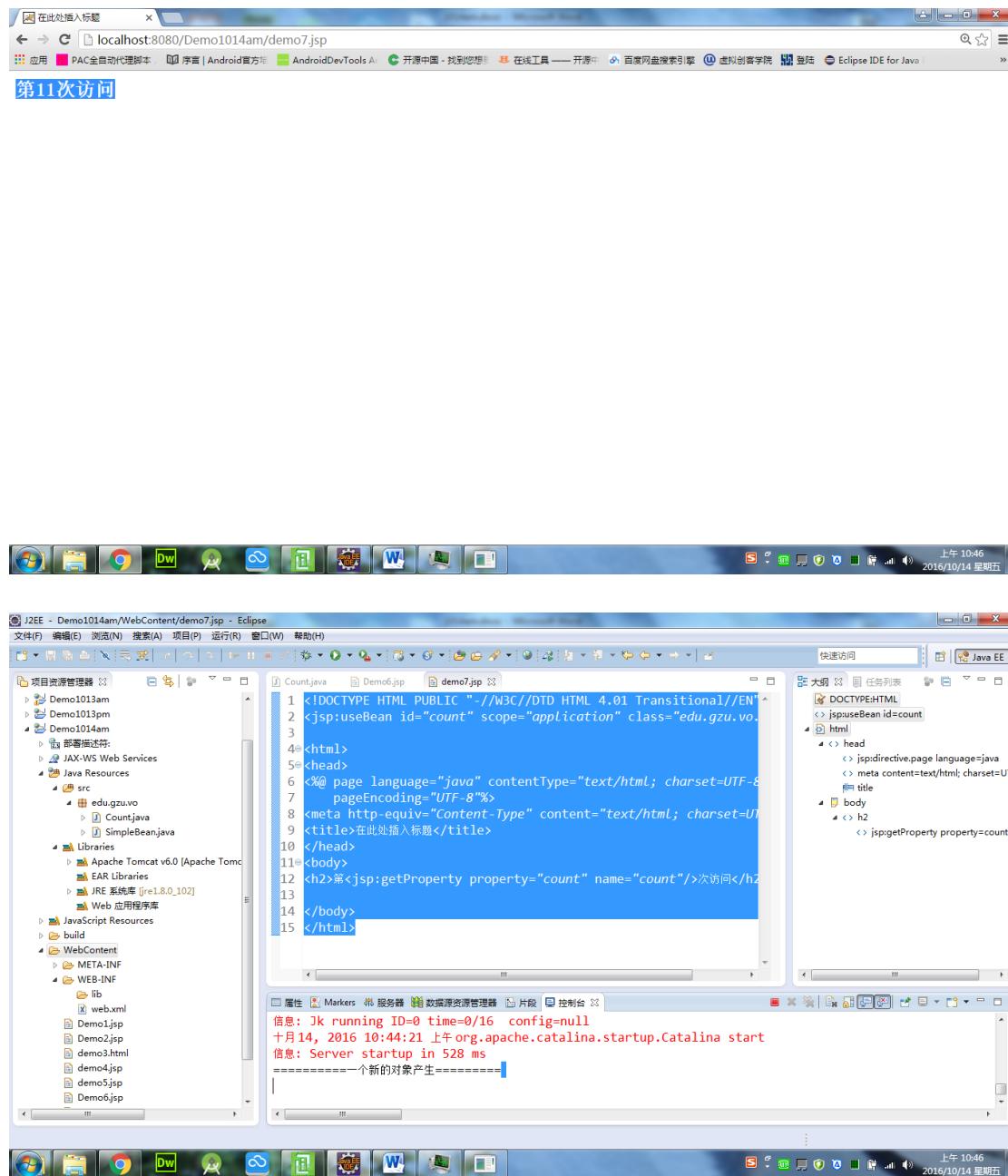
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<jsp:useBean id="count" scope="application">
```

```
class="edu.gzu.vo.Count"/>

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<h2>第<jsp:getProperty property="count" name="count"/>
次访问</h2>
</body>
</html>
```



关闭浏览器：



## 5.7 JavaBean 的删除

JavaBean 是通过<jsp:useBean>标签进行创建的，但是其操作依靠的仍然是 4 种属性范围，如果一个 JavaBean 不再使用的话，则可以直接使用 4 种属性范围的 removeAttribute()删除。

```
page: pageContext.removeAttribute("id");
```

request: request.removeAttribute("id");  
session: session.removeAttribute("id");  
application: application.removeAttribute("id");

demo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<jsp:useBean id="count" scope="session"
class="edu.gzu.vo.Count"/>

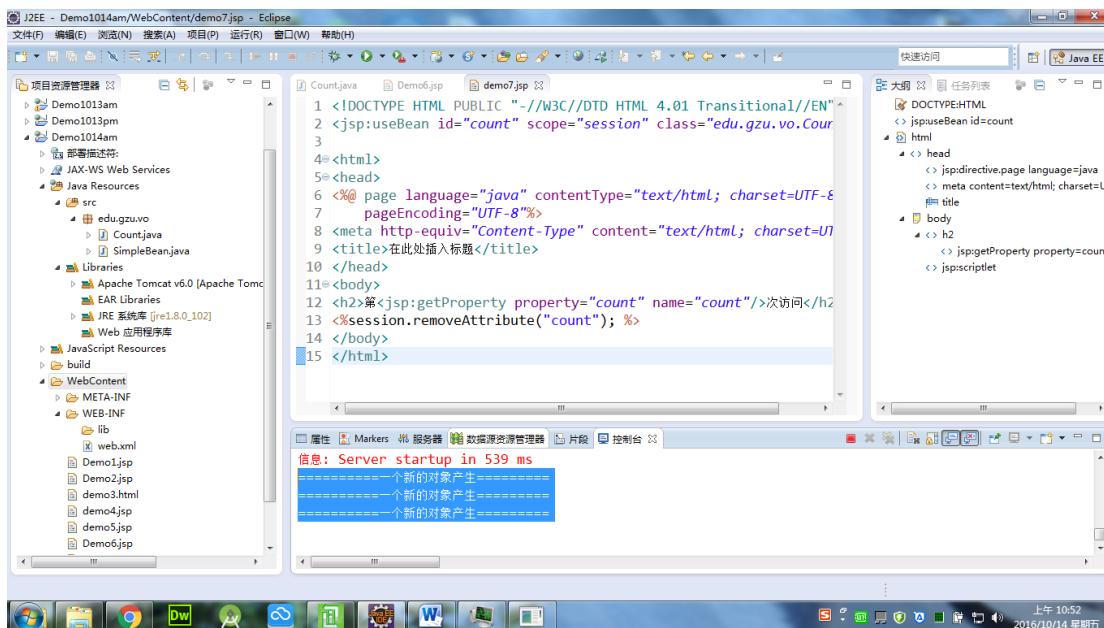
<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<h2>第<jsp:getProperty property="count" name="count"/>
次访问</h2>
<%session.removeAttribute("count"); %>
```

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

多次刷新页面：



第1次访问



## 5.8 作业：注册验证

分析：使用 JSP+JavaBean 完成一个注册验证的程序。用户在表单中填

写用户名，年龄，email。如果用户输入正确，则进行输入内容的显示，如果输入的不正确。则在错误的地方显示，而正确的内容要继续保存。

所需要的程序：

Register.java

注册使用的 JavaBean，可以接受参数，同时进行判断，并且返回错误的结果。

```
package edu.gzu.vo;

import java.util.HashMap;
import java.util.Map;

public class Register {
    private String name;
    private String age;
    private String email;
    private Map<String, String> errors = null;

    public Register() {
        this.name = "";
        this.age = "";
        this.email = "";
    }
}
```

```
this.errors = new HashMap<String, String>();  
}  
  
public boolean isTrue() {  
    boolean flag = true;  
    if (!this.name.matches("\\w{6,15}")) {  
        flag = false;  
        this.name = "";  
        errors.put("errname", "用户名是6–15位字母  
或数字。");  
    }  
    if (!this.age.matches("\\d+")) {  
        flag = false;  
        this.age = "";  
        errors.put("errage", "年龄只能是数字！");  
    }  
    if  
    (!this.email.matches("\\w+@\\w+\\.\\w+\\.?\\w"))  
    {  
        flag = false;  
        this.email = "";  
        errors.put("erremail", "输入的email地址不
```

```
    合法! ");
}

return flag;
}

public String getErrorMsg(String key) {
    String value = this.errors.get(key);
    return value == null ? "" : value;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getAge() {
    return age;
}
```

```
public void setAge(String age) {  
    this.age = age;  
}  
  
public String getEmail() {  
    return email;  
}  
  
public void setEmail(String email) {  
    this.email = email;  
}  
}
```

index.jsp

注册信息的填写，同时会对输入错误的数据进行错误提示

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
<%request.setCharacterEncoding("utf-8"); %>  
<jsp:useBean id="reg" scope="request"  
class="edu.gzu.vo.Register"/>
```

```
<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<form action="check.jsp" method="post">

username:<input type="text" name="name"
value='<jsp:getProperty property="name"
name="reg"/>'><br>

<%=reg.getErrorMsg("errname") %><br>

age:<input type="text" name="age"
value='<jsp:getProperty property="age"
name="reg"/>'><br>

<%=reg.getErrorMsg("errage") %><br>

E-mail:<input type="text" name="email"
value='<jsp:getProperty property="email"
name="reg"/>'><br>
```

```
<%=reg.getErrorMsg("erreemail") %><br>

<input type="submit" value="注册"><input type="reset"
value="重置">

</form>

</body>

</html>
```

### check.jsp

将输入表单的数据自动赋值给 JavaBean，同时进行验证，如果失败返回 index.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<%
    request.setCharacterEncoding("utf-8");
%>

<jsp:useBean id="reg" scope="request"
class="edu.gzu.vo.Register" />

<jsp:setProperty property="*" name="reg" />

<html>
<head>

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

```
pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<%
if (reg.isTrue()) {

%>

<jsp:forward page="success.jsp" />

<%
} else {

%>

<jsp:forward page="index.jsp" />

<%
}

%>

</body>

</html>
```

success.jsp

注册成功页，可以显示用户注册成功的信息。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<%
    request.setCharacterEncoding("utf-8");
%>

<jsp:useBean id="reg" scope="request"
class="edu.gzu.vo.Register" />

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
username:<jsp:getProperty property="name"
name="reg"/><br>
age:<jsp:getProperty property="age" name="reg"/><br>
E-mail:<jsp:getProperty property="email" name="reg"/>
```

```
</body>
```

```
</html>
```

用到 Map 来保存错误信息

复习：（List、Set、Map）

复习：反射机制

预习：DAO 设计模式 文件上传

# 6 DAO 设计模式

## 6.1 DAO 设计模式简介

DAO (Data Access Object, 数据访问对象) 的主要功能是数据操作。

在程序的标准开发中属于数据层的操作。



客户层：现在是 B/S 架构，所以在客户层一般指的是浏览器

显示层：JSP/Servlet 进行显示的页面效果

业务层：(BO: Business Object) 会将对各原子性的 DAO 操作进行组合，组合成一个完整的业务逻辑。

数据层：(DAO) 提供多个原子性的 DAO 操作，增删改查。

在整个 DAO 中实际上是以接口为操作的标准。也就是客户端依靠 DAO 实现的接口进行操作，而服务端要将接口具体实现。在 DAO 中一般由以下几个部分组成。

DataBaseConnection：专门负责数据库的打开与关闭。edu.gzudbc.

DataBaseConnection

VO：主要是属性 getter、setter 组成，VO 中的类的属性是由数据库的表的字段相对应的。edu.gzu.vo.类名

DAO：主要是定义操作的接口。edu.gzu.dao.XxxDAO.

Impl：DAO 接口的真实实现类。完成的是数据库的具体操作，但是不负责数据库的打开与关闭。edu.gzu.impl.XxxDAOImpl

Proxy：代理实现类，主要是完成数据库的打开、关闭，并且调用真实实现类对象操作。edu.gzu.dao.proxy.XxxDAOProxy

Factory：工厂类，通过工厂类取得一个 DAO 的实例化对象。  
edu.gzu.factory.DAOFactory。

## 6.2 DAO 开发

首先在数据库创建一个 emp 的表

```
CREATE TABLE emp (
    empno INT(4) PRIMARY KEY,
    ename VARCHAR(10),
    job VARCHAR(9),
    hiredate DATE,
    sal FLOAT(7,2)
);
```

定义 VO 类，VO 类的名称与表的名称一致，里面的属性就是表里面

的字段。

```
package edu.gzu.vo;

import java.util.Date;

public class Emp {
    private int empno;
    private String ename;
    private String job;
    private Date hiredate;
    private float sal;

    public int getEmpno() {
        return empno;
    }

    public void setEmpno(int empno) {
        this.empno = empno;
    }

    public String getName() {
        return ename;
    }
}
```

```
}
```

```
public void setEname(String ename) {
```

```
    this.ename = ename;
```

```
}
```

```
public String getJob() {
```

```
    return job;
```

```
}
```

```
public void setJob(String job) {
```

```
    this.job = job;
```

```
}
```

```
public Date getHiredate() {
```

```
    return hiredate;
```

```
}
```

```
public void setHiredate(Date hiredate) {
```

```
    this.hiredate = hiredate;
```

```
}
```

```
public float getSal() {  
    return sal;  
}  
  
public void setSal(float sal) {  
    this.sal = sal;  
}  
}
```

接下来定义 DatabaseConnection 类

```
package edu.gzudbc;

import java.sql.Connection;
import java.sql.DriverManager;

public class DatabaseConnection {
    private static final String DBDRIVER =
"org.gjt.mm.mysql.Driver";
    private static final String DBURL =
"jdbc:mysql://127.0.0.1:3306/school";
    private static final String DEUSER = "root";
    private static final String DEPWD = "123456";
}
```

```
private static final String DBPASS = "123456";  
  
private Connection conn = null;  
//在构造方法中进行数据库连接  
  
public DatabaseConnection() throws Exception {  
    try {  
        Class.forName(DBDRIVER); //加载驱动程序  
        this.conn =  
DriverManager.getConnection(DBURL, DEUSER,  
DBPASS);  
    } catch (Exception e) {  
        throw e;  
    }  
}  
//取得数据库连接  
  
public Connection getConnection() {  
    return this.conn;  
}  
//关闭数据库  
  
public void close() throws Exception {  
    if (this.conn != null) { //避免空指针异常  
        try {
```

```
this.conn.close(); //数据库关闭

} catch (Exception e) {

    throw e;
}

}

}

}
```

接下来我们该定义我们 DAO 接口，在 DAO 设计模式中，最重要的部分就是定义 DAO 接口。在定义 DAO 接口之前，你必须对业务进行详细的分析，要知道你操作的这张表在整个系统中应该具备哪些功能。

```
package edu.gzu.dao;

import java.util.List;

import edu.gzu.vo.Emp;

public interface EmpDAO {

    // 数据增加操作接口

    public boolean doCreate(Emp emp) throws
Exception;
```

```
// 查询全部的数据

public List<Emp> findAll(String keyword) throws
Exception;

// 按empno查询

public Emp findById(int empno) throws
Exception;
}
```

DAO 接口的实现类：

```
package edu.gzu.dao.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;

import edu.gzu.dao.EmpDAO;
import edu.gzu.vo.Emp;

public class EmpDAOImpl implements EmpDAO {
```

```
private Connection conn = null;//数据库连接对象
private PreparedStatement ps = null;//数据库操作对象

public EmpDAOImpl(Connection conn){//通过构造方法取得数据库连接

    this.conn = conn;//取得数据库连接
}

@Override
public boolean doCreate(Emp emp) throws Exception {

    boolean flag = false;//定义一个标志位
    String sql = "INSERT INTO emp
(empno,ename,job,hiredate,sal)VALUES(?, ?, ?, ?, ?)
";
    //实例化ps对象
    this.ps = this.conn.prepareStatement(sql);
    this.ps.setInt(1, emp.getEmpno());//设置empno
    this.ps.setString(2, emp.getEname());
    this.ps.setString(3, emp.getJob());
    this.ps.setDate(4, new
```

```
java.sql.Date(emp.getHiredate().getTime()));

    this.ps.setFloat(5, emp.getSal());

    if(this.ps.executeUpdate()>0){//更新记录的
行数大于0

        flag = true;//修改标志位
    }

    this.ps.close();//关闭ps操作。

    return flag;
}

@Override

public List<Emp> findAll(String keyword) throws
Exception {

    List<Emp> all = new ArrayList<Emp>();//定义
一个集合，接受查询到的全部数据

    String sql = "SELECT
empno,ename,job,hiredate,sal FROM emp WHERE ename
LIKE ? OR job LIKE ?";

    this.ps = this.conn.prepareStatement(sql);

    this.ps.setString(1, "%" + keyword + "%");

    this.ps.setString(2, "%" + keyword + "%");//设
置查询的关键字
```

```
ResultSet rs = this.ps.executeQuery(); //执行查询操作

    Emp emp = null; //定义emp对象

    while(rs.next()){

        emp = new Emp(); //每次循环都实例化一个emp对象

        emp.setEmpno(rs.getInt(1)); //设置empno

        emp.setEname(rs.getString(2));

        emp.setJob(rs.getString(3));

        emp.setHiredate(rs.getDate(4));

        emp.setSal(rs.getFloat(5));

        all.add(emp); //向集合中增加对象

    }

    this.ps.close(); //关闭ps操作。

    return all;

}

@Override

public Emp findById(int empno) throws Exception

{

    Emp emp = null;

    String sql = "SELECT
```

```
empno,ename,job,hiredate,sal FROM emp WHERE
empno=?;

    this.ps = this.conn.prepareStatement(sql);
    this.ps.setInt(1, empno);

    ResultSet rs = this.ps.executeQuery(); //执行查询操作

    if(rs.next()){
        emp = new Emp(); //实例化一个emp对象
        emp.setEmpno(rs.getInt(1)); //设置empno
        emp.setEname(rs.getString(2));
        emp.setJob(rs.getString(3));
        emp.setHiredate(rs.getDate(4));
        emp.setSal(rs.getFloat(5));
    }

    this.ps.close(); //关闭ps操作。
    return emp;
}

}
```

上面都分别实现增加数据、查询所有的数据、`findById`，由于在方法声明的时候使用了 `throws` 关键字，所以说所有的异常都交给被调用

者处理。

而且在上面的实现类中并没有处理数据库的打开与关闭，只是通过构造方法取得了数据库的链接，接下来真正的数据库打开关闭由代理类来完成。

```
package edu.gzu.dao.proxy;

import java.util.List;

import edu.gzu.dao.EmpDAO;
import edu.gzu.dao.impl.EmpDAOImpl;
import edu.gzu.jdbc.DatabaseConnection;
import edu.gzu.vo.Emp;

public class EmpDAOProxy implements EmpDAO {
    // 定义数据库连接类
    private DatabaseConnection dbc = null;
    // 声明 DAO 对象
    private EmpDAO dao = null;
    // 在构造方法中实例化连接，同时实例化 dao 对象
    public EmpDAOProxy() throws Exception {
        this.dbc = new DatabaseConnection();// 连接数据库
    }
}
```

```
this.dao = new EmpDAOImpl(this.jdbc.getConnection());// 实例化真实主题类

}

@Override
public boolean doCreate(Emp emp) throws Exception {
    boolean flag = false;//定义标志位
    try {
        if (this.dao.findById(emp.getEmpno()) == null) {//在插入数据之前先查询数据库中有无该数据
            flag = this.dao.doCreate(emp);//调用真实主题类中方法进行插入
        }
    } catch (Exception e) {
        throw e;
    } finally {
        this.jdbc.close();//关闭数据库连接
    }
    return flag;
}

@Override
```

```
public List<Emp> findAll(String keyWord) throws Exception {  
  
    List<Emp> all = null;  
  
    try {  
  
        all = this.dao.findAll(keyWord);  
  
    } catch (Exception e) {  
  
        throw e;  
  
    } finally {  
  
        this.jdbc.close();  
  
    }  
  
    return all;  
  
}
```

