一、servlet

1、servlet api概述

servlet有4个包

javax.servlet包含了servlet和servlet容器之间的类和接口

主要成员：接口servlet、servletRequest、servletResponse、servletContext、servletConfig、Fileter、RequestDispatchar，类GenericServlet

javax.servlet.http包含了httpservlet和servlet容器之间的接口

javax.servlet.annocation包含了servlet、filter、listtener进行标注的注解

javax.servlet.descriptor包含了为web应用程序的配置信息提供编程式访问的类型

2、servlet

servlet接口定义了5个方法

init（ServletConfig config）初始化信息

service（setvletRequest request，ServletResponse response）响应请求

destroy（）

getServletinfo（）返回java.long.string的返回值

getServletConfig()返回 servletConfig的返回值

每一个Http请求，servlet容器都会创建一个servletequest实例，并将它传给servlet的service方法。servletRequest封装了请求信息

3、servletRequest接口中的方法：

（1）int getContentLength（）返回请求主体中的字节数

（2）String getContType（）返回主体的类型

（3）String getParameter（java.long.Stringname）返回指定请求参数的值

<http://127.0.0.1:8080/App03/myServlet?id=123&sname=tom>

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

String sname = req.getParameter("sname");

另一种常用方式用来返回一个html表单域的值

（4）String getProtocol（）返回这个http请求的协议名称及版本号

5、Servletresponse

Servletresponse接口表示一个servlet响应。在调用一个servlet的service方法之前，servlet容器会先创建一个servletResponse，并将它作为service的第二个参数传给他。serviceResponse隐藏了响应发给浏览器的复杂性

servletresponse中定义的其中一个方法getwriter方法，他返回可以将文本传给客户端的java.io.printwriter流，在默认情况下，Printwriter对象采用ISO-8859-1编码

6、在servlet容器初始化servlet时，servlet容器将servletConfig传递给servlet中info方法

在web.xml中配置初始化参数。

<init-param>

<param-name>paramname</param-name>

<param-value>db1</param-value>

</init-param>

为了从一个Servlet内部获得初始化的值，应该在Servlet容器传递给Servlet的init方法的ServletConfig中调用getInitParameter（参数），获得值。

7、servletContext

servletContext表示servlet应用程序，每个web应用程序只有一个Context

在servletconfig中调用getservletcontext方法可以获得ServletServletContext。

ServletContext中处理属性的一些方法：

java.lang.Object getAttribute(java.lang.String name)

java.util.Enumeration<java.lang.String> getAttributeNames()

setAttribute(java.lang.String name,java.lang.Object object)

removeAttribute(java.lang.String name)

8、GenericServlet

GenericServlet是继承了Servlet接口的类。

9、HttpServlet

servlet应用程序大多数都会用到http协议，servlet.http包包含了servlet应用程序的类和接口，大部分都覆盖了servlet的接口和类

httpservlet（继承Generic）、httpServletRequest（继承servletrequest）、httpservletresponse、httpsession、cookie

HttpServlet类覆盖了GenericServlet类。在使用HttpServlet时，还用使用到HttpServletRequest和HttpServletResponse对象，它们分别表示Servlet请求和响应。

doGet（HttpServletRequest request，HttpServletResponse response）

doPost（HttpServletRequest request，HttpServletResponse response）

HttpServletRequest表示HTTP环境中的Servlet请求，它继承ServletRequest接口。增加的几个方法：

（1）、java.lang.String getContextPath（）：返回表示请求context请求URL部分。

（2）、Cookie[] getCookies（）:返回一个Cookie对象数组。

（3）java.lang.String getHeader（java.lang.String name）:返回指定HTTP标头的值。

（4）java.lang.String getMethod（）：返回发出这条请求的HTTP方法的名称。

（5）java.lang.String getQueryString（）：返回请求URL中的查询字符串。

（6）HttpSession getSession（）：返回与这个请求有关的session对象。如果没有找到，则新建一个session对象。

（7）HttpSession getSession（boolean create）：当参数create为true时，与上面方法类似。

HttpServletResponse表示HTTP环境下的Servlet响应。主要方法：

addCookie（Cookie cookie）：给响应对象添加cookie。

addHeader（java.lang.String name,java.lang.String value）:给响应对象添加标头。

sendRedirect（java.lang.String location）:发送响应代号，将浏览器重定向到指定的位置。

二、session管理

1、网页重写

网址重写是一种Session追踪技术，需要将一个或多个token作为一个查询字符串添加到URL中。Token的格式一般是键=值：

url?key1=value1&ke2=value2

缺点：

（1）、在有些web浏览器中，URL限制2000个字符。

（2）、仅当有链接要插入值时，值才能转换成后面的资源。

（3）、某些字符，例如空格、&符号及问号都必须进行转码。

（4）、添加到URL中的信息隐蔽性不好。

（5）、网址重写必须在服务器端有效。所有的链接都必须带有值，这样可能出现一个问题，即一个页面可能有许多个链接。

2、隐藏域

利用隐藏域来保持状态，与采用网址重写技术类似。将要传递的信息放在HTML表单的隐藏域中。当用户提交表单时，隐藏域中的值也传递到服务器端。只有当页面包含表单，或者可以在页面中添加表单时，才适合隐藏域。

