ORACLE WORKFORCE DEVELOPMENT PROGRAM

Java企业级应用开发

3

第3章：JSP核心技术

CSG
CAMPUS SOLUTION GROUP

本章内容

- 第1节 JSP基础
- 第2节 访问JSP
- 第3节 JSP工作原理
- 第4节 作用域与隐式对象
- 第5节 JSP服务端脚本
- 第6节 JSP Model2标准

本章目标

- 理解Jsp技术的原理特性和优点。
- 能理解Jsp和Servlet的区别和联系。
- 掌握如何创建Jsp页面以及执行过程。
- 掌握作用域与隐式对象。
- 熟练应用JSP服务端脚本。
- 熟练应用Jsp+Servlet+JavaBean技术。
- 掌握MVC设计模式在Jsp技术中的应用。

第1节 JSP基础

⑩ 知识点预览

#	知识点	难点	重点	应用	说明
1	JSP概述				介绍JSP的渊源和特点
2	第一个JSP应用	√	√	√	初步体掌握JSP页面的构成
3	JSP与Servlet的区别	√	√		明确JSP和Servlet的区别

JSP概述

- 什么是动态网页？
  - 动态网页是指在服务器端运行的程序或者网页，会随客户、时间、地点等信息的不同返回不同的网页。
    - 比如：登录论坛时，普通客户只能看帖子，而管理员同样的页面会呈现“删除”、“修改”等多种操作提示。另外，在不同的时间登录，看到的内容也是不一样的
  - 动态网页的特点？
    - 交互性：根据用户的选
    - 自动更新：无需手动修
    - 随机性：不同的时间、

- 如何实现动态网页的开发？
  - 传统的HTML页面文件中加入服务器脚本语言，比如Jsp等。

JSP概述

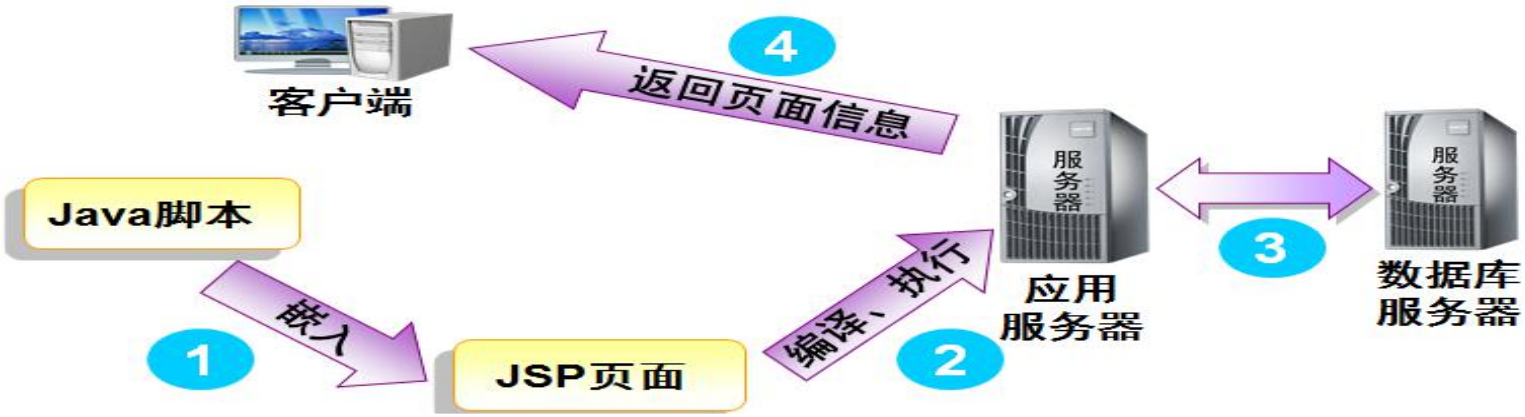
什么是Jsp ？

- Jsp是Java Server Pages的缩写，由Sun公司倡导、许多公司参与，于1999年推出的一种动态网页技术标准。
- Jsp是基于Java Servlet以及整个Java体系的Web开发技术。
- 可以建立安全的、跨平台的先进动态网站。
- Jsp定义了许多用于Web应用有用的标准元素，例如访问JavaBeans的组件元素，在页面间传递控制权的元素。以及在请求、页面和用户间共享信息的元素。
- Jsp规范允许自定义元素，和预定义元素的组合确保了可以开发出更强大的Web应用程序。
- 其实，通俗的讲，Jsp技术是指在HTML页面中嵌入Java脚本语言，然后由应用服务器中的Jsp引擎来编译和运行，之后再将生成的整个页面返回给客户端。

JSP概述

Jsp运行原理概述

- 将包含有Jsp页面资源的WEB站点，部署到安装了tomcat容器的应用程序服务器上。客户端通过浏览器向服务器中的站点发送请求。
- Jsp页面被第一次请求执行时，服务器引擎首先将Jsp页面转译成一个Java文件，这个Java文件其实就是Servlet类。
- 然后再编译成字节码文件，响应客户请求，而再次被请求时，Jsp引擎将直接执行该字节码文件。 Java服务器脚码将由翻译后对应Servlet的service方法执行，并把Jsp页面中的静态内容以及动态业务数据结果交给客户端浏览器显示。



```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%
String path = request.getContextPath();
String basePath =
request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
    <title>First Jsp</title>
  </head>  <body>
    <%
        out.print("Hello Jsp");
    %>
  </body>
</html>
```

- 浏览器访问index.jsp时，服务器首先将index.jsp翻译成一个index_jsp.class，在Tomcat服务器的work\Catalina\localhost\项目名\org\apache\jsp目录下可以看到index_jsp.class的源代码文件index_jsp.java

```
public final class index_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {
public void _jspInit() {
    _el_expressionfactory =
_jspxFactory.getJspApplicationContext(getServletConfig().getServletContext()).getExpressionFactory();
    _jsp_annotationprocessor = (org.apache.AnnotationProcessor)
getServletConfig().getServletContext().getAttribute(org.apache.AnnotationProcessor.class.getName());
  }
 public void _jspService(HttpServletRequest request, HttpServletResponse response)
       throws java.io.IOException, ServletException {
    PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;
    PageContext _jspx_page_context = null; try {
      response.setContentType("text/html;charset=UTF-8");
      pageContext = _jspxFactory.getPageContext(this, request, response,
                null, true, 8192, true);
      _jspx_page_context = pageContext;
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
      _jspx_out = out;

      out.write('\r');
      out.write('\n');
。 。 。
```

JSP概述

- 为什么用Jsp？
  - Jsp技术和其他技术对比性能优越:
    - 可以开发出动态的、高性能Web服务应用程序。
    - 克服了一些类似于Asp技术中将HTML代码嵌入程序，而造成不易开发和维护的弊端，而是采用向HTML代码中嵌入Java代码的方式，从而实现动静分离，便于开发的同时，也提高了效率。
    - 利用Java语言实现了跨平台，编译后运行的优点，提高通用性和运行效率。
    - 通过Java服务程序代码的方式，克服了脚本执行带来的不宜维护和安全隐患。
    - 基于Java Servlet的特性，以使Servlet基于一个应用程序进程，每个Servlet请求，对应于进程中的一个线程，克服了传统的CGI模式中对于每一个请求加载一个编译器和目标脚本而造成的资源消耗。
    - 便于团队开发。

第一个JSP应用

- 一个Jsp页面内容示例
  - 该段代码被客户端请求后，可以实现，将服务器端运行页面中Java程序而得到的动态计数结果显示到客户端浏览器上。

```
<html>
<head><title>The Counter Page</title></head>
<body>
<%! int counter = 0; %>
<H1>Welcome to the counter page.</H1>
<% counter++; %>
This page was visited <%= counter %> times.
</body>
</html>
```

第一个JSP应用

- 体现出优越性
  - 交互性：随着用户的要求和选择而改变和响应。
  - 自动更新：无需手动编辑，自动生成新的页面。
  - 随机性：随时间、地点、人的改变而改变。
  - 动静分离：提高了效率，便于分工协同开发，便于维护 。
  - 重复使用：一次编译，到处使用。
  - 安全性：先编译后运行，避免脚本级执行的缺点。

JSP与Servlet的区别

- **Jsp与Servlet的区别？**
  - 运作原理上的区别：
    - Sun首先开发的是Servlet，功能强劲，但是页面信息输出还是采用传统的CGI模式，也就是输出流动态生成HTML页面，包括每个标签和内容。而Jsp页面是把Java Tag镶嵌到HTML代码中，可以实现了动静分离，方便了网页的 开发和修改。
  - 业务功能上的区别：
    - Jsp页面可以划归视图层，但也可以说是Servlet的一种特殊形式。主要用途是提供可交互的客户界面，向客户显示模型数据。
    - Servlet是Java类，可以划归控制器层，根据客户的请求操作模型，并把模型的响应结果经由视图展现给客户。

第2节 JSP服务端脚本

⑩ 知识点预览

#	知识点	重点	难点	应用	说明
1	JSP代码脚本		√	√	掌握JSP页面服务器脚本的创建和应用
2	JSP申明脚本		√		掌握JSP页面创建成员变量和方法
3	JSP输出脚本		√	√	理解并掌握JSP页面中表达式的应用
4	JSP指令元素脚本	√	√	√	掌握page指令和include指令的应用
5	JSP注释		√	√	掌握JSP注释的书写和特点
6	JSP动作脚本			√	掌握并应用动作指令include

JSP代码脚本

Jsp服务端脚本

- JSP页面中嵌入Java业务代码的方式：
  - 在“<%”以及“%>”之间嵌入Java业务逻辑代码。
- 将Java业务代码嵌入JSP页面中的运行原理：
  - 在一个Jsp页面中可以有多处Java程序片， Jsp引擎将页面翻译成Java文件时，Java服务器脚本程序，将作为service方法中的语句，用作业务逻辑处理，这些Java程序片将被Jsp引擎按顺序执行。
  - 在一个程序片中声明的变量称做Jsp页面的局部变量，他们从声明位置向后，所有程序部分以及表达式部分都有效。因为Jsp引擎将Jsp页面转译成servlet类文件时，这些变量作为servlet类service方法的变量，即局部变量。
  - 当多个客户请求一个Jsp页面时，Jsp引擎为每个客户启动一个线程，不同的客户对应各自的局部变量（被分配在不同的内存空间）。因此，一个客户对Jsp页面局部变量操作的结果，不会影响到其他客户的这个局部变量。

控制流语句

- JSP 提供了强有力的 Java 工具来嵌入到你的 web 应用程序中。你可以在 JSP 编程中使用所有的 API 和 Jave
- 构建块，包括条件语句、循环等。

条件语句

- if…else 块像普通的 Scriptlet 一样开始，但 Scriptlet 结束于包含在 Scriptlet 标签间的 HTML 文本每一行。
- <%! int day = 3; %>
- <html>
- <head><title>IF...ELSE Example</title></head>
- <body>
- <% if (day == 1 | day == 7) { %>
- Today is weekend
- <% } else { %>
- Today is not weekend
- <% } %>
- </body>
- </html>

switch...case

使用 out.println()语句和内部 Scriptletas 编写，与上述例子有一点区别

```
:
<%! int day = 3; %>
<html>
<head><title>SWITCH...CASE Example</title></head>
<body>
<%
switch(day) {
case 0:
out.println("It\'s Sunday.");
break;
case 1:
out.println("It\'s Monday.");
break;
case 2:
out.println("It\'s Tuesday.");
break;
case 3:
out.println("It\'s Wednesday.");
break;
case 4:
out.println("It\'s Thursday.");
break;
case 5:
out.println("It\'s Friday.");
break;
default:
out.println("It's Saturday.");
}
%>
</body>
</html>
```

循环语句

- 可以在 Java 中使用循环块的三种基本类型来实现 JSP 编程：for，while，和 do…while
- <%! int fontSize; %>
- <html>
- <head><title>FOR LOOP Example</title></head>
- <body>
- <%for (fontSize = 1; fontSize <= 3; fontSize++){ %>
- <font color="green" size="<%= fontSize %>">
- JSP Tutorial
-
- <%}%>
- </body>
- </html>

```
<%! int fontSize; %>
<html>
<head><title>WHILE LOOP Example</title></head>
<body>
<%while ( fontSize <= 3){ %>
<font color="green" size="<%= fontSize %>">
JSP Tutorial
</font><br />
<%fontSize++;%>
<%}%>
</body>
</html>
```

JSP声明脚本

Jsp声明脚本

- 在"<%!"和"%>"标记之间放置声明脚本。
  - 声明格式举例：<%! Declarations %>

Jsp声明脚本分类

- 声明变量：
  - 就是在"<%!"和"%>"标记之间放置Java的变量声明语句，变量的类型可以是Java的任何数据类型。
  - 在Jsp页面翻译成servlet类以后，将作为成员变量，变量占用的内存空间直到JSP引擎关闭时才释放。
  - 当多个客户访问同一个页面时，JSP将为客户创建的线程之间共享页面成员变量，每个客户线程对页面成员变量的操作，都会影响它的值。

```
<%!
        public Date date=new Date();
        private int count=0;
%>
```

JSP声明脚本

Jsp声明脚本分类

- 声明方法：
  - 就是在”<%!”和”%>”标记之间放置Java的方法声明语句，这些 在整个页面内有效，称为页面的成员方法。
  - 可以在Java服务脚码中调用，在方法内声明的变量成为局部变量，只在方法内有效，方法调用时为局部变量分配空间，调用完毕释放变量空间。

```
<%!
        public String showTime(int hour){
                if(hour<12){
                        return "早上好！ ";
                }else if(hour<18){
                        return "下午好！ ";
                }else{
                        return "晚上好！ ";
                }
        }
%>
```

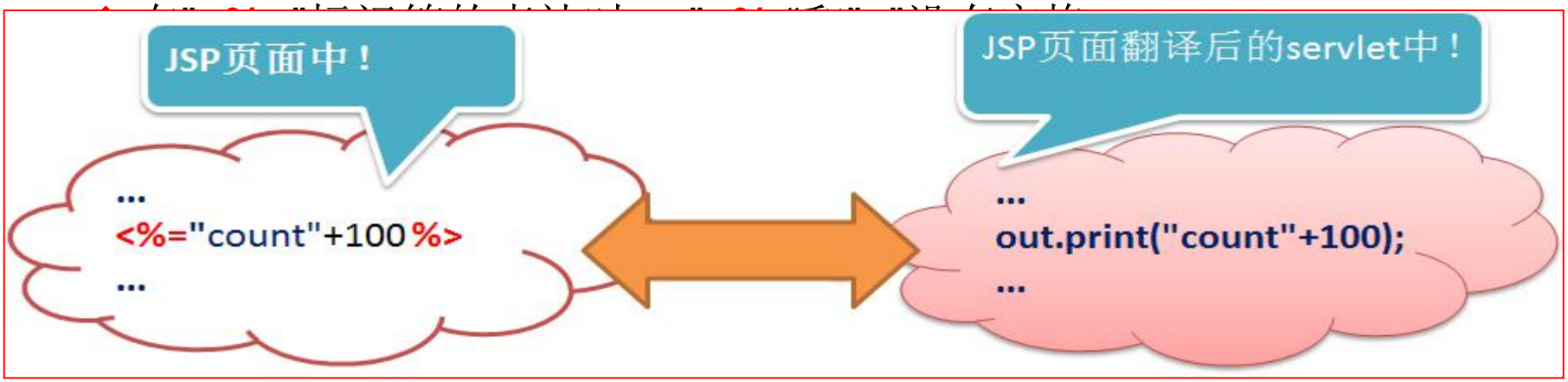
JSP输出脚本

Jsp输出脚本

- 在”<%=”和”%>”标记之间放置Java表达式。
  - 表达式的格式举例： <%=Expression %>

Jsp输出脚本的运行原理

- 在Jsp页面中的”<%=”和”%>”标记之间放置Java表达式，在该页面被客户端请求时，表达式的值由服务器负责运算，并将计算值转换成字符串发送给客户端显示。
- 表达式可以按照值来处理，放置到静态页面元素的合适位置。



课堂练习

课堂练习1：第一个JSP应用

- 任务描述
  - 编写JSP网页，输出“HelloWorld”
  - 每次输出HelloWorld的数量随登录次数增加，每次增加1个
- 关联知识点
  - 编写简单的JSP网页
  - 理解JSP与一般HTML网页的区别
  - 在JSP中使用简单的java语句
- 实现步骤
  - 创建工程FirstJsp
  - 在WebContent文件夹下建立first.jsp
  - 编写first.jsp