@Override

```
public Emp findById(int empno) throws Exception {  
  
    Emp emp = null;  
  
    try {  
  
        emp = this.dao.findById(empno);  
  
    } catch (Exception e) {  
  
        throw e;  
  
    } finally {  
  
        this.jdbc.close();  
  
    }  

```

```
    return emp;  
}  
}
```

现在在代理类的构造方法中实例化的数据库连接类的对象以及真实主题实现类，而且在代理类中的各个方法也只是调用了真实主题类中相应的方法。

DAO 的真实实现类和代理类编写完成以后需要编写工厂类，来降低代码间的耦合度。

工厂类：

```
package edu.gzu.factory;

import edu.gzu.dao.EmpDAO;
import edu.gzu.dao.proxy.EmpDAOProxy;

public class DAOFactory {
    //取得DAO接口实例
    public static EmpDAO getEmpDAOInstance() throws
Exception{
    //取得代理类的实例
    return new EmpDAOProxy();
}
```

}

测试：

```
package edu.gzu.dao.test;

import edu.gzu.factory.DAOFactory;
import edu.gzu.vo.Emp;

public class TestDaoInsert {

    public static void main(String[] args) {
        Emp emp = null;
        for (int i = 0; i < 9; i++) {
            emp = new Emp();
            emp.setEmpno(1000 + i);
            emp.setEname("hunk-" + i);
            emp.setJob("程序员-" + i);
            emp.setHiredate(new java.util.Date());
            emp.setSal(500 * i);
            try {
                DAOFactory.getEmpDAOInstance().doCreate(emp)
            }
        }
    }
}
```

```
;  
    } catch (Exception e) {  
        // TODO 自动生成的 catch 块  
        e.printStackTrace();  
    }  
}  
  
}  
}
```

## 6.3 JSP 调用 DAO

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
  
pageEncoding="UTF-8"%>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">
```

```
<title>添加用户</title>

</head>

<body>

<form action="emp_insert_do.jsp" method="post">

empno:<input type="text" name="empno"><br>

ename:<input type="text" name="ename"><br>

job:<input type="text" name="job"><br>

hiredate:<input type="text" name="hiredate"><br>

sal:<input type="text" name="sal"><br>

<input type="submit" value="submit">

</form>

</body>

</html>
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<%@page import="java.text.SimpleDateFormat"%>
<%@ page import="edu.gzu.factory.* , edu.gzu.vo.*" %>

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

```
pageEncoding="UTF-8">%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>执行插入</title>

</head>

<body>

<%request.setCharacterEncoding("utf-8"); %>

<%
Emp emp = new Emp();

emp.setEmpno(Integer.parseInt(request.getParameter("empno")));

emp.setEname(request.getParameter("ename"));

emp.setJob(request.getParameter("job"));

emp.setHiredate(new

SimpleDateFormat("yyyy-MM-dd").parse(request.getParameter("hiredate")));

emp.setSal(Float.parseFloat(request.getParameter("sal")));

try{

if(DAOFactory.getEmpDAOInstance().doCreate(emp)){

%>

<h2>雇员信息添加成功! </h2>
```

```
<%  
  
}else{  
  
%>  
  
<h2>雇员信息添加失败! </h2>  
  
<%  
  
}  
  
}catch(Exception e){  
  
e.printStackTrace();  
  
}  
  
%>  
  
</body>  
  
</html>
```

列出所有数据：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<%@ page import="edu.gzu.factory.* ,edu.gzu.vo.*" %>  
  
<%@page import="java.util.*" %>  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
  
pageEncoding="UTF-8"%>
```

```
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>列出数据</title>

</head>

<body>

<%request.setCharacterEncoding("utf-8"); %>

<%
try{

String keyword = request.getParameter("kw");

if(keyword==null){

    keyword="";

}

List<Emp> all =
DAOFactory.getEmpDAOInstance().findAll(keyword);

Iterator<Emp> iter = all.iterator();

%>

<center>

<form action="emp_list.jsp" method="post">

请输入查询的关键字: <input type="text" name="kw"><input
type="submit" value="查询">

</form>

<table border="1">
```

```
<tr>

<td>雇员编号</td>

<td>雇员姓名</td>

<td>工作</td>

<td>雇佣日期</td>

<td>薪酬</td>

</tr>

<%
while(iter.hasNext()){

    Emp emp = iter.next();

    %>

    <tr>

        <td><%=emp.getEmpno() %></td>

        <td><%=emp.getEname() %></td>

        <td><%=emp.getJob() %></td>

        <td><%=emp.getHiredate() %></td>

        <td><%=emp.getSal() %></td>

    </tr>

    <%
}

%>

</table>
```

```
</center>

<%
}catch(Exception e){

    e.printStackTrace();

}

%>

</body>

</html>
```

作业：写一个 JSP 页面，调用我们写好的 findById()。

## 7 文件上传

文件上传，在实际的开发中一般有两种，一种是 SmartUpload，一种是 FileUpload。SmartUpload 是由 [www.jspsmart.com](http://www.jspsmart.com) 网站开发的，可以轻松实现上传。很遗憾现在网上已经关闭。虽然说上面的 SmartUpload 已经关闭了，但是现在还有很多程序员喜欢使用。掌握 SmartUpload 上传。FileUpload 是 Apache 官方给的一个免费的上传组件，但是这个上传组件很麻烦。交给大家自学。理解即可。

### 7.1 上传单个文件

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
```

```
<form action="demo2.jsp" method="post"
enctype="multipart/form-data">

<input type="file" name="file1" ><input type="submit">

</form>

</body>

</html>
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<%@page import="org.lxh.smart.SmartUpload"%>

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<%
//实例化SmartUpload上传组件
```

```
SmartUpload su = new SmartUpload();  
  
//初始化上传操作  
  
su.initialize(pageContext);  
  
//上传准备  
  
su.upload();  
  
//将上传文件保存在upload文件夹中，该文件夹是手动创建的。  
  
su.save("upload");  
  
%>  
  
</body>  
  
</html>
```

## 7.2 混合表单

表单要上传文件的时候，是必须使用 `enctype` 进行封装的。但是我们表单中如果还有其他的表单控件的话，这个时候就无法使用 `request` 对象来获取了。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"
```

```
pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<form action="demo2.jsp" method="post"
enctype="multipart/form-data">

<input type="text" name="name"><br>

<input type="file" name="file1" ><input type="submit">

</form>

</body>

</html>
```

在接收 name 的时候，使用我们的 request 对象就接收不到了。因为我们的表单进行了二进制封装。必须依靠 SmartUpload 组件中的 getRequest.getParameter()方法来取得请求的参数。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<%@page import="org.Lxh.smart.SmartUpload"%>

<html>
<head>

<%@ page language="java" contentType="text/html;

```

```
charset=UTF-8"

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<%
request.setCharacterEncoding("utf-8");

%>

<%
SmartUpload su = new SmartUpload();

su.initialize(pageContext);

su.upload();

//通过request组件已经无法取得内容

String name = request.getParameter("name");

String uname =
su.getRequest().getParameter("name");

su.save("upload");

%>

<%=name%>

<%=uname%>
```

```
</body>
```

```
</html>
```

## 7.3 为上传文件自动命名

为了防止重名，我们在上传的时候就需要我们给上传的文件自动命名。

IP 地址+时间戳+三位随机数

我们需要写一个自动命名的类：

```
package edu.gzu.util;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Random;

public class IPTimeStamp {
    private SimpleDateFormat sdf = null;
    private String ip = null;

    public IPTimeStamp() {
    }

    public IPTimeStamp(String ip) {
        this.ip = ip;
    }
}
```

}

```
}

public String getTimeStamp() {
    this.sdf = new
SimpleDateFormat("yyyyMMddHHmmssSSS");
    return this.sdf.format(new Date());
}

private String addZero(String str, int len) {
    StringBuffer s = new StringBuffer();
    s.append(str);
    while (s.length() < len) {
        s.insert(0, "0");
    }
    return s.toString();
}

}
```

修改一下我们上面的例子，为文件上传自动命名。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
```

```
<%@page import="java.io.File"%>

<%@page import="edu.gzu.util.IPTimeStamp"%>

<%@page import="org.Lxh.smart.SmartUpload"%>

<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<%
    request.setCharacterEncoding("utf-8");
    String ip = request.getRemoteAddr();
    String ip1 = "127.0.0.1";
%>

<%
    SmartUpload su = new SmartUpload();
    su.initialize(pageContext);
    su.upload();
%>
```

```
String uname =  
  
su.getRequest().getParameter("name");  
  
    IPTimeStamp its = new IPTimeStamp(ip);  
  
    String ext =  
  
su.GetFiles().getFile(0).getFileExt();  
  
    String filename = its.getIPTimeRand()+"."+ext;  
  
    su.GetFiles().getFile(0).saveAs(getServletContext()  
.getRealPath("/")+"upload"+java.io.File.separator  
+filename);  
  
%>  
  
<%=uname%>  
  
      
  
</body>  
  
</html>
```

## 7.4 批量上传

前端：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>
```

```
<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<form action="demo2.jsp" method="post"
enctype="multipart/form-data">

<input type="file" name="file1" ><br>
<input type="file" name="file2" ><br>
<input type="file" name="file3" ><br><input
type="submit">

</form>

</body>

</html>
```

后台：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<%@page import="java.io.File"%>
```

```
<%@page import="edu.gzu.util.IPTimeStamp"%>

<%@page import="org.Lxh.smart.SmartUpload"%>

<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<%
    request.setCharacterEncoding("utf-8");
    String ip = request.getRemoteAddr();
    String ip1 = "127.0.0.1";
%>

<%
    SmartUpload su = new SmartUpload();
    su.initialize(pageContext);
    su.upload();
    IPTimeStamp its = new IPTimeStamp(ip);
%
```

```
for (int i = 0; i < su.GetFiles().getCount(); i++)  
{  
    String ext =  
    su.GetFiles().getFile(i).getFileExt();  
  
    String filename = its.getIPTimeRand() + "." +  
ext;  
  
    su.GetFiles()  
        .getFile(i)  
  
        .saveAs(getServletContext().getRealPath("/") +  
"upload"  
        + java.io.File.separator +  
filename);  
%>  
  
  
<%  
}  
%>  
</body>  
</html>
```

## 8 Servlet

### 8.1 Servlet 简介

Servlet(服务器端小程序)是由 Java 编写的服务器端程序, 可以向 JSP 一样生产动态的 Web 页。Servlet 主要是运行在服务端, 并且由服务器调用。Servlet 采取了多线程, 大大提升了执行速度。并且保留了 Java 的可移植性。

Servlet 最大的好处就是它可以处理客户端传来的 HTTP 请求, 并且返回响应。

Servlet 程序的执行流程

- 1、 客户端通过 HTTP 发出请求
- 2、 Web 服务器接收该请求并将其发送给 Servlet, 如果这个 Servlet 没有被加载, web 服务器将把加载到 Java 虚拟机并执行
- 3、 Servlet 将接收 HTTP 请求并且执行
- 4、 Servlet 会将处理后的结果想 web 服务器返回应答
- 5、 Web 服务器将从 Servlet 收到的应答发回给客户端。

在 Servlet 中最重要的是 Servlet 接口。

### 8.2 第一个 Servlet 程序 (手动创建)

如果要开发一个可以处理 HTTP 请求的 Servlet 程序, 则肯定要继承 HttpServlet 类, 而且在自定义的 Servlet 类中至少还要覆写 HttpServlet 类中提供的 doGet()方法。

HttpServlet 中的方法:

protected void	<u>doGet</u> (HttpServletRequest req, HttpServletResponse resp)	Called by the server (via the service method) to allow a servlet to handle a GET request.
protected void	<u>doPost</u> (HttpServletRequest req, HttpServletResponse resp)	Called by the server (via the service method) to allow a servlet to handle a POST request.

手动创建第一个 Servlet 程序:

```
package edu.gzu.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloServlet extends HttpServlet {

    public void doGet(HttpServletRequest
req,HttpServletResponse resp) throws
```

```
ServletException, IOException{  
    PrintWriter out = resp.getWriter();  
    out.println("<html>");  
  
    out.println("<head><title>HelloServlet</title></head>");  
    out.println("<body>");  
    out.println("<h2>Hello, Servlet!</h2>");  
    out.println("</body>");  
    out.println("</html>");  
    out.close();  
}  
  
}
```

编译完成之后，是无法立即访问的，因为所有的 Servlet 都是.class 文件，所以在我们的 WEB-INF\web.xml 中进行 Servlet 程序的映射配置。

```
<servlet>  
    <servlet-name>helloservlet</servlet-name>  
  
<servlet-class>edu.gzu.servlet.HelloServlet</servlet-class>
```

```
</servlet>

<servlet-mapping>
    <servlet-name>helloservlet</servlet-name>
    <url-pattern>/helloServlet</url-pattern>
</servlet-mapping>
```



注意：每次配置完 web.xml 都需要重启服务器。

## 8.3 Servlet 与表单

Servlet 本身就存在 HttpServletRequest 和 HttpServletResponse 对象的声明，所以可以使用 servlet 接收用户所提交的内容  
新建一个表单：

```
<!doctype html>
<html>
<head>
<meta charset="UTF-8"/>
```

```
<title>在此插入标题</title>

</head>

<body>

<form action="Hello" method="post">

name:<input type="text" name="name"><input

type="submit">

</form>

</body>

</html>
```

hello:

```
package edu.gzu.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Hello extends HttpServlet {

    private static final long serialVersionUID = 1L;
```

```
public Hello() {  
    super();  
}  
  
protected void doGet(HttpServletRequest request,  
                      HttpServletResponse response) throws ServletException,  
IOException {  
  
}  
  
protected void doPost(HttpServletRequest request,  
                      HttpServletResponse response) throws ServletException,  
IOException {  
  
    String name = request.getParameter("name");  
  
    PrintWriter out = response.getWriter();  
  
    out.println("<html>");  
  
    out.println("<head><title>HelloServlet</title></head>");  
  
    out.println("<body>");  
  
    out.println("<h2>" + name + "</h2>");  
  
    out.println("</body>");  
  
    out.println("</html>");
```

```

        out.close();

    }

}

```

## 8.4 Servlet 生命周期

Servlet 是运行在服务端的 java 程序，其生命周期受到 Web 容器的控制，生命周期包括加载程序、初始化、服务、销毁、卸载 5 个阶段。

	<b>void <u>init()</u></b> <b>A convenience method which can be overridden so that there's no need to call super.init(config).</b>
	<b>void <u>init(ServletConfig config)</u></b> <b>Called by the servlet container to indicate to a servlet that the servlet is being placed into service.</b> <b>两个 init 方法同时出现的时候，调用的是 <u>init(ServletConfig config)</u></b>
abstract void	<b><u>service(HttpServletRequest req, HttpServletResponse res)</u></b> <b>Called by the servlet container to allow the servlet to respond to a request.</b>
void	<b><u>destroy()</u></b> <b>Called by the servlet container to indicate to a servlet that the servlet is being taken out of service.</b>

### 1、加载 Servlet

Web 容器负责加载，当 Web 容器启动时或者是在第一次使用这个 Servlet 时，容器负责创建 Servlet，但是用户必须通过部署描述 web.xml

指定的位置，成功加载之后，web 容器会通过反射的方式对 Servlet 进行实例化。

## 2、 初始化

当一个 Servlet 被实例化之后，容器将调用 init 方法初始化这个对象，初始化的目的是为了让 servlet 对象在的护理客户端请求前完成一些初始化的工作，例如建立数据库连接、读取资源文件，如果初始化失败，则该 Servlet 就会被直接卸载。

## 3、 服务

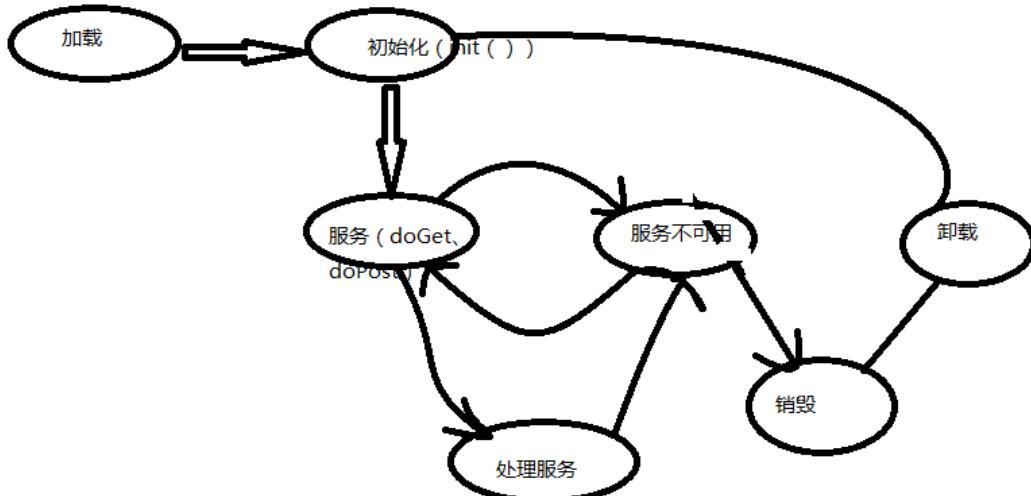
当有请求提交时，Servlet 将调用 service()方法，常用的是 doGet 和 doPost 进行处理。通过 request 来接受用户请求，通过 response 来设置响应信息。

## 4、 销毁

当 web 容器关闭或者检测到一个 servlet 要从容器中被删除时，会自动调用 destroy()方法，以便让该实例释放所占用的资源。

## 5、 卸载

当一个 Servlet 调用完 destroy()方法后，此实例将等待被垃圾回收，如果需要再次使用此 servlet，会重新调用 init()方法。



注意：在正常情况下，Servlet 只会被初始化一次，而处理服务会调用多次。销毁也只调用一次，但是当一个 servlet 长期不用的时候，会被容器自动销毁，而再次使用的时候，会重新实例化。

demo：

```
package edu.gzu.servlet;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class LifeCycleServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
```

```
public LifeCycleServlet() {  
  
    super();  
  
}  
  
  
public void init()throws ServletException {  
  
    System.out.println("=====1Servlet 初始化=====");  
  
}  
  
  
public void doGet(HttpServletRequest request,  
HttpServletResponse response) throws ServletException, IOException {  
  
    System.out.println("=====2Servlet 服务=====");  
  
}  
  
  
public void doPost(HttpServletRequest request,  
HttpServletResponse response) throws ServletException, IOException {  
  
    this.doGet(request, response);  
  
}  
  
  
public void destroy(){  
  
    System.out.println("=====3Servlet 销毁=====");  
  
}
```

}

## 8.5 取得初始化配置信息

在将 JSP 的 9 大内置对象的时候，我们讲过 config 对象，通过

void init(ServletConfig config)

Called by the servlet container to indicate to a servlet that the servlet is being placed into service.

实际上 config 对象就是 ServletConfig 接口的实例。通过 config 对象可以读取配置信息。在这里也可以通过 init 方法找到 ServletConfig 接口的实例。