优势：（1）、能传递更多的信息。

（2）、隐蔽性更好。

（3）、传递的信息不需要进行编码。

3、cookie

网页重写和隐藏域都只适合于保持哪些不需要跨越许多页面的信息。如果这些信息需要跨越很多页面，这两种技术就变得很难实现，因为必须管理每一个页面的信息。cookie能解决网页重写和隐藏域都无法解决的问题。

cookie是自动在web服务器和浏览器之间来回传递的一小块信息。Cookie适用于跨越多个页面的信息。传输的过程由HTTP协议处理。每台Web服务器最多可以支持20个cookie。

Cookie的不足之处在于，用户可以通过修改浏览器设置来拒绝接收cookie。

要使用cookie，必须熟悉javax.servlet.http.cookie类，以及HttpServletRequest和HttpServletResponse接口中的几个方法。

创建cookie:

Cookie cookie = new Cookie(name,value);

例：创建一个选择语言的cookie：

Cookie languageSelectCookie=new Cookie(“language”,”Italian”);

创建Cookie后，可以设置它的domain、path及maxAge属性。其中maxAge比较重要，它决定了cookie的有效期。

为了将cookie发送到浏览器，需要在HttpServletResponse

上调用addCookie（cookie对象）；

第二次请求同一台服务器中的同一个资源或不同资源时，浏览器会同时把从web服务器处收到的cookie再传回去。

也可以利用javaScript在客户端创建或删除cookie。

要访问浏览器发出的cookie，可以在HttpServletRequest中使用getCookie方法。返回Cookie数组，如果请求中没有cookie，返回null。为了找到某个名称的cookie，需要迭代数组。例如：找一个名为maxRecords的cookie。

Cookie[] cookies = request.getCookies();

Cookie maxRecordsCookie = null;

if (cookies != null){

for (Cookie cookie:cookies) {

if (cookie.getName().equals(“maxRecords”)) {

maxRecordsCookie = cookie;

break;

}

}

}

没有直接删除cookie的方法。为了删除cookie，需要创建一个同名cookie，将它的maxAge属性设置为0，并在HttpServletResponse中添加一个新的cookie。

Cookie cookie = new Cookie(“name”,”value”);

cookie.setMaxAge(0);

response.addCookie(cookie);

4、HttpSession对象

在所有Session追踪技术中，HttpSession对象是最强大的， 也是功能最多的。用户可以没有或者有一个HttpSession，并且只能访问自己的HttpSession。

HttpSession是当一个用户第一次访问某个网站时自动创建的。通过HttpServletRequest调用getSession方法。

HttpSession的setAttribute方法将一个值放到HttpSession中：

void setAttribute(java.lang.String name,java.lang.object value) 。

通过在HttpSession中调用getAttribute方法，同时传递一个属性名称，可以获得HttpSession中保存的对象：

java.lang.Object getAttribute(java.lang.String name)

HttpSession中另外一个有用的方法是getAttributeNames，它返回一个Enumeration，迭代一个HttpSession中的所有属性：

Java.util.Enumeration<java.lang.String> getAttributeNames()

HttpSession中保存的值不发送到客户端，这是与其他Session管理方式不同的地方。Servlet容器为它所创建的HttpSeion生成一个唯一的标识，并将这个标识符作为一个token发送给客户端，一般是作为一个名为JSESSIONID的cookie，或者作为一个jsessionid参数添加到URL后面。在后续的请求中，浏览器会将这个token发送回服务器，使服务器能够知道是哪个用户发出的请求。

通过在HttpSession中调用getId方法，可以获取HttpSession的标识符。

java.lang.String getId();

HttpSession中定义了一个invalidate方法，强制Session过期，并将绑定到他的所有对象解除掉。

在默认的情况下，HttpSession是在用户一段时间不操作后过期。可以在部署描述符设置session-timeout中设置超时时间。

5、购物车

public class Product {

private int id;

private String name;

private String description;

private double price;

public Product(int id,String name,String description,double price){

this.id = id;

this.name = name;

this.description = description;

this.price = price;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

}

public class ShoppingItem {

private Product product;

private int quantity;

public ShoppingItem(Product product,int quantity){

this.product = product;

this.quantity = quantity;