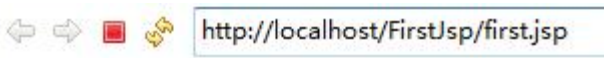
课堂练习

页面运行效果

第一次登录

HelloWorld

第二次登录



HelloWorld

HelloWorld

JSP指令元素脚本

Jsp指令元素脚本

- Jsp指令概述：
  - 向容器发送消息，设置全局变量，不产生输出，将有关页面的特殊处理信息，发送给容器，一个指令影响整个页面。
- Jsp指令的要点：
  - Jsp指令以”<%@”标记开始，以”%>”标记结束，标记和”@”之间要分开。
  - 标记中确定某些方面的特征叫做属性，属性的取值方式，以属性值对的方式来实现，采用attribute-name=”value”的模式，同一指令的不同属性间空格隔开。
    - 标记语法举例:

```
<%@ directive-name
        attribute-name1=“attribute-value1”
        attribute-name2=“attribute-value2”
        …
%>
```

- 如果一个指令有多个属性，这多个属性可以写在一个指令中，也可以分开写。
- **<%@ page contentType="text/html;charset=gb2312"%>**
- **<%@ page import="java.util.Date"%>**

<%@ page contentType="text/html;charset=gb2312" import="java.util.Date"%>

有三种指令标签

- **<%@ page ... %>**
- **page指令用于定义JSP页面的各种属性.**
- **<%@ include ... %>** 包括转换阶段的一个文件。
- **<%@ taglib ... %>** 声明一个在页面中使用的标签库，包含自定义操作。

JSP指令元素脚本

Jsp指令元素脚本

- 常用Jsp指令 ：
  - page指令：无论page指令出现在JSP页面中的什么地方，它作用的都是整个JSP页面，为了保持程序的可读性和遵循良好的编程习惯，page指令最好是放在整个JSP页面的起始位置。
    - 用来定整个Jsp页面的一些属性，这些属性除import外只能出现一次。
    - 通过page指令的contentType属性，告诉web容器，返回给客户端页面的类型以及编码格式。

```
<%@ page contentType="text/html;charset=GBK" … %>
```

    - 通过import属性导入Java包，可以一次导入多个包，逗号分开。

```
<%@ page import="java.util.Date , java.text.SimpleDateFormat" … %>
<%@ page import="java.util.Date" , import="java.io.File" … %>
```

    - Import属性是page指令中唯一可以重复出现的属性。

```
<%@ page import="java.util.Date" … %>
<%@ page import=java.text.SimpleDateFormat" … %>
```

JSP指令元素脚本

Jsp指令元素脚本

- 常用Jsp指令：
  - page指令：
    - page指令language属性，定义Jsp页面使用的脚本语言。

```
<%@ page language="java" … %>
```

- 通过page指令的session属性，用于设置是否需要使用内置的session对象。

```
<%@ page session="true" … %>
```

- 通过isThreadSafe属性取值ture或者false，设置页面允许或禁止多用户。

```
<%@ page isThreadSafe="true" … %>
```

- Info属性可以为页面准备一个字符串，页面中可以通过getServletInfo()方法获取。

```
<%@ page info="I love jsp!" … %>
```

```
<%
        …
        String info=geServletInfo();  %>
```

@include指令

- include指令用于引入其它JSP页面，如果使用include指令引入了其它JSP页面，那么JSP引擎将把这两个JSP翻译成一个servlet。所以include指令引入通常也称之为静态引入。
- 语法：<%@ include file="relativeURL"%>，其中的file属性用于指定被引入文件的路径。路径以“/”开头，表示代表当前web应用

include指令细节注意问题：

- 被引入的文件必须遵循JSP语法。
- 被引入的文件可以使用任意的扩展名，即使其扩展名是html，JSP引擎也会按照处理jsp页面的方式处理它里面的内容，为了见明知意，JSP规范建议使用.jspf（JSP fragments(片段)）作为静态引入文件的扩展名。
- 由于使用include指令将会涉及到2个JSP页面，并会把2个JSP翻译成一个servlet，所以这2个JSP页面的指令不能冲突（除了pageEncoding和导包除外）。

- head.jspf
- <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
- <h1 style="color:red;">网页头部</h1>
- foot.jspf
- <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
- <h1 style="color:blue;">网页尾部</h1>

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>jsp的Include指令测试</title>
  </head>  <body>
  <%--使用include标签引入引入其它JSP页面--%>
    <%@include file="/jspfragments/head.jspf" %>
    <h1>网页主体内容</h1>
    <%@include file="/jspfragments/foot.jspf" %>
  </body>
</html>
```

JSP注释

- **Jsp注释**
  - 适当的注释可以增加程序的可读性，它可以方便程序的调试和维护。
- **Jsp注释分类**
  - 输出型注释：
    - 输出型注释的内容写在”<!- -”和”- ->”之间。格式如图：

```
…
<!– 注释内容 [<%=表达式 %>] -->
…
```

    - 输出型是指会被Jsp引擎发送给客户端浏览器的注释，这种注释可以在浏览器的源码中看到，浏览器将其作为HTML的注释处理， 注意[<%=表达式 %>]是可以将服务端计算后的结果，发送到客户端浏览器，并出现在注释部分。

- <html>
- <head><title>A Comment Test</title></head>
- <body>
- <h2>A Test of Comments</h2>
- <%-- This comment will not be visible in the page source --%>
- </body>
- </html>

JSP注释

- **Jsp注释**
  - 适当的注释可以增加程序的可读性，它可以方便程序的调试和维护。
- **Jsp注释分类**
  - 隐藏型注释：
    - 在标记”<%– –”和”– –%>”之间加入的内容称为隐藏型注释，会被Jsp引擎忽略，不会发送到客户端浏览器中。其格式如图：

```
<%– 注释内容 --%>
```

    - 注意：在Jsp页面服务端脚码中可以使用“//、/*内容*/、/**内容*/”。

```
<%
        //单行注释
        /*
         *多行注释
         */
        /**
          *JavaDoc注释
          */
%>
```

JSP - 操作

- 语法 目的
- **jsp:include** 当请求页面时，包含一个文件
- **jsp:useBean** 发现或实例化一个 **JavaBean**
- **jsp:setProperty JavaBean** 的属性集
- **jsp:getProperty** 将 **JavaBean** 的属性嵌入到输出中
- **jsp:forward** 将请求转发给一个新页面
- **jsp:plugin** 生成浏览器-特定代码，为 **Java** 插件创建 **OBJECT** 或 **EMBED** 标签
- **jsp:element** 动态的定义 **XML** 元素
- **jsp:attribute** 定义了动态定义的 **XML** 元素的属性
- **jsp:body** 定义了动态定义 **XML** 元素的 **body**
- **jsp:text** 用于在 **JSP** 页面和文档中编写模板

JSP动作脚本

Jsp动作脚本

- Jsp动作标记是一种特殊的标签，它影响Jsp运行时的功能，是在运行时才对Jsp动作标签指定的业务进行处理，更易操作和维护。

Jsp常用动作标记

- jsp:include动作标记：该动作标记用来在Jsp页面中动态包含一个文件，包含页面程序和被包含程序是彼此独立的，互不影响。格式如图：

```
<jsp:include page="url" />
或
<jsp:include page="url" >...</jsp:include>
```

- jsp:include动作标记和include指令标记的区别？
  - 处理原理？
    - jsp:include动态包含是在执行时，将被包含文件的运行结果传送给客户端，而include指令是先和被包含文件合并，再运行。
  - 处理特性？
    - Jsp:include可以包含动、静态文件，使用灵活，后者执行速度快。

JSP页面的构造示例

- 示例业务：
  - 声明的变量，作为创建页面中一个获取服务器端时间的对象。
  - 声明方法，作为创建页面中的一个根据输入的小时数，返回“上午好！”、“下午好！”、“晚上好！”三种问候信息。
  - 为了美观，使用jsp指令**动态导入**一个可以作为页面头部的页面文件。
  - 在页面Java程序片中添加业务，调用可以根据**系统时间返回问候信息的方法，传递可以获得系统时间的变量为参数**，获取问候信息。
  - 通过表达式，将问候信息显示到页面指定为位置。

JSP页面的构造示例

Jsp页面代码内容构成简介

```
<%@ page language="java" import="java.util.Date" contentType="text/html;
charset=GBK" %>
<%!
private Date date=new Date();
public String showTime(int hour){
if(hour<12){
return "早上好！";
}else if(hour<18){
return "下午好！";
}else{
return "晚上好！";
}
}
%>
<html>
<head>
<title>Get information by time !</title>
</head>
<body>
<%-- JSP注释--%>
<%@ include file="top.htm" %>
<%
String infor=showTime(date.getHours());
if(infor!=null){
%>
<p>现在时间为：<%=date.getHours() %>点，得到的问候是<%=infor %></p>
<%
}
%>
</body></html>
```

JSP页面的构造示例

- Jsp页面示例运行效果

课堂练习

课堂练习2：动态包含的使用

- 任务描述
  - 定义2个要包含的文件：
  - Info.html：

```
<h2><font color =“red”>
Info.html</font></h2>
```

  - info.jsp：

```
<h2><font color =“green”>
<%=“info.jsp”%></font></h2>
```

  - 在index.jsp中使用动态包含指令包含以上两个文件
- 关联知识点
  - 动态包含指令的使用
  - 动态包含和静态包含的区别
- 实现步骤
  - 创建两个要包含的文件，并在index.jsp中使用动态包含指令。

课堂练习

- 页面运行效果


Insert title here - Windows Internet Explorer
http://localhost/IncludeTest/
收藏夹
建议网站
网页快讯库
Insert title here
动态包含操作
info.html
info.jsp

JavaBean

- JavaBean是一个遵循特定写法的Java类
- 它通常具有如下特点：

- 这个Java类必须具有一个无参的构造函数

属性必须私有化。

私有化的属性必须通过public类型的方法暴露给其它程序，并且方法的命名也必须遵守一定的命名规范。

```
package gacl.javabean.study;
public class Person {
    private String name;
    private String sex;
    private int age;
    private boolean married;
    public Person() {
    }
    public String getName() {
        return name;
    }
   public void setName(String name) {
        this.name = name;
    }
    public String getSex() {
        return sex;
    }
    public void setSex(String sex) {
        this.sex = sex;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public boolean isMarried() {
```

- JavaBean在J2EE开发中，通常用于封装数据，对于遵循以上写法的JavaBean组件，其它程序可以通过反射技术实例化JavaBean对象，并且通过反射那些遵守命名规范的方法，从而获知JavaBean的属性，进而调用其属性保存数据。

JavaBean的属性

- **JavaBean的属性可以是任意类型，并且一个JavaBean可以有多个属性。每个属性通常都需要具有相应的setter、getter方法，setter方法称为属性修改器，getter方法称为属性访问器。**

属性修改器必须以小写的set前缀开始，后跟属性名，且属性名的第一个字母要改为大写，例如，name属性的修改器名称为setName，password属性的修改器名称为setPassword。

属性访问器通常以小写的get前缀开始，后跟属性名，且属性名的第一个字母也要改为大写，例如，name属性的访问器名称为getName，password属性的访问器名称为getPassword。

一个JavaBean的某个属性也可以只有set方法或get方法，这样的属性通常也称之为只写、只读属性

在JSP中使用JavaBean

- <jsp:useBean>标签：用于在JSP页面中查找或实例化一个JavaBean组件。
- <jsp:setProperty>标签：用于在JSP页面中设置一个JavaBean组件的属性。
- <jsp:getProperty>标签：用于在JSP页面中获取一个JavaBean组件的属性。

- **<jsp:useBean>标签用于在指定的域范围内查找指定名称的JavaBean对象，如果存在则直接返回该JavaBean对象的引用，如果不存在则实例化一个新的JavaBean对象并将它以指定的名称存储到指定的域范围中。**
- 常用语法：
- **<jsp:useBean id="beanName" class="package.class" scope="page|request|session|application"/>**
- **"id"属性用于指定JavaBean实例对象的引用名称和其存储在域范围中的名称。**
- **"class"属性用于指定JavaBean的完整类名（即必须带有包名）。**
- **"scope"属性用于指定JavaBean实例对象所存储的域范围，其取值只能是page、request、session和application等四个值中的一个，其默认值是page。**

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%--在jsp中使用jsp:useBean标签来实例化一个Java类的对象
<jsp:useBean id="person" class="gacl.javabean.study.Person" scope="page"/>
    ├<jsp:useBean>：表示在JSP中要使用JavaBean。
    ├id:表示生成的实例化对象，凡是在标签中看见了id，则肯定表示一个实例对象。
    ├class：此对象对应的包.类名称
    ├scope：此javaBean的保存范围，四种范围：page、request、session、application
--%>
<jsp:useBean id="person" class="gacl.javabean.study.Person" scope="page"/>
<%
   //person对象在上面已经使用jsp:useBean标签实例化了，因此在这里可以直接使用person对象
   //使用setXxx方法为对象的属性赋值
   //为person对象的name属性赋值
   person.setName("孤傲苍狼");
   //为person对象的Sex属性赋值
   person.setSex("男");
   //为person对象的Age属性赋值
   person.setAge(24);
   //为person对象的married属性赋值
   person.setMarried(false);
%>
<!DOCTYPE HTML>
<html>
  <head>
    <title>jsp:useBean标签使用范例</title>
  </head>
  <body>
    <%--使用getXxx()方法获取对象的属性值--%>
```

<jsp:useBean> 操作

- **useBean 操作具有多种用途。它首先利用 id 和 scope 变量搜索现有对象。如果没有找到一个对象，那么它会试图创建指定的对象。**

加载 bean 的最简单的方式如下：

```
<jsp:useBean id="name" class="package.class" />
```

加载 bean 类完成后，你可以使用 jsp:setProperty 和 jsp:getProperty 操作来修改和检索 bean 属性。

<jsp:setProperty>标签

- **<jsp:setProperty>标签用于设置和访问JavaBean对象的属性。**

语法格式一：

```
<jsp:setProperty name="beanName" property="propertyName" value="string字符串"/>
```

语法格式二：

```
<jsp:setProperty name="beanName" property="propertyName" value="<%= expression %>" />
```

语法格式三：

```
<jsp:setProperty name="beanName" property="propertyName"
param="parameterName"/>
```

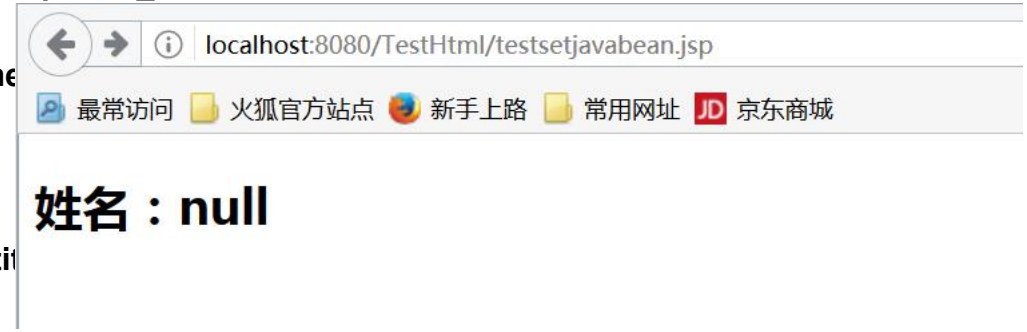
语法格式四：

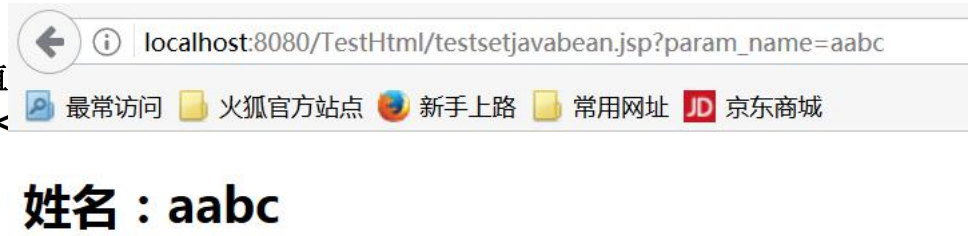
```
<jsp:setProperty name="beanName" property= "*" />
```

setProperty 操作

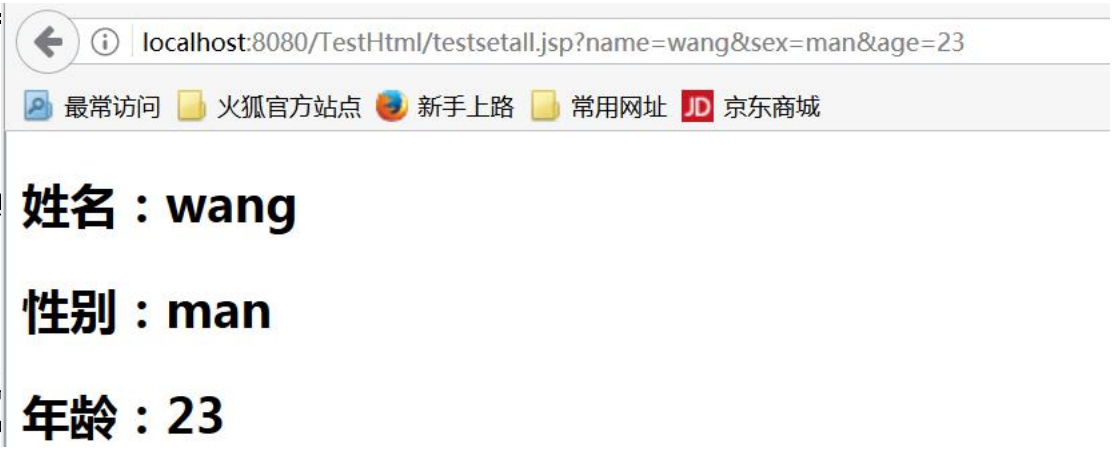
- name属性用于指定JavaBean对象的名称。
- property属性用于指定JavaBean实例对象的属性名。
- value属性用于指定JavaBean对象的某个属性的值，value的值可以是字符串，也可以是表达式。为字符串时，该值会自动转化为JavaBean属性相应的类型，如果value的值是一个表达式，那么该表达式的计算结果必须与所要设置的JavaBean属性的类型一致。
- param属性用于将JavaBean实例对象的某个属性值设置为一个请求参数值，该属性值同样会自动转换成要设置的JavaBean属性的类型。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<jsp:useBean id="person" class="gacl.javabean.study.Person" scope="page"/>
<%--
   jsp:setProperty标签可以使用请求参数为bean的属性赋值
   param="param_name"用于接收参数名为param_name的参数值，然后将接收到的值赋给name属性
--%>
<jsp:setProperty property="name" name
<!DOCTYPE HTML>
<html>
  <head>
    <title>jsp:setProperty标签使用范例</ti
  </head>