```
package edu.gzu.servlet;

import java.io.IOException;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class InitParamServlet extends HttpServlet {
    private String initParam = null;
```

```
private static final long serialVersionUID = 1L;

public InitParamServlet() {
    super();
}

public void init(ServletConfig config) throws ServletException {
    this.initParam = config.getInitParameter("ref");
}

protected void doGet(HttpServletRequest request,
                      HttpServletResponse response) throws ServletException,
IOException {
    System.out.println("初始化参数: "+this.initParam);
}

protected void doPost(HttpServletRequest request,
                      HttpServletResponse response) throws ServletException,
IOException {
    this.doGet(request, response);
}
```

{}

配置信息：

```
<servlet>

    <description></description>

    <display-name>InitParamServlet</display-name>

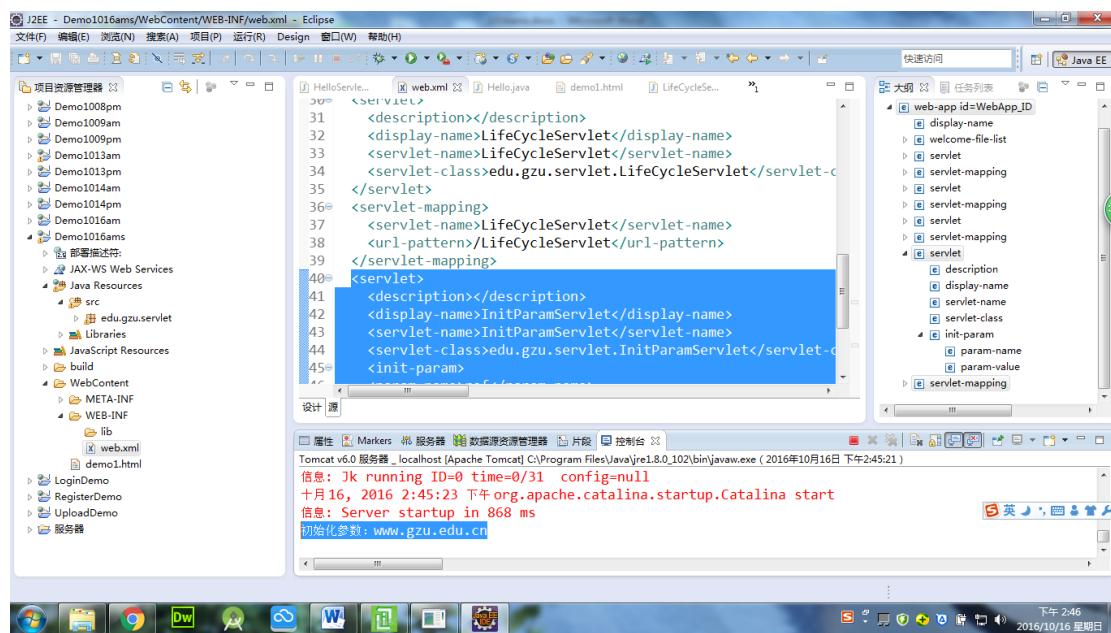
    <servlet-name>InitParamServlet</servlet-name>

    <servlet-class>edu.gzu.servlet.InitParamServlet</servlet-class>

    <init-param>
        <param-name>ref</param-name>
        <param-value>www.gzu.edu.cn</param-value>
    </init-param>
</servlet>

<servlet-mapping>
    <servlet-name>InitParamServlet</servlet-name>
    <url-pattern>/InitParamServlet</url-pattern>
</servlet-mapping>
```

运行结果：



## 8.6 取得其他内置对象

### 8.6.1 取得 HttpSession 实例

Interface `HttpServletRequest` 取得 Session 依靠的 `HttpServletRequest` 接口实例化的方法。

<u><a href="#">HttpSession</a></u> <u><a href="#">getSession()</a></u>	<p>Returns the current session associated with this request, or if the request does not have a session, creates one. 返回当前的 session</p>
<u><a href="#">HttpSession</a></u> <u><a href="#">getSession(boolean create)</a></u>	<p>Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session. 返回当前的 session, 如果没有则创建一个新的 session。</p>

**demo:**

```
package edu.gzu.servlet;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class GetSessionDemo extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public GetSessionDemo() {
        super();
    }

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws ServletException,
IOException {
        HttpSession ses = request.getSession(); //取得 Session 对象
        System.out.println("SessionID:" + ses.getId());
    }
}
```

```
ses.setAttribute("name", "hunk");

System.out.println("name 属性的值 : 

"+ses.getAttribute("name"));

}

protected void doPost(HttpServletRequest request,
                      HttpServletResponse response) throws ServletException,
IOException {
    this.doGet(request, response);
}

}
```

配置：

```
<servlet>

<description></description>

<display-name>GetSessionDemo</display-name>

<servlet-name>GetSessionDemo</servlet-name>

<servlet-class>edu.gzu.servlet.GetSessionDemo</servlet-class>
```

```

</servlet>

<servlet-mapping>

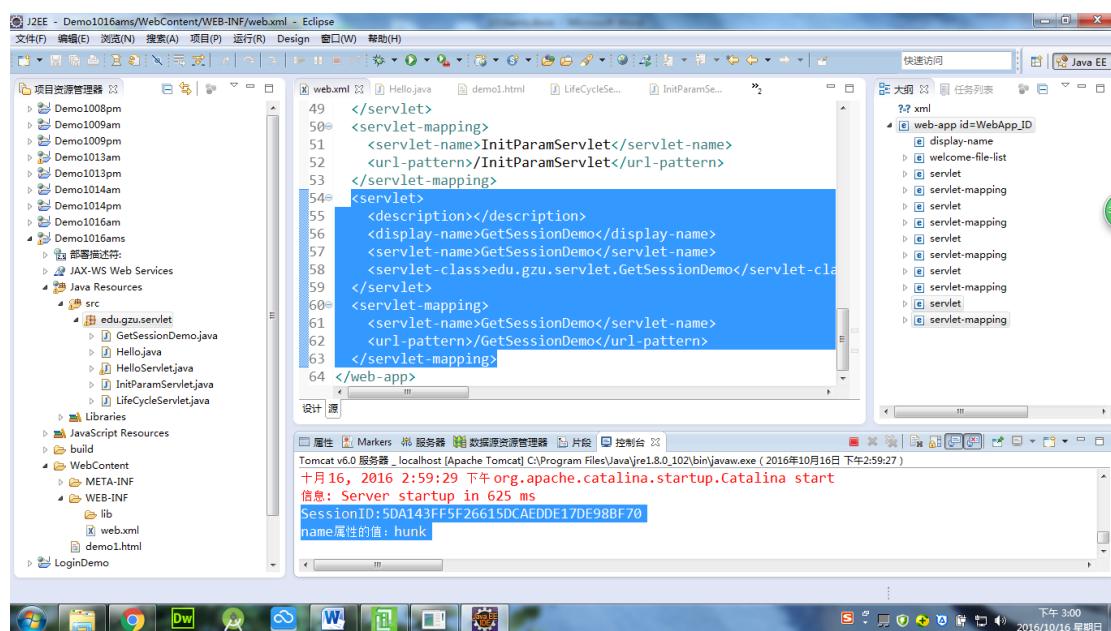
    <servlet-name>GetSessionDemo</servlet-name>

    <url-pattern>/GetSessionDemo</url-pattern>

</servlet-mapping>

```

运行结果：



## 8.6.2 取得 ServletContext 实例

javax.servlet

Class GenericServlet

java.lang.Object

└ javax.servlet.GenericServlet

All Implemented Interfaces:

java.io.Serializable, [Servlet](#), [ServletConfig](#)

**Direct Known Subclasses:**[HttpServlet](#)ServletContext getServletContext()

Returns a reference to the ServletContext in which this servlet is running.

**demo:**

```
package edu.gzu.servlet;

import java.io.IOException;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class GetApplicationServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public GetApplicationServlet() {
        super();
    }
}
```

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

    ServletContext application = super.getServletContext();
    System.out.println("真实路径: "+application.getRealPath("/"));

}

protected void doPost(HttpServletRequest request,
                      HttpServletResponse response) throws ServletException,
IOException {
    this.doGet(request, response);
}

}
```

配置：

```
<servlet>

    <description></description>
    <display-name>GetApplicationServlet</display-name>
    <servlet-name>GetApplicationServlet</servlet-name>

    <servlet-class>edu.gzu.servlet.GetApplicationServlet<
```

```
/servlet-class>

</servlet>

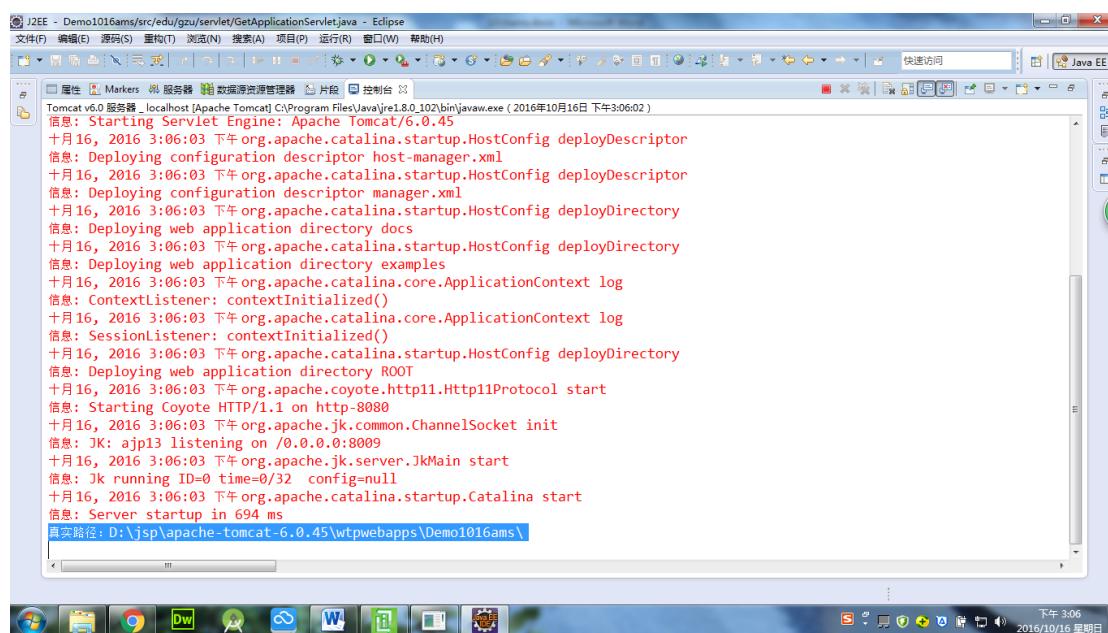
<servlet-mapping>

    <servlet-name>GetApplicationServlet</servlet-name>

    <url-pattern>/GetApplicationServlet</url-pattern>

</servlet-mapping>
```

运行结果：



## 8.7 Servlet 跳转

### 8.7.1 客户端跳转

在 Servlet 中要想进行客户端跳转，直接使用 `HttpServletResponse` 接口的 `sendRedirect()`方法即可。这是客户端跳转，只能传递 session 范围的属性，对于 request 范围属性无能为力。

demo：

```
package edu.gzu.servlet;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ClientRedirectServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public ClientRedirectServlet() {
        super();
    }

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws ServletException,
IOException {
        request.getSession().setAttribute("name", "hunk");
        request.setAttribute("age", 32);
        response.sendRedirect("demo1.jsp");
    }
}
```

```
protected void doPost(HttpServletRequest request,
                      HttpServletResponse response) throws ServletException,
IOException {
    this doGet(request, response);
}
```

配置：

```
<servlet>
    <description></description>
    <display-name>ClientRedirectServlet</display-name>
    <servlet-name>ClientRedirectServlet</servlet-name>

    <servlet-class>edu.gzu.servlet.ClientRedirectServlet<
    /servlet-class>
</servlet>

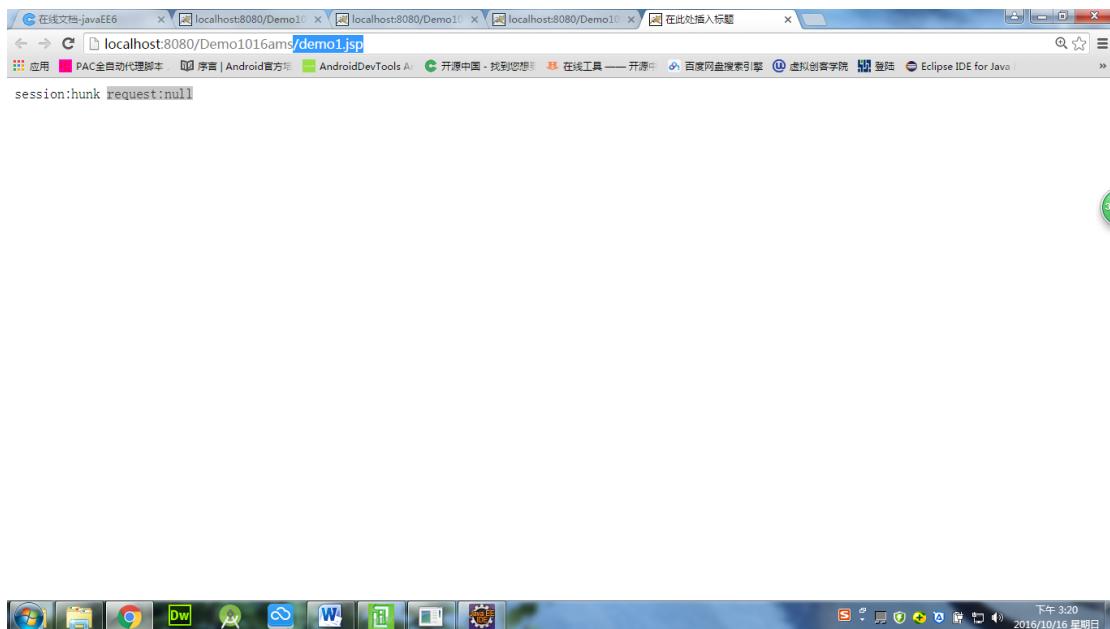
<servlet-mapping>
    <servlet-name>ClientRedirectServlet</servlet-name>
    <url-pattern>/ClientRedirectServlet</url-pattern>
</servlet-mapping>
```

demo1.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<% request.setCharacterEncoding("utf-8"); %>
session:<%=session.getAttribute("name") %>
request:<%=request.getAttribute("age") %>
</body>
</html>
```

运行结果：



客户端跳转。

### 8.7.2 服务器端跳转

在 JSP 中服务器端跳转时是采用的<jsp:forward>指令，在 servlet 中没有这个指令，所以想要在 servlet 中实现服务端跳转就必须依靠 Interface RequestDispatcher 来完成。

void	<u><a href="#">forward</a></u> ( <u><a href="#">ServletRequest</a></u> request, <u><a href="#">ServletResponse</a></u> response) Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server. 页面跳转
void	<u><a href="#">include</a></u> ( <u><a href="#">ServletRequest</a></u> request, <u><a href="#">ServletResponse</a></u> response) Includes the content of a resource (servlet, JSP page, HTML file) in the response. 页面包含

要想使用 [forward](#) 方法，还得使用 ServletRequest 接口实现的 [getRequestDispatcher](#)

<u><a href="#">RequestDispatcher</a></u>	<u><a href="#">getRequestDispatcher</a></u> (java.lang.String path)
--	---

Returns a RequestDispatcher object that acts as a wrapper for the resource located at the given path.  
取得 [RequestDispatcher](#) 的实例

demo:

```
package edu.gzu.servlet;

import java.io.IOException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ServerRedirectServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public ServerRedirectServlet() {
        super();
    }

    protected void doGet(HttpServletRequest request,
```

```
HttpServletResponse response) throws ServletException,  
IOException {  
  
    request.getSession().setAttribute("name", "hunk");// 设置一个  
Session 范围的属性  
  
    request.setAttribute("age", 32);// 设置一个 request 范围属性  
  
    RequestDispatcher rd =  
request.getRequestDispatcher("demo1.jsp");// 实例化  
RequestDispatcher 对象，并且指定跳转路径  
  
    rd.forward(request, response);// 服务器端跳转  
}  
  
  
  