}

public Product getProduct() {

return product;

}

public void setProduct(Product product) {

this.product = product;

}

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

}

public class ShoppingCartServlet extends HttpServlet {

private static final String CART\_ATTRIBUTE = "cart";

private List<Product> products = new ArrayList<Product>();

public void init(){

products.add(new Product(1,"Apple iPhone7 Plus",

"iPhone7 Plus 128GB",6368));

products.add(new Product(2,"Apple iPhone7",

"Apple iPhone 7 (A1660) 128G",5199));

products.add(new Product(3,"Apple iPhone 6s Plus",

"Apple iPhone 6s Plus (A1699) 128G",4799));

products.add(new Product(4,"Apple iPhone6",

"Apple iPhone6 32GB",2489));

}

private void sendProductList(HttpServletResponse response) throws IOException{

response.setContentType("text/html");

PrintWriter writer = response.getWriter();

writer.println("<html><head><title>Products</title></head><body>");

writer.println("<h2>Products</h2>");

writer.println("<ul>");

for (Product p:products) {

writer.println("<li>"+p.getName()+","+p.getPrice()

+","+"<a href='shoppingCartServlet?id="+p.getId()

+"&type=viewProductDetails'>Details</a></li>");

}

writer.println("</ul>");

writer.println("<a href='shoppingCartServlet?type=viewCart'>View Cart</a>");

writer.println("</body></html>");

}

private Product getProduct(int id){

for (Product p:products) {

if (p.getId() == id) {

return p;

}

}

return null;

}

private void sendProductDetails(HttpServletRequest request,

HttpServletResponse response) throws IOException{

response.setContentType("text/html");

PrintWriter writer = response.getWriter();

int id = Integer.parseInt(request.getParameter("id"));

Product p = getProduct(id);

if (p != null) {

writer.println("<html><head><title>Product Details</title></head><body>");

writer.println("<h2>Product Details</h2>");

writer.println("<form method='post'>");

writer.println("<input type='hidden' name='id' value='"+id+"'/>");

writer.println("<table>");

writer.println("<tr><td>Name:</td><td>"+p.getName()+"</td></tr>");

writer.println("<tr><td>Desciption:</td><td>"+p.getDescription()+"</td></tr>");

writer.println("<tr><td>Price:</td><td>"+p.getPrice()+"</td></tr>");

writer.println("<tr><td><input name='quantity'/></td>" +

"<td><input type='submit' value='Buy'/></td></tr>");

writer.println("<tr><td colspan='2'>" +

"<a href='shoppingCartServlet?type=products'>Product list</a></td></tr>");

writer.println("</table></form></body></html>");

} else {

writer.println("No product found");

}

}

private void showCart(HttpServletRequest request, HttpServletResponse response) throws IOException{

response.setContentType("text/html");

PrintWriter writer = response.getWriter();

writer.println("<html><head><title>Shopping cart</title></head><body>");

writer.println("<h2>Shopping cart</h2>");

writer.println("<a href='shoppingCartServlet?type=products'>Product list</a>");

HttpSession session = request.getSession();

List<ShoppingItem> cart = (List<ShoppingItem>)

session.getAttribute(CART\_ATTRIBUTE);

double total = 0.0;

if (cart != null) {

writer.println("<table>");

writer.println("<tr><td style='width:150px'>quantity</td>");

writer.println("<td style='width:150px'>name</td>");

writer.println("<td style='width:150px'>price</td>");

writer.println("<td style='width:150px'>amount</td></tr>");

for(ShoppingItem s:cart) {

Product p = s.getProduct();

int quantity = s.getQuantity();

if (quantity != 0) {

writer.println("<tr>");

writer.println("<td>"+quantity+"</td>");

writer.println("<td>"+p.getName()+"</td>");

writer.println("<td>"+p.getPrice()+"</td>");

double subPrice = p.getPrice()\*quantity;

writer.println("<td>"+subPrice+"</td>");

writer.println("</tr>");

total+=subPrice;

}

}

writer.println("<tr><td colspan='4' style='text-align:right'>"

+"Total:"+total+"</td></tr>");

writer.println("</table>");

writer.println("</body></html>");

}

}

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

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

if ("products".equals(type)) {

this.sendProductList(response);

} else if ("viewProductDetails".equals(type)) {

this.sendProductDetails(request, response);

} else if ("viewCart".equals(type)) {

this.showCart(request, response);

} else {

this.sendProductList(response);

}

}

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

int id = Integer.parseInt(request.getParameter("id"));

int quantity = Integer.parseInt(request.getParameter("quantity"));