  <body>
     <%--使用getXxx()方法获取对象的属性值
      <h2>姓名：<%=person.getName()%><
  </body>
</html>
```





```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<jsp:useBean id="person" class="gacl.javabean.study.Person" scope="page"/>
<%--
   jsp:setProperty标签用所有的请求参数为bean的属性赋值
   property="*"代表bean的所有属性
--%>
<jsp:setProperty property="*" name
<!DOCTYPE HTML>
<html>
 <head>
   <title>jsp:setProperty标签使用范例
 </head>

 <body>
   <%--使用getXxx()方法获取对象的属
   <h2>姓名：<%=person.getName()
   <h2>性别：<%=person.getSex()%></h2>
   <h2>年龄：<%=person.getAge()%></h2>
 </body>
</html>
```



```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<jsp:useBean id="person" class="gacl.javabean.study.Person" scope="page"/>
<%--使用jsp:setProperty标签设置person对象的属性值
jsp:setProperty在设置对象的属性值时会自动把字符串转换成8种基本数据类型
但是jsp:setProperty对于复合数据类型无法自动转换--%>
<jsp:setProperty property="name" name="person" value="白虎神皇"/>
<jsp:setProperty property="sex" name="person" value="男"/>
<jsp:setProperty property="age" name="person" value="24"/>
<jsp:setProperty property="married" name="person" value="false"/>
<%--
birthday属性是一个Date类型，这个属                                                    是会
报错的
<jsp:setProperty property="birthda
--%>
<jsp:setProperty property="birthda
<!DOCTYPE HTML>
<html>
  <head>
    <title>jsp:setProperty标签使用范
  </head>

  <body>
    <%--使用getXxx()方法获取对象的属性值 --%>
    <h2>姓名：<%=person.getName()%></h2>
    <h2>性别：<%=person.getSex()%></h2>
    <h2>年龄：<%=person.getAge()%></h2>
    <h2>已婚：<%=person.isMarried()%></h2>
```

姓名：白虎神皇

性别：男

年龄：24

已婚：false

出生日期：Thu Apr 20 22:25:29 CST 2017

<jsp:getProperty>标签

- **<jsp:getProperty>标签用于读取JavaBean对象的属性，也就是调用JavaBean对象的getter方法，然后将读取的属性值转换成字符串后插入进输出的响应正文中。**

语法：
<jsp:getProperty name="beanInstanceName" property="PropertyName" />
name属性用于指定JavaBean实例对象的名称，其值应与<jsp:useBean>标签的id属性值相同。
property属性用于指定JavaBean实例对象的属性名。

如果一个JavaBean实例对象的某个属性的值为null，那么，
使用<jsp:getProperty>标签输出该属性的结果将是一个内容为“null”的字符串。

- <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
- <%--
- 在jsp中使用jsp:useBean标签来实例化一个Java类的对象
- <jsp:useBean id="person" class="gacl.javabean.study.Person" scope="page"/>
- ├<jsp:useBean>：表示在JSP中要使用JavaBean。
- ├id:表示生成的实例化对象，凡是在标签中看见了id，则肯定表示一个实例对象。
- ├class：此对象对应的包.类名称
- ├scope：此javaBean的保存范围，四种范围：page、request、session、application
- --%>
- <jsp:useBean id="person" class="gacl.javabean.study.Person" scope="page"/>

- <%--
- 使用jsp:setProperty标签设置person对象的属性值
- jsp:setProperty在设置对象的属性值时会自动把字符串转换成8种基本数据类型
- 但是jsp:setProperty对于复合数据类型无法自动转换
- --%>
- <jsp:setProperty property="name" name="person" value="白虎神皇"/>
- <jsp:setProperty property="sex" name="person" value="男"/>
- <jsp:setProperty property="age" name="person" value="24"/>
- <jsp:setProperty property="married" name="person" value="false"/>
- <%--
- birthday属性是一个Date类型，这个属于复合数据类型，因此无法将字符串自动转换成Date ，用下面这种写法是会报错的
- <jsp:setProperty property="birthday" name="person" value="1988-05-07"/>
- --%>
- <jsp:setProperty property="birthday" name="person" value="<%=new Date()%>"/>

```
package action;
public class TestBean {
private String message = "No message specified";
public String getMessage() {
return(message);
}
public void setMessage(String message) {
this.message = message;
}
}
```

main.jsp

```
<html>
<head>
<title>Using JavaBeans in JSP</title>
</head>
<body>
<center>
<h2>Using JavaBeans in JSP</h2>
<jsp:useBean id="test" class="action.TestBean" />
<jsp:setProperty name="test"
property="message"
value="Hello JSP..." />
<p>Got message....</p>
<jsp:getProperty name="test" property="message" />
</center>
</body>
</html>
```

第3节访问JSP

⑩ 知识点预览

#	知识点	难点	重点	应用	说明
1	客户端访问JSP		√	√	掌握JSP页面的创建和访问
2	Servlet访问JSP		√	√	掌握Servlet访问JSP页面
3	访问安全性	√	√	√	掌握安全访问JSP资源方式
4	服务端跳转		√	√	掌握servlet服务端传递请求
5	服务端跳转与重定向的区别	√	√		能够区分服务传递和重定向的区别

客户端访问JSP

客户端访问JSP页面：

- 通过客户端浏览器：
  - 在浏览器地址栏中，输入欲访问的资源URL。通用的格式如图：

```
http://服务器地址:端口号/虚拟目录/资源名称
```

  - 如果将容器中，conf目录下的server.xml中的服务端口号为默认80，可以省略端口号，格式如图：

```
http://服务器地址/虚拟目录/资源名称
```

  - 通过页面中的超链接访问其他站点Jsp页面，格式如图：

```
<a href="http://服务器地址/虚拟目录/资源名称">链接文本</a>
```

  - 通过页面中的超链接访问本站点Jsp页面，格式如图：

```
<a href="/虚拟目录/…目录/XXX.jsp">链接文本</a>
```

客户端访问JSP

- 客户端访问JSP页面示例：
  - login.jsp文件
  - 提交用户名，密码到servlet
  - servlet实现跳转控制

```
<body>
    <form action="login">
username:<input type="text"  name="username">
<br>
 password:<input type="password" name="pwd">
<br>
    <input type="submit">
    </form>
  </body>
```

Servlet访问JSP

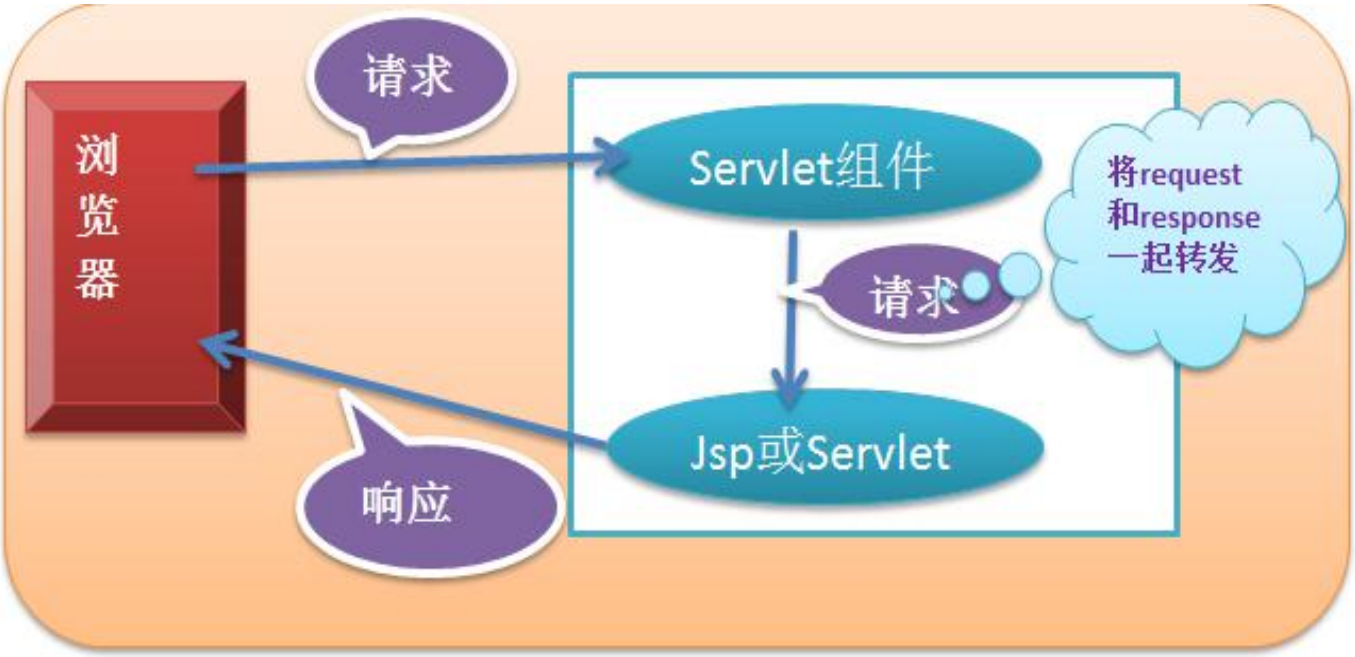
Servlet访问JSP：

- Servlet服务端程序，通过服务端传递请求访问指定JSP。
  - 重写Servlet类中的doGet (request,response)或者doPost(request,response)方法。让另外一个方法调用被重写的方法，这样，可以实现对get请求和post请求都能响应。
  - 在重写的服务响应方法中，通过接收客户端请求后而创建的线程体中的request请求对象，调用getRequestDispatcher(destFilePath)方法，获得可以转发到目标组件的转发器对象。
  - 通过获得可以实现服务端传递请求的RequestDispatcher对象，调用forward(request,response)方法，实现请求的传递，而且请求对象还是第一个request对象，在此过程中，可以添加新的信息实现传递。
  - 格式如图：

```
…
request.getDispatcher(destFilePath).forward(request,response);
…
```

Servlet访问JSP实例

Servlet访问JSP实例步骤：



Servlet访问JSP

- **Servlet访问JSP时doGet(request,response)主要内容：**
  - 通过转发器实现服务器端转发请求步骤
    - 声明转发器接口RequestDispatcher对象。
    - 通过request.getRequestDispatcher(String path)方法获得转发器对象引用。
    - 通过dispatcher.forward(request,response)方法，实现Servlet访问Jsp页面。

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        //声明转发器
        RequestDispatcher dispatcher=null;
        //通过request请求对象调用getRequestDispathcer(String path)方法，
                //代理转发器的引用
        dispatcher=request.getRequestDispatcher("/success.jsp");
        if(dispatcher!=null){
                //通过请求转发器调用forward()方法，实现服务器端servlet
                //传递HTTP请求。从当前的servlet到指定的Jsp。
                dispatcher.forward(request, response);
        }
}
```

```
public class CheckAccount extends HttpServlet {
 @Override
 protected void doPost(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException {
 doGet(req,resp);
 }
 @Override
 public void doGet(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException {
 HttpSession session = req.getSession();
 String username = req.getParameter("username");
 String pwd = req.getParameter("pwd");
 RequestDispatcher dispatcher=null;
 if((username != null)&&(username.trim().equals("abc"))) {
  if((pwd != null)&&(pwd.trim().equals("123"))) {
   System.out.println("success"+username);
   session.setAttribute("username", username);
  //resp.sendRedirect("success.jsp");
   req.getRequestDispatcher("success.jsp").forward(req, resp);
   return;
  }
 }
 resp.sendRedirect("fail.jsp");
 return;
 }
}
```

web.xml配置文件

- <servlet>
- <servlet-name>login</servlet-name>
- <servlet-class>com.oraclesdp.csg.webapp.action.CheckAccount</servlet-class>

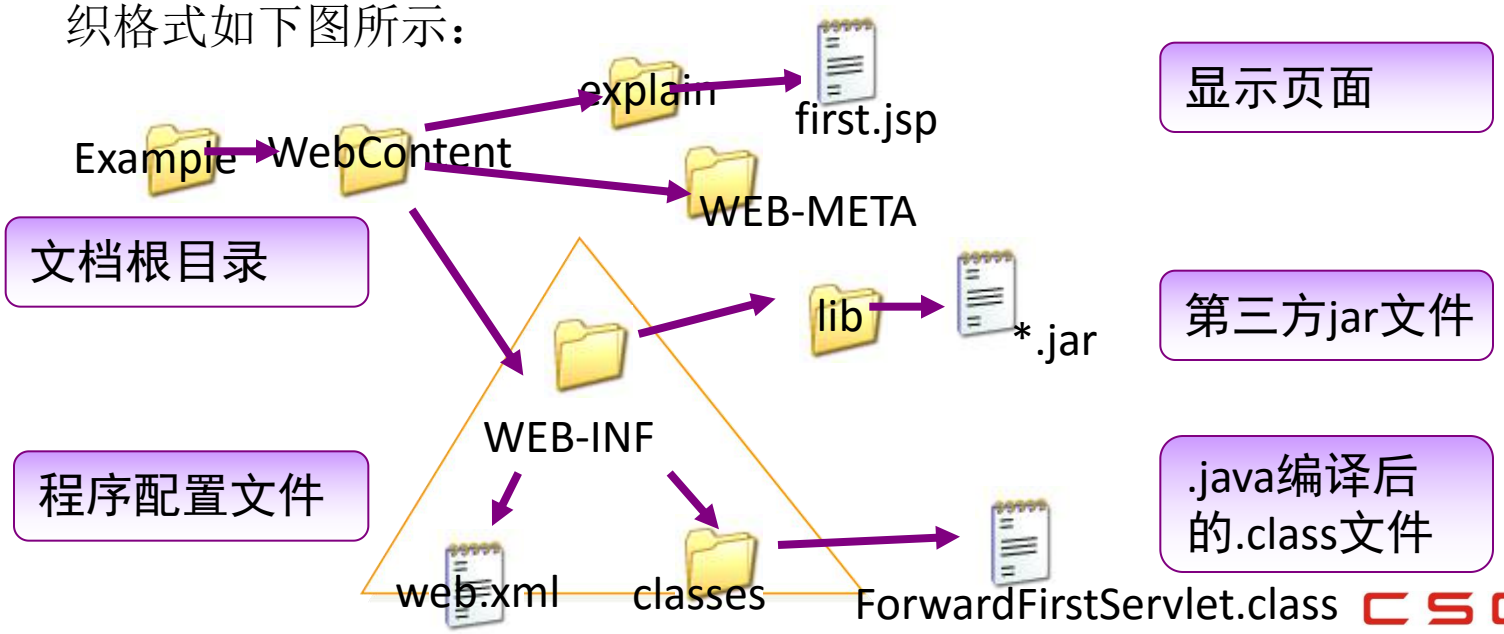
</servlet>

- <servlet-mapping>
- <servlet-name>login</servlet-name>
- <url-pattern>/login</url-pattern>
- </servlet-mapping>

