# 一、JSP相关技术详解

## 1.1构建web应用

引用：要想成为一个优秀的程序员，应该从基本功练起，所有的代码应该用简单的文本编辑器（EidtPlus、UItraEdit等工具来完成）。

坚持用最原始的工具来学习技术，会让你对整个技术的每个细节有更准确的把握。

你是否知道Eclipse创建web应用时为你做了些什么？如果不知道，那你最好还是不要用它。

正确对待IDE工具的态度是：可以使用工具单绝不依赖工具，即使用最简单的vi（UNIX下无格式编辑器）、记事本也一样可以完成非常优秀的项目。------当然本人达不到。

对于IDE工具，业内有一个说法：IDE工具会加快高手的开发效率，但会使初学者更白痴。

接下来徒手搭建一个Web应用：

1. 建立一个web应用的文件夹，以WebDemo文文件名的文件夹，任意目录即可。
2. 在WebDemo文件夹内，建立WEB-INF文件夹。（大小写要注意）。
3. 进入tomcat或其他任何的web容器内，找到任何一个web应用，在其WEB-INF文件夹下面找到web.xml复制其到刚刚建立的WEB-INF文件夹下。
4. 将上面复制的xml文件其更改为一个最原始的xml。修改后的xml内容为

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app xmls:xsi="http://xmlns.jcp.org/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://java.sun.com/xml/ns/javaee"  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_3\_1.xsd" version="3.1">  </web-app> |

1. 在WEB-INF文件夹下面创建一个classes文件夹和lib文件夹。其实这两个文件夹中放的

这时我们已经建立以个空的的web项目

所以项目的结构应该是这样：

WEB-INF

Classes

Lib

Web.xml

Test.Jsp

把整个WebDemo文件夹复制到 tomcat的webapp下，一个简单的web应用就建好了。

## 1.2配置描述符web.xml

Web.xml对于java web项目应用十分重要，在servlet2.5之前，每个java web应用中都必须包含一个web.xml文件，并且必须在WEB-INF下面。

从servlet3.0开始，WEB-INF路径下的web.xml文件不再是必须，但通常还是建议保留个配置文件。

对于java Web而言，WEB-INF是一个特殊的文件夹，web容器包含它下面的所有内容，客户浏览器无法访问该路径下的任何内容。

在servlet2.5之前，几乎所有的组件都通过web.xml文件来配置管理，从servlet3.0开始加入了注解来配置管理组件。因此web.xml变得更加简洁。

Web.xml文件的根元素是<web-app.../> 元素，在servlet3.0中，该元素新增了如下属性。

Metadata-complete: 该属性接受true或fasle两个属性值。当该属性为true时，该web应用将不会加载注解配置的web组件。（如：servlet，filter，listener等）

在web.xml中配置首页使用welcome-file-list元素，该元素包含多个welcome-file子元素。

|  |
| --- |
| <welcome-file-list>  <welcome-file>a.html</welcome-file> <welcome-file>a.htm</welcome-file>  <welcome-file>a.jsp</welcome-file>  </welcome-file-list> |

按上面顺序依次寻找。

# JSP的基本原理

JSP的本质是servlet。当用户向指定的servlet发送请求的时候servlet利用输出流动态生成html页面，包括每一个静态的html标签和所有在HTML页面出现的内容。

因为包含大量的静态标签，内容，导致开发效率及其低下。所有的表现逻辑，布局都必须耦合在java代码中。

Jsp解决了这个问题。

Jsp有两个部分组成：静态部分：标准的html标签，静态页面的内容，

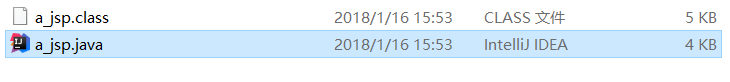
动态内容：受java控制的内容

|  |
| --- |
| <%@ page contentType="text/html; charset=UTF-8" language="java" errorPage=""%>  <html>  <head>  <title>欢迎</title>  </head>  <body>  欢迎学习java jsp  <%out.println(new java.util.Date());%>  </body>  </html> |