Product p = this.getProduct(id);

if (p != null && quantity > 0) {

ShoppingItem item = new ShoppingItem(p,quantity);

HttpSession session = request.getSession();

List<ShoppingItem> cart =

(List<ShoppingItem>)session.getAttribute(CART\_ATTRIBUTE);

if (cart == null) {

cart = new ArrayList<ShoppingItem>();

session.setAttribute(CART\_ATTRIBUTE, cart);

}

cart.add(item);

this.showCart(request, response);

}

}

}

jsp（全称Java Server Pages）

Servlet有两个缺陷，1，在Servlet中编写的所有HTML标签都必须包含在java字符串中，这使得发送HTTP响应变成一项十分繁琐的工作。2，所有文本和HTML标签必须进行硬编码；因此，即使只是对其很小的修改，也需要重新编译。

JavaServer Pages（JSP）解决了Servlet中的两个缺陷问题。JSP并没有取代Servlet，互补。

3.1JSP概述

JSP页面其实是一个Servlet。但是，我们编写JSP页面要比编写Servlet要容易很多。JSP优势：1、不需要编译JSP页面。2、JSP页面的扩展名称为.jsp的文件，可以利用任何文本编辑器编写。

JSP和Servlet运行的容器一样，称为应用程序服务器。例如：开源的Tomcat。

第一次请求一个jsp页面，容器做两件事情：

1、将jsp页面转换成一个jsp页面的实现类，这个类实现了javax.servlet.jsp.JspPage、javax.servlet.jsp.HttpjspPage的java类。JspPage是javax.servlet.Servlet的子接口。所以jsp页面的实现类也是一个Servlet。

2、如果转换成功后，容器会编译Servlet。然后，加载、初始化，执行Servelt的生命周期。

JSP API，其中包含4个包：

（1）、javax.servlet.jsp:包含核心类和接口。

（2）、javax.servlet.jsp.tagext:包含开发定制标签的类型。

（3）、javax.el：为Unified Expression Language 提供API。

（4）、javax.servlet.jsp.el:提供容器必须支持的类，以便支持JSP中的Expression Language。

JSP API的网站：

<http://download.oracle.com/docs/cd/E17802_01/products/products/jsp/2.1/docs/jsp-2_1-pfd2/index.html>

jsp和html的区别

html静态页面，jsp动态生成

jsp和javascript的区别

javascript源代码由服务器发送到客户端，有客户端执行。客户端可以看到JavaScript代码而jsp不会

JSP语法

<%代码段 %> 等价的代码 <jsp:scriptlet> 代码片 </jsp:scriptlet>

任何文本、HTML标签、JSP元素必须写在脚本程序的外面

运行中文时吧格式改为UTF-8不然会出现乱码

JSP声明

一个声明语句可以声明一个或多个变量、方法，供后面的Java代码使用。在JSP文件中，您必须先声明这些变量和方法然后才能使用它们。

JSP声明的语法格式：

<%! declaration; [ declaration; ]+ ... %>

等价的XML语句，就像下面这样：

<jsp:declaration> 代码片段 </jsp:declaration> <%! int i = 0; %>

jsp表达式作用将内容的运算结果输出到客户端

<%=mag%>等价于<%out.println(mag);%>

一个JSP表达式中包含的脚本语言表达式，先被转化成String，然后插入到表达式出现的地方。

由于表达式的值会被转化成String，所以您可以在一个文本行中使用表达式而不用去管它是否是HTML标签。

表达式元素中可以包含任何符合Java语言规范的表达式，但是不能使用分号来结束表达式。

<%= 表达式 %>等价的XML语句：<jsp:expression> 表达式</jsp:expression>

不能在jsp程序段中定义方法

注释

<%-- 注释 --%>JSP注释，注释内容不会被发送至浏览器甚至不会被编译

<!-- 注释 --> HTML注释，通过浏览器查看网页源代码时可以看见注释内容

<\% 代表静态 <%常量

%\> 代表静态 %> 常量

\' 在属性中使用的单引号

\" 在属性中使用的双引号

JSP指令

指示jsp转换器应该如何将某个jsp页面转换成servlet的命令

JSP指令用来设置与整个JSP页面相关的属性。

JSP指令语法格式：

<%@ directive attribute="value" %>

|  |  |
| --- | --- |
| **指令** | **描述** |
| <%@ page ... %> | 定义页面的依赖属性，比如脚本语言、error页面、缓存需求等等  Page指令为容器提供当前页面的使用说明。一个JSP页面可以包含多个page指令。 |
| <%@ include ... %> | 包含其他文件 |
| <%@ taglib ... %> | 引入标签库的定义，可以是自定义标签 |