访问安全性

访问Jsp页面时的安全机制

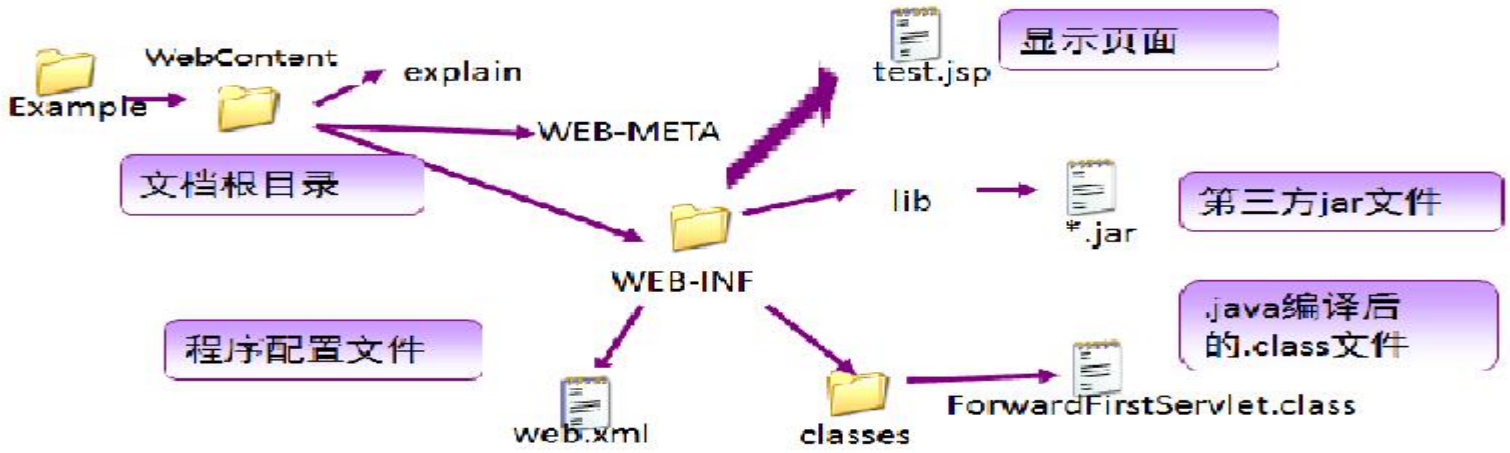
- 访问Jsp页面资源时的安全隐患：
  - 直接在浏览器地址栏中输入站点容器中的资源，**容易暴露信息**。
  - 一些**带有后台管理页面的内容，如果事先知道了页面的名字，而站点中又没有安全机制，那么就可以不通过验证而直接操作，这样是非常危险的。**
  - 造成以上现象的原因，是Jsp页面资源直接"裸露"。应用程序的资源组织格式如下图所示：



访问安全性

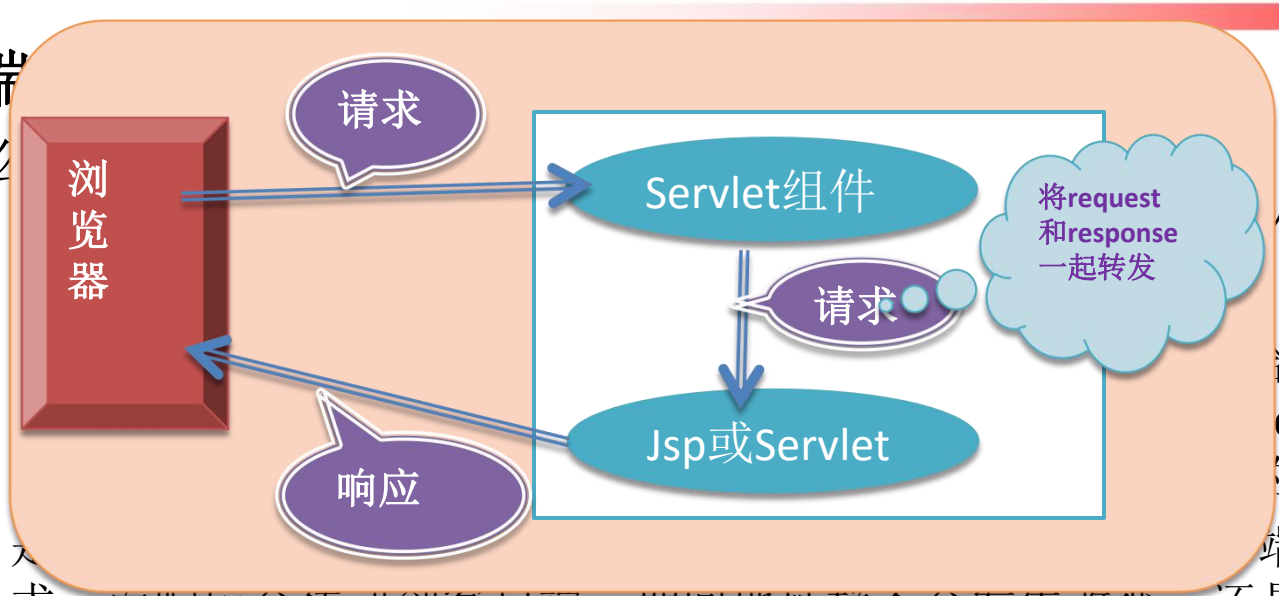
访问Jsp页面时的安全机制

- 安全访问Jsp页面资源的解决方案：
  - 直接将Jsp页面存储在WEB-INF目录中，此时，在客户端浏览器中，即便是写出从站点目录到资源的绝对路径，也是访问不到的，会报路径有误的“404”错误。
  - 通过servlet来访问受限的Jsp页面资源，就可以通过访问servlet来访问资源了。而且Jsp资源得到了有效的保护，在方法中加入业务逻辑控制语句，这样就达到了安全访问控制。
  - 避免Jsp页面资源直接”裸露”的解决方案。应用程序的资源组织格式如图所示：



服务端跳转

- 服务端
  - 什么
    - 个页面，
    - 另外一 equest) 的请求。
    - 端转发请求，而响应会传递到客户端。期间地址栏不会发生改变，还是原来的url地址。
  - 实现服务端跳转主要步骤：
    - 通过request.getRequestDispatcher(转发目标rul)获得转发器对象。
    - 通过RequestDispatcher对象，调用forward(request,response)方法转发请求。



服务端跳转与重定向的区别

- 服务
  - 服
  - 重

从当前的Servlet
作用范围内。转发

发送一个新请求，
let中，通过响应

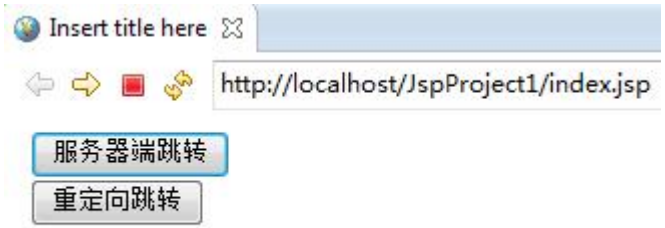
课堂练习

课堂练习3：服务器端跳转和重定向跳转

- 任务描述
  - 编写一个简单的JSP网页index.jsp，网页中只有两个按钮
  - 按下两个按钮后分别实现，服务器段跳转和重定向跳转
  - 服务器端跳转到WEB-INF目录下的forward.jsp
  - 重定向跳转到WebContent目录下的redirect.jsp
- 关联知识点
  - 编写简单的JSP网页
  - 理解服务器端跳转和重定向跳转的区别
  - 掌握如何访问WEB-INF目录下的文件
- 实现步骤
  - 在相应目录下创建文件
  - 编写servlet类实现跳转

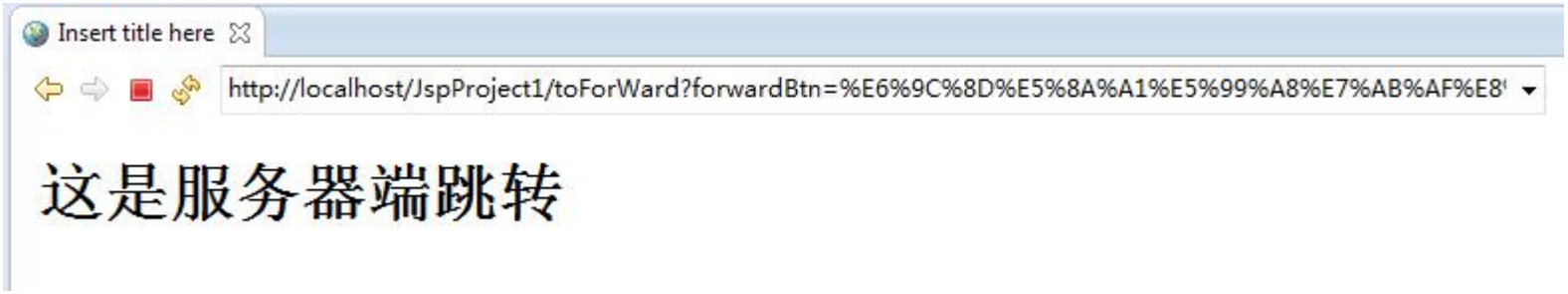
课堂练习

页面运行效果



重定向跳转

服务器端跳转



第4节 JSP工作原理

⑩ 知识点预览

#	知识点	重点	难点	应用	说明
1	嵌入服务端代码		√	√	掌握JSP页面中嵌入服务器代码方法
2	JSP翻译过程	√	√		理解JSP执行过程的翻译阶段
3	JSP编译过程	√	√		理解JSP执行过程的编译阶段
4	JSP响应过程	√	√		理解JSP执行过程的响应阶段

JSP嵌入服务端代码

JSP嵌入服务端代码

- 将Java业务代码嵌入JSP页面中的方式
  - 在“<%”以及“%>”之间嵌入Java业务逻辑代码。
    - 代码举例：

```
<%@ page language="java" contentType="text/html; charset=GBK"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
<%
    long counter = 0;
    cou
%>
<html>
<head>
<title>
</head>
<body>
    <p>
    <%
        counter = counter + 10;
    %>
    <p>counter=counter+10之后的结果为<br /><%=counter%></p>
</body>
</html>
```

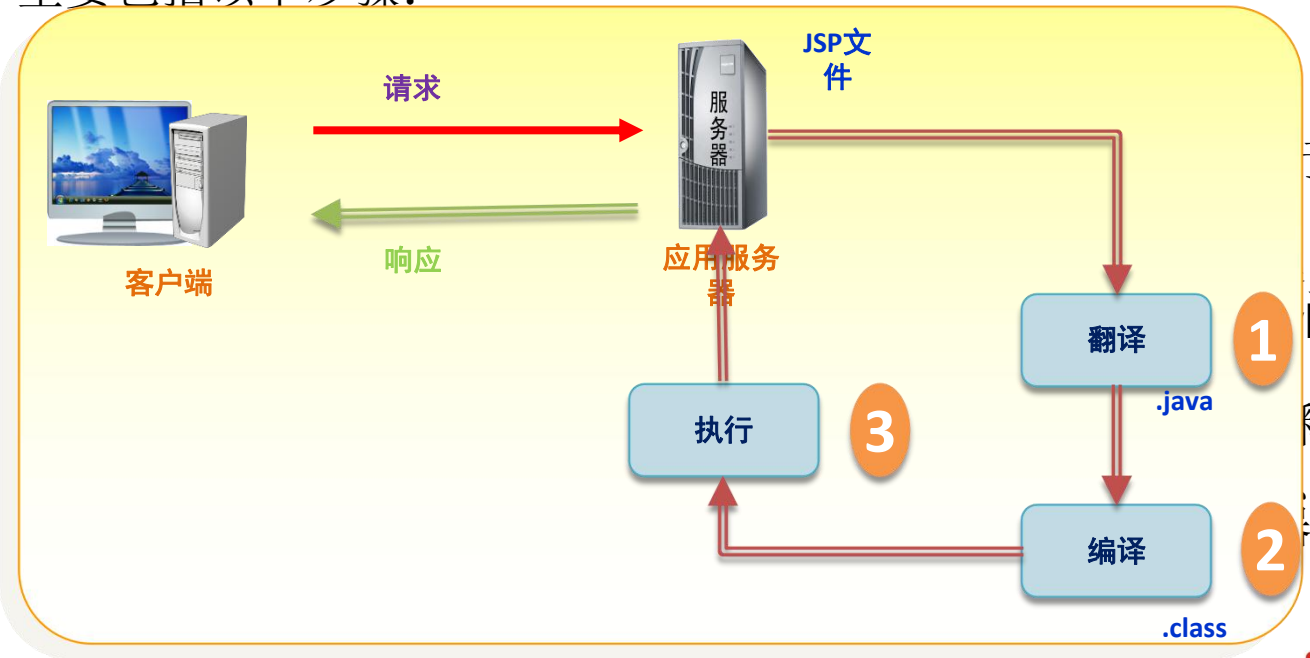
JSP嵌入服务端代码

JSP嵌入服务端代码以及运作机理

- 将Java业务代码嵌入JSP页面中的运行原理。
  - 在一个Jsp页面中可以有多处Java程序片，这些Java程序片将被Jsp引擎按顺序执行。
  - 在一个程序片中声明的变量称做Jsp页面的局部变量，他们从声明位置向后，所有程序部分以及表达式部分都有效。
  - 因为Jsp引擎将Jsp页面转译成servlet类文件时，这些变量作为servlet类service方法的变量，即局部变量。
  - 有时候可以将一个程序片分割成几个更小的程序片，然后在这些小的程序片之间再插入Jsp页面的一些其他标记元素。这样增加程序段代码的可读性。
  - Jsp引擎将页面翻译成Java文件时，Java服务器脚本程序，将作为service方法中的语句，用作业务逻辑处理。

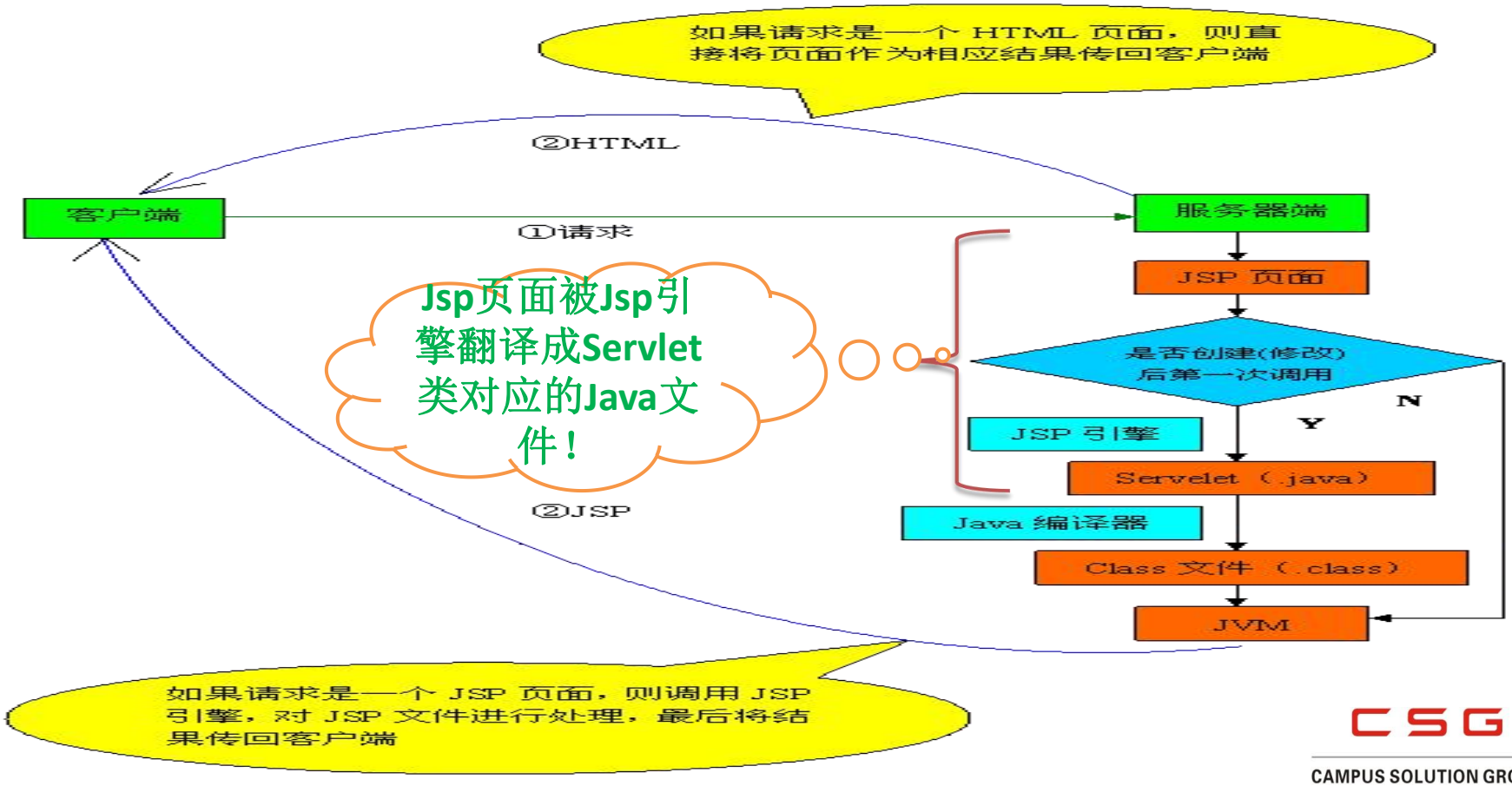
JSP执行过程概述

- **Web容器处理JSP文件请求需要经过3个阶段：**
  - 翻译阶段：JSP文件会被Web容器中的JSP引擎转换成Java源码。
  - 编译阶段：Java源码会被编译成可执行的字节码。
  - 执行阶段：容器接受了客户端的请求后，执行编译成字节码的JSP文件。处理完请求后，容器把生成的页面反馈给客户端进行显示。
  - 运作机理。
    - 主要包括以下步骤：