protected void doPost(HttpServletRequest request,  
  
HttpServletResponse response) throws ServletException,  
IOException {  
  
    this doGet(request, response);  
}  
  
}
```

配置：

```
<servlet>  
  
<description></description>  
  
<display-name>ServerRedirectServlet</display-name>
```

```
<servlet-name>ServerRedirectServlet</servlet-name>
```

```
<servlet-class>edu.gzu.servlet.ServerRedirectServlet<
```

```
/servlet-class>
```

```
</servlet>
```

```
<servlet-mapping>
```

```
<servlet-name>ServerRedirectServlet</servlet-name>
```

```
<url-pattern>/ServerRedirectServlet</url-pattern>
```

```
</servlet-mapping>
```

运行结果：

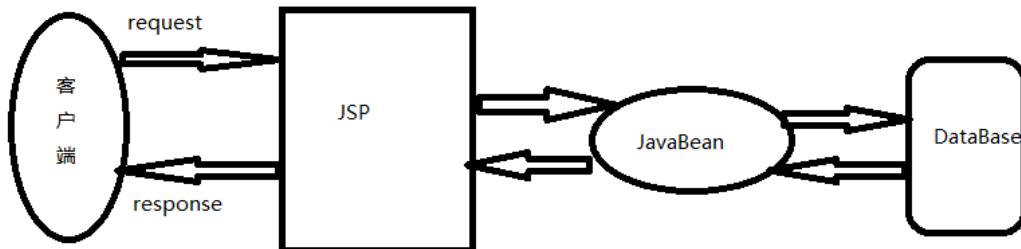


## 8.8 Web 开发模式

在实际的 Web 开发中，有两种模式一种叫 Model，另外一种叫 ModelII

### 8.8.1 Model

Model 就是指在开发中将显示层、控制层、数据层的操作同意交给 JSP 或 JavaBean 来进行处理。



Model 的处理情况分为两种，一种是完全是 JSP，另外一种是 JSP+JavaBean 的模式开发。

(一): 用户发出的请求交给 JSP 页面进行处理。如果是开发小型 Web 程序，为了快速与便利，通常是将显示层与逻辑运算层都写在 JSP 中。这种做法的优点是：1、开发速度快，程序员不需要编写 JavaBean 与 Servlet，只要写 JSP 就好。2、小幅度的修改代码还是比较方便的。缺点：程序的可读性低，因为 Java 代码和 HTML 代码混合在一起，读起来确实不方便。最要命的是程序的重复利用性很低。所有的程序代码全部写在了 JSP 中，并没有把常用的程序写成组件，代码繁杂，难以维护。

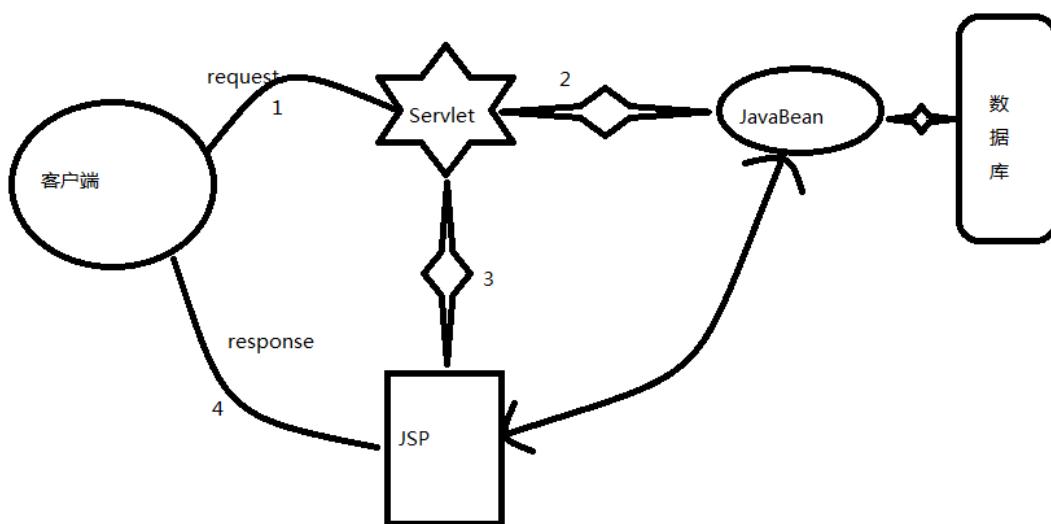
(二) JSP+JavaBean，JSP 主要负责页面显示层，而业务层全部写成了 JavaBean，将程序代码封装成组件。这样处理，JavaBean 就负责了大部分的数据处理。这种做法的优点是：1、程序的可读性高。因为逻辑性的代码全部放进了 JavaBean 中，不会和网页显示的代码混合在

一起，后期维护也比较轻松。2、可重复利用性相对较高，核心的业务代码是 JavaBean 完成的。缺点是：没有流程控制，程序中的每一个 JSP 都需要检查请求的参数是否正确、条件判断、异常发生时的处理。日后维护起来也并不方便。

总的来说，Model 是适合小型的程序开发，或者复杂度较低的程序开发。Model 最大的优点就是开发速度快。但是进行维护的时候要付出代价。

### 8.8.2 ModelII: Model-View-Controller

在 ModelII 中所有的开发都是以 Servlet 为主体展开的，由 Servlet 接收所有的客户端请求，然后根据相应的请求来调用相应的 JavaBean，并且将所有的显示结果交给 JSP 来完成，也就是俗称的 MVC 设计模式。



MVC 是一个设计模式，它强制性的将应用程序的输入、处理和输出分开。MVC 设计模式被分成 3 个核心层，即模型层、显示层、控制层。这三层分别处理自己的任务。

显示层(**view**): 主要负责接收 **Servlet** 传递的内容, 并且调用 **JavaBean**, 将内容显示给用户。

控制层 (**controller**): 主要负责所有的用户请求参数, 判断请求参数是否合法, 根据请求的类型调用 **JavaBean** 执行操作并将最终的处理结果交给显示层进行显示。

模型层(**model**): 完成一个独立的业务操作组件, 一般都是以 **JavaBean** 或者 **EJB** 的形式进行定义的。

**EJB** 属于 **SUN** 的分布式开发, 主要负责业务中心的编写, 分为会话 *bean*、实体 *bean* 和消息驱动 *bean* 3 种。

在 **MVC** 中, 最关键的是使用 **RequestDispatcher** 接口, 因为内容都是通过此接口保存到 **JSP** 页面上进行显示的。

当用户请求提出时, 所有的请求都会交给 **Servlet** 进行处理, 然后有 **Servlet** 调用 **JavaBean**, 并将 **JavaBean** 的操作结果通过 **RequestDispatcher** 接口传递到 **JSP** 页面。由于这些要显示的内容只在一次的请求-响应中有效, 所以在 **MVC** 设计模式中, 所有的属性传递都使用 **request** 属性范围传递, 这样可以提升代码的操作性能。

## 8.9 mvc 设计模式应用

用户登录: 用户输入登录信息提交给 **Servlet** 进行接收, **Servlet** 接收到请求之后先对其合法性进行验证, 如果验证失败, 则将错误信息传递给登录页面, 如果数据合法, 则调用 **DAO** 层完成数据库的验证, 根据验证的结果跳转到登录成功或失败页面。

- 1、 User JavaBean 用户登录的 VO 类
- 2、 DatabaseConnection JavaBean 负责数据库的连接和关闭
- 3、 UserDAO JavaBean 定义登录操作的接口
- 4、 UserDAOImpl JavaBean DAO 接口的真实实现类，完成具体的登录验证
- 5、 UserDAOProxy JavaBean 定义代理操作，负责是数据库的打开和关闭并且调用真实主题类
- 6、 DAOFactory JavaBean 工厂类，取得 DAO 接口的实例。
- 7、 LoginServlet Servlet 接受请求参数，进行参数验证，调用 DAO 完成具体的登录验证，并根据 DAO 的验证结果返回登录信息。
- 8、 login.jsp JSP 提供用户输入的表单，可以显示用户登录成功或失败的信息。

第一步：数据库设计

User 表： userid name password

第二步：定义 VO 类

```
package edu.gzu.vo;

public class User {
    private String userid;
    private String name;
    private String password;
```

```
public String getUserid() {  
    return userid;  
}  
  
public void setUserid(String userid) {  
    this.userid = userid;  
}  
  
public String getName() {  
    return name;  
}  
  
public void setName(String name) {  
    this.name = name;  
}  
  
public String getPassword() {  
    return password;  
}  
  
public void setPassword(String password) {  
    this.password = password;
```

```
}
```

```
}
```

第三步：定义 DatabaseConnection 类

```
package edu.gzudbc;
```

```
import java.sql.Connection;
```

```
import java.sql.DriverManager;
```

```
public class DatabaseConnection {
```

```
    private static final String DBDRIVER =
```

```
        "org.gjt.mm.mysql.Driver";
```

```
    private static final String DBURL =
```

```
        "jdbc:mysql://127.0.0.1:3306/school";
```

```
    private static final String DEUSER = "root";
```

```
    private static final String DBPASS = "123456";
```

```
    private Connection conn = null;
```

```
    // 在构造方法中进行数据库连接
```

```
    public DatabaseConnection() throws Exception {
```

```
try {

    Class.forName(DBDRIVER); // 加载驱动程序

    this.conn =
DriverManager.getConnection(DBURL, DEUSER,
DBPASS);

} catch (Exception e) {

    throw e;
}

}

// 取得数据库连接

public Connection getConnection() {

    return this.conn;
}

}

// 关闭数据库

public void close() throws Exception {

    if (this.conn != null) { // 避免空指针异常

        try {

            this.conn.close(); // 数据库关闭
        } catch (Exception e) {

            throw e;
        }
    }
}
```

}

第四步：定义 DAO 接口类 IUserDAO

```
package edu.gzu.dao;

import edu.gzu.vo.User;

public interface IUserDAO {
    public boolean findLogin(User user) throws
Exception;
}
```

第五步：定义 DAO 实现类

```
package edu.gzu.dao.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import edu.gzu.dao.IUserDAO;
```

```
import edu.gzu.vo.User;

public class UserDaoImpl implements IUserDAO {

    private Connection conn = null;
    private PreparedStatement ps = null;

    public UserDaoImpl(Connection conn) {
        this.conn = conn;
    }

    @Override
    public boolean findLogin(User user) throws
Exception {
        boolean flag = false;
        try {
            String sql = "SELECT name FROM user WHERE
userid = ? AND password = ?";
            this.ps =
this.conn.prepareStatement(sql);
            this.ps.setString(1, user.getUserid());
            this.ps.setString(2,
```

```
user.getPassword());  
  
    ResultSet rs = this.ps.executeQuery();  
  
    if (rs.next()) {  
  
        user.setName(rs.getString(1));  
  
        flag = true;  
  
    }  
  
    rs.close();  
  
} catch (Exception e) {  
  
    throw e;  
  
} finally {  
  
    if (this.ps != null) {  
  
        try {  
  
            this.ps.close();  
  
        } catch (Exception e) {  
  
            throw e;  
  
        }  
  
    }  
  
}  
  
  
return flag;  
}
```

```
}
```

第六步：定义 DAO 代理操作类

```
package edu.gzu.dao.proxy;

import edu.gzu.dao.IUserDAO;
import edu.gzu.dao.impl.UserDAOImpl;
import edu.gzu.jdbc.DatabaseConnection;
import edu.gzu.vo.User;

public class UserDAOProxy implements IUserDAO {
    private DatabaseConnection dbc = null;
    private IUserDAO dao = null;

    public UserDAOProxy() {
        try {
            thisdbc = new DatabaseConnection();
        } catch (Exception e) {
            e.printStackTrace();
        }
        thisdao = new
UserDAOImpl(thisdbc.getConnection());
    }
}
```

```
}

@Override
public boolean findLogin(User user) throws
Exception {
    boolean flag = false;
    try {
        flag = this.dao.findLogin(user);
    } catch (Exception e) {
        throw e;
    } finally {
        thisdbc.close();
    }
    return flag;
}

}
```

第七步：定义工厂类，取得 DAO 实例

```
package edu.gzu.factory;

import edu.gzu.dao.IUserDAO;
```

```
import edu.gzu.dao.proxy.UserDAOProxy;

public class DAOFactory {

    public static IUserDAO getInstance() {
        return new UserDAOProxy();
    }
}
```

上面 DAO 的操作仅仅是完成了数据层的操作，下面编写 Servlet，在 Servlet 接收客户端发来的数据，同时调用 DAO，并且要根据 DAO 的结果返回相应的信息。

#### 第八步：定义 LoginServlet

```
package edu.gzu.servlet;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;

import edu.gzu.factory.DAOFactory;
import edu.gzu.vo.User;

public class LoginServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public LoginServlet() {
        super();
    }

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws ServletException,
IOException {
        String path = "login.jsp";
        String userid = request.getParameter("userid");
        String password = request.getParameter("password");
        List<String> info = new ArrayList<String>();
        if (userid == null || "".equals(userid)) {
            info.add("用户 Id 不能为空");
        }
    }
}
```

```
if (password == null || "".equals(password)) {  
  
    info.add("密码不能为空");  
  
}  
  
if (info.size() == 0) {  
  
    User user = new User();  
  
    user.setUserId(userid);  
  
    user.setPassword(password);  
  
  
  
    try {  
  
        if (DAOFactory.getIUserDAOinstance().findLogin(user)) {  
  
            info.add("用户登录成功，欢迎" + user.getName() + "  
光临！");  
  
        } else {  
  
            info.add("用户登录失败，用户名或密码错误！");  
  
        }  
  
    } catch (Exception e) {  
  
        // TODO 自动生成的 catch 块  
  
        e.printStackTrace();  
  
    }  
  
}  
  
request.setAttribute("info", info);  
  
request.getRequestDispatcher(path).forward(request, response);
```

```
}

protected void doPost(HttpServletRequest request,
                      HttpServletResponse response) throws ServletException,
IOException {
    this doGet(request, response);
}

}
```

### 第九步：定义 logi.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ page import="java.util.*"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
```

```
<script type="text/javascript">

    function isTrue(f) {
        if (!(/^\w{5,15}$/, test(f.userid.value))) {
            alert("用户ID必须是5到15位");
            f.userid.focus();
            return false;
        }
        if (!(/^\w{5,15}$/, test(f.password.value))) {
            alert("用户ID必须是5到15位");
            f.userid.focus();
            return false;
        }
        return true;
    }

</script>

<body>

    <h2>用户登录</h2>

    <%
        request.setCharacterEncoding("utf-8");
    %>

    <%
        List<String> info =
```

```
(List<String>)request.getAttribute("info");

if(info != null){

    Iterator<String> iter = info.iterator();

    while(iter.hasNext()){

        %>

        <h4><%=iter.next()%></h4>

        <%
    }

}

%>

<form action="LoginServlet" method="post"

onsubmit="return isTrue(this)">

    userid:<input type="text" name="userid"><br>

    password:<input

        type="password" name="password"><br> <input

        type="submit">

</form>

</body>

</html>
```

## 9 表达式语言

MVC 的 demo 的话，其实在 JSP 中还存在大量的 java 代码。仍然在 JSP 中还需要导入一些 Java 类，为了让整个界面更加简洁，在开发中可以使用表达式语言。

### 9.1 表达式语言

表达式语言（EL, Expression Language）是在 JSP2.0 以后增加的功能。

使用表达式语言可以很方便的访问（page、request、session、application）中的属性内容。也就是说使用 EL，必须得把属性写在上面的标志位中。如果通过 EL 来访问属性，就会避免很多的 Scriptlet 代码。

语法：

`${属性名称}`

使用 EL 可以很方便的访问到对象中的属性，提交的参数或者进行数学运算。EL 的优点就是如果输出的内容为 null，就会自动使用""空字符串代替。

demo：不使用表达式语言

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
```

```
charset=UTF-8"

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<%
request.setAttribute("info", "时间过得好快啊，一转眼就10
月下旬了，离交JavaWeb作业只剩下10天了！蓝瘦，香菇！");
if(request.getAttribute("info")!=null){

%>

<h2><%=request.getAttribute("info") %></h2>

<%
}

%>

</body>

</html>
```

使用表达式语言修改上述代码：

```
<body>

<%
request.setAttribute("info",
```

"时间过得好快啊，一转眼就10月下旬了，离交JavaWeb作业只剩下10天了！蓝瘦，香菇！");

```
%>

<h2>${info }</h2>

</body>
```

## 9.2 表达式语言的内置对象

EL 的主要功能就是进行内容的显示，为了显示方便，在 EL 中提供了许多内置对象，通过不同的内置对象的设置，EL 可以输出不同的内容。

EL 常用的内置对象有：

pageContext 表示的就是 javax.servlet.jsp.PageContext 对象

pageScope 表示的是 page 属性范围

requestScope

sessionScope

applicationScope

param 接收传递到本页面的一个参数

paramValues 接收传递到本页面的一组参数

header 取得一个头信息数据

headerValues 取得一组头信息数据

cookie 取出 cookie 中的数据

initParam 取得配置的初始化参数

## 9.2.1 访问 4 种属性范围的内容

使用 EL 可以输出 4 种属性范围的内容，如果在不同的属性范围中设置相同的属性名称，如何取得？

demo：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<%
pageContext.setAttribute("info", "page属性范围");
request.setAttribute("info", "request");
session.setAttribute("info", "session");
application.setAttribute("info", "application");
%>
```

```
<h2>${info }</h2>  
  
</body>  
  
</html>
```

发现取得的属性范围的顺序是 page-» request-» session-» application。

如果你就想 4 个都输出，怎么办？

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
  
pageEncoding="UTF-8"%>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
  
<title>在此处插入标题</title>  
  
</head>  
  
<body>  
  
<%  
  
pageContext.setAttribute("info", "page属性范围");  
  
request.setAttribute("info", "request");  
  
session.setAttribute("info", "session");  
  
application.setAttribute("info", "application");
```

```
%>

<h2>${pageScope.info }</h2>

<h2>${requestScope.info }</h2>

<h2>${sessionScope.info }</h2>

<h2>${applicationScope.info }</h2>

</body>

</html>
```

## 9.2.2 调用内置对象操作

在介绍 JSP 内置对象的时候就说过 `pageContext` 内置对象可以取得 `request`、`session`、`application` 的实例。所以在 EL 中，可以通过 `pageContext` 这个表达式的内置对象调用 JSP 内置对象的方法。（反射机制）

demo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;

```

```
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<h3>IP地址: ${pageContext.request.remoteAddr }</h3>

<h2>Session ID:${pageContext.session.id }</h2>

<h1>是否是新的Session: ${pageContext.session.new}</h1>

</body>

</html>
```

### 9.2.3 接收请求参数

使用 EL 可以接受请求的参数。语法:

```
${param.参数名称}
```

demo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<html>

<head>

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

<meta http-equiv="Content-Type" content="text/html;
```

```
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<h2><%=request.getParameter("info") %></h2>

${param.info }

</body>

</html>
```

```
http://localhost:8080/Demo1016am/demo4.jsp?info=%E5%8F%AF%E8%
83%BD%E6%98%AF%E8%A6%81%E4%B8%8B%E8%AF%BE%E4%BA%86
%EF%BC%8C%E5%A4%A7%E5%AE%B6%E4%B8%8D%E6%89%93%E6%8%
B%9B%E5%91%BC%E5%B0%B1%E5%87%BA%E5%8E%BB%E4%BA%86
```



现在接受一组参数：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<form action="demo6.jsp" method="post">
兴趣: <input type="checkbox" name="inst" value="唱歌">
唱歌<br>
<input type="checkbox" name="inst" value="跳舞">跳舞<br>
<input type="checkbox" name="inst" value="足球">足球<br>
<input type="checkbox" name="inst" value="篮球">篮球<br>
<input type="checkbox" name="inst" value="A站">A站<br>
<input type="checkbox" name="inst" value="B站">B站<br>
<input type="submit">
</form>
```

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

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
  
pageEncoding="UTF-8"%>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
  
<title>在此处插入标题</title>  
  
</head>  
  
<body>  
  
<%request.setCharacterEncoding("utf-8"); %>  
  
<h2>第1个参数: ${paramValues.inst[0]}</h2>  
  
<h2>第2个参数: ${paramValues.inst[1]}</h2>  
  
<h2>第3个参数: ${paramValues.inst[2]}</h2>  
  
<h2>第4个参数: ${paramValues.inst[3]}</h2>  
  
<h2>第5个参数: ${paramValues.inst[4]}</h2>  
  
<h2>第6个参数: ${paramValues.inst[5]}</h2>
```

```
<h2>第7个参数: ${paramValues.inst[6]}</h2>
</body>
</html>
```

## 9.3 集合操作

输出 List:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

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

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<%
List all = new ArrayList();
```

```
all.add("hunk");
all.add("15335061230");
all.add("1417441001");
request.setAttribute("allinfo", all);
%>
<h2>第1个元素: ${allinfo[0]} </h2>
<h2>第2个元素: ${allinfo[1]} </h2>
<h2>第3个元素: ${allinfo[2]} </h2>
</body>
</html>
```

EL 只能访问保存在属性范围中的内容，所以要将集合保存在 request 范围中，使用 EL 输出的时候，直接用过集合的下标访问。

输出 Map:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<%@page import="java.util.*"%>
<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
```

```
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<%
Map map = new HashMap();

map.put("name", "hunk");

map.put("phone", "15335061230");

request.setAttribute("mapinfo", map);

%>

<h2>name:${mapinfo.name }</h2>

<h2>name:${mapinfo["name"] }</h2>

<h2>phone:${mapinfo.phone }</h2>

</body>

</html>
```

利用 Map 集合保存数据，在访问 Map 数据的时候，可以通过 key 找到相应的 value，在 EL 中，处理可以采用“.”之外，也可以采用“[]”来访问。

## 9.4 在 MVC 中应用 EL

EL 的强大功能还在于，可以直接通过发射的方式调用保存在属性范围中的 Java 对象内容。

定义个 VO 类：

```
package edu.gzu.vo;

public class Person {

    private String name;
    private String phone;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
}
```

取出内容：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<%@page import="edu.gzu.vo.Person"%>

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<%
Person p = new Person();
p.setName("hunk");
p.setPhone("15335061230");
request.setAttribute("pinfo", p);
%>
<h2>name:${pinfo.name }</h2>
<h2>phone:${pinfo.phone }</h2>
```

```
</body>
```

```
</html>
```

下面结合 MVC 来编写

写一个 Servlet 传递 request 属性：

```
package edu.gzu.servlet;

import java.io.IOException;

import javax.persistence.PersistenceContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import edu.gzu.vo.Person;

public class ELServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public ELServlet() {
        super();
    }
}
```