Page指令的语法格式：

<%@ page attribute="value" %>

等价的XML格式：

<jsp:directive.page attribute="value" />

下表列出与Page指令相关的属性：

|  |  |
| --- | --- |
| **属性** | **描述** |
| buffer | 指定out对象使用缓冲区的大小，单位kb |
| autoFlush | 控制out对象的 缓存区，默认true，缓冲区满时，被缓存的内容自动刷新，为false时只有调用response的flush才刷新 |
| contentType | 指定当前JSP页面的MIME类型和字符编码，就是response的内容类型，默认”text/html ” |
| errorPage | 指定当JSP页面发生异常时需要转向的错误处理页面 |
| isErrorPage | 指定当前页面是否可以作为另一个JSP页面的错误处理页面 |
| extends | 指定servlet从哪一个类继承 |
| import | 导入要使用的Java类 |
| info | 定义JSP页面的描述信息 |
| isThreadSafe | 指定对JSP页面的访问是否为线程安全 |
| language | 定义JSP页面所用的脚本语言，默认是Java |
| session | 指定JSP页面是否使用session |
| isELIgnored | 指定是否执行EL表达式 |
| isScriptingEnabled | 确定脚本元素能否被使用 |

include指令的语法格式如下：

<%@ include file="文件相对 url 地址" %>

**include** 指令中的文件名实际上是一个相对的 URL 地址。

插入外部文件

如果您没有给文件关联一个路径，JSP编译器默认在当前路径下寻找。

等价的XML语法：

<jsp:directive.include file="文件相对 url 地址" />

Taglib指令

JSP API允许用户自定义标签，一个自定义标签库就是自定义标签的集合。

Taglib指令引入一个自定义标签集合的定义，包括库路径、自定义标签。

Taglib指令的语法：

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

uri属性确定标签库的位置，prefix属性指定标签库的前缀。

等价的XML语法：

<jsp:directive.taglib uri="uri" prefix="prefixOfTag" />

JSP动作

JSP行为标签使用XML语法结构来控制servlet引擎。它能够动态插入一个文件，重用JavaBean组件，引导用户去另一个页面，为Java插件产生相关的HTML等等。

行为标签只有一种语法格式，它严格遵守XML标准：

<jsp:action\_name attribute="value" />

利用JSP动作可以动态地插入文件、重用JavaBean组件、把用户重定向到另外的页面、为Java插件生成HTML代码。

动作元素只有一种语法，它符合XML标准：

|  |  |
| --- | --- |
| **语法** | **描述** |
| jsp:include | 用于在当前页面中包含静态或动态资源，在页面被请求时引如一个文件 |
| jsp:useBean | 寻找和初始化一个JavaBean组件 |
| jsp:setProperty | 设置 JavaBean组件的值 |
| jsp:getProperty | 将 JavaBean组件的值插入到 output中 |
| jsp:forward | 从一个JSP文件向另一个文件传递一个包含用户请求的request对象 |
| jsp:plugin | 用于在生成的HTML页面中包含Applet和JavaBean对象 |
| jsp:element | 动态创建一个XML元素 |
| jsp:attribute | 定义动态创建的XML元素的属性 |
| jsp:body | 定义动态创建的XML元素的主体 |
| jsp:text | 用于封装模板数据 |

常见的属性

所有的动作要素都有两个属性：id属性和scope属性。

id属性：

id属性是动作元素的唯一标识，可以在JSP页面中引用。动作元素创建的id值可以通过PageContext来调用。

scope属性：

该属性用于识别动作元素的生命周期。 id属性和scope属性有直接关系，scope属性定义了相关联id对象的寿命。 scope属性有四个可能的值： (a) page, (b)request, (c)session, 和 (d) application。

<jsp:include>动作元素

<jsp:include>动作元素用来包含静态和动态的文件。该动作把指定文件插入正在生成的页面。语法格式如下：

<jsp:include page="相对 URL 地址" flush="true" />

　前面已经介绍过include指令，它是在JSP文件被转换成Servlet的时候引入文件，而这里的jsp:include动作不同，插入文件的时间是在页面被请求的时候。

以下是include动作相关的属性列表。

|  |  |
| --- | --- |
| **属性** | **描述** |
| page | 包含在页面中的相对URL地址。 |
| flush | 布尔属性，定义在包含资源前是否刷新缓存区。 |