JSP翻译过程

- **JSP翻译过程。**
  - ◆ 翻译阶段。
    - ➢ Jsp文件将会被Web容器中的Jsp引擎转换成Java源码。原理如图：



JSP翻译过程

JSP翻译过程

- Jsp文件将会被Web容器中的Jsp引擎转换成Java源码。
  - 将Jsp页面中的Java变量的声明，以及方法的声明，转译成servlet的属性和行为。

```
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import java.util.Date;

public final class first_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {
        private Date date=new Date();
        public String showTime(int hour){
                if(hour<12){
                        return "早上好！";
                }else if(hour<18){
                        return "下午好！";
                }else{
                        return "晚上好！";
                }
        }
```

JSP翻译过程

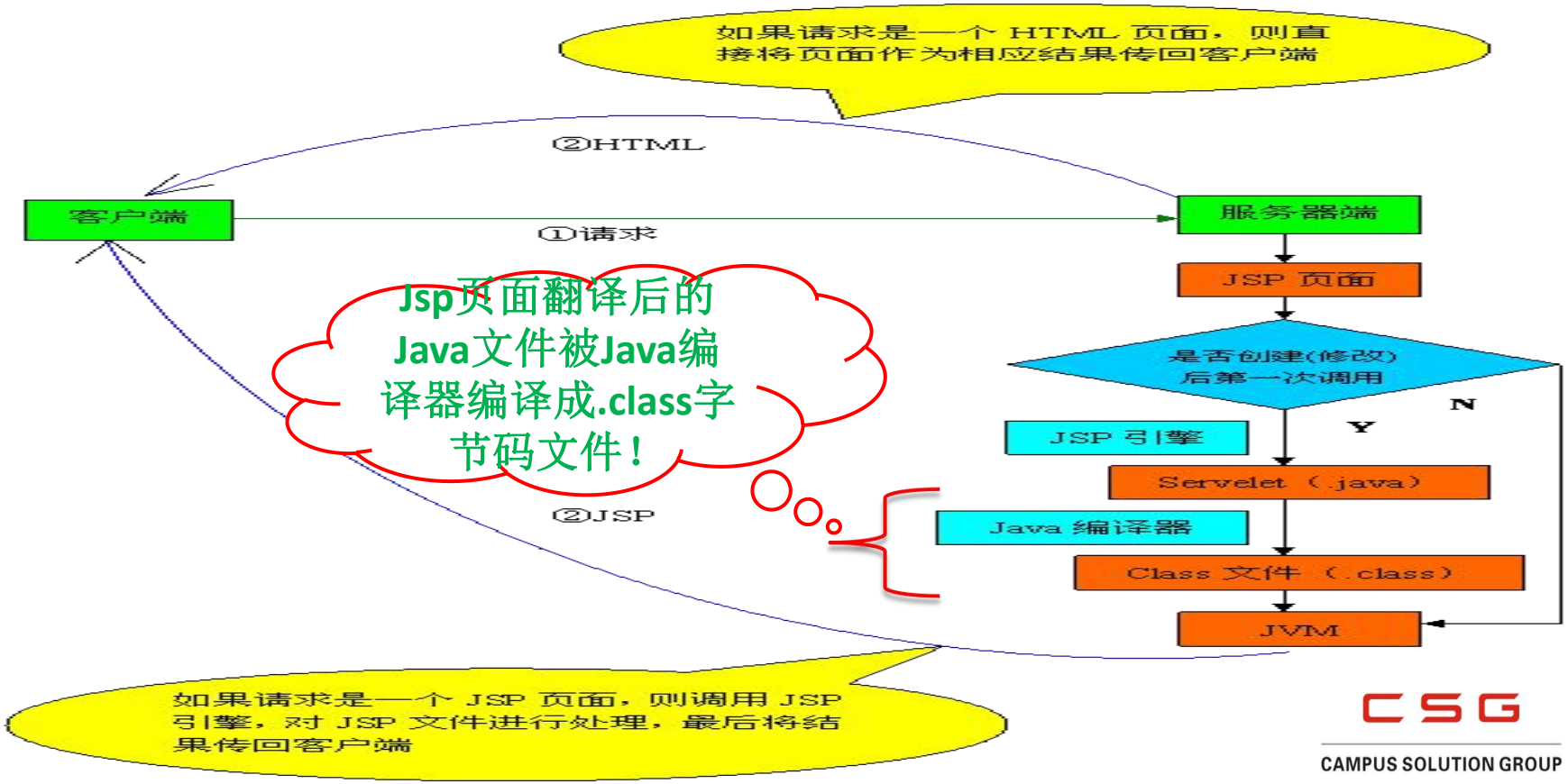
JSP翻译过程

- Jsp文件将会被Web容器中的Jsp引擎转换成Java源码。
  - 将Jsp页面中的Java脚码程序片，转译成servlet的service方法的输出到客户端的输出语句。

```
public void _jspService(HttpServletRequest request, HttpServletResponse response)
      throws java.io.IOException, ServletException {

    try {
      response.setContentType("text/html; charset=GBK");
       out.write("<html>\r\n");
       out.write("<head>\r\n");
       out.write("<title>Get information by time !</title>\r\n");
       out.write("</head>\r\n");
       out.write("<body>\r\n");
       out.write("\r\n");
       out.write("<!--  %@ include file=\"top.htm\" %-->\r\n");
       org.apache.jasper.runtime.JspRuntimeLibrary.include(request, response, "../../html/top.htm", out, false);
       out.write('\r');
       out.write('\n');

         String infor=showTime(date.getHours());
         if(infor!=null){

       out.write("\r\n");
       out.write("\t\t<p>现在时间为：<font color=\"green\" >");
       out.print(date.getHours() );
       out.write("</font>点，得到的问候是<font color=\"green\">");
       out.print(infor );
       out.write("</font></p>\r\n");
       out.write("\t\t");

         }

       out.write("\r\n");
       out.write("</body>\r\n");
       out.write("</html>");
```

JSP编译过程

- **JSP编译过程。**
  - 编译阶段。
    - 翻译之后的Servlet对应Java类，被编译成可执行的字节码。然后，该字节码程序将驻留在容器进程中，被多个客户端进程共享。



JSP响应过程

响应过程

- 执行阶段的业务过程：
  - 当Jsp页面对应的服务器程序，被请求执行时，先执行_jspInit()方法。

```
public void _jspService(HttpServletRequest request, HttpServletResponse response)
      throws java.io.IOException, ServletException {

    try {
      response.setContentType("text/html; charset=GBK");
       out.write("<html>\r\n");
       out.write("<head>\r\n");
       out.write("<title>Get information by time !</title>\r\n");
       out.write("</head>\r\n");
       out.write("<body>\r\n");
       out.write("\r\n");
       out.write("<!--  %@ include file=\"top.htm\" %-->\r\n");
       org.apache.jasper.runtime.JspRuntimeLibrary.include(request, response, "../../html/top.htm", out, false);
       out.write('\r');
       out.write('\n');

         String infor=showTime(date.getHours());
         if(infor!=null){

       out.write("\r\n");
       out.write("\t\t<p>现在时间为：<font color=\"green\" >");
       out.print(date.getHours() );
       out.write("</font>点，得到的问候是<font color=\"green\">");
       out.print(infor );
       out.write("</font></p>\r\n");
       out.write("\t\t");

         }

       out.write("\r\n");
       out.write("</body>\r\n");
       out.write("</html>");
```

Jsp运行的三个阶段

演示Jsp运行的三个阶段

Jsp运行的三个阶段案例演示

案例1： Jsp运行的三个阶段

- 任务描述
  - 在Web项目中创建Jsp页面，然后通过浏览器访问，从而掌握Jsp运行的三个阶段。
- 关联知识点
  - 创建HTML页面静态内容。
  - Jsp项目组织格式，欢迎界面的设置，虚拟目录的设置。
  - Jsp页面构成要素简介。
  - Jsp页面执行过程与原理。
- 实现步骤
  - 创建一个Java Web项目Example，项目Example的WebContent下创建explain目录，然后其中创建一个Jsp页面(test.jsp)。
  - 在创建的test.jsp页面中修改模板自带的内容。
  - 在page指令中通过import属性导入java.util.Date，在页面Java程序片中创建一个Date对象，用于请求时获取服务器系统时间，然后通过表达式取出并显示结果到客户端。
  - 通过观察Web容器资源了解Jsp运行的三个阶段以及生成的相应资源。

```
<html>
<head><title>A Comment Test</title></head>
<body>
<p>Today's date:
<%= (new java.util.Date()).toLocaleString()%>
</p>
</body>
</html>
```

第5节 作用域与隐式对象

⑩ 知识点预览

#	知识点	重点	难点	应用	说明
1	使用服务端脚本		√	√	掌握服务端脚本的使用
2	Java Web四大作用域概述		√		了解Java Web四大作用域
3	page作用域		√	√	掌握并应用page作用域
4	request作用域		√	√	掌握并应用request作用域
5	session作用域		√	√	掌握并应用session作用域
6	application作用域		√	√	掌握并应用application作用域
7	JSP隐式对象		√	√	掌握JSP隐式对象在业务中的应用

使用服务端脚本

- Java服务端脚本的**运行特征**
  - Jsp引擎将页面翻译成Java文件时，将程序段中的变量作为页面Servlet类的**service方法中的局部变量**处理。
  - 程序段中的Java语句作为service方法中的语句处理，最终将页面所有程序段中的变量和语句依次转译到service方法中，被Jsp引擎顺序执行。
  - **当多个客户请求一个Jsp页面时，Jsp引擎为每个客户启动一个线程，不同的客户对应各自的局部变量（被分配在不同的内存空间）。因此，一个客户对Jsp页面局部变量操作的结果，不会影响到其他客户。**
- Java程序段在Jsp应用中常用业务方式
  - 操作页面的成员变量，页面的成员变量在客户线程之间共享。
  - 调用方法，调用的方法必须是页面成员方法或内置对象的方法。
  - 声明和操作程序段变量。

使用服务端脚本

使用服务端脚本

- Java程序段在Jsp应用中常用业务方式示例
  - Eclipse中创建java web项目EgScriptlet
    - 操作页面的成员变量，页面的成员变量在客户线程之间共享。
      - ✓ 在WebContent/egscriptlet下创建counter.jsp，并在其中创建成员变量int count=100，在页面的服务器程序段中修改count。并通过表达式，显示在页面的指定位置，并模拟多客户请求。
    - 调用方法，调用的方法必须是页面成员方法或内置对象的方法。
      - ✓ 在counter.jsp页面中的声明部分，定义void addValue(int value)方法，实现向count变量上累加value 值的业务。然后在服务器脚本中调用该方法，并在页面的指定部分显示调用后的影响。
    - 声明和操作程序段变量。
      - ✓ 在counter.jsp页面中的服务端脚本部分，创建一个整型变量localValue=200，然后，在页面的指定位置显示localValue。同时，模仿多用户访问，掌握localValue由访问而生存且不能共享的特点。

JSP属性范围

- 所谓的属性范围就是一个属性设置之后，可以经过多少个其他页面后仍然可以访问的保存范围.
- 四种属性范围分别指以下四种：

当前页：一个属性只能在一个页面中取得，跳转到其他页面无法取得

一次服务器请求：一个页面中设置的属性，只要经过了服务器跳转，则跳转之后的页面可以继续取得。

一次会话：一个用户设置的内容，只要是与此用户相关的页面都可以访问（一个会话表示一个人，这个人设置的东西只要这个人不走，就依然有效）

上下文中：在整个服务器上设置的属性，所有人都可以访问

属性的操作方法

public void setAttribute(String name,Object value)

设置属性

public object getAttribute(String name)取得属性

public void removeAttribute(String name)删除属性

Java Web四大作用域概述

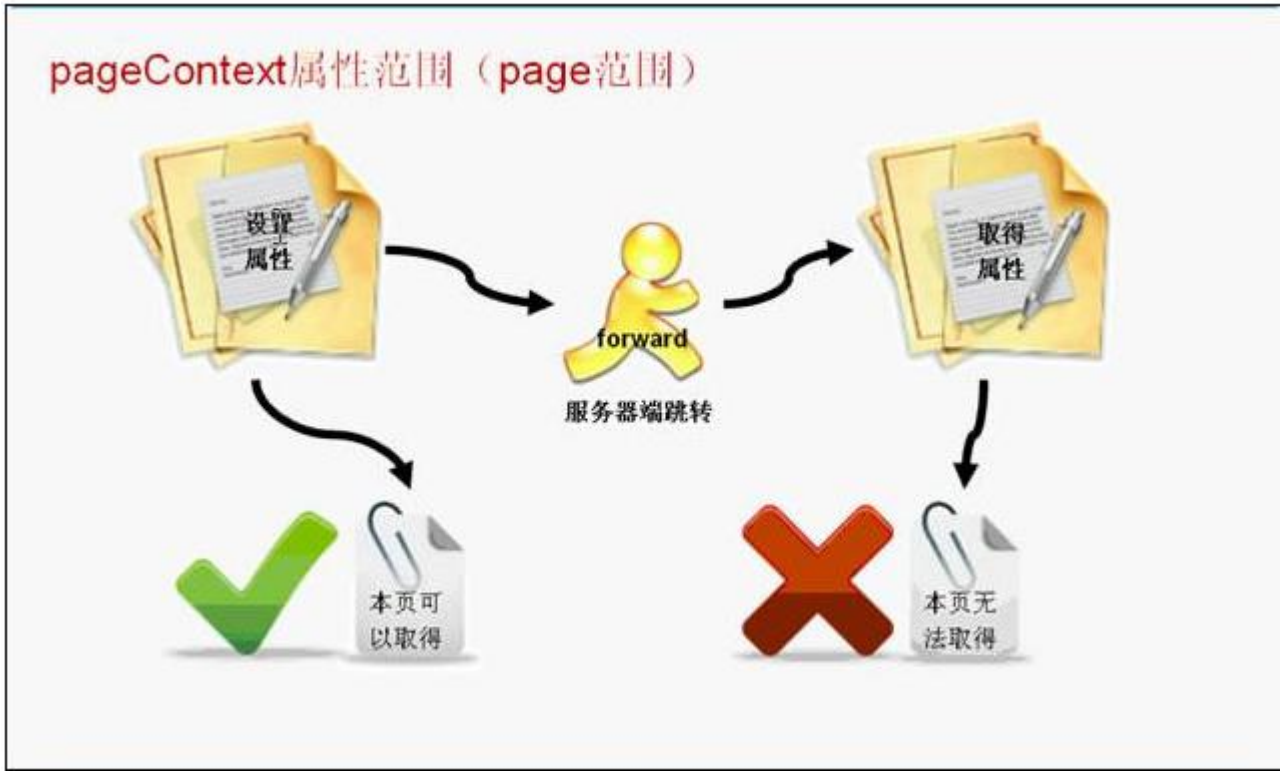
Java Web四大作用域概述

- 什么是作用域？
  - 所谓"作用域"就是"信息共享的范围"，也就是说一个信息能够在多大的范围内有效。
- JSP Web应用作用域表：
  - Web交互的最基本单位为HTTP请求。每个用户从进入网站到离开网站这段过程称为一个HTTP会话，一个服务器的运行过程中会有多个用户访问，就是多个HTTP会话。