其中%保卫的部分为java脚本。脚本把静态的页面变成了动态的。

当启动tomcat后你会发现在work下面会有一个.java文件和一个.class文件

这个java类就是servlet类。



|  |
| --- |
| /\*  \* Generated by the Jasper component of Apache Tomcat  \* Version: Apache Tomcat/7.0.77  \* Generated at: 2018-01-16 07:53:56 UTC  \* Note: The last modified time of this file was set to  \* the last modified time of the source file after  \* generation to assist with modification tracking.  \*/  package org.apache.jsp;  import javax.servlet.\*;  import javax.servlet.http.\*;  import javax.servlet.jsp.\*;  public final class a\_jsp extends org.apache.jasper.runtime.HttpJspBase  implements org.apache.jasper.runtime.JspSourceDependent {  private static final javax.servlet.jsp.JspFactory \_jspxFactory =  javax.servlet.jsp.JspFactory.getDefaultFactory();  private static java.util.Map<java.lang.String,java.lang.Long> \_jspx\_dependants;  private volatile javax.el.ExpressionFactory \_el\_expressionfactory;  private volatile org.apache.tomcat.InstanceManager \_jsp\_instancemanager;  public java.util.Map<java.lang.String,java.lang.Long> getDependants() {  return \_jspx\_dependants;  }  public javax.el.ExpressionFactory \_jsp\_getExpressionFactory() {  if (\_el\_expressionfactory == null) {  synchronized (this) {  if (\_el\_expressionfactory == null) {  \_el\_expressionfactory = \_jspxFactory.getJspApplicationContext(getServletConfig().getServletContext()).getExpressionFactory();  }  }  }  return \_el\_expressionfactory;  }  public org.apache.tomcat.InstanceManager \_jsp\_getInstanceManager() {  if (\_jsp\_instancemanager == null) {  synchronized (this) {  if (\_jsp\_instancemanager == null) {  \_jsp\_instancemanager = org.apache.jasper.runtime.InstanceManagerFactory.getInstanceManager(getServletConfig());  }  }  }  return \_jsp\_instancemanager;  }  public void \_jspInit() {  }  public void \_jspDestroy() {  }  public void \_jspService(final javax.servlet.http.HttpServletRequest request, final javax.servlet.http.HttpServletResponse response)  throws java.io.IOException, javax.servlet.ServletException {  final javax.servlet.jsp.PageContext pageContext;  javax.servlet.http.HttpSession session = null;  final javax.servlet.ServletContext application;  final javax.servlet.ServletConfig config;  javax.servlet.jsp.JspWriter out = null;  final java.lang.Object page = this;  javax.servlet.jsp.JspWriter \_jspx\_out = null;  javax.servlet.jsp.PageContext \_jspx\_page\_context = null;  try {  response.setContentType("text/html; charset=UTF-8");  pageContext = \_jspxFactory.getPageContext(this, request, response,  "", 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\n");  out.write("<html>\r\n");  out.write("\t<head>\r\n");  out.write("\t\t<title>欢迎</title>\r\n");  out.write("\t</head>\r\n");  out.write("\t<body>\r\n");  out.write("\t\t欢迎学习java jsp\r\n");  out.write("\t\t");  out.println(new java.util.Date());  out.write("\r\n");  out.write("\t</body>\r\n");  out.write("</html>");  } catch (java.lang.Throwable t) {  if (!(t instanceof javax.servlet.jsp.SkipPageException)){  out = \_jspx\_out;  if (out != null && out.getBufferSize() != 0)  try {  if (response.isCommitted()) {  out.flush();  } else {  out.clearBuffer();  }  } catch (java.io.IOException e) {}  if (\_jspx\_page\_context != null) \_jspx\_page\_context.handlePageException(t);  else throw new ServletException(t);  }  } finally {  \_jspxFactory.releasePageContext(\_jspx\_page\_context);  }  }  } |



其中主要就是三个方法，init（）初始，destroy（）销毁，service（）对用的请求生成相应的方法。

Jsp的4个结论：

1. jsp文件必须在jsp服务器中运行。
2. Jsp文件必须生成servlet才能执行。
3. 每个jsp的第一个访问者都很慢，因为必须等待jsp编译成servlet。
4. Jsp页面的访问者无需安装任何客户端，因为到客户端的是html页面。

# jsp的4中基础语法

|  |
| --- |
| <%@ page contentType="text/html; charset=utf-8" language="java" errorPage=""%>  <html>  <head>  <title>欢迎</title>  </head>  <body>  欢迎学习java jsp  <%out.println(new java.util.Date());%>  <%-- jsp 注释--%>  <!-- html注释 -->  <%!  //这是一个声明  public int count;  //声明一个方法  public int getNum(){  return 7468;  }  %>  <br />  这里是一个数字：  <%  //脚本  out.println(count);  %>  <br />  输出方法的返回值：  <%  out.println(getNum());  %>  </body>  </html> |