实例

以下我们定义了两个文件 **date.jsp** 和 **main.jsp**，代码如下所示：

date.jsp文件代码：

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

pageEncoding="UTF-8"%>

<p>

今天的日期是: <%= (new java.util.Date()).toLocaleString()%>

</p>

main.jsp文件代码：

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

pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<h2>include 动作实例</h2>

<jsp:include page="date.jsp" flush="true" />

</body>

</html>

现在将以上两个文件放在服务器的根目录下，访问main.jsp文件。显示结果如下：

include 动作实例

今天的日期是: 2016-6-25 14:08:17

<jsp:useBean>动作元素

**jsp:useBean** 动作用来加载一个将在JSP页面中使用的JavaBean。

这个功能非常有用，因为它使得我们可以发挥 Java 组件复用的优势。

将表现逻辑和业务逻辑分割开来的方法之一

jsp:useBean动作最简单的语法为：

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

在类载入后，我们既可以通过 jsp:setProperty 和 jsp:getProperty 动作来修改和检索bean的属性。

以下是useBean动作相关的属性列表。

|  |  |
| --- | --- |
| **属性** | **描述** |
| class | 指定Bean的完整包名。 |
| type | 指定将引用该对象变量的类型。 |
| beanName | 通过 java.beans.Beans 的 instantiate() 方法指定Bean的名字。 |

在给出具体实例前，让我们先来看下 jsp:setProperty 和 jsp:getProperty 动作元素：

<jsp:setProperty>动作元素

jsp:setProperty用来设置已经实例化的Bean对象的属性，有两种用法。首先，你可以在jsp:useBean元素的外面（后面）使用jsp:setProperty，如下所示：

<jsp:useBean id="myName" ... />

...

<jsp:setProperty name="myName" property="someProperty" .../>

此时，不管jsp:useBean是找到了一个现有的Bean，还是新创建了一个Bean实例，jsp:setProperty都会执行。第二种用法是把jsp:setProperty放入jsp:useBean元素的内部，如下所示：

<jsp:useBean id="myName" ... >

...

<jsp:setProperty name="myName" property="someProperty" .../>

</jsp:useBean>

此时，jsp:setProperty只有在新建Bean实例时才会执行，如果是使用现有实例则不执行jsp:setProperty。

jsp:setProperty动作有下面四个属性,如下表：

|  |  |
| --- | --- |
| **属性** | **描述** |
| name | name属性是必需的。它表示要设置属性的是哪个Bean。 |
| property | property属性是必需的。它表示要设置哪个属性。有一个特殊用法：如果property的值是"\*"，表示所有名字和Bean属性名字匹配的请求参数都将被传递给相应的属性set方法。 |
| value | value 属性是可选的。该属性用来指定Bean属性的值。字符串数据会在目标类中通过标准的valueOf方法自动转换成数字、boolean、Boolean、 byte、Byte、char、Character。例如，boolean和Boolean类型的属性值（比如"true"）通过 Boolean.valueOf转换，int和Integer类型的属性值（比如"42"）通过Integer.valueOf转换。 　　value和param不能同时使用，但可以使用其中任意一个。 |
| param | param 是可选的。它指定用哪个请求参数作为Bean属性的值。如果当前请求没有参数，则什么事情也不做，系统不会把null传递给Bean属性的set方法。因此，你可以让Bean自己提供默认属性值，只有当请求参数明确指定了新值时才修改默认属性值。 |

<jsp:getProperty>动作元素

jsp:getProperty动作提取指定Bean属性的值，转换成字符串，然后输出。语法格式如下：

<jsp:useBean id="myName" ... />

...

<jsp:getProperty name="myName" property="someProperty" .../>

下表是与getProperty相关联的属性：

|  |  |
| --- | --- |
| **属性** | **描述** |
| name | 要检索的Bean属性名称。Bean必须已定义。 |
| property | 表示要提取Bean属性的值 |

实例

以下实例我们使用了Bean:

package com.runoob.main;

public class TestBean {

private String message = "菜鸟教程";

public String getMessage() {

return(message);

}

public void setMessage(String message) {

this.message = message;

}

}

编译以上实例文件 TestBean.java ：

$ javac TestBean.java

编译完成后会在当前目录下生成一个 **TestBean.class** 文件， 将该文件拷贝至当前 JSP 项目的 **WebContent/WEB-INF/classes/com/runoob/main** 下( com/runoob/main 包路径，没有需要手动创建)。

下面是一个很简单的例子，它的功能是装载一个Bean，然后设置/读取它的message属性。

现在让我们在main.jsp文件中调用该Bean:

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

pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<h2>Jsp 使用 JavaBean 实例</h2>