Java Web四大作用域表		
名称	作用域	解释说明
applicationScope	在WEB应用程序中有效	服务器启动到停止这段时间
sessionScope	在当前会话中有效	客户端浏览器开启HTTP会话，到关闭浏览器这段时间
requestScope	在当前请求中有效	HTTP请求开始到结束这段时间
pageScope	在当前页面有效	当前页面从打开到关闭这段时间

page属性范围（pageContext）

- 在一个页面设置的属性，跳转到其他页面就无法访问了。



page作用域

pageContext对象

- 它相当于页面中所有功能的集大成者,本类名也叫pageContext。
  - pageContext对象提供了对JSP页面内所有的对象及名字空间的访问，也就是说他可以访问到本页所在的session，也可以取本页面所在的application的某一属性值。
- pageContext常用方法：
  - JspWriter getOut() 返回当前客户端响应被使用的JspWriter流(out) 。
  - Httpsession getsession() 返回当前页中的Httpsession对象(session) 。
  - Object getPage() 返回当前页的Object对象(page) 。
  - ServletRequest getRequest() 返回当前页的ServletRequest对象(request) 。
  - ServletResponse getResponse() 返回当前页的ServletResponse对象(response) 。
  - Exception getException() 返回当前页的Exception对象(exception) 。

page作用域

pageContext对象

- pageContext常用方法：
  - ServletConfig getServletConfig() 返回当前页的ServletConfig对象(config)。
  - ServletContext getServletContext() 返回当前页的ServletContext对象(application)。
  - void setAttribute(String name,Object attribute) 设置属性及属性值。
  - void setAttribute(String name,Object obj,int scope) 在指定范围内设置属性及属性值。
  - public Object getAttribute(String name) 取属性的值。
  - Object getAttribute(String name,int scope) 在指定范围内取属性的值。

page作用域

page作用域

- page作用范围仅限于用户请求的当前页面，就是指向当前Jsp页面本身，有点象类中的this指针。
- 对page对象的引用通常存储在pageContext对象中。
- 使用pageContext调用setAttribute()以及getAttribute()方法，可以直接设置以及获取page作用域信息。
- 对于page对象的引用将在响应返回给客户端之后被释放。
  - 存在于Java服务脚码中的变量应用示例：
    - 示例步骤概要：
      - ü 在Jsp页面Java服务程序中创建变量 int number=10。
      - ü 使用pageContext.setAttribute(String,Object)方法，将number值存入pageScope作用域。
      - ü 分别在本页以及其他页面使用pageContext.getAttribute(String)，进行验证。

page作用域

page作用域

- page作用范围仅限于用户请求的当前页面，对于page对象的引用将在响应返回给客户端之后被释放。
  - 使用pageContext实现page作用域存取变量的操作。
    - 示例详细步骤如下：
      - ✓ Eclipse中创建Dynamic Web Project项目PageScopeExplain。
      - ✓ 在PageScopeExplain下的WebContent目录下创建pageexample目录。
      - ✓ 在pageexample目录中创建pagescopetest.jsp文件。
      - ✓ 在pagescopetest.jsp 代码中，添加Java服务程序片，其中创建变量 int number=10。
      - ✓ 使用pageContext.setAttribute(“number”,number)方法，将number值存入pageScope作用域。
      - ✓ 分别在本页(pagescopetest.jsp)以及其他页面(another.jsp)使用pageContext.getAttribute(String)，验证page作用域的特点。

测试page范围

- test1.jsp
- <%
- pageContext.setAttribute("name","yxkong");
- out.println("test1.jsp: ");
- out.println(pageContext.getAttribute("name"));
- out.println("<p>");
- pageContext.include("test2.jsp");
- %>

- test2.jsp
- <%
- out.println("test2.jsp: ");
- out.println(pageContext.getAttribute("name"));
- %>

访问test1.jsp，将看到如下的输出：

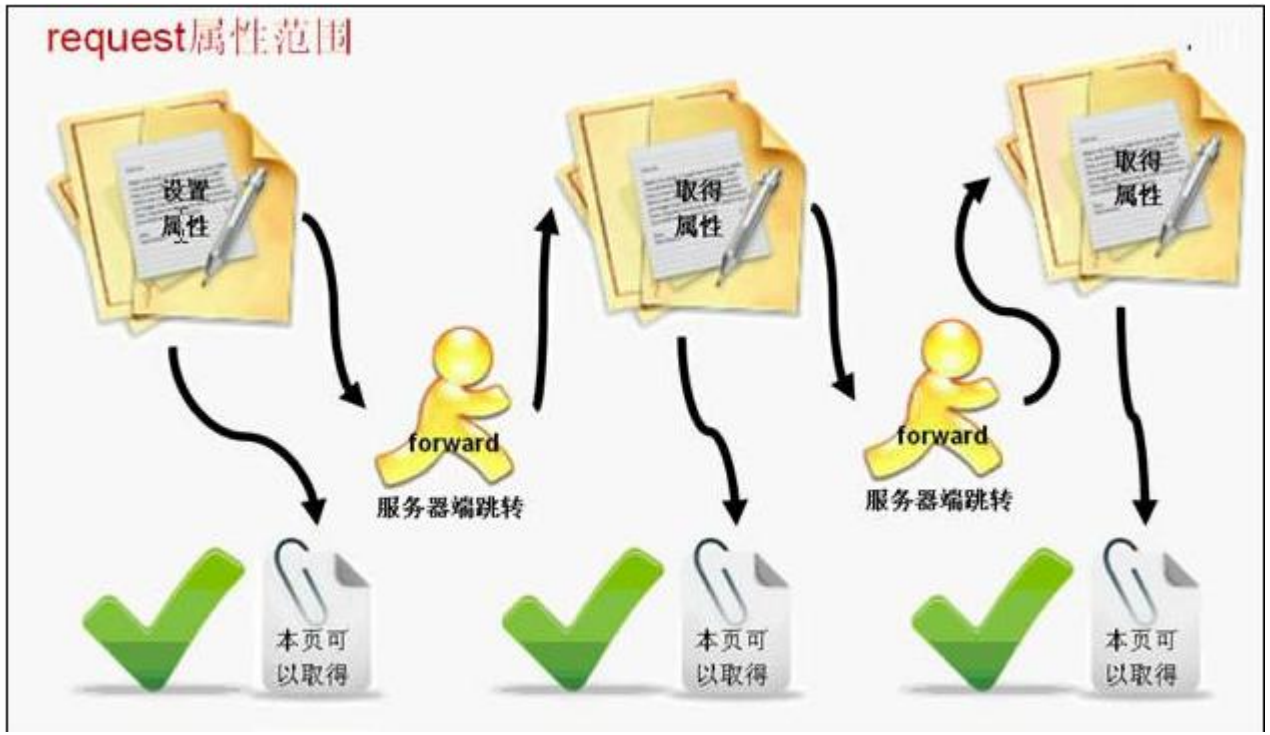
test1.jsp: yxkong

test2.jsp: null

说明保存在pageContext对象中的属性具有page范围，只能在同一个页面中被访问。

request属性范围

- request属性范围表示在一次服务器跳转中有效，只要是服务器跳转，则设置的request属性可以一直传递下去。



request作用域

request作用域

- 客户端的请求信息被封装在request对象中，通过它才能了解到客户的需求，然后做出响应。它是HttpServletRequest类的实例，Servlet之间的信息共享是通过HttpServletRequest接口的两个方法来实现的。
  - void setAttribute（String name, Object value）：将对象value以name为名称保存到request作用域中。
  - Object getAttribute（String name）：从request作用域中取得指定名字的信息。
  - String getParameter(String name)：返回name指定参数的参数值。
- Servlet中的doGet()、doPost()方法的第一个参数就是HttpServletRequest对象，使用这个对象可以获得请求信息，也可传递信息。可以通过RequestDispatcher接口的forward()方法，将请求转发给其他Servlet。
- 在新的目标组件中，可以获取传递来的请求。但这些信息在请求结束后就无效了，即使重定向，也是浏览器重新发生一次请求，不会附带上次请求的内容。

request作用域

request作用域

- 客户端的请求信息被封装在request对象中，但是这些信息在请求结束后就无效了，所以要体会request作用域，需要servlet的参与，也可以通过多个servlet实现request作用域属性值得传递。
- 使用pageContext实现request作用域存取变量的操作。
  - 示例详细步骤如下：
    - Eclipse中创建Dynamic Web Project项目RequestScopeExplain。
    - 在RequestScopeExplain下的WebContent目录下创建requestexample目录，其中创建requestscopetest.jsp文件。
    - 在Java Resources下src目录中创建com.oracle.teach.servlet包，其中创建servlet的子类RequestScopeServlet类。
    - 重写doGet方法，调用request.setAttribute(“requestInfor”, 100) 添加request作用域属性值。
    - request.getRequestDispatcher(“requestexample/requestscopetest.jsp”)方法获得RequestDispatcher转发器对象，通过该对象调用forward(request,response)。
    - 在requestscopetest.jsp中，使用pageContext.getRequest().getAttribute(“requestInfor”)方法，获得requestScope作用域的属性值。
    - 通过浏览器直接访问RequestScopeExplain.java。

```
<%@page contentType="text/html;charset=UTF-8"%>
<%@page import="java.util.*"%>
<%  request.setAttribute("name","孤傲苍狼");
    request.setAttribute("date",new Date()); %>
<%--使用jsp:forward标签进行服务器端跳转--%>
<jsp:forward page="/requestScopeDemo02.jsp" />
```

requestScopeDemo02.jsp

```
<%@page contentType="text/html;charset=UTF-8"%>
<%@page import="java.util.*"%>
<%
   //取得requestScopdemo01.jsp设置的属性
   String refName = (String)request.getAttribute("name");
   Date refDate = (Date)request.getAttribute("date");
%>
<h1>姓名：<%=refName%></h1>
<h1>日期：<%=refDate%></h1>
```

测试request范围

- test1.jsp
- <%
- request.setAttribute("name","yxkong");
- out.println("test1.jsp: ");
- out.println(request.getAttribute("name"));
- out.println("<p>");
- pageContext.include("test2.jsp");
- %>

- test2.jsp
- <%
- out.println("test2.jsp: ");
- out.println(request.getAttribute("name"));
- %>

访问test1.jsp，将看到如下的输出：
test1.jsp: yxkong
test2.jsp: yxkong
说明保存在request对象中的属性具有request范围，在请求对象存活期间，可以访问这个范围内的对象。
将pageContext.include("test2.jsp"); 这一句注释起来，先访问test1.jsp，再访test2.jsp，可以看到如下输出：
test2.jsp: null
这是因为客户端开始了一个新的请求。

session属性范围

- session设置的属性不管如何跳转，都可以取得的。当然，session只针对一个用户.

在第一个页面上设置的属性，跳转(服务器跳转/客户端跳转)到其他页面之后，其他的页面依然可以取得第一个页面上设置的属性。

session作用域

session作用域应用

- Session作用域指的是客户端与服务器的一次会话，从客户连到服务器的一个WebApplication开始，直到客户端与服务器断开连接为止。Session对象是Httpsession类的实例。
- 一浏览器对服务器进行多次访问，在这多次访问之间传递信息，就是session作用域的体现，获得session的实例，通过pageContext.getsession。会话对象session常用方法如下：
  - Object Httpsession.getAttribute（String name）：从session中获取信息。
  - void Httpsession.setAttribute（String name, Object value）：向session中保存信息。
  - Httpsession HttpServletRequest.getSession()：获取当前请求所在的session的对象。
- session的开始时刻比较容易判断，但结束时刻就不好判断了，因为浏览器关闭时并不会通知服务器，所以只能通过如下这种方法判断。
  - 通过setMaxInactiveInterval(int second)方法来设置活动时间。
  - 如果想主动让会话结束，例如用户单击“注销”按钮的时候，可以使用Httpsession 的 invalidate()方法，用于强制结束当前session。

- <%@page contentType="text/html;charset=UTF-8"%>
- <%@page import="java.util.*"%>
- <% session.setAttribute("name","孤傲苍狼");
  session.setAttribute("date",new Date());%>
- <%--使用服务器端跳转--%>
- <jsp:forward page="/sessionScopeDemo02.jsp"/>

sessionScopeDemo02.jsp

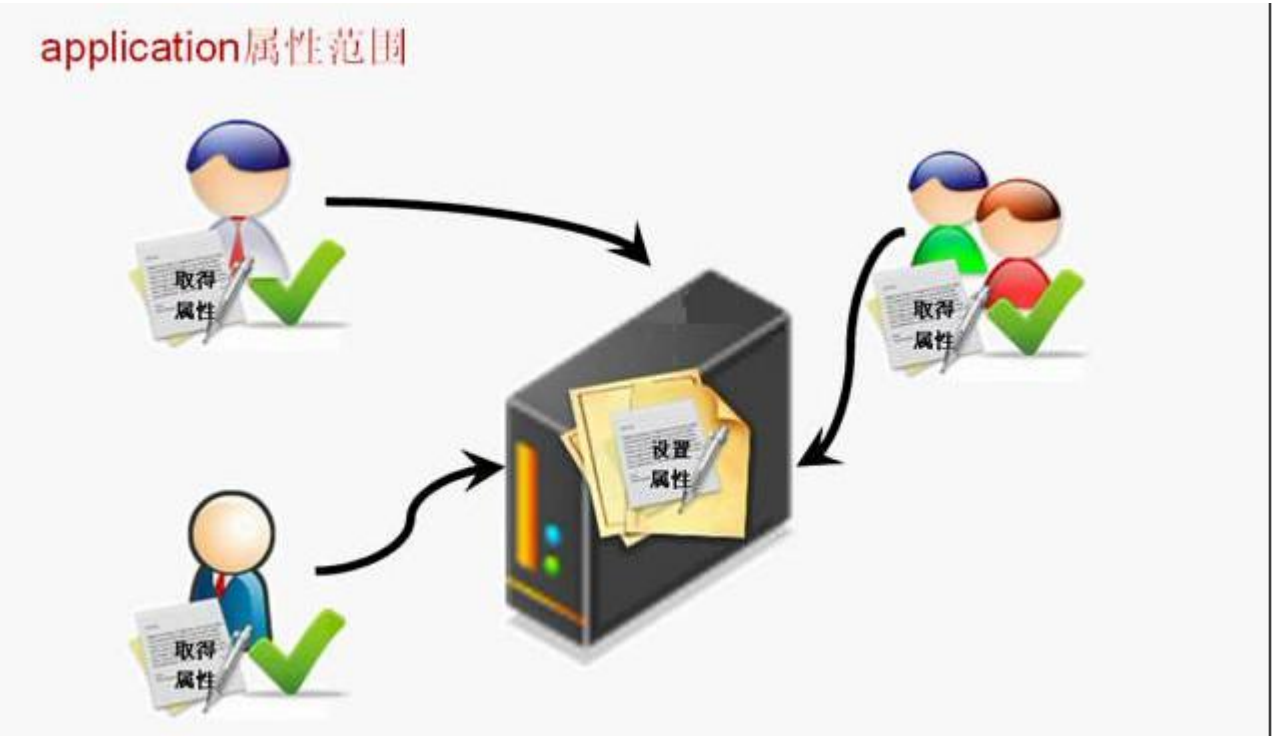
- <%@page contentType="text/html;charset=UTF-8"%>
- <%@page import="java.util.*"%>
- <%String refName = (String)session.getAttribute("name");
- Date refDate = (Date)session.getAttribute("date");%>
- <h1>姓名：<%=refName%></h1>
- <h1>日期：<%=refDate%></h1>
- <%--使用超链接这种客户端跳转--%>
- <h1><a
  href="${pageContext.request.contextPath}/sessionScopeDemo03.jsp">sessionScopeDemo
  03</h1>

测试session范围

- test1.jsp
- <%
- session.setAttribute("name","yxkong");
- %>
- test2.jsp
- <%
- out.println("test2.jsp: ");
- out.println(session.getAttribute("name"));
- %>

先访问test1.jsp，然后在同一个浏览器窗口中访问test2.jsp，可以看到如下输出：
test2.jsp: yxkong
说明保存在session对象中的属性具有session范围，在会话期间，可以访问这个范围内的对象。
如果我们在访问完test1.jsp后，关闭浏览器，重新打开浏览器窗口，访问test2.jsp，将看到如下输出：
test2.jsp: null
这是因为客户端与服务器开始了一次新的会话。

application属性范围



因为application属性范围是在服务器上设置的一个属性，所以一旦设置之后任何用户都可以浏览到此属性

application作用域

application作用域

- application作用域实现了用户间数据的共享，可存放全局变量。它开始于服务器的启动，直到服务器的关闭，在此期间，此对象将一直存在。
- 在用户的前后连接或不同用户之间的连接中，可以对此对象的同一属性进行操作，在任何地方对此对象属性的操作，都将影响到其他用户对此的访问。
- 服务器的启动和关闭决定了application对象的生命。它是ServletContext类的实例。常用方法如下：
  - 获通过pageContext.getServletContext()，得application的实例。
  - Object getAttribute（String name）：从application中获取信息。
  - void setAttribute（String name, Object value）：向application作用域中设置信息。

application作用域

application作用域

- application作用域实现了用户间数据的共享，可存放全局变量。它开始于服务器的启动，直到服务器的关闭。
- 使用pageContext调用servletContext()，而对application作用域存取变量的操作。
  - 示例详细步骤如下：
    - Eclipse中创建Dynamic Web Project项目ApplicationScopeExplain。
    - 在ApplicationScopeExplain下的WebContent目录下创建applicationexample目录，其中创建applicationscopetest1.jsp、applicatioscopetest2.jsp文件。
    - 在applicatioscopetest1.jsp页面Java程序片中，通过pageContext的servletContext()方法获得application对象，由application对象来调用setAttribute(“applicationInfor”,”application服务！”)方法，添加application作用域属性值。
    - 在applicatioscopetest2.jsp中的服务器程序中通过application对象调用getAttribute(“applicationInfor”)，获得applicationInfor的值并显示。来感受application作用域。

applicationScopeDemo01.jsp

```
<%@ page contentType="text/html;charset=GBK"%>
<%@ page import="java.util.*"%>
<%
   //此时设置的属性任何用户都可以取得
   application.setAttribute("name","孤傲苍狼");  //设置属性
   application.setAttribute("date",new Date());
%>
<h1><a href="${pageContext.request.contextPath}/applicationScopeDemo02.jsp">applicationScopeDemo02</a></h1>
```

applicationScopeDemo02.jsp

```
<%@ page contentType="text/html;charset=GBK"%>
<%@ page import="java.util.*"%>
<%
    String refName =
(String)application.getAttribute("name");
    Date refDate = (Date)application.getAttribute("date");
%>
<h1>姓名：<%=refName%></h1>
<h1>日期：<%=refDate%></h1>
```