## 3.1 jsp注释

<%-- jsp注释 --%>

<!-- html注释 -->

html的注释是可以通过源代码看到的，但jsp的注释是看不到的，说明jsp注释是不会被发送到客户端的。

## 3.2 jsp声明

<%! 声明部分 %>

## 3.3 输出jsp表达式

<%= 表达式 %>

注意：表达式的后面不能有分号。

## 3.4脚本

以前jsp的脚本应用非常广泛，脚本里面几乎可以包含任何可以执行的java代码。

脚本会转换为jspservice方法里面的内容，整个静态页面都是jspservice方法里面通过输出语句来输出，因为java中不允许在方法中定义方法所以脚本里面不可以定义方法。

在脚本中声明的变量是局部变量，不可以使用修饰符。

页面：

|  |
| --- |
| <%@ page contentType="text/html; charset=utf-8" language="java" errorPage=""%>  <html>  <head>  <title>欢迎</title>  </head>  <body>  欢迎学习java jsp  <%out.println(new java.util.Date());%>  <%-- jsp 注释--%>  <!-- html注释 -->  <%!  //这是一个声明  public int count;  //声明一个方法  public int getNum(){  return 7468;  }  %>  <br />  这里是一个数字：  <%  //脚本  out.println(count);  %>  <br />  输出方法的返回值：  <%  out.println(getNum());  %>  <br />  这里是输出表达式  <%= count++ %>  <br />  输出方法的值：  <%= getNum() %>  <br />  循环5个tr：  <% for(int a=0;a<6;a++){ %>    <tr>  <td>姓名：</td>  <td> </td>  </tr>  <% } %>  </body>  </html> |