<jsp:useBean id="test" class="com.runoob.main.TestBean" />

<jsp:setProperty name="test"

property="message"

value="菜鸟教程..." />

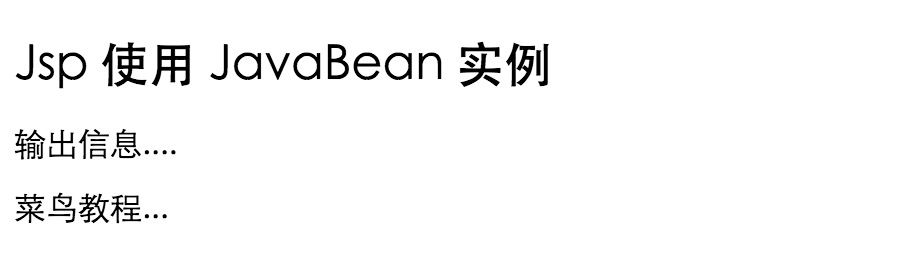
<p>输出信息....</p>

<jsp:getProperty name="test" property="message" />

</body>

</html>

浏览器访问，执行以上文件，输出如下所示：



<jsp:forward> 动作元素

　jsp:forward动作把请求转到另外的页面。jsp:forward标记只有一个属性page。语法格式如下所示：

<jsp:forward page="相对 URL 地址" />

以下是forward相关联的属性：

|  |  |
| --- | --- |
| **属性** | **描述** |
| page | page属性包含的是一个相对URL。page的值既可以直接给出，也可以在请求的时候动态计算，可以是一个JSP页面或者一个 Java Servlet. |

实例

以下实例我们使用了两个文件，分别是： date.jsp 和 main.jsp。

date.jsp 文件代码如下：

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

pageEncoding="UTF-8"%>

<p>

今天的日期是: <%= (new java.util.Date()).toLocaleString()%>

</p>

main.jsp文件代码：

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

pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<h2>forward 动作实例</h2>

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

</body>

</html>

现在将以上两个文件放在服务器的根目录下，访问main.jsp文件。显示结果如下：

今天的日期是: 2016-6-25 14:37:25

<jsp:plugin>动作元素

jsp:plugin动作用来根据浏览器的类型，插入通过Java插件 运行Java Applet所必需的OBJECT或EMBED元素。

如果需要的插件不存在，它会下载插件，然后执行Java组件。 Java组件可以是一个applet或一个JavaBean。

plugin动作有多个对应HTML元素的属性用于格式化Java 组件。param元素可用于向Applet 或 Bean 传递参数。

以下是使用plugin 动作元素的典型实例:

<jsp:plugin type="applet" codebase="dirname" code="MyApplet.class"

width="60" height="80">

<jsp:param name="fontcolor" value="red" />

<jsp:param name="background" value="black" />

<jsp:fallback>

Unable to initialize Java Plugin

</jsp:fallback>

</jsp:plugin>

如果你有兴趣可以尝试使用applet来测试jsp:plugin动作元素，<fallback>元素是一个新元素，在组件出现故障的错误是发送给用户错误信息。

<jsp:element> 、 <jsp:attribute>、 <jsp:body>动作元素

<jsp:element> 、 <jsp:attribute>、 <jsp:body>动作元素动态定义XML元素。动态是非常重要的，这就意味着XML元素在编译时是动态生成的而非静态。

以下实例动态定义了XML元素：

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

pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>

<jsp:element name="xmlElement">

<jsp:attribute name="xmlElementAttr">

属性值

</jsp:attribute>

<jsp:body>

XML 元素的主体

</jsp:body>

</jsp:element>

</body>

</html>

浏览器访问以下页面，输出结果如下所示：



<jsp:text>动作元素

<jsp:text>动作元素允许在JSP页面和文档中使用写入文本的模板，语法格式如下：

<jsp:text>模板数据</jsp:text>

以上文本模板不能包含其他元素，只能只能包含文本和EL表达式（注：EL表达式将在后续章节中介绍）。请注意，在XML文件中，您不能使用表达式如 ${whatever > 0}，因为>符号是非法的。 你可以使用 ${whatever gt 0}表达式或者嵌入在一个CDATA部分的值。

<jsp:text><![CDATA[<br>]]></jsp:text>

如果你需要在 XHTML 中声明 DOCTYPE,必须使用到<jsp:text>动作元素，实例如下：

<jsp:text><![CDATA[<!DOCTYPE html

PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"DTD/xhtml1-strict.dtd">]]>