测试application范围

- test1.jsp
- <%
- application.setAttribute("name","yxkong");
- %>
- test2.jsp
- <%
- out.println("test2.jsp: ");
- out.println(application.getAttribute("name"));
- %>

- 先访问test1.jsp，然后关闭浏览器，再打开浏览器窗口，访问test2.jsp，可以看到如下输出：

- test2.jsp: yxkong

- 说明保存在application对象中的属性具有application范围，在Web应用程序运行期间，都可以访问这个范围内的对象。

jsp四种属性范围的使用场合

request：如果客户向服务器发请求，产生的数据，用户看完就没用了，像这样的数据就存在**request**域,像新闻数据，属于用户看完就没用的。

session：如果客户向服务器发请求，产生的数据，用户用完了等一会儿还有用，像这样的数据就存在**session**域中，像购物数据，用户需要看到自己购物信息，并且等一会儿，还要用这个购物数据结帐。

application(servletContext)：如果客户向服务器发请求，产生的数据，用户用完了，还要给其它用户用，像这样的数据就存在**application(servletContext)**域中，像聊天数据。

- JSP技术的设计者为 便于开发人员在编写JSP页面时获得这些web对象的引用，特意定义了9个相应的变量，开发人员在JSP页面中通过这些变量就可以快速获得这9大对象的引 用。

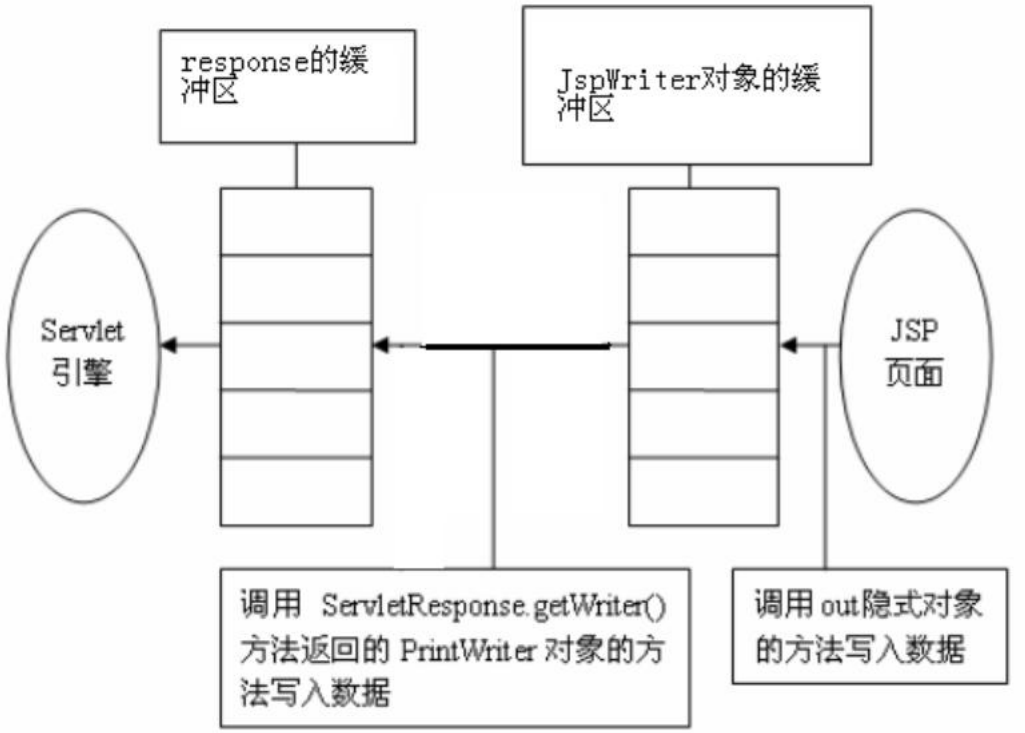
Jsp隐式对象

- **Jsp隐式对象概述**
  - 由JSP规范提供,不用编写者实例化。
  - 通过Web容器实现和管理。
  - 所有JSP页面均可使用。
  - 只有在脚本元素的表达式或代码段中才可使用(<%=使用内置对象%>或<%使用内置对象%>)。
- **常用内置对象**
  - 输入输出对象:request对象、response对象、out对象。
  - 通信控制对象:pageContext对象、session对象、application对象。
  - Servlet对象:page对象、config对象。
  - 错误处理对象:exception对象。

Jsp隐式对象out

- out对象用于向客户端发送文本数据。

out对象的工作原理图



Jsp隐式对象out

- **Jsp隐式对象out的特点**
  - out对象是输出流对象。
  - Out对象所属类型是(数据流 javax.servlet.jsp.jspWriter)。
- **out对象的常用方法**
  - void print(value)向客户端输出各种数据。
  - void clear() 清除缓冲区的内容 。
  - void clearBuffer() 清除缓冲区的当前内容。
  - void flush() 清空流 。
  - int getBufferSize() 返回缓冲区以字节数的大小，如不设缓冲区则为0。
  - int getRemaining() 返回缓冲区还剩余多少可用 。
  - boolean isAutoFlush() 返回缓冲区满时，是自动清空还是抛出异常。
  - void close() 关闭输出流 。

Jsp隐式对象out

- Jsp隐式对象out应用

```
<%@ page buffer=”1kb” autoFlush=”true”
contentType=”text/html;charSet=GBK” %>
<html>
<body>
<% for(int i=0;i<135;i++) //迭代输出
out.println(“Hello world, “+i+” “);
%>
<br>BufferSize: <%=out.getBufferSize() %>
<br>BufferRemain: <%=out.getRemaining() %>
<br>AutoFlush: <%=out.isAutoFlush() %>
<% out.clearBuffer(); %>
</body>
</html>
```

Jsp隐式对象request

- **Jsp隐式对象request的特点**
  - 客户端的请求信息被封装在request对象中，通过它才能了解到客户的需求，然后做出响应。
  - 它是(请求信息 javax.servlet.http.HttpServletrequest)类的实例。
- **request对象的常用方法**
  - object getAttribute(String name) 返回指定属性的属性值 。
  - Enumeration getAttributeNames() 返回所有可用属性名的枚举 。
  - String getCharacterEncoding() 返回字符编码方式 。
  - int getContentLength() 返回请求体的长度（以字节数） 。
  - String getContentType() 得到请求体的MIME类型 。
  - ServletInputStream getInputStream() 得到请求体中一行的二进制流 。
  - String getParameter(String name) 返回name指定参数的参数值 。
  - Enumeration getParameterNames() 返回可用参数名的枚举。

Jsp隐式对象request

Jsp隐式对象request应用举例

```
<%@ page contentType="text/html;charset=GBK" %>
<html>
<head><title> request内置对象的实例 </title></head>
<body>
<form aciton="request.jsp">
<br>Get request results:
<br><input type="text" name="myname">
<br><input type="submit" name="get value">
</form>
返回HTTP请求信息中使用的方法名称:<%=request.getMethod()%>
<br>
返回请求信息中调用Servlet的URL部分:<%=request.getServletPath()%>
<br>
返回HTTP GET请求信息中URL之后的查询字符串:<%=request.getQueryString()%>
<br>
返回请求实体的MIME类型:<%=request.getContentType()%>
<br>
返回请求信息中的协议名名字和版本号:<%=request.getProtocol()%>
<br>
有关任何路径信息:<%=request.getPathInfo()%>
<br>
返回接受请求的服务器主机:<%=request.getServerName()%>
<br>
返回服务器的端口号:<%=request.getServerPort()%>
<br>
返回提交请求的客户机的规范名字:<%=request.getRemoteHost()%>
<br>
返回提交请求的客户机的IP地址:<%=request.getRemoteAddr()%>
<br>
返回请求中使用的模式（协议）名字:<%=request.getScheme()%>
<br>
返回这个request值，提交过来的值:<%=request.getParameter("myname")%>
</body>
</html>
```

Jsp隐式对象response

- **Jsp隐式对象response的应用**

```
<%@ page language="java" contentType="text/html;charSet=GBK" %>
<html>
<body>
<center><h3>response.sendRedirect()使用例子</h3></center>
<form action="index4.jsp">
<table border=1><tr><td>
<select name="pg">
<option value=0>本页</option>
<option value=1>hello页面</option>
<option value=2>goodbye页面</option>
</select>
</td></tr>
<tr><td><input type="submit" value="提交"></td></tr></table>
</form>
<%
String pg = request.getParameter("pg"); //获取传递参数pg
if("1".equals(pg)) //如果pg等于1
response.sendRedirect("hello.jsp"); //则页面重定向为hello.jsp
else if("2".equals(pg)) //如果pg等于2
response.sendRedirect("goodbye.jsp"); //则页面重定向为goodbye.jsp
else //否则不进行页面重定向，即还显示本页
out.println("没有进行页面重定向");
%>
</body>
</html>
```

Jsp隐式对象session

- **Jsp隐式对象session的特点**
  - session对象指的是客户端与服务器的一次会话，从客户连到服务器的一个WebApplication开始，直到客户端与服务器断开连接为止。
  - 它是(会话javax.servlet.http.Httpsession)类的实例。
- **session对象的常用方法**
  - long getCreationTime() 返回session创建时间。
  - public String getId() 返回session创建时JSP引擎为它设的惟一ID号。
  - long getLastAccessedTime() 返回此session里客户端最近一次请求时间。
  - int getMaxInactiveInterval() 返回两次请求间隔多长时间此session被取消(ms)。
  - String[] getValueNames() 返回一个包含此session中所有可用属性的数组。

Jsp隐式对象session

- **session对象的常用方法**
  - void invalidate() 取消session，使session不可用。
  - boolean isNew() 返回服务器创建的一个session,客户端是否已经加入。
  - void removeValue(String name) 删除session中指定的属性。
  - void setMaxInactiveInterval() 设置两次请求间隔多长时间此session被取消(ms)。
  - Object HttpSession.getAttribute（String name）：从session中获取信息。
  - void HttpSession.setAttribute（String name, Object value）：向session中保存信息。

Jsp隐式对象session

Jsp隐式对象session的应用

- session对象指的是客户端与服务器的一次会话。
- 从客户连到WebApplication开始，直到客户端与服务器断开连接为止，它是(会话javax.servlet.http.Httpsession)类的实例。
- session对象应用实例具体步骤：
  - 通过session调用isNew()方法，实现判断是否是同一个会话。
    - 创建Web项目SessionExplain，WebContent目录下创建sessioneg目录，其中创建index.jsp模拟登录页，创建manage.jsp模拟后台页面。
    - 通过浏览器直接访问manage.jsp页面，如果没有经过index.jsp登录页，在manage.jsp页面服务程序中，实现重定向到index.jsp页面。
  - 通过session调用setAttribute(key,obj)方法，实现保存会话信息。
    - 向index.jsp页面中添加业务，在session.jsp页面服务程序中，添加session.setAttribute(“sessionId”,session.getId())语句，将当前会话的id号保存到session的属性sessionId。
    - 通过manage.jsp页面读取当前会话中的属性sessionId对应的值。

Jsp隐式对象application

- **Jsp隐式对象application的特点**
  - application对象实现了用户间数据的共享，可存放全局变量。它开始于服务器的启动，直到服务器的关闭，在此期间，此对象将一直存在。
  - 这样在用户的前后连接或不同用户之间的连接中，可以对此对象的同一属性进行操作,服务器的启动和关闭决定了application对象的生命。
  - 它是(应用服务javax.servlet.http.ServletContext)类的实例。
- **application对象的常用方法**
  - Object getAttribute(String name) 返回给定名的属性值 。
  - Enumeration getAttributeNames() 返回所有可用属性名的枚举 。
  - void setAttribute(String name,Object obj) 设定属性的属性值 。
  - void removeAttribute(String name) 删除一属性及其属性值 。
  - String getServerInfo() 返回JSP(SERVLET)引擎名及版本号。
  - String getRealPath(String path) 返回一虚拟路径的真实路径 。
  - ServletContext getContext(String uripath) 返回指定WebApplication的application对象 。

Jsp隐式对象application

Jsp隐式对象application的应用

- application对象实现了用户间数据的共享，可存放全局变量。它开始于服务器的启动，直到服务器的关闭,服务器的启动和关闭决定了application对象的生命，它是javax.servlet.http.ServletContext类的实例。
- application对象应用实例具体步骤：
  - 创建Web项目ApplicationExplain，WebContent目录下创建applicationeg目录，目录中创建index.jsp页面。
  - 在index.jsp页面中创建一个站点访问人数成员变量count。然后，业务语句中避免同一个会话重复刷新count数据。

Jsp隐式对象Exception

Jsp隐式对象exception的特点

- exception对象是一个例外对象，当一个页面在运行过程中发生了例外，就产生这个对象。如果一个JSP页面要应用此对象，就必须把isErrorPage设为true，否则无法编译。
- 它实际上是java.lang.Throwable的对象。
- exception对象的常用方法：
  - String getMessage() 返回描述异常的消息 。
  - String toString() 返回关于异常的简短描述消息 。
  - void printStackTrace() 显示异常及其栈轨迹 。
  - Throwable FillInStackTrace() 重写异常的执行栈轨迹。

Jsp隐式对象Exception

Jsp隐式对象exception的应用

- 页面在运行过程中发生了例外，就产生这个exception对象。在发生异常页的page属性中，添加errorPage=“异常处理页面“，在负责处理异常的页面page指令中，添加isErrorPage=”*true*”。
- exception对象应用步骤：
  - 在异常处理页负责异常处理。
    - 创建Web项目ExceptionExplain，在WebContent目录下创建exceptioneg目录。
    - 在exceptioneg目录中创建exceptionsrc.jsp页面，页面服务程序代码中添加业务语句，设计一个除数为零的异常。
    - exceptioneg目录中，创建负责异常处理的页面exceptionserver.jsp，在该页面page指令属性中isErrorPage=”*true*”，通过内置异常对象在页面中显示异常信息。

课堂练习

课堂练习4：简单的登录页面

- 任务描述
  - 在网页login.jsp中输入账号密码，提交至login_auth.jsp。
  - 在login_auth.jsp中处理登录数据。
  - 如果输入的用户名为csg，密码为123，则登录成功。
  - 其他情况登录失败，并跳转到login.jsp。
  - login.jsp与login_auth.jsp都在WebContent目录下。
- 关联知识点
  - 掌握Java Web中四大作用域
  - 掌握Jsp中隐式对象的使用
- 实现步骤
  - 创建login.jsp和login_auth.jsp
  - 编写代码实现跳转和验证

课堂练习

页面运行效果

登录画面

登录成功

登录失败

```
<%@ 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=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<form name="loginForm" method="POST" action="login_auth.jsp">
<table>
<tr>
<td>账号:</td>
<td><input type="text" name="loginName" value=""></td>
</tr>
<tr>
<td>密码:</td>
<td><input type="password" name="loginPasswd" value=""></td>
</tr>
<tr>
<td><input type="submit" name="commit" value="提交"></td>
</tr>
</table>
</form>
</body>
</html>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.o
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <%!
        String name = null;
        String password = null;
        boolean firstLogin = false;
    %>
    <%
        //获取request请求中的name
        name = request.getParameter("name");
        //获取request请求中的stu_name
        password = request.getParameter("password");
        //匹配用户信息
        if(name.equals("csg")){
            //匹配密码
            if(password.equals("123")){
                out.println("<h1>欢迎登陆用户: "+n
            }
            else{
                //密码输入错误，重新登陆
                out.println("<h1>密码错误</h1>");
                out.println("<h1>3秒后跳转到登陆页
```

使用session对象的简易购物车

shop.jsp：商品选择页面

```
<body>
        <form id="form1" name="form1" method="post" action="shop_do.jsp">
                <p><strong>请选择你要购买的商品</strong></p>
                <table width="300" border="1">
                        <tr>
                                <td>商品名：</td>
                                <td><input type="text" name="goods"></td>
                        </tr>
                        <tr>
                                <td colspan="2">
                                        <div align="center">
                                                <input type="submit"
name="Submit" value="加入购物车">
                                                <input type="reset"
name="Submit2" value="重选">
                                        </div>
                                </td>
                        </tr>
                </table>
        </form>
</body>
```