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

    Person p = new Person();
    p.setName("hunk");
    p.setPhone("15335061230");
    request.setAttribute("pinfo", p);
    request.getRequestDispatcher("demo10.jsp").forward(request,
response);

}

protected void doPost(HttpServletRequest request,
                      HttpServletResponse response) throws ServletException,
IOException {
    this.doGet(request, response);
}

}
```

demo10.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
```

```
Transitional//EN">

<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<h2>name:${pinfo.name }</h2>

<h2>phone:${pinfo.phone }</h2>

</body>

</html>
```

使用表达式语言后，在 Jsp 文件中并没有再导入 VO 包。

现在可以将一个对象的内容输出，但是在 DAO 中，如果执行的是查询的操作，则传到 JSP 中的将是一个对象的集合 List。那么如何输出？

之前学习 JSP 的时候，JSP 页面只做 3 件事，接收属性、判断、输出。

而且在 JSP 中最好只导入一个 util 包，所以要输出一个集合，还是要依靠 Iterator 完成，通过 Iterator 找出集合中的每一个元素。但是 EL 只能操作属性范围的内容，所以在使用了 iterator 找到一个内容之后

还需要将这个内容给他存放在 page 中，因为每一个要输出的内容只在本页中有效。然后再使用 EL 输出。

定义个 ELListServlet:

```
package edu.gzu.servlet;

import java.io.IOException;
import java.util.*;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import edu.gzu.vo.Person;

public class ELListServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public ELListServlet() {
        super();
    }
}
```

```
protected void doGet(HttpServletRequest
request,
                      HttpServletResponse response) throws
ServletException, IOException {
    List<Person> all = new ArrayList<Person>();
    Person p = null;
    p = new Person();
    p.setName("hunk");
    p.setPhone("15335061230");
    all.add(p);
    p = new Person();
    p.setName("何王科");
    p.setPhone("0513-68115983");
    all.add(p);
    request.setAttribute("pAll", all);

    request.getRequestDispatcher("demo11.jsp").forward(request, response);
}

protected void doPost(HttpServletRequest
request,
```

```
HttpServletResponse response) throws  
ServletException, IOException {  
    this.doGet(request, response);  
}  
}
```

demo11.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
<%@page import="java.util.Iterator"%>  
<%@page import="java.util.List"%>  
<html>  
<head>  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
pageEncoding="UTF-8"%>  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
<title>在此处插入标题</title>  
</head>
```

```
<body>

<%
List all = (List)request.getAttribute("pAll");

if(all != null){

    Iterator iter = all.iterator();

    %>

    <table border="1">

        <tr>

            <td>name</td>

            <td>phone</td>

        </tr>

        <%
        while(iter.hasNext()){

            pageContext.setAttribute("p", iter.next());

            %>

            <tr>

                <td>${p.name }</td>

                <td>${p.phone }</td>

            </tr>

            <%
        }

    }
}
```

```
%>

</table>

<%
}

%>

</body>

</html>
```

## 9.5 运算符

在表达式中为了方便用户的显示操作定义了许多算术运算符、关系运算符、逻辑运算符等，使用这些运算符将使得 JSP 页面更加简洁，但是对于太复杂的操作还是在 Servlet 或者 JavaBean 中完成。在使用这些运算符时，所有的操作内容也可以直接使用设置的属性，而不用考虑转型的问题。

算数运算符有 5 种：+、-、\*、/、%

demo：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

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

```
pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<%
    pageContext.setAttribute("num1", 20);
    pageContext.setAttribute("num2", 30);

%>

<h3>加法: ${num1 + num2 }</h3>
<h3>减法: ${num1 - num2 }</h3>
<h3>乘法: ${num1 * num2 }</h3>
<h3>除法: ${num1 / num2 }和${num1 div num2 }</h3>
<h3>取余法: ${num1 % num2 }和${num1 mod num2 }</h3>

</body>

</html>
```

关系运算符: ==, !=, <, >, <=, >=

demo:

```
<%
    pageContext.setAttribute("num1", 20);
    pageContext.setAttribute("num2", 30);
```

%>

<h3>相等: \${num1 == num2 }</h3>

<h3>不相等: \${num1 != num2 }</h3>

<h3>小于: \${num1 < num2 }</h3>

<h3>大于: \${num1 > num2 }</h3>

<h3>小于等于: \${num1 <= num2 }</h3>

<h3>大于等于: \${num1 >= num2 }</h3>

逻辑运算符: &&, ||, !

demo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<%
```

```
pageContext.setAttribute("flagA", true);  
  
pageContext.setAttribute("flagB", false);  
  
%>  
  
<h3>与: ${flagA && flagB}</h3>  
  
<h3>或: ${flagA || flagB}</h3>  
  
<h3>非: ${!flagA}</h3>  
  
</body>  
  
</html>
```

其他运算符: empty, ?:, ()

demo

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"    pageEncoding="UTF-8"%>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
  
<title>在此处插入标题</title>  
  
</head>  
  
<body>  
  
<%
```

```
pageContext.setAttribute("num1", 10);
pageContext.setAttribute("num2", 20);
pageContext.setAttribute("num3", 30);
%>
<h2>empty:${empty info}</h2>
<h2>三目:${num1>num2?"大于":"小于"}</h2>
<h2>括号:${num1*(num2+num3)}</h2>
</body>
</html>
```

## 预习：JSTL

# 10 JSP 标准标签库

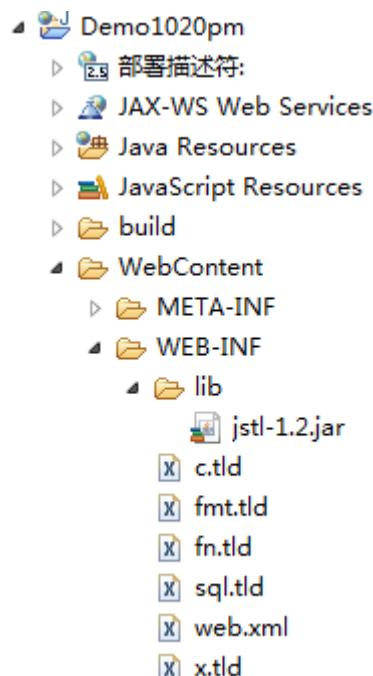
## 10.1 JSTL 简介

JSTL (JSP Standard Tag Library) 是一个开源的标签组件，由 Apache 的 jakarta 小组开发的。可以直接在 <http://tomcat.apache.org/taglibs> 下载。

目前 JSTL 主要支持标签库有核心标签库 (c)、SQL 标签库 (sql)、XML 标签库 (xml)、函数标签库 (fn) 以及 I18N 格式标签库 (fmt)。

## 10.2 安装 JSTL 1.2

下载的 JSTL 是以 jar 包的形式存在的，直接将此 jar 包保存在 WEB-INF\lib 目录中，之后直接解压这个 jar 包，并将其中的几个主要的标签配置文件 (c.tld、sql.tld、x.tld、fn.tld、fmt.tld) 保存在 WEB-INF 文件夹中。此时，就安装好了。



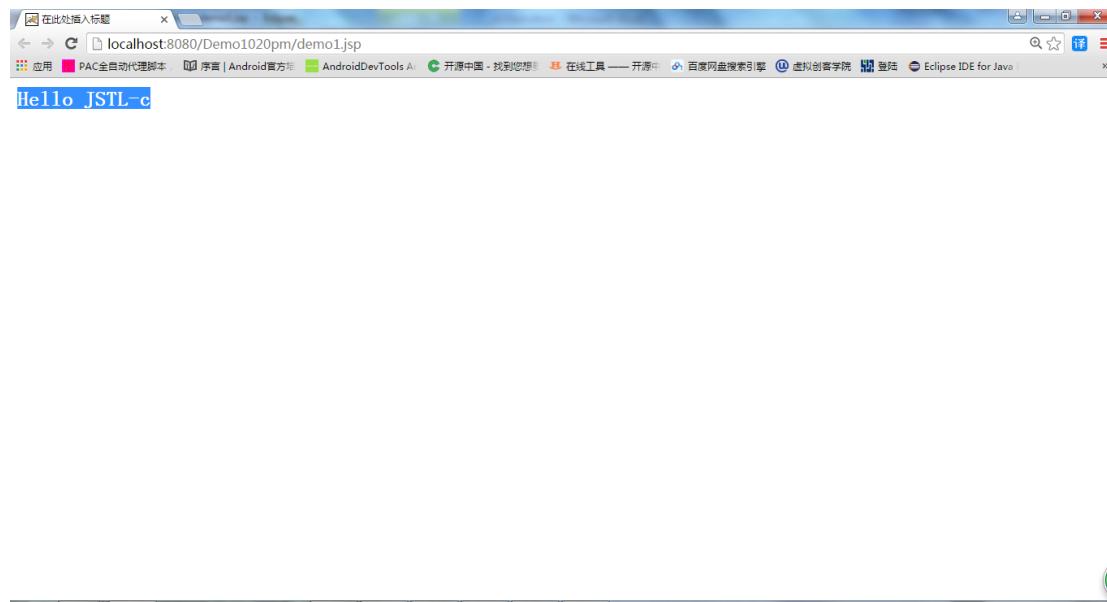
测试：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<%@ taglib uri="/WEB-INF/c.tld" prefix="c" %>

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<h2><c:out value="Hello JSTL-c"></c:out></h2>
</body>
</html>
```

运行结果：



每次写 jsp 之前都需要导入 c.tld,我们可以配置一下 Web.xml 中文件。

```
<jsp-config>
  <taglib>
    <taglib-uri>c</taglib-uri>
    <taglib-location>/WEB-INF/c.tld</taglib-location>
  </taglib>
</jsp-config>
```

### 10.3 核心标签库

核心标签库是 JSTL 中最重要的部分，在开发中也最长使用，c 标签库完成的有流程控制、迭代输出。

序号	功能分类	标签名称	描述
1	基本标签	<c:out>	输出属性内容
2		<c:set>	设置属性内容
3		<c:remove>	删除指定属性

4		<c:catch>	异常处理
5	流程控制标签	<c:if>	条件判断
6		<c:choose>	多条件判断，可以设置<c:when>和<c:otherwise>
7	迭代标签	<c:forEach>	输出数组、集合
8		<c:forTokens>	字符串拆分及输出
9	包含标签	<c:import>	将一个指定的路径包含到当前页面进行显示
10	生成 URL 标签	<c:url>	根据路径和参数生成一个新的 URL
11	客户端跳转	<c:redirect>	客户端跳转

### 10.3.1 <c:out>

<c:out>输出内容，语法：

有标签体：

```
<c:out values="打印输出的内容" [escapeXml="true/false"]>
    标签体:默认值
</c:out>
```

无标签体：

```
<c:out values="打印输出的内容" [escapeXml="true/false"] [default="默
    认值"]/>
```

- **values:** 设置打印输出的内容。支持 EL

- escapeXml: 是否转换字符串，> ➔&gt;,默认值是 true。支持 EL
- default: 如果 values 为空，则显示 default 定义的内容。支持 EL

**demo:**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<%@ taglib uri="/WEB-INF/c.tld" prefix="c" %>

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<%
pageContext.setAttribute("info", "Hello,JSTL!");
%>
<h2><c:out value="${info }">你好</c:out></h2>
<h2><c:out value="${ref }">你好</c:out></h2>
```

```
<h2><c:out value="${ref }" default="我好"/></h2>  
</body>  
</html>
```

### 10.3.2<c:set>

<c:set>主要是用来将属性保存在 4 种属性范围中。

语法：

没有标签体：

```
<c:set var=" 属性名称 " value=" 属性内容 "  
[scope="[page|request|session|application]"]/>  
  
<c:set value="属性内容" target="属性名称" property="属性名称"/>
```

有标签体：

```
<c:set var="属性名称" [scope="[page|request|session|application]"]>  
属性内容</c:set>  
  
<c:set target="属性名称" property="属性名称">  
属性内容  
</c:set>
```

#### <c:set>标签的属性

- **var:** 设置属性名称，不支持 EL
- **value:** 设置属性内容，如果为 null 则表示删除属性，支持 EL
- **scope:** 设置属性的保存范围，不支持 EL
- **target:** 存储的目标属性，支持 EL， request 里面的属性名称

- **property:** 指定的 target 属性支持 EL, javaBean 中的属性名称

**demo:**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<%@ taglib uri="/WEB-INF/c.tld" prefix="c" %>  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
  
pageEncoding="UTF-8"%>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
  
<title>在此处插入标题</title>  
  
</head>  
  
<body>  
  
<c:set var="info" value="Hello" scope="request"/>  
  
<h2>属性内容: ${info}</h2>  
  
</body>  
  
</html>
```

还可以通过<c:set>将内容设置到一个 JavaBean 中的属性, 这个时候我们就要借助 target 和 property。

定义一个 JavaBean:

```
package edu.gzu.vo;

public class SimpleInfo {

    private String info;

    public String getInfo() {
        return info;
    }

    public void setInfo(String info) {
        this.info = info;
    }
}
```

demo3.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<%@page import="edu.gzu.vo.SimpleInfo"%>

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

<html>
<head>

<%@ page language="java" contentType="text/html;

```

```
charset=UTF-8"

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<%
SimpleInfo si = new SimpleInfo();
request.setAttribute("sinfo", si);
%>

<c:set value="hello" target="${sinfo }"
property="info"/>

<!--<c:set target="${sinfo }"
property="info">hello</c:set>-->

<h3>属性内容: ${sinfo.info_}</h3>

</body>

</html>
```

### 10.3.3<c:remove>标签

<c:remove>标签在程序中的主要作用是删除指定范围中的属性，功能与 removeAttribute()方法类似。

语法：

```
<c:remove var="属性名称" [scope="[page|request|session|application]"/>
```

属性：

var：要删除的属性的名称，必须指定，不支持 EL

scope：删除属性的保存范围，默认保存在 page 中，不支持 EL

demo：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

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

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<c:set var="info" value="Hello" scope="request"/>
<c:remove var="info" scope="request"/>
```

```
<h2>属性内容: ${info_}</h2>  
</body>  
</html>
```

### 10.3.4<c:catch>标签

<c:catch>标签主要用来处理程序中产生的异常，并进行相关的异常处理，语法如下：

```
<c:catch var="保存异常信息的属性名称">  
有可能发生异常的语句  
</c:catch>
```

属性 var 不支持 EL

**demo:**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
<%@ taglib uri="c" prefix="c" %>  
<html>  
<head>  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
pageEncoding="UTF-8"%>  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">
```

```
<title>在此处插入标题</title>

</head>

<body>

<c:catch var="errmsg">

<%
int result = 10/0;

%>

</c:catch>

<h2>异常信息: ${errmsg}</h2>

</body>

</html>
```

### 10.3.5<c:if>标签

<c:if>标签主要用来判断语句，功能与在程序中使用 if 语法一样，语法：

没有标签体：

```
<c:if test=" 判断条件 " var=" 存储判断结果 "
[scope="[page|request|session|application]"/>
```

有标签体：

```
<c:if test=" 判断条件 " var=" 存储判断结果 "
[scope="[page|request|session|application]"/>
```

满足条件时执行的语句

```
</c:if>
```

**属性:**

- **test:** 用于判断条件, 如果条件为 true, 则执行标签体里面的内容, 支持 EL;
- **var:** 保存判断的结果, 不支持 EL
- **scope:** 指定判断结果的保存范围, 默认值是 page, 不支持 EL

**demo:**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

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

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<c:if test="${param.ref == 'gzu' }" var="res1">
```

```
scope="page">

<h3>欢迎${param.ref_}光临!</h3>

</c:if>

</body>

</html>
```

http://localhost:8080/Demo1020pm/demo6.jsp?ref=gzu

### 10.3.6<c:choose>、<c:when>、<c:otherwise>标签

<c:if>标签只能针对一个条件的判断，如果要同时判断多个条件可以使用<c:choose>标签。但是<c:choose>标签只能作为<c:when>和<c:otherwise>的父标签出现。语法：

```
<c:choose>

标签体内容 (<c:when>和<c:otherwise>)

</c:choose>
```

在本标签中没有任何的属性内容，而且本标签中的内容只能有以下的内容：

**<c:when>**：可以出现一次或多次，用于进行条件判断。

**<c:otherwise>**：可以出现0此或多次，用于所有条件都不满足时的操作。（与switch中的default语句的功能类似）

<c:when>与<c:if>类似，都需要通过test进行条件判断，<c:when>的语法如下：

```
<c:when test="判断条件">
```

满足条件时执行的语句

```
</c:when>
```

属性 **test** 用于判断条件，若条件为 **true**，则执行标签体的内容。支持 EL。

**<c:otherwise>**语法：

```
<c:otherwise>
```

当所有**<c:when>**都不满足条件时，执行本标签体内容。

```
</c:otherwise>
```

**demo:**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

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

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
```

```
<c:choose>

<c:when test="#{param.num == 10}">
    <h3>num:10</h3>
</c:when>

<c:when test="#{param.num == 20}">
    <h3>num:20</h3>
</c:when>

<c:otherwise>
    <h3>总是没有满足条件的。</h3>
</c:otherwise>
</c:choose>
</body>
</html>
```

<http://localhost:8080/Demo1020pm/demo7.jsp?num=201>

注意：`<c:otherwise>`必须放在`<c:when>`后面。

### 10.3.7<c:forEach>标签

`<c:forEach>`标签的主要功能是循环控制。可以将集合中的成员进行迭代输出，功能与 `Iterator` 接口类似。语法：

```
<c:forEach [var="每一个对象的属性名称" items="集合" [varStatus="保存相关成员信息"] [begin="集合的开始输出位置"] [end="集合的结束输出位置"] [step="每次增长的步长"]]>
```

标签体

```
</c:forEach>
```

属性：

- **var**: 用来存放集合中的每一个对象，不支持 EL
- **items**: 保存所有的集合，主要是数组、List、Set 及 Map，支持 EL
- **varStatus**: 用于存放当前对象的成员信息，不支持 EL
- **begin**: 集合的开始位置，默认是 0，支持 EL
- **end**: 集合的结束位置，默认是集合的最后一个元素
- **step**: 每次迭代的间隔数，默认是 1

**demo:**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

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

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
```

```
<body>

<%

String info[] = { "周一", "周二", "周三" };

pageContext.setAttribute("ref", info);

%>

<h3>

输出全部:

<c:forEach items="${ref }" var="xingqi">

${xingqi }、

</c:forEach>

</h3>

<h2>

间隔为2的输出全部:

<c:forEach items="${ref }" var="xingqi" step="2">

${xingqi }、

</c:forEach>

</h2>

<h1>输出前俩个:

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

${xingqi }、

</c:forEach>
```

```
</h1>

</body>

</html>
```

demo：输出List

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<%@page import="java.util.ArrayList"%>
<%@page import="java.util.List"%>
<%@ taglib uri="c" prefix="c"%>

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<%
//String info[] = { "周一", "周二", "周三" };
List all = new ArrayList();
```

```
all.add("周一");
all.add("周2");
all.add("周3");

pageContext.setAttribute("ref", all);

%>

<h3>
    输出全部:
    <c:forEach items="${ref }" var="xingqi">
        ${xingqi }、
    </c:forEach>
</h3>

<h2>
    间隔为2的输出全部:
    <c:forEach items="${ref }" var="xingqi" step="2">
        ${xingqi }、
    </c:forEach>
</h2>

<h1>输出前俩个:
    <c:forEach items="${ref }" var="xingqi" begin="0"
end="1">
        ${xingqi }、
    </c:forEach>
```