</jsp:text>

<head><title>jsp:text action</title></head>

<body>

<books><book><jsp:text>

Welcome to JSP Programming

</jsp:text></book></books>

</body>

</html>

JSP隐含对象

对象 描述

request HttpServletRequest类的实例

response HttpServletResponse类的实例

out PrintWriter类的实例，用于把结果输出至网页上

session HttpSession类的实例

application ServletContext类的实例，与应用上下文有关

config ServletConfig类的实例

pageContext PageContext类的实例，提供对JSP页面所有对象以及命名空间的访问

page 类似于Java类中的this关键字

Exception Exception类的对象，代表发生错误的JSP页面中对应的异常对象

## request对象

request对象是javax.servlet.http.HttpServletRequest 类的实例。每当客户端请求一个JSP页面时，JSP引擎就会制造一个新的request对象来代表这个请求。

request对象提供了一系列方法来获取HTTP头信息，cookies，HTTP方法等等。

response对象

response对象是javax.servlet.http.HttpServletResponse类的实例。当服务器创建request对象时会同时创建用于响应这个客户端的response对象。

response对象也定义了处理HTTP头模块的接口。通过这个对象，开发者们可以添加新的cookies，时间戳，HTTP状态码等等。

out对象

out对象是 javax.servlet.jsp.JspWriter 类的实例，用来在response对象中写入内容。

最初的JspWriter类对象根据页面是否有缓存来进行不同的实例化操作。可以在page指令中使用buffered='false'属性来轻松关闭缓存。

JspWriter类包含了大部分java.io.PrintWriter类中的方法。不过，JspWriter新增了一些专为处理缓存而设计的方法。还有就是，JspWriter类会抛出IOExceptions异常，而PrintWriter不会。

下表列出了我们将会用来输出boolean，char，int，double，String，object等类型数据的重要方法：

|  |  |
| --- | --- |
| 方法 | 描述 |
| out.print(dataType dt) | 输出Type类型的值 |
| out.println(dataType dt) | 输出Type类型的值然后换行 |
| out.flush() | 刷新输出流 |

session对象

session对象是 javax.servlet.http.HttpSession 类的实例。和Java Servlets中的session对象有一样的行为。

session对象用来跟踪在各个客户端请求间的会话。

application对象

application对象直接包装了servlet的ServletContext类的对象，是javax.servlet.ServletContext 类的实例。

这个对象在JSP页面的整个生命周期中都代表着这个JSP页面。这个对象在JSP页面初始化时被创建，随着jspDestroy()方法的调用而被移除。

通过向application中添加属性，则所有组成您web应用的JSP文件都能访问到这些属性。

config对象

config对象是 javax.servlet.ServletConfig 类的实例，直接包装了servlet的ServletConfig类的对象。

这个对象允许开发者访问Servlet或者JSP引擎的初始化参数，比如文件路径等。

以下是config对象的使用方法，不是很重要，所以不常用：

config.getServletName();

它返回包含在<servlet-name>元素中的servlet名字，注意，<servlet-name>元素在 WEB-INF\web.xml 文件中定义。

pageContext 对象

pageContext对象是javax.servlet.jsp.PageContext 类的实例，用来代表整个JSP页面。

这个对象主要用来访问页面信息，同时过滤掉大部分实现细节。

这个对象存储了request对象和response对象的引用。application对象，config对象，session对象，out对象可以通过访问这个对象的属性来导出。

pageContext对象也包含了传给JSP页面的指令信息，包括缓存信息，ErrorPage URL,页面scope等。

PageContext类定义了一些字段，包括scope： PAGE\_SCOPE，REQUEST\_SCOPE，SESSION\_SCOPE， APPLICATION\_SCOPE。它也提供了40余种方法，有一半继承自javax.servlet.jsp.JspContext 类。主要用来存取属性。

其中一个重要的方法就是removeArribute()，它可接受一个或两个参数。比如，pageContext.removeArribute("attrName")移除四个scope中相关属性，但是下面这种方法只移除特定scope中的相关属性：

pageContext.removeAttribute("attrName", PAGE\_SCOPE);

getAttribute和setAttribute方法。属性可以指定一下4种范围：page ，request，session，application。page范围最小，保存的属性只能在同一个jsp页面使用，request是servletrequest，session是httpsession。application是servletcontext。

public abstract void setAttribute（String name，Objiectvalue，int scope ）

page 对象

这个对象就是页面实例的引用。它可以被看做是整个JSP页面的代表。

page 对象就是this对象的同义词。

exception 对象

exception 对象包装了从先前页面中抛出的异常信息。它通常被用来产生对出错条件的适当响应

控制流语句

<% if (day == 1 | day == 7) { %>

<p>今天是周末</p>

<% } else { %>

<p>今天不是周末</p>

<% } %>

<%for ( fontSize = 1; fontSize <= 3; fontSize++){ %>

<font color="green" size="<%= fontSize %>">

菜鸟教程

</font><br />

<%}%>

<%while ( fontSize <= 3){ %>

<font color="green" size="<%= fontSize %>">

菜鸟教程

</font><br />

<%fontSize++;%>

<%}%>