|  |
| --- |
| */\*  \* Generated by the Jasper component of Apache Tomcat  \* Version: Apache Tomcat/7.0.77  \* Generated at: 2018-01-17 06:36:03 UTC  \* Note: The last modified time of this file was set to  \* the last modified time of the source file after  \* generation to assist with modification tracking.  \*/* **package** org.apache.jsp;  **import** javax.servlet.\*; **import** javax.servlet.http.\*; **import** javax.servlet.jsp.\*;  **public final class** a\_jsp **extends** org.apache.jasper.runtime.HttpJspBase  **implements** org.apache.jasper.runtime.JspSourceDependent {     *//这是一个声明* **public int** count;  *//声明一个方法* **public int** getNum(){  **return** 7468;   }    **private static final** javax.servlet.jsp.JspFactory \_jspxFactory =  javax.servlet.jsp.JspFactory.getDefaultFactory();   **private static** java.util.Map<java.lang.String,java.lang.Long> \_jspx\_dependants;   **private volatile** javax.el.ExpressionFactory \_el\_expressionfactory;  **private volatile** org.apache.tomcat.InstanceManager \_jsp\_instancemanager;   **public** java.util.Map<java.lang.String,java.lang.Long> getDependants() {  **return** \_jspx\_dependants;  }   **public** javax.el.ExpressionFactory \_jsp\_getExpressionFactory() {  **if** (\_el\_expressionfactory == **null**) {  **synchronized** (**this**) {  **if** (\_el\_expressionfactory == **null**) {  \_el\_expressionfactory = \_jspxFactory.getJspApplicationContext(getServletConfig().getServletContext()).getExpressionFactory();  }  }  }  **return** \_el\_expressionfactory;  }   **public** org.apache.tomcat.InstanceManager \_jsp\_getInstanceManager() {  **if** (\_jsp\_instancemanager == **null**) {  **synchronized** (**this**) {  **if** (\_jsp\_instancemanager == **null**) {  \_jsp\_instancemanager = org.apache.jasper.runtime.InstanceManagerFactory.getInstanceManager(getServletConfig());  }  }  }  **return** \_jsp\_instancemanager;  }   **public void** \_jspInit() {  }   **public void** \_jspDestroy() {  }   **public void** \_jspService(**final** javax.servlet.http.HttpServletRequest request, **final** javax.servlet.http.HttpServletResponse response)  **throws** java.io.IOException, javax.servlet.ServletException {   **final** javax.servlet.jsp.PageContext pageContext;  javax.servlet.http.HttpSession session = **null**;  **final** javax.servlet.ServletContext application;  **final** javax.servlet.ServletConfig config;  javax.servlet.jsp.JspWriter out = **null**;  **final** java.lang.Object page = **this**;  javax.servlet.jsp.JspWriter \_jspx\_out = **null**;  javax.servlet.jsp.PageContext \_jspx\_page\_context = **null**;    **try** {  response.setContentType(**"text/html; charset=utf-8"**);  pageContext = \_jspxFactory.getPageContext(**this**, request, response,  **""**, **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\n"**);  out.write(**"<html>\r\n"**);  out.write(**"\t<head>\r\n"**);  out.write(**"\t\t<title>欢迎</title>\r\n"**);  out.write(**"\t</head>\r\n"**);  out.write(**"\t<body>\r\n"**);  out.write(**"\t\t欢迎学习java jsp\r\n"**);  out.write(**"\t\t"**); out.println(**new** java.util.Date());  out.write(**"\r\n"**);  out.write(**"\t\t"**);  out.write(**"\r\n"**);  out.write(**"\t\t<!-- html注释 -->\r\n"**);  out.write(**"\t\t"**);  out.write(**"\r\n"**);  out.write(**"\t\t <br />\r\n"**);  out.write(**"\t\t 这里是一个数字：\r\n"**);  out.write(**"\t\t "**);    *//脚本* out.println(count);    out.write(**"\r\n"**);  out.write(**"\t\t <br />\r\n"**);  out.write(**"\t\t 输出方法的返回值：\r\n"**);  out.write(**"\t\t "**);   out.println(getNum());    out.write(**"\r\n"**);  out.write(**"\t\t <br />\r\n"**);  out.write(**"\t\t 这里是输出表达式\r\n"**);  out.write(**"\t\t "**);  out.print( count++ );  out.write(**"\r\n"**);  out.write(**"\t\t <br />\r\n"**);  out.write(**"\t\t 输出方法的值：\r\n"**);  out.write(**"\t\t "**);  out.print( getNum() );  out.write(**"\r\n"**);  out.write(**"\r\n"**);  out.write(**"\t\t <br />\r\n"**);  out.write(**"\r\n"**);  out.write(**"\t\t循环5个tr：\r\n"**);  out.write(**"\t\t"**);  **for**(**int** a=0;a<6;a++){   out.write(**"\r\n"**);  out.write(**"\t\t\r\n"**);  out.write(**"\t\t\t<tr>\r\n"**);  out.write(**"\t\t\t\t<td>姓名：</td>\r\n"**);  out.write(**"\t\t\t\t<td> </td>\r\n"**);  out.write(**"\t\t\t</tr>\t\r\n"**);  out.write(**"\r\n"**);  out.write(**"\t\t"**);  }   out.write(**"\r\n"**);  out.write(**"\r\n"**);  out.write(**"\r\n"**);  out.write(**"\t</body>\r\n"**);  out.write(**"</html>"**);  } **catch** (java.lang.Throwable t) {  **if** (!(t **instanceof** javax.servlet.jsp.SkipPageException)){  out = \_jspx\_out;  **if** (out != **null** && out.getBufferSize() != 0)  **try** {  **if** (response.isCommitted()) {  out.flush();  } **else** {  out.clearBuffer();  }  } **catch** (java.io.IOException e) {}  **if** (\_jspx\_page\_context != **null**) \_jspx\_page\_context.handlePageException(t);  **else throw new** ServletException(t);  }  } **finally** {  \_jspxFactory.releasePageContext(\_jspx\_page\_context);  }  } } |

# JSP的三个编译指令

Page:该指令是针对当前页面的指令。

Include:用于指定包含另一个页面。

Taglib：用于定义和访问自定义标签。

使用编译指令的语法格式如下：

<%@ 编译指令名 属性名=”属性值”……%>

## 4.1 page指令

|  |
| --- |
| <%@  [language=”Java”]  [extends=”package.class”]  [import=”package.class | package.\*,……”]  [session=”true|false”]  [buffer=”none | 8KB | size Kb”]  [autoFlush=”true | false”]  [isThreadSafe=”true | false”]  [info=”text”]  [errorPage=”relativeURL”]  [contenType=””mimeType[;charset=characterSet]” | “text/html;charSet=ISO-8859-1 ”]  [pageEncoding=”ISO-8859-1”]  [isErrorPage=”true | false”]  %> |