```
</h1>  
  
</body>  
  
</html>
```

demo输出Map

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
  
<%@page import="java.util.HashMap"%>  
  
<%@page import="java.util.Map"%>  
  
<%@page import="java.util.ArrayList"%>  
  
<%@page import="java.util.List"%>  
  
<%@ taglib uri="c" prefix="c"%>  
  
<html>  
  
<head>  
  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
  
pageEncoding="UTF-8"%>  
  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">  
  
<title>在此处插入标题</title>  
  
</head>  
  
<body>  
  
<%
```

```
//String info[] = { "周一", "周二", "周三" };

//List all = new ArrayList();

//all.add("周一");

//all.add("周2");

//all.add("周3");

Map m = new HashMap();

m.put("xingqi1", "周1");

m.put("xingqi2", "周2");

m.put("xingqi3", "周3");

pageContext.setAttribute("ref", m);

%>

<h3>

    输出全部:

    <c:forEach items="${ref }" var="xingqi">

${xingqi.key }-->${xingqi.value }

</c:forEach>

</h3>

</body>

</html>
```

### 10.3.8<c:forTokens>标签

<c:forTokens>标签也是用于输出，类似于 String 类中的 split 方法和循

环输出的一个集合。语法：

```
<c:forTokens items="输出的字符串" delims="字符串分隔符" [var="存放每一个字符串变量"] [varStatus="存放当前对象的相关信息"]  
[begin="输出位置"] [end="结束位置"] [step="输出间隔"]>  
标签体内容  
</c:forTokens>
```

- var: 用来存放集合中的每一个对象，不支持 EL
- items: 要输出的字符串，支持 EL
- delims: 定义分割字符串的内容，支持 EL
- varStatus: 用于存放当前对象的成员信息，不支持 EL
- begin: 开始的输出位置，默认是 0，支持 EL
- end: 结束的输出位置，默认是集合的最后一个成员
- step: 每次迭代的间隔数，默认是 1

**demo:**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">  
<%@ taglib uri="c" prefix="c" %>  
<html>  
<head>  
<%@ page language="java" contentType="text/html;  
charset=UTF-8"  
pageEncoding="UTF-8"%>
```

```
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<%
String s = "www.gzu.edu.cn";
pageContext.setAttribute("ref", s);

%>

<h3>

拆分结果是:

<c:forTokens items="${ref }" delims="." var="con">
${con}_、
</c:forTokens>

</h3>

</body>

</html>
```

### 10.3.9<c:import>标签

<c:import>标签可以将其他页面的内容包含进来一起显示。与<jsp:include>类似，但是<c:import>还可以包含外部的页面。语法：

```
<c:import url="包含地址的 URL" [context="上下文路径"] [var="保存内"]>
```

```
容的属性名称"] [scope="page|request|session|application"]
[charEncoding="字符编码"] [varReader="以 Reader 方式读取内容"]>
标签体内容
[<c:param name="参数名称" value="参数内容"/>]
</c:import>
```

### 属性:

**url:** 指定要包含文件的路径, 支持 EL (不能设置成 null, 否则会出现 JasperException)

**cotext:** 如果要访问在同一个 Web 容器下的其他资源时设置, 必须以"/"开头。

**var:** 存储导入的文件内容, 不支持 EL

**scope:** 定义 var 的保存范围, 默认为 page, 不支持 EL

**charEncoding:** 定义字符编码, 支持 EL

**varReader:** 储存导入文件的内容, 以 Reader 类型存入, 不支持 EL

### demo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

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

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

```
pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<c:import url="http://www.gzu.edu.cn"
charEncoding="utf-8"/>

</body>

</html>
```

上述程序通过<c:import>将贵大的站点的内容导入进行显示，但是只是将 HTML 代码导入进来了，而图片等相关资源都会出现显示问题。如果现在要导入的页面中可以接受参数，则可以使用它的子标签<c:param>进行传递。

**demo:**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<html>

<head>

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

pageEncoding="UTF-8"%>
```

```
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<h2>name:${param.name }</h2>

<h2>age:${param.age }</h2>

</body>

</html>
```

demo12:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

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

<html>

<head>

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

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>
```

```
<body>

<c:import url="demo11.jsp" charEncoding="utf-8">

<c:param name="name" value="hunk"/>
<c:param name="age" value="31"/>

</c:import>

</body>

</html>
```

### 10.3.10< c:url >标签

< c:url >标签可以直接在产生一个 URL，语法：

```
<c:url value="操作的 URL" [context="上下文路径"] [var="保存的属性名
称"] [scope="[page|rquest|session|application]"]>
<c:param name="参数名称" value="参数值"/>
</c:url>
```

```
<c:url value="操作的 URL" [context="上下文路径"] [var="保存的属性名
称"] [scope="[page|rquest|session|application]"]/>
```

**属性：**

**value:** 要执行的 URL，支持 EL

**context:** 如果要访问在同一个 Web 容器下的其他资源时设置，必须以/开头。支持 EL

**var:** 存储导入的文件内容，不支持 EL

scope: 定义 var 的保存范围, 默认是 page, 不支持 EL

**demo:**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

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

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<c:url value="http://gzu.gaoxiaobang.com" var="info">
<c:param name="username" value="hewangke0502@126.com"/>
<c:param name="password" value="123456"/>
</c:url>
<a href="${info }">新的URL</a>
</body>
</html>
```



### 10.3.11< c:redirect>标签

在 JSP 中学习过 `request.sendRedirect()` 进行客户端跳转，`<c:redirect>` 标签与之类似。语法：

```
<c:redirect url="跳转的地址" context="上下文路径"/>  
  
<c:redirect url="跳转的地址" context="上下文路径">  
  <c:param name="参数名称" value="参数值"/>  
</c:redirect>
```

**属性：**

**url:** 跳转的地址，支持 EL

**context:** 如果要访问在同一个 Web 容器下的其他资源时设置，必须以 / 开头。支持 EL

**demo:**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01  
Transitional//EN">
```

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

<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<c:redirect url="http://www.baidu.com">

<c:param name="key" value="贵州大学"/>

</c:redirect>

</body>

</html>
```

本标签完成了客户端跳转，并且传递了两个参数，由于是客户端跳转，所以跳转之后地址栏发生了改变。

# 11 分页

## 11.1 JSP 中分页实现

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<%@page import="java.sql.*"%>

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<%
public static final String DBDRIVER =
"org.gjt.mm.mysql.Driver" ;
public static final String DBURL =
"jdbc:mysql://127.0.0.1:3306/school" ;
public static final String DBUSER = "root" ;
```

```
public static final String DBPASS = "123456" ;  
  
%>  
  
<%  
  
Connection conn = null ;  
  
PreparedStatement ps = null ;  
  
ResultSet rs = null ;  
  
%>  
  
<%  
  
int currentPage = 1 ; // 为当前所在的页， 默认在第1页  
  
int lineSize = 3 ; // 每次显示的记录数  
  
int allRecorders = 0 ; // 表示全部的记录数  
  
int pageSize = 1 ; // 表示全部的页数（尾页）  
  
int lsData[] = {1,3,5,7,9,10,15,20,25,30,50,100} ;  
  
%>  
  
<%  
  
try{  
  
    currentPage =  
  
Integer.parseInt(request.getParameter("cp")) ;  
  
} catch(Exception e) {}  
  
try{  
  
    lineSize =  
  
Integer.parseInt(request.getParameter("ls")) ;
```

```
        } catch(Exception e) {}

%>

<%

Class.forName(DBDRIVER) ;

conn = DriverManager.getConnection(DBURL, DBUSER,
DBPASS) ;

String sql = "SELECT COUNT(empno) FROM emp" ;

ps = conn.prepareStatement(sql) ;

rs = ps.executeQuery() ;

if(rs.next()){ // 取得全部的记录数

    allRecorders = rs.getInt(1) ;

}

%>

<center>

<h1>雇员列表</h1>

<script type="text/javascript">

function go(num){

    document.getElementById("cp").value = num ;

    document.spform.submit() ; // 表单提交

}

</script>

<%
```

```
pageSize = (allRecorders + lineSize -1) / lineSize ;  
  
%>  
  
<%  
  
    sql = "SELECT empno,ename,job,hiredate,sal FROM emp  
limit ? , ?";  
  
    ps = conn.prepareStatement(sql) ;  
  
    ps.setInt(1,(currentPage-1) * lineSize) ;  
  
    ps.setInt(2,lineSize) ;  
  
    rs = ps.executeQuery() ;  
  
%>  
  
<form name="spform" action="myfenye.jsp" method="post">  
  
    <input type="button" value="首页" onclick="go(1)"  
    <%=currentPage==1?"DISABLED":""%>>  
  
    <input type="button" value="上一页"  
    onclick="go(<%=currentPage-1%>)"  
    <%=currentPage==1?"DISABLED":""%>>  
  
    <input type="button" value="下一页"  
    onclick="go(<%=currentPage+1%>)"  
    <%=currentPage==pageSize?"DISABLED":""%>>  
  
    <input type="button" value="尾页"  
    onclick="go(<%=pageSize%>)"  
    <%=currentPage==pageSize?"DISABLED":""%>>
```

```
跳转到第<select id="cp" name="cp"
onchange="go(this.value)">

<%
for(int x=1;x<=pageSize;x++){

%>

<option value="<%="x%">">
<%=x==currentPage?"SELECTED":""%><%=x%"></option>

<%
}

%>

</select>页

每页显示

<select name="ls" onchange="go(1)">

<%
for(int x=0;x<lsData.length;x++){

%>

<option value="<%="lsData[x]"%>">
<%=lsData[x]==lineSize?"SELECTED":""%><%=lsData[x]"%>

</option>

<%
}

%>
```

```
</select>

条

<input type="hidden" name="selcp" value="1">

</form>

<TABLE BORDER="1" cellpadding="5" cellspacing="0"
bgcolor="#F2F2F2" width="100%">

<TR onMouseOver="changeColor(this,'white')"
onMouseOut="changeColor(this,'F2F2F2')">

    <td align="center" valign="middle"><span
class="STYLE10">编号</span></td>

    <td align="center" valign="middle"><span
class="STYLE10">姓名</span></td>

    <td align="center" valign="middle"><span
class="STYLE10">职位</span></td>

    <td align="center" valign="middle"><span
class="STYLE10">雇佣日期</span></td>

    <td align="center" valign="middle"><span
class="STYLE10">工资</span></td>

    </td>
</TR>

<%
while(rs.next()){


```

```
int empno = rs.getInt(1) ;  
  
String ename = rs.getString(2) ;  
  
String job = rs.getString(3) ;  
  
Date hiredate = rs.getDate(4) ;  
  
double sal = rs.getDouble(5) ;  
  
%>  
  
<TR onMouseOver="changeColor(this,'white')"  
onMouseOut="changeColor(this,'F2F2F2')">  
  
    <td align="center" valign="middle"><span  
class="STYLE6"><%=empno%></span></td>  
  
    <td align="center" valign="middle"><span  
class="STYLE6"><%=ename%></span></td>  
  
    <td align="center" valign="middle"><span  
class="STYLE6"><%=job%></span></td>  
  
    <td align="center" valign="middle"><span  
class="STYLE6"><%=hiredate%></span></td>  
  
    <td align="center" valign="middle"><span  
class="STYLE6"><%=sal%></span></td>  
  
</TR>  
<%
```

```
}

%>

</table>

<%
    conn.close() ;
%>

</center>

</body>

</html>
```

## 11.2 作业：抽象出分页控件

## 12Session 购物车

用户登录到服务器上打开所有的产品列表，然后选择将产品添加到购物车，所有已购买的产品将在用户的购物车中列出。

### 12.1product 表：

```
CREATE TABLE product(
    pid      INT      PRIMARY KEY ,
    pname    VARCHAR(50) NOT NULL ,
    note     VARCHAR(500)      ,
    price    FLOAT      NOT NULL ,
    amount   INT
);

-- 插入测试数据

INSERT      INTO      product(pid, pname, note, price, amount)      VALUES
(101, 'Oracle 数据库开发', '基本 SQL、DBA 入门', 69.8, 30);

INSERT      INTO      product(pid, pname, note, price, amount)      VALUES (102, 'Java
开发', 'Java 入门书籍', 79.8, 30);

INSERT      INTO      product(pid, pname, note, price, amount)      VALUES (103, 'Java
WEB 开发', 'JSP、Servlet、AJAX、Struts', 99.8, 20);

INSERT      INTO      product(pid, pname, note, price, amount)      VALUES
(104, 'Spring 开发手册', 'Spring、MVC、标签', 57.9, 20);
```

```
INSERT INTO product(pid, pname, note, price, amount) VALUES
(105, 'Hibernate 实战精讲', 'ORMapping', 87.3, 10);

INSERT INTO product(pid, pname, note, price, amount) VALUES (106, 'Struts
2.0 权威开发', 'WebWork、Struts 2.0', 70.3, 23);

INSERT INTO product(pid, pname, note, price, amount) VALUES (107, 'SQL
Server 指南', 'SQL Server 数据库', 29.8, 11);

INSERT INTO product(pid, pname, note, price, amount) VALUES
(108, 'windows 指南', '基本使用', 23.2, 20);

INSERT INTO product(pid, pname, note, price, amount) VALUES (109, 'linux
操作系统', '原理、内核、基本命令', 37.9, 10);

INSERT INTO product(pid, pname, note, price, amount) VALUES (110, '企业
开发架构', '企业开发原理、成本、分析', 109.5, 20);

INSERT INTO product(pid, pname, note, price, amount) VALUES (111, '分布
式开发', 'RMI、EJB、Web 服务', 200.8, 10);

INSERT INTO product(pid, pname, note, price, amount) VALUES (112, 'SEAM
(JSF + EJB 3.0)', 'JSF、SEAM、EJB 3.0', 80.2, 15);
```

## 12.2image.jsp 验证码页面：

```
<%@ page contentType="image/jpeg"
import="java.awt.*, java.awt.image.*, java.util.*, javax
.imageio.*" pageEncoding="utf-8"%>
<%!
```

```
Color getRandColor(int fc,int bc){//给定范围获得随机颜色

    Random random = new Random();

    if(fc>255) fc=255;

    if(bc>255) bc=255;

    int r=fc+random.nextInt(bc-fc);

    int g=fc+random.nextInt(bc-fc);

    int b=fc+random.nextInt(bc-fc);

    return new Color(r,g,b);

}

%>

<%

//设置页面不缓存

response.setHeader("Pragma","No-cache");

response.setHeader("Cache-Control","no-cache");

response.setDateHeader("Expires", 0);

// 在内存中创建图象

// 通过这里可以修改图片大小

int width=80, height=25;

BufferedImage image = new BufferedImage(width, height,

BufferedImage.TYPE_INT_RGB);
```

```
// 获取图形上下文  
  
// g相当于笔  
  
Graphics g = image.getGraphics();  
  
  
//生成随机类  
  
Random random = new Random();  
  
  
// 设定背景色  
  
g.setColor(getRandColor(200,250));  
  
// 画一个实心的长方，作为北京  
  
g.fillRect(0, 0, width, height);  
  
  
//设定字体  
  
g.setFont(new Font("宋体",Font.PLAIN,18));  
  
  
//画边框  
  
//g.setColor(new Color());  
  
//g.drawRect(0,0,width-1,height-1);  
  
  
// 随机产生155条干扰线，使图象中的认证码不易被其它程序探测到
```

```
g.setColor(getRandColor(160, 200));  
  
for (int i=0;i<155;i++)  
{  
  
    int x = random.nextInt(width);  
  
    int y = random.nextInt(height);  
  
    int xl = random.nextInt(12);  
  
    int yl = random.nextInt(12);  
  
    g.drawLine(x,y,x+xl,y+yl);  
  
}  
  
  
  
// 取随机产生的验证码(4位数字)  
  
//String rand = request.getParameter("rand");  
  
//rand = rand.substring(0,rand.indexOf("."));  
  
String sRand="";  
  
// 如果要使用中文，必须定义字库，可以使用数组进行定义  
  
// 这里直接写中文会出乱码，必须将中文转换为unicode编码  
  
String[] str =  
  
{"A", "B", "C", "D", "E", "F", "G", "H", "J", "K", "L", "M", "N",  
"P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z", "a", "b", "  
c", "d", "e", "f", "g", "h", "i", "j", "k", "m", "n", "p", "s", "t  
", "u", "v", "w", "x", "y", "z", "1", "2", "3", "4", "5", "6", "7"  
, "8", "9"} ;
```

```
for (int i=0;i<4;i++){  
  
    String rand=str[random.nextInt(str.length)];  
  
    sRand+=rand;  
  
    // 将验证码显示到图象中  
  
    g.setColor(new  
  
Color(20+random.nextInt(110),20+random.nextInt(110),2  
0+random.nextInt(110))); //调用函数出来的颜色相同，可能是  
因为种子太接近，所以只能直接生成  
  
    g.drawString(rand,16*i+6,19);  
  
}  
  
  
// 将验证码存入SESSION  
  
session.setAttribute("rand",sRand);  
  
  
  
  
// 图象生效  
  
g.dispose();  
  
  
  
  
// 输出图象到页面  
  
ImageIO.write(image, "JPEG",
```

```
response.getOutputStream();

out.clear();

out = pageContext.pushBody();

%>
```

## 12.3split\_page\_plugin.jsp 分页页面

```
<%@ page contentType="text/html" pageEncoding="utf-8"%>

<%-- 包含以下的内容即可完成分页

<jsp:include page="split_page_plugin.jsp">

    <jsp:param name="allRecorders"
value="<%=allRecorders%>" />

    <jsp:param name="url" value="<%=URL%>" />

    <jsp:param name="currentPage"
value="<%=currentPage%>" />

    <jsp:param name="lineSize" value="<%=lineSize%>" />

</jsp:include>

--%>

<%
    int currentPage = 1 ; // 为当前所在的页， 默认在第1页
    int lineSize = 15 ;   // 每次显示的记录数
    long allRecorders = 0 ; // 表示全部的记录数
```

```
long pageSize = 1 ; // 表示全部的页数（尾页）

int lsData[] = {1,3,5,7,9,10,15,20,25,30,50,100} ;

String keyword = request.getParameter("kw") ; // 接
收查询关键字

String url = request.getParameter("url") ;

%>

<%
try{

    currentPage =
Integer.parseInt(request.getParameter("currentPage"))

;

} catch(Exception e) {}

try{

    lineSize =
Integer.parseInt(request.getParameter("lineSize")) ;

} catch(Exception e) {}

System.out.println(lineSize) ;

try{

    allRecorders =
Long.parseLong(request.getParameter("allRecorders")) ;

} catch(Exception e) {}

if(keyword == null){
```

```
keyWord = "" ; // 如果模糊查询没有关键字，则表示查
询全部

}

%>

<%
pageSize = (allRecorders + lineSize -1) / lineSize ;

if(pageSize == 0){

    pageSize = 1 ;

}

%>

<script language="javascript">

function go(num){

    document.getElementById("cp").value = num ;

    document.spform.submit() ; // 表单提交

}

</script>

<form name="spform" action="<%=url%>" method="post">

输入查询关键字: <input type="text" name="kw"
value="<%=keyWord%>">

<input type="submit" value="查询"><br>

<input type="button" value="首页" onclick="go(1)">
```

```
<%=currentPage==1?"DISABLED":""%>>

<input type="button" value="上一页"
onclick="go(<%=currentPage-1%>)">

<%=currentPage==1?"DISABLED":""%>>

<input type="button" value="下一页"
onclick="go(<%=currentPage+1%>)">

<%=currentPage==pageSize?"DISABLED":""%>>

<input type="button" value="尾页"
onclick="go(<%=pageSize%>)">

<%=currentPage==pageSize?"DISABLED":""%>>

跳转到第<select id="cp" name="cp"
onchange="go(this.value)">

<%
for(int x=1;x<=pageSize;x++){

%>

<option value=<%=x%>">

<%=x==currentPage?"SELECTED":""%><%=x%></option>

<%
}

%>

</select>页

每页显示
```

```
<select name="ls" onchange="go(1)">

<%
    for(int x=0;x<lsData.length;x++){

%>

    <option value="<%="lsData[x]%">">
<%=lsData[x]==lineSize?"SELECTED":""%><%=lsData[x]%">
</option>