该页面有两个功能，一个是获得客户提交的商品数据并加入到购物车，另一个是实现购物流程的控制（继续购物或结账）

shop_do.jsp：

```
<body>
        <%
        String goodsName = request.getParameter("goods");//获取商品名称
        if(!goodsName.equals("")){
        goodsName = new String(goodsName.getBytes("ISO-8859-1"),"UTF-8");  //解决中文乱码问题
                ArrayList list = null;  //定义保存商品的动态数组
                list = (ArrayList)session.getAttribute("list");  //通过list属性取得购物车
                if(list==null){
                        list = new ArrayList();
                        list.add(goodsName);
                        session.setAttribute("list", list);
                }else{
                        list.add(goodsName);
                }
        %>
        <%
        }else{
                response.sendRedirect("shop.jsp");
        }
        %>
        <center>
```

pay.jsp：该页面完成结账处理。主要功能是把购物车里面所有商品名称显示给客户

```
<body>
        非常感谢您的光临！您本次在我们这里购买了一下商品：<br>
        <%
        ArrayList list = (ArrayList)session.getAttribute("list");
        for(int i=0;i<list.size();i++){
                String goodsName = (String)list.get(i);
        %>
        商品：<%=i %>:<%=goodsName %><br>
        <%} %>
```

第6节 JSP Model2标准

⑩ 知识点预览

#	知识点	重点	难点	应用	说明
1	JSP的两种模式		√		理解Model1和Model2设计模式的特点
2	MVC模式概述	√	√		理解并掌握MVC模式
3	设计三层结构	√	√	√	理解并掌握MVC模式在JSP技术的应用

JSP的两种模式

设计模式

- 什么是设计模式？
  - 设计模式是一套被反复使用、成功设计经验的总结。是某一类问题特殊再现的解决方案，该方案提供了一个经过充分验证的通用方案。
  - 使编程活动既有对具体问题的针对性，又有对将来问题和需求的通用性。
- 设计模式的力量源泉。
  - 源自用户需要保持交互操作界面的稳定性，又需要根据需求的变化调整显示内容和形式，是数据的显示和处理松耦合，便于系统的维护和扩展。
- 设计模式的优点。
  - 提高了代码的重性，便于扩展，易操作以维护，便于开发。

JSP的两种模式

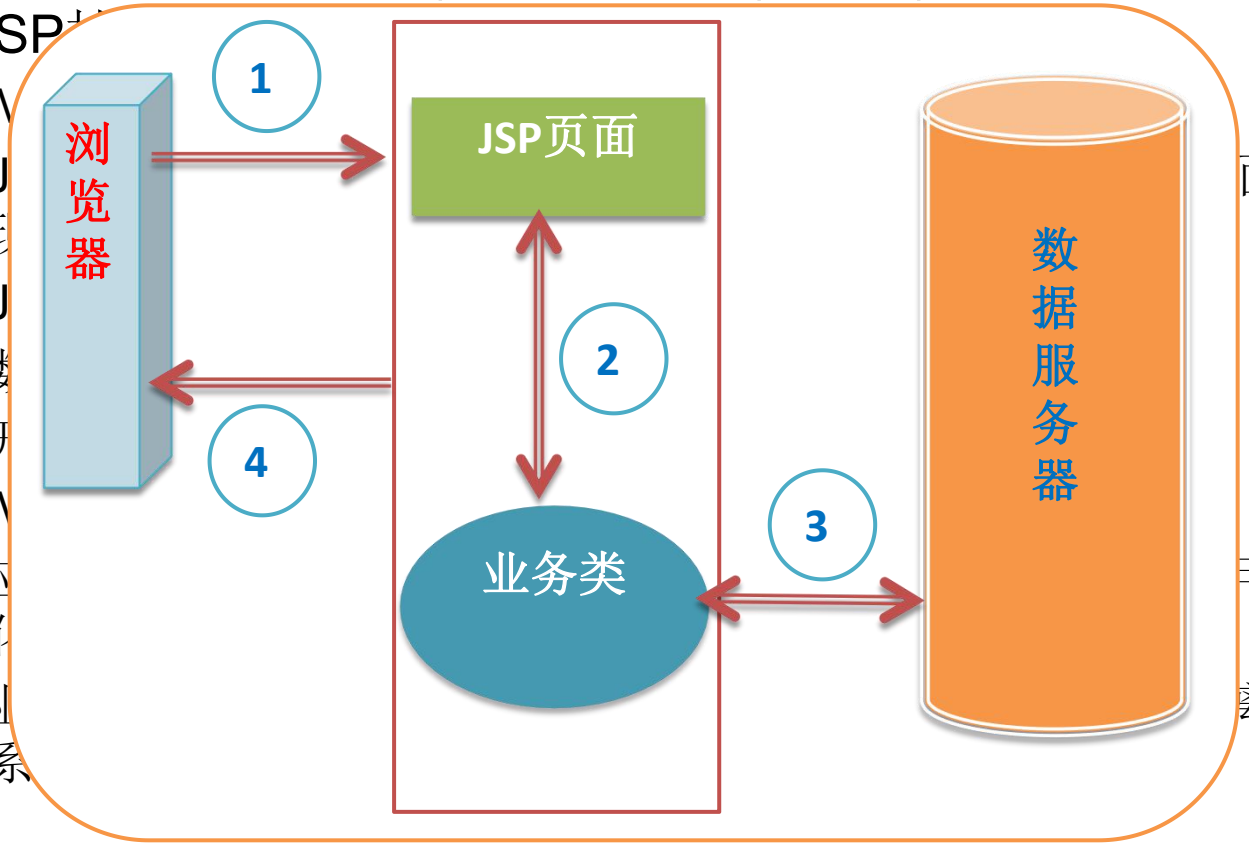
JSP的两种模式

- Sun公司的JSP规范提出了(JSP Model1(模式1)、 JSP Model2(模式2))两种用JSP
  - JSP M
    - J
    - J
    - 数
    - 开
  - JSP M
    - 应
    - 业

面中同时实

用户需求变

离，不利于



在jsp+javabean架构中，JSP负责控制逻辑、表现逻辑、业务对象（javabean）的调用。

JSP+JavaBean模式适合开发业务逻辑不太复杂的web应用程序，这种模式下，JavaBean用于封装业务数据，JSP即负责处理用户请求，又显示数据。

```java
public class CalculatorBean {

    //用户输入的第一个数
    private double firstNum;
    //用户输入的第二个数
    private double secondNum;
    //用户选择的操作运算符
    private char operator = '+';
    //运算结果
    private double result;

    public double getFirstNum() {
        return firstNum;
    }

    public void setFirstNum(double firs
        this.firstNum = firstNum;
    }

    public double getSecondNum() {
        return secondNum;
    }

    public void setSecondNum(double sec
        this.secondNum = secondNum;
```

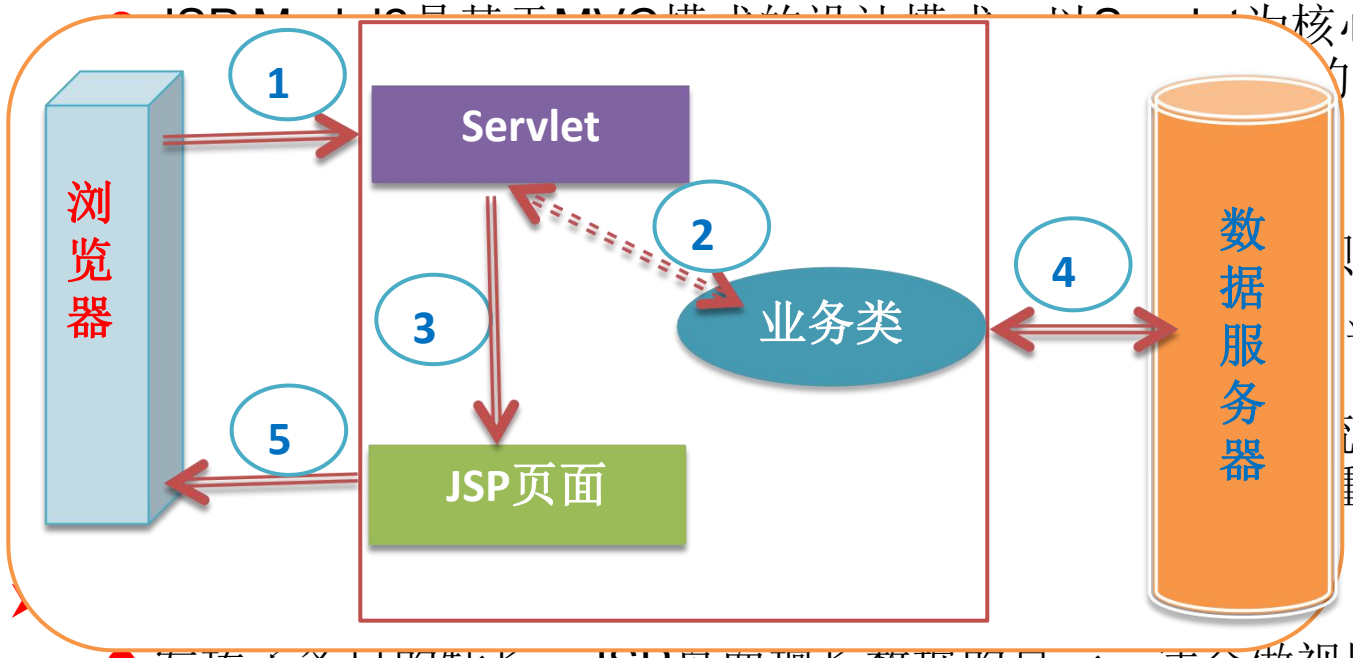
按 Ctrl+C 复制代码

```jsp
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%--使用me.gacl.domain.CalculatorBean --%>
<jsp:useBean id="calcBean" class="me.gacl.domain.CalculatorBean"/>
<%--接收用户输入的参数 --%>
<jsp:setProperty name="calcBean" property="*"/>
<%
//使用CalculatorBean进行计算
 calcBean.calculate();
%>
<!DOCTYPE HTML>
<html>
  <head>
    <title>使用【jsp+javabean开发模式】开发的简单计算器</title>
  </head>

  <body>
     <br/>
    计算结果是：
    <jsp:getProperty name="calcBean" property="firstNum"/>
    <jsp:getProperty name="calcBean" property="operator"/>
    <jsp:getProperty name="calcBean" property="secondNum"/>
      =
    <jsp:getProperty name="calcBean" property="result"/>

     <br/><hr> <br/>
    <form action="${pageContext.request.contextPath}/calculator.jsp" method="post">
        <table border="1px">
            <tr>
                <td colspan="2">简单的计算器</td>
            </tr>
            <tr>
                <td>第一个参数</td>
                <td><input type="text" name="firstNum"></td>
```

JSP的两种模式

Sun公司的JSP的两种模式

- JSP规范提出了(JSP Model1(模式1)、 JSP Model2(模式2))两种用JSP技术建立应用程序的方式。
  - JSP Model2(模式2)：



核心。通过这种
的组件实现，这
则和数据操作。
显示和获取用
流程控制，完成
重定向的方式

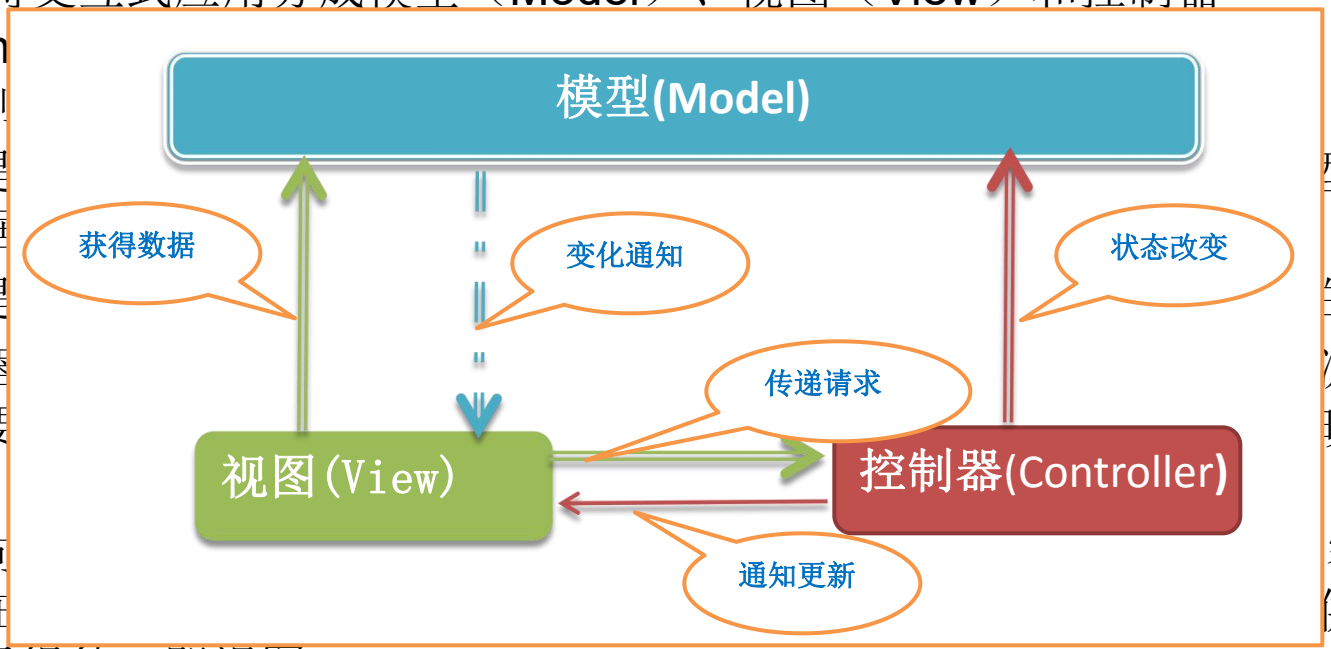
    - 发挥了各自的特长，JSP页面擅长数据的显示，适合做视图Servlet擅长数据处理。
    - 清晰地分离了视图层和业务层，使视图、控制、业务之间有较低的耦合性，利于维护扩展，利于开发。

MVC模式概述

MVC模式概述

什么是MVC模式？

- MVC将交互式应用分成模型（Model）、视图（View）和控制器（Con
显示和控制
- 模型是
型封装了数据
- 视图是
的状态。
- 控制器
流程。它主要
映到视图。
- 从面向
容易维护。在
供合适的显示组件，即视图。
- MVC模式比较适合开发大型应用，便于扩展，也便于维护，同时比较适合团队开发，是开发工程化。



设计三层结构

MVC模式的应用

- 创建视图：
  - 视图的职责？
    - 主要负责呈现结果以及获得数据。
  - 视图的表现形式？
    - JSP页面或者HTML页面。
  - 视图获取客户请求信息的方式？
    - 通过表单控件。
    - 通过超链接传递参数。
    - 通过请求对象。
  - 视图传递的目标组件？
    - 对应的Servlet类对象。

设计三层结构

MVC模式的应用

- 创建Model模型：
  - Model的职责？
    - 主要负责业务处理，数据操作。
  - Model的表现形式？
    - 实体类业务类以及数据访问类。
  - Model的“工作”方式？
    - Model对象调用相关业务方法。
    - 封装数据。
  - Model的业务对象？
    - 被控制器调用。

- 在平时的JavaWeb项目开发中，在不使用第三方mvc开发框架的情况下，通常会选择Servlet+JSP+JavaBean开发模式来开 发JavaWeb项目，Servlet+JSP+JavaBean组合开发就是一种MVC开发模式了，控制器(Controller)采用 Servlet、模型(Model)采用JavaBean、视图(View)采用JSP。在讲解Servlet+JSP+JavaBean开发模式之前，先简单了解一下MVC开发模式

设计三层结构

MVC模式的应用

- 创建Servlet：
  - Servlet的职责？
    - 主要负责业务调度，接收视图的请求，调度Model完成业务，通知视图更新显示。
  - Servlet的表现形式？
    - Servlet类的子类。
  - Servlet获取客户请求信息的方式？
    - 通过请求对象。
  - Servlet传递信息的目标与方式？
    - 通过重定向通知视图更新，同时传递Model数据模型。

课堂练习

课堂练习5：MVC设计模式实现简单的登录

- 任务描述
  - 使用MVC模式，编写一个网站，可实现简单的登录功能。
  - 用户的信息需保存在mysql的数据库中
- 关联知识点
  - 掌握MVC设计模式
  - 复习使用JDBC操作数据库
- 实现步骤
  - 根据脚本创建数据库表
  - 根据MVC模式分层
  - 编写代码

课堂练习

页面运行效果

登录界面

登录成功

登录失败

JDBC复习