<%
    }

%>

</select>

条

<input type="hidden" name="selcp" value="1">

</form>
```

## 12.4login.jsp 登录页面：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<html>

<head>

<%@ page language="java" contentType="text/html;

```

```
charset=UTF-8"

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>登录页面</title>

</head>

<body>

<center>

<% // 乱码解决

    request.setCharacterEncoding("utf-8") ;

%>

    <h1>登陆程序</h1>

    <hr>

    <%=request.getAttribute("info")!=null?request.getAttribute("info"):""%>

    <form action="check.jsp" method="post">

        用户ID: <input type="text" name="mid"><br>

        密 &nbsp;&nbsp;码: <input type="password"
name="password"><br>

        验证码: <input type="text" name="code"
maxlength="4" size="4"><br>

        <input type="submit" value="登陆">
```

```
<input type="reset" value="重置">

</form>

</center>

</body>

</html>
```

## 12.5check.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<%@page import="java.sql.*"%>

<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<body>

<center>

<% // 乱码解决
```

```
request.setCharacterEncoding("GBK") ;  
%>  
<h1>登陆程序</h1>  
<hr>  
  
<%  
String rand = (String) session.getAttribute("rand") ;  
// 从session中取出验证码  
String code = request.getParameter("code") ;  
if(!rand.equalsIgnoreCase(code)){  
    request.setAttribute("info","请输入正确的验证码！  
") ;  
%>  
<jsp:forward page="Login.jsp"/>  
<%  
}  
%>  
<%!  
public static final String DBDRIVER =  
"org.gjt.mm.mysql.Driver" ;  
public static final String DBURL =  
"jdbc:mysql://127.0.0.1:3306/school" ;
```

```
public static final String DBUSER = "root" ;  
  
public static final String DBPASS = "123456" ;  
  
%>  
  
<%  
  
Connection conn = null ;  
  
PreparedStatement pstmt = null ;  
  
ResultSet rs = null ;  
  
%>  
  
<%  
  
String userid = request.getParameter("mid") ;  
  
String password = request.getParameter("password") ;  
  
String sql = "SELECT name FROM user WHERE userid=? AND  
password=? " ;  
  
%>  
  
<%  
  
boolean flag = false ;  
  
Class.forName(DBDRIVER) ;  
  
conn =  
  
DriverManager.getConnection(DBURL,DBUSER,DBPASS) ;  
  
pstmt = conn.prepareStatement(sql) ;  
  
pstmt.setString(1,userid) ;  
  
pstmt.setString(2,password) ;
```

```
rs = pstmt.executeQuery() ; // 进行查询

if(rs.next()){ // 现在可以查找到内容

    session.setAttribute("id",userid) ; // 保存mid

    flag = true ;

}

conn.close() ; // 关闭数据库连接

%>

<%

if(flag){ // 现在已经登陆成功

%>

    <jsp:forward page="welcome.jsp"/>

%>

} else {

    request.setAttribute("info","错误的用户名或密码!");

" ) ;

%>

    <jsp:forward page="Login.jsp"/>

%>

}

%>

</center>

</body>
```

```
</html>
```

## 12.6welcome.jsp 欢迎页面

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<% // 乱码解决
request.setCharacterEncoding("utf-8") ;
%>
<center>
<h1>登陆程序</h1>
<br>
```

```
<%

    if(session.getAttribute("id") != null){

%>

    <h2>欢迎<font

color="RED"><%=session.getAttribute("id")%></font>光

临! </h2>

    <h3><a href="product_list.jsp">进入商品列表

</a></h3>

    <h3><a href="Logout.jsp">登陆注销</a></h3>

%>

    } else {

        request.setAttribute("info","请先登陆! ");

%>

    <jsp:forward page="Login.jsp"/>

%>

}

</center>

</body>

</html>
```

## 12.7logout.jsp 注销页面：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<% // 乱码解决
request.setCharacterEncoding("utf-8") ;
%>
<center>
<h1>登录程序</h1>
<br>
<%
session.invalidate() ; // 让session失效
response.sendRedirect("login.jsp") ;
```

```
%>

</center>

</body>

</html>
```

## 12.8product\_list.jsp 产品列表页

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">

<%@page import="java.sql.*"%>

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<script language="javascript">
    function changeColor(obj,color){
        obj.bgColor = color ;
    }

```

```
</script>

<body>

<%!

    public static final String URL = "product_list.jsp" ;

%>

<%!

public static final String DBDRIVER =
"org.gjt.mm.mysql.Driver" ;

public static final String DBURL =
"jdbc:mysql://127.0.0.1:3306/school" ;

public static final String DBUSER = "root" ;

public static final String DBPASS = "123456" ;

%>

<%
    Connection conn = null ;
    PreparedStatement pstmt = null ;
    ResultSet rs = null ;
%>

<%
    int currentPage = 1 ; // 为当前所在的页， 默认在第1页
    int lineSize = 15 ; // 每次显示的记录数
    int allRecorders = 0 ; // 表示全部的记录数
```

```
String keyword = request.getParameter("kw") ; // 接  
收查询关键字  
  
%>  
  
<%  
  
try{  
  
    currentPage =  
  
Integer.parseInt(request.getParameter("cp")) ;  
  
} catch(Exception e) {}  
  
try{  
  
    lineSize =  
  
Integer.parseInt(request.getParameter("ls")) ;  
  
} catch(Exception e) {}  
  
if(keyword == null){  
  
    keyword = "" ; // 如果模糊查询没有关键字，则表示查  
询全部  
  
}  
  
%>  
  
<%  
  
Class.forName(DBDRIVER) ;  
  
conn =  
  
DriverManager.getConnection(DBURL,DBUSER,DBPASS) ;  
  
String sql = "SELECT COUNT(pid) FROM product " +
```

```
" WHERE pname LIKE ? OR note LIKE ? OR price
LIKE ? OR amount LIKE ? " ;

stmt = conn.prepareStatement(sql) ;

stmt.setString(1, "%" +keyWord+ "%") ;

stmt.setString(2, "%" +keyWord+ "%") ;

stmt.setString(3, "%" +keyWord+ "%") ;

stmt.setString(4, "%" +keyWord+ "%") ;

rs = stmt.executeQuery() ;

if(rs.next()){// 取得全部的记录数

    allRecorders = rs.getInt(1) ;

}

%>

<center>

    <h1>商品列表</h1>

<jsp:include page="split_page_plugin.jsp">

    <jsp:param name="allRecorders"
value="<%="allRecorders%>" />

    <jsp:param name="url" value="<%="URL%>" />

    <jsp:param name="currentPage"
value="<%="currentPage%>" />

    <jsp:param name="LineSize" value="<%="lineSize%>" />

</jsp:include>
```

```
<TABLE BORDER="1" cellpadding="5" cellspacing="0"
bgcolor="#F2F2F2" width="100%>

<TR onMouseOver="changeColor(this,'white')"
onMouseOut="changeColor(this,'F2F2F2')">
    <td align="center" valign="middle"><span
class="STYLE10">编号</span></td>
    <td align="center" valign="middle"><span
class="STYLE10">名称</span></td>
    <td align="center" valign="middle"><span
class="STYLE10">价格</span></td>
    <td align="center" valign="middle"><span
class="STYLE10">数量</span></td>
    <td align="center" valign="middle"><span
class="STYLE10">简介</span></td>
    <td align="center" valign="middle"><span
class="STYLE10">购买</span></td>
</TR>

<%
sql = " SELECT pid,pname,note,price,amount" +
" FROM product WHERE (pname LIKE ? OR note LIKE ?
OR price LIKE ? OR amount LIKE ? ) " +
" limit ?,?" ;
```

```
stmt = conn.prepareStatement(sql) ;  
  
stmt.setString(1, "%" +keyWord+ "%") ;  
  
stmt.setString(2, "%" +keyWord+ "%") ;  
  
stmt.setString(3, "%" +keyWord+ "%") ;  
  
stmt.setString(4, "%" +keyWord+ "%") ;  
  
stmt.setInt(5,(currentPage-1) * lineSize) ;  
  
stmt.setInt(6,lineSize) ;  
  
rs = stmt.executeQuery() ;  
  
%>  
  
<%  
  
while(rs.next()){  
  
    int pid = rs.getInt(1) ;  
  
    String name = rs.getString(2) ;  
  
    String note = rs.getString(3) ;  
  
    double price = rs.getDouble(4) ;  
  
    int amount = rs.getInt(5) ;  
  
%>  
  
<TR onMouseOver="changeColor(this,'white')"  
onMouseOut="changeColor(this,'F2F2F2')">  
    <td align="center" valign="middle"><span  
class="STYLE6"><%=pid%></span></td>  
  
    <td align="center" valign="middle"><span
```

```
class="STYLE6">><%=name%></span></td>

    <td align="center" valign="middle"><span

class="STYLE6">><%=price%></span></td>

    <td align="center" valign="middle"><span

class="STYLE6">><%=amount%></span></td>

    <td align="center" valign="middle"><span

class="STYLE6">><%=note%></span></td>

    <td align="center" valign="middle"><span

class="STYLE6">><a href="add_car.jsp?pid=<%=pid%>"

target="_abLank">增加到购物车</a></span></td>

</TR>

<%
}

%>

</table>

<%
    conn.close() ;
%>

</center>

</body>

</html>
```

## 12.9add\_car.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

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

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<% // 要购买的商品编号
int pid =
Integer.parseInt(request.getParameter("pid")) ;
Map<Integer, Integer> cars = (Map<Integer, Integer>)
session.getAttribute("allpid") ; // 取得全部已经购买的商品
// 但如果是第一次购买的话，肯定取得不到
if(cars == null){ // 现在没有购买任何的商品
```

```
cars = new HashMap<Integer, Integer>() ; // 实例化一个新的Map对象

}

if(cars.get(pid) == null){ // 还没有购买此商品

    cars.put(pid,1) ; // 此商品购买了一个

} else { // 现在已经购买了

    int val = cars.get(pid) ; // 取出原本的数量

    val++ ; // 增加

    cars.put(pid,val) ; // 覆盖已有的内容

}

session.setAttribute("allpid",cars) ; // 重新保存更改后的数据

%>

<jsp:forward page="product_cars.jsp"/>

</body>

</html>
```

## 12.10product\_cars.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

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

<%@page import="java.sql.*"%>
```

```
<html>

<head>

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

pageEncoding="UTF-8"%>

<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

<title>在此处插入标题</title>

</head>

<script language="javascript">

    function changeColor(obj,color){

        obj.bgColor = color ;

    }

</script>

<body>

<%!

    public static final String URL = "product_list.jsp" ;

%>

<%!

public static final String DBDRIVER =

"org.gjt.mm.mysql.Driver" ;

public static final String DBURL =
```

```
"jdbc:mysql://127.0.0.1:3306/school" ;  
  
public static final String DBUSER = "root" ;  
  
public static final String DBPASS = "123456" ;  
  
%>  
  
<%  
  
Connection conn = null ;  
  
PreparedStatement pstmt = null ;  
  
ResultSet rs = null ;  
  
%>  
  
<%  
  
Map<Integer, Integer> cars = (Map<Integer, Integer>)  
session.getAttribute("allpid") ;  
  
if(cars != null) {  
  
    Set<Integer> key = cars.keySet() ; // 取得全部的  
key  
%>  
  
<%  
  
Class.forName(DBDRIVER) ;  
  
conn =  
  
DriverManager.getConnection(DBURL, DBUSER, DBPASS) ;  
  
StringBuffer sql = new StringBuffer() ; // 循环修改  
内容的时候使用StringBuffer
```

```
%>

<center>

    <h1>商品列表</h1>

<form action="update_car.jsp" method="post">

<TABLE BORDER="1" cellpadding="5" cellspacing="0"
bgcolor="#F2F2F2" width="100%">

    <TR onMouseOver="changeColor(this,'white')"
onMouseOut="changeColor(this,'F2F2F2')">

        <td align="center" valign="middle"><span
class="STYLE10">编号</span></td>

        <td align="center" valign="middle"><span
class="STYLE10">名称</span></td>

        <td align="center" valign="middle"><span
class="STYLE10">价格</span></td>

        <td align="center" valign="middle"><span
class="STYLE10">数量</span></td>

        <td align="center" valign="middle"><span
class="STYLE10">简介</span></td>

        <td align="center" valign="middle"><span
class="STYLE10">数量</span></td>

    </TR>

<%
```

```
sql.append("SELECT pid, pname, note, price, amount FROM  
product WHERE pid IN (" );  
  
int count = 0 ;  
  
Iterator<Integer> iter = key.iterator() ;  
  
while(iter.hasNext()){  
  
    count++ ;  
  
    sql.append(iter.next()) ;  
  
    if(count <= key.size()-1){  
  
        sql.append(",") ;  
  
    }  
  
}  
  
sql.append(")") ;  
  
pstmt = conn.prepareStatement(sql.toString()) ;  
  
rs = pstmt.executeQuery() ;  
  
%>  
  
<%  
  
while(rs.next()){  
  
    int pid = rs.getInt(1) ;  
  
    String name = rs.getString(2) ;  
  
    String note = rs.getString(3) ;  
  
    double price = rs.getDouble(4) ;  
  
    int amount = rs.getInt(5) ;
```

```
%>

<TR onMouseOver="changeColor(this,'white')"
onMouseOut="changeColor(this,'F2F2F2')">

    <td align="center" valign="middle"><span
class="STYLE6"><%=pid%></span></td>

    <td align="center" valign="middle"><span
class="STYLE6"><%=name%></span></td>

    <td align="center" valign="middle"><span
class="STYLE6"><%=price%></span></td>

    <td align="center" valign="middle"><span
class="STYLE6"><%=amount%></span></td>

    <td align="center" valign="middle"><span
class="STYLE6"><%=note%></span></td>

    <td align="center" valign="middle"><span
class="STYLE6">
        <input type="text" name=<%=pid%>
value=<%=cars.get(pid)%> size="3" maxlength="3">
    </td>
</TR>

<%
}
%>
```

```
<%  
  
    if(count == 0){  
  
%>  
  
    <TR onMouseOver="changeColor(this,'white')"  
onMouseOut="changeColor(this,'F2F2F2')">  
  
        <td align="center" valign="middle"  
  
colspan="6"><span class="STYLE6">暂时没有购买任何商品！  
  
</span></td>  
  
    </TR>  
  
<%  
  
    } else {  
  
%>  
  
    <TR onMouseOver="changeColor(this,'white')"  
onMouseOut="changeColor(this,'F2F2F2')">  
  
        <td align="center" valign="middle"  
  
colspan="6"><span class="STYLE6">  
  
            <input type="submit" value="修改数量">  
  
</span></td>  
  
    </TR>  
  
<%  
  
    }  
  
%>
```

```
</table>

</form>

<%
    conn.close() ;

} else {

%>

<TABLE BORDER="1" cellpadding="5" cellspacing="0"
bgcolor="#F2F2F2" width="100%">

    <TR onMouseOver="changeColor(this,'white')"
onMouseOut="changeColor(this,'#F2F2F2')">

        <td align="center" valign="middle"
colspan="6"><span class="STYLE6">暂时没有购买任何商品！

</span></td>

    </TR>

</TABLE>

<%
}

%>

</center>

</body>

</html>
```

## 12.11update\_car.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

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

<html>
<head>
<%@ page language="java" contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>在此处插入标题</title>
</head>
<body>
<% // 要购买的商品编号
Enumeration allpid = request.getParameterNames() ;
// 取得所有的提交数据
Map<Integer, Integer> cars = (Map<Integer, Integer>)
session.getAttribute("allpid") ; // 取得全部已经购买的商品
if(cars == null){ // 现在没有购买任何的商品
cars = new HashMap<Integer, Integer>() ; // 实例
```

化一个新的Map对象

```
}

while(allpid.hasMoreElements()){

    String paramName = (String) allpid.nextElement();

    try{

        int pid = Integer.parseInt(paramName) ;

        int val =

Integer.parseInt(request.getParameter(paramName)) ;

        cars.put(pid,val) ;// 重新设置相同的PID，则表示

覆盖

    }catch(Exception e){}

}

session.setAttribute("allpid",cars) ; // 重新保存更

改后的数据

%>

<jsp:forward page="product_cars.jsp"/>

</body>

</html>
```

**12.12 作业：**将上述代码使用 MVC 设计模式实现

# 13 项目实战——购物网站

## 13.1 项目需求概述

### 13.1.1 概述

本系统为购物网站的前台部分，用户可以查看商品信息，可以购物。

### 13.1.2 主要功能

本系统的整体设计可分为：商品显示模块、用户模块、购物车模块、订单模块。以下是各个模块的功能介绍：

序号	模块名称	主要功能
1	商品显示模块	分页显示所有商品 查看单个商品详细信息 搜索商品
2	用户模块	新用户注册 用户登录 用户退出 用户修改注册信息
3	购物车模块	加入商品到购物车 从购物车中移除商品 修改所购商品数量 提交订单并显示订单信息

4	订单模块	修改订单信息  用户确认后生成正式订单  用户查看历史订单	
---	------	---	--

### 13.1.3 运行环境

软件环境：

分类	名称	语种
操作系统	无要求	简体中文
数据库平台	Mysql	-
应用服务器	Tomcat6	-
Java 开发工具	Eclipse3 以上	
框架	无	

硬件环境：

开发电脑	最低配置	推荐配置
硬件配置	-	-
	-	-

## 13.2 功能需求

### 13.2.1 商品显示模块

#### 1 分页显示所有商品

需求编号	Shopping_001
功能名称	分页显示所有商品
功能描述	点击“首页”，打开商品列表页面； 能够分页显示所有商品概要信息（包括序号、商品名称、价格、操作一共四项）
备注	本截图中序号和商品名称未显示完整



商品名称	价格	操作
对象持久化技术详解	59.0	加入购物车
反	39.0	加入购物车
	39.0	加入购物车
	59.0	加入购物车
	95.0	加入购物车
版) 卷I：基础知识	75.0	加入购物车
技术详解	45.0	加入购物车
	88.0	加入购物车

[上一页] 1 2 3 ... [下一页]

#### 2 查看单个商品详细信息

需求编号	Shopping_002
功能名称	查看单个商品详细信息

功能描述	<p>点击“首页”，打开商品列表页面；</p> <p>点击一个商品的商品名称，可以查看该商品的详细信息，具体内容参看截图</p>
备注	



The screenshot shows a web-based e-commerce platform. At the top, there's a blue header bar with navigation links: 首页 (Home), 用户管理 (User Management), 购物车 (Cart), 订单 (Orders), and 退出 (Logout). Below the header, a green banner displays the text "欢迎访问 电子商务门户" and a breadcrumb trail: 电子商务门户 → 商品明细. The main content area has a title "精通Hibernate: Java对象持久化技术详解". Below the title is a table with product details:

【作者】	孙卫琴
【价格】	59.0
【出版社】	电子工业出版社
【书号】	2
【页码】	600
【所属类别】	软件与程序设计
【简介】	<p>孙卫琴的计算机书籍创作心得：如果说书的结构好比房屋的框架，书的内容则好比房屋的具体组成元素。计算机书的内容的形式分为：文字、表格、形式的元素……</p> <p>继《Tomcat与Java Web开发技术详解(含光盘)》和《精通Struts：基于MVC的Java Web设计与开发(含光盘)》之后，Java对象持久化技术详解，正处在迎接面市的准备当中。Hibernate是一个基于Java的开放源代码的持久化中间件，它查询和数据缓存功能，Java开发人员可以方便的通过Hibernate API来操纵数据库。现在，越来越多的Java开发人员把Hibernate 作为优秀的类库和组件，荣获了第15届Jolt大奖。Hibernate之所以能结合大量的典型的实例，详细介绍了运用目前最成熟的hibernate2.1版本进行Java对象持久化的技术。Hibernate是仅能掌握用Hibernate工具对这两种模型进行映射的技术，还能获得设计与开发Java对象模型和关系数据模型的先进经验。</p>

Below the table, there's a thumbnail image of the book cover for "精通Hibernate: Java对象持久化技术详解", which features a cartoon character of a bear holding a book. At the bottom right of the page, there's a button labeled "加入购物车" (Add to Cart).

### 3 搜索商品

需求编号	Shopping_003
功能名称	搜索商品
功能描述	<p>点击“首页”，打开商品列表页面；</p> <p>能够按照商品名称模糊查询，如果不输入查询条件，则查询所有</p>
备注	



The screenshot shows a product search interface. At the top, there is a navigation bar with links: 首页 (Home), 用户管理 (User Management), 购物车 (Cart), 订单 (Orders), and 退出 (Logout). Below the navigation bar is a search bar with the placeholder "商品名称:" and a red-bordered "搜索" (Search) button. A table lists products with columns: 商品名称 (Product Name), 价格 (Price), and 操作 (Operations). Each row contains a small image of a book and a "加入购物车" (Add to Cart) button.

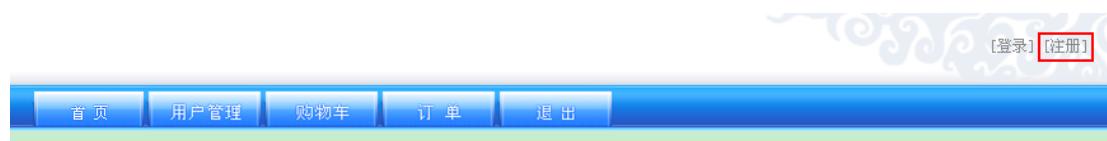
商品名称	价格	操作
对称持久化技术详解	59.0	加入购物车
	39.0	加入购物车
	39.0	加入购物车
	59.0	加入购物车
	95.0	加入购物车
版1 卷I：基础知识	75.0	加入购物车
技术详解	45.0	加入购物车
	88.0	加入购物车

[上一页] [下一页]

## 13.2.2 用户模块

### 1 新用户注册

需求编号	Shopping_004
功能名称	新用户注册
功能描述	点击注册项，打开新用户注册页面； 按照要求填写好注册信息后，点击“注册”按钮（其中用户名和密码是必填项）
备注	注意注册时用户名不能重复



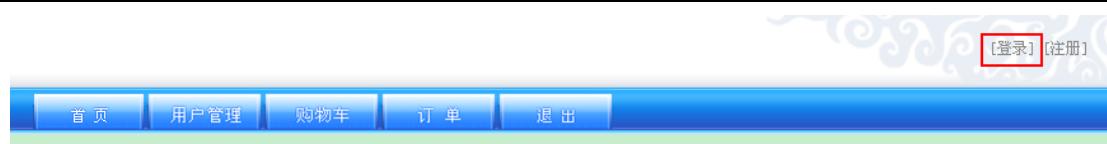
The screenshot shows a user registration page. At the top, there is a navigation bar with links: 首页 (Home), 用户管理 (User Management), 购物车 (Cart), 订单 (Orders), and 退出 (Logout). Below the navigation bar is a registration form with fields for username, password, and other personal information. At the bottom right of the form area, there is a red-bordered "注册" (Register) button.

新用户注册

用户名:	<input type="text"/>
注册用户名长度限制为0- 16字节	*
密码(至少6位, 至多8位):	<input type="password"/>
请输入密码, 区分大小写。	*
请不要使用任何类似 VVV、VV 或 HTML 字符!	
密码(至少6位, 至多8位):	<input type="password"/>
请再输入一遍确认	*
真实姓名:	<input type="text"/>
所在地区:	<input type="text"/>
联系地址1:	<input type="text"/>
联系地址2:	<input type="text"/>
邮编:	<input type="text"/>
家庭电话:	<input type="text"/>
办公室电话:	<input type="text"/>
手机:	<input type="text"/>
E-mail地址:	<input type="text"/>
<input type="button" value="注册"/> <input type="button" value="清除"/>	

## 2 用户登录

需求编号	Shopping_005
功能名称	用户登录
功能描述	点击登录项, 打开用户登录页面;  输入用户名和密码, 实现用户登录功能
备注	



首页 用户管理 购物车 订单 退出

> 欢迎访问 电子商务门户

● 电子商务门户 → 用户登录

输入您的用户名、密码登录

请输入您的用户名	<input type="text"/>	没有注册?
请输入您的密码	<input type="password"/>	
<input type="button" value="登 录"/>		

### 3 用户退出

需求编号	Shopping_006
功能名称	用户退出
功能描述	点击退出项，实现用户退出功能
备注	本功能需要依赖用户登录功能



### 4 用户修改注册信息

需求编号	Shopping_007
功能名称	用户修改注册信息
功能描述	点击“用户管理”，打开注册信息修改页面（页面中要能正确显示该用户的信息）； 修改一些信息，点击“修改”按钮
备注	本功能需要依赖用户登录功能； 用户名不能修改

首 页    **用户管理**    购物车    订 单    退 出

>> 欢迎 sunxun 访问

● 电子商务门户 → 用户信息修改

用户信息修改	
用户名:	<input type="text" value="sunxun"/> *
注册用户名长度限制为0-16字节	
密码(至少6位, 至多8位): 请输入密码, 区分大小写。 请不要使用任何类似`  `、`  `或 HTML 字符	<input type="password"/> *
密码(至少6位, 至多8位): 请再输一遍确认	<input type="password"/> *
真实姓名:	<input type="text" value="孙逊"/>
所在地区:	<input type="text" value="南京"/>
联系地址1:	<input type="text" value="南京"/>
联系地址2:	<input type="text" value="南京"/>
邮编:	<input type="text" value="161000"/>
家庭电话:	<input type="text" value="1235345"/>
办公室电话:	<input type="text" value="1235334"/>
手机	<input type="text" value="1234345"/>
E-mail地址	<input type="text" value="sunxun@sina.com"/>
<input type="button" value="修改"/>	

### 13.2.3 购物车模块

#### 1 加入商品到购物车

需求编号	Shopping_008
功能名称	加入商品到购物车
功能描述	可以在商品列表页面中点击“加入购物车”图标；  可以在单个商品详细信息页面中点击“加入购物车”图标；  点击“购物车”，打开购物车页面，可以在购物车页面中点击“继续购物”按钮，继续购物。
备注	



首页 用户管理 **购物车** 订单 退出

产品名称	价格	数量	合计	操作
术详解	59.0	1	¥ 59	<b>取消</b> <b>保存修改</b>

**提交订单** **继续购物** **清空购物车**

## 2 从购物车中移除商品

需求编号	Shopping_009
功能名称	从购物车中移除商品
功能描述	点击“购物车”，打开购物车页面； 在购物车页面中点击“取消”按钮，可以移除购物车中一种商品； 在购物车页面中点击“清空购物车”按钮，可以移除购物车中所有商品
备注	



首页 用户管理 **购物车** 订单 退出

产品名称	价格	数量	合计	操作
术详解	59.0	1	¥ 59	<b>取消</b> <b>保存修改</b>

**提交订单** **继续购物** **清空购物车**

### 3 修改所购商品数量

需求编号	Shopping_010
功能名称	修改所购商品数量
功能描述	点击“购物车”，打开购物车页面； 修改数量，点击“保存修改”按钮，可以修改一种商品的购买数量
备注	注意数量只能输入正整数



产品名称	价格	数量	合计	操作
未详解	59.0	1	¥ 59	<input type="button" value="取消"/> <input style="border: 2px solid red;" type="button" value="保存修改"/>

### 4 提交订单并显示订单信息

需求编号	Shopping_011
功能名称	提交订单并显示订单信息
功能描述	点击“购物车”，打开购物车页面； 点击“提交订单”按钮，打开订单信息确认页面
备注	如果用户没有登录，不能提交订单； 如果购物车为空，不能提交订单



首页 用户管理 购物车 订单 退出

产品名称	价格	数量	合计	操作
术详解	59.0	1	¥ 59	<input type="button" value="取消"/> <input type="button" value="保存修改"/>
			¥ 59	

### 13.2.4 订单模块

#### 1 修改订单信息

需求编号	Shopping_012
功能名称	修改订单信息
功能描述	在订单信息确认页面中，可以修改和该订单相关的三项（用户信息：跳转到修改注册信息页面、付款方式：邮局汇款，货到付款，银行转账、购物清单：跳转回购物车页面）
备注	本功能需要依赖用户登录功能

[首页](#) [用户管理](#) [购物车](#) [订单](#) [退出](#)

欢迎访问 电子商务门户

电子商务门户 →  确认定单

用户信息		<a href="#">修改</a>
用户名:	张三	
联系地址:	street1	
邮编:	200001	
家庭电话:	12345678	
办公室电话:	22446688	
手机:	13588888888	
Email地址:	bl@abc.coms	

付款方式	
<input style="width: 100px; height: 20px;" type="button" value="邮局汇款"/>	

商品购物清单		<a href="#">修改</a>
1	精通Hibernate：Java对象持久化技术详解	价格: 59.0

请认真检查以上订单信息，确认无误后，点击 →  提交订单

## 2 用户确认后生成正式订单

需求编号	Shopping_013
功能名称	用户确认后生成正式订单
功能描述	在订单信息确认页面中，点击“提交订单”图标，生成正式订单
备注	本功能需要依赖用户登录功能

首 页    用户管理    购物车    订 单    退 出

> 欢迎访问 电子商务门户

① 电子商务门户 → ② 确认定单

用户信息 [修改]	
用户名:	张三
联系地址:	street1
邮编:	200001
家庭电话:	12345678
办公室电话:	22446688
手机:	13588888888
Email地址:	b1@abc.com

付款方式

1	精通Hibernate：Java对象持久化技术详解	价格: 59.0
---	---------------------------	----------

商品购物清单 [修改]

请认真检查以上订单信息，确认无误后，点击 → **提交订单**

### 3 用户查看历史订单

需求编号	Shopping_014
功能名称	用户查看历史订单
功能描述	点击“订单”，打开历史订单列表页面，可以查看订单明细和删除订单。
备注	本功能需要依赖用户登录功能；

首 页    用户管理    购物车    **订 单**    退 出

>> 欢迎 sunsun 访问

① 电子商务门户 → ② 订单列表

序号	订单编号	订单金额	付款方式	操作
1	BCA8A8F7DE0041D9BD53EACF0EFBC668	127.0	邮局汇款	<b>删除</b> <b>查看明细</b>

查看订单明细：

序号	商品名称	单价	数量
1	Effective Java中文版	39.0	1
2	Java与模式	88.0	1

## 13.3 引言

### 13.3.1 编写目的

本说明书是根据购物网站需求说明书的要求编写的，是为了实现系统功能而设计一个体系架构，以满足需求设计中规定的各种需求。本说明书作为软件开发工程师进一步作详细设计的基础，也是编写代码的重要依据，同时它也是需求设计人员、测试人员和管理人员的参考材料。

### 13.3.2 术语或缩写

MVC（业务模型层、视图层、控制层）

## 13.4 总体设计

### 13.4.1 系统说明

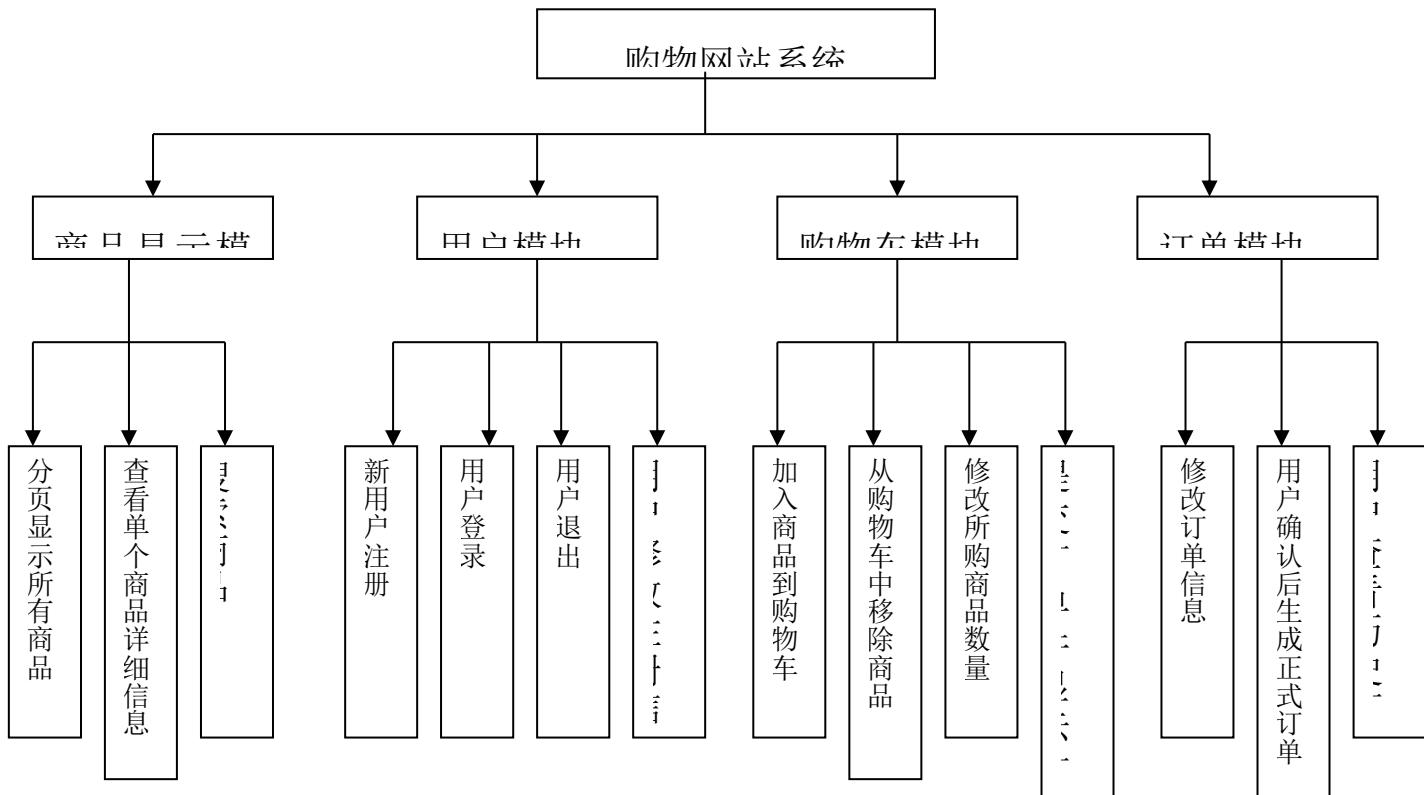
本系统的整体设计可分为：商品显示模块、用户模块、购物车模块、订单模块。以下是各个模块的功能介绍：

序号	模块名称	主要功能
1	商品显示模块	分页显示所有商品

		查看单个商品详细信息 搜索商品
2	用户模块	新用户注册  用户登录  用户退出  用户修改注册信息
3	购物车模块	加入商品到购物车  从购物车中移除商品  修改所购商品数量  提交订单并显示订单信息
4	订单模块	修改订单信息  用户确认后生成正式订单  用户查看历史订单

### 13.4.2 总体架构

本系统包括商品显示模块、用户模块、购物车模块、订单模块，具体的示意图如下所示：



## 13.5 包结构、类分析

### 13.5.1 业务模型层

- 实体包
  - 商品实体类
  - 用户实体类
  - 购物车实体类
  - 主订单实体类
  - 子订单实体类
- Dao 包

- 商品 Dao 类
- 用户 Dao 类
- 主订单 Dao 类
- 子订单 Dao 类
- Service 包
  - 商品 Service 类
  - 用户 Service 类
  - 订单 Service 类
- 工具包
  - Jdbc 帮助类

### 13.5.2 控制层

- Servlet 包
  - 商品相关 Servlet 类
  - 用户相关 Servlet 类
  - 购物车相关 Servlet 类
  - 订单相关 Servlet 类

### 13.5.3 视图层

- JSP
  - 商品列表页面
  - 商品明细页面

- 用户注册页面
- 用户登录页面
- 用户修改注册信息页面
- 购物车页面
- 提交订单确认页面
- 历史订单列表页面

历史订单明细页面

## 13.6 实现思路

- 1、商品显示模块：这个模块相对独立，用到商品表 product
- 2、用户模块：用户登录成功后可以把用户信息放到 session 中，用到用户表 user
- 3、购物车模块：所有购物信息最好以一个整体放到 session 中，点击 "加入购物车"按钮后,要修改 session 中的购物信息，然后在购物车页面中,可以从 session 中取出所有购物信息并显示，这个模块主要和 session 打交道
- 4、订单模块：在"确认订单"页面中,当用户点击"提交订单"按钮后,需要把该订单信息保存到订单主表 orders 和订单明细表 orderdetail 两张表中（注意:保存到两张表是一个事务）

## 13.7 数据库设计

### 13.7.1 表结构

商品表 product:

字段名称	字段类型	字段长度	字段注释
productid (主键)	varchar2	40	商品 id
name	varchar2	64	名称
description	varchar2	2000	简介
baseprice	number		单价
writer	varchar2	32	作者
publish	varchar2	64	出版社
pages	number		页码
images	varchar2	100	封面

用户表 t\_user:

字段名称	字段类型	字段长度	字段注释
userid (主键)	varchar2	40	用户 id
username	varchar2	32	用户名
truename	varchar2	32	真实姓名
password	varchar2	32	密码
street1	varchar2	64	联系地址 1
street2	varchar2	64	联系地址 2

city	varchar2	32	所在地区
zip	varchar2	8	邮编
email	varchar2	32	Email 地址
homephone	varchar2	16	家庭电话
cellphone	varchar2	16	手机
officephone	varchar2	16	办公室电话

订单主表 orders:

字段名称	字段类型	字段长度	字段注释
orderid (主键)	bigint	20	订单 id
cost	double	15	订单金额
payType	varchar	16	付款方式
userid	bigint	20	所属用户 id
time	timestamp		订单时间

订单明细表 orderdetail:

字段名称	字段类型	字段长度	字段注释
detailid (主键)	varchar2	40	明细 id
orderid	varchar2	40	所属订单 id
productid	varchar2	40	购买商品 id
productname	varchar2	64	商品名称
baseprice	number		购买商品时单价

num	number		购买商品数量
-----	--------	--	--------

## 13.7.2 表关系

- 1、订单主表的 `userid` 字段作为外键参照用户表的主键；
- 2、订单明细表的 `orderid` 字段作为外键参照订单主表的主键；
- 3、订单明细表的 `productid` 字段作为外键参照商品表的主键。