下面依次介绍page指令各属性的意义。

Language：声明当前JSP页面使用的脚本语言的种类，因为页面是JSP页面，该属性的值通常都是java，该属性的默认值也是java，所以通常无需设置。

Extends：指定该JSP页面转换成的java类所继承的父类，或所实现的接口。

Import：用来导入包。下面几个包是默认自动导入的，不需要显示导入。默认导入的包有：java.lang.\*、javax.servlet.\*、Javax.servlet.jsp.\*、java.servlet.html.\*。

session：设计这个JSP页面是否需要 HTTP Session.

buffer；指定输出缓冲区的大小。输出缓冲区的内部对象：out用于缓冲JSP页面对客户浏览器的输出，默认值为8KB，可以设置为none，也可以社遏制为其他值，单位为KB。

autoFlus：当输出缓冲区将溢出时，是否需要强制输出缓冲区的内容。设置为true时为正常输出：如果设置为false，则会在buffer溢出时产生一个异常。

Info：设置该JSP程序的信息，也可以看做其说明，可以通过Servlet.getServletInfo()方法获取该值。如果在JSP页面中，可直接调用getServletInfo()方法去该值，因为JSP页面的实质就是Servlet。

ErrorPage：指定错误处理页面。如果本页面产生了异常或者错误，而该JSP页面没有对应的处理代码，则会自定调用该属性所指定的JSP页面。

因为JSP内建了异常机制支持，所以JSP可以不处理异常，即使是checked异常。

IsErrorPage：设置本JSP页面是否为错误处理程序。如果该页面本身已是错误处理页面，则通常无需指定errorPage属性。

ContentType：用于设定生成网页的文件格式和编码字符集，即MIME类型和页面字符集类型，默认的MIME类型是text/html:默认的字符集类型为ISO-8859-1.

PageEncoding:指定生成网页的编码字符集。

Include指令:可以将一个外部文件嵌入到当前JSP文件中，同时接卸这个页面中的JSP语句（如果有的话）。这是个静态的include语句，它会把目标页面的其他编译指令也包含进来，动态include则不会。 include既可以静态的文本，也可以引用动态的JSP页面。

## 4.2 include指令

使用include指令可一把一个外部文件引入到当前的jsp当中。

可以引入静态文件，也可以引入jsp页面。

# JSP 的7个动作指令

动作指令和编译指令不一样，编译指令是通知servlet引擎的处理消息，而动作指令只是运行时的动作。

## 5.1 forward 指令

|  |
| --- |
| 第一种不带参数  <jsp:forward page="forward\_result.jsp"/> |
| 第二种带参数  <jsp:forward page="forward\_result.jsp">  <jsp:param name="age" value="29"/>  </jsp:forward> |

结果页：

|  |
| --- |
| <%=request.getParameter("age") %> |

## 5.2 include指令

|  |
| --- |
| <%@ page contentType="text/html;charst=utf-8" language="java" %> |

带参数的

|  |
| --- |
| <jsp:include page="jspinclude1.jsp">  <jsp:param name="username" value="lipw"/>  </jsp:include> |

注意：由上面看，forward和include很相似。

通过查看生成的servlet代码可以发现，

forward指令使用\_jspx\_page\_context的forward()方法引入目标页；

Include 指令则使用JspRuntimeLibrary的include()方法引入界面；

区别在于forward是新页面完全替代旧页面，而include是旧页面包含新页面。

## 5.3 useBean、setproperty、getProperty 指令

这三个指令都是与javabean相关的指令，其中userbean指令用于在jsp中初始化一个java实体类。

setProperty指令用于为Javabean实例的属性设置属性值：getProperty指令用于输出javabean实例的属性。

|  |
| --- |
| <%@ page contentType="text/html;charset=utf-8" language="java" %>  <!DOCTYPE html>  <html>  <head>  <title>userBean</title>  </head>  <body>  <jsp:useBean id="p1" class="com.Person" scope="page" />  <jsp:setProperty name="p1" property="name" value="lipw" />  <jsp:setProperty name="p1" property="age" value="23" />  <jsp:getProperty name="p1" property="name" /><br/>  <jsp:getProperty name="p1" property="age" />  </body>  </html> |

脚本方式

|  |
| --- |
| <%@ page contentType="text/html;charset=utf-8" language="java" %>  <!DOCTYPE html>  <html>  <head>  <title>bean1</title>  </head>  <body>  <%  Person p1=new Person();  pageContext.setAttribute("p1",p1);  p1.setName("sdsad");  p1.setAge(253);  %>  输出：  <%= p1.getName() %>  <%= p1.getAge() %>  </body>  </html> |

## 5.4 plugin 指令

## 5.5 param指令

## 5.6 JSP脚本中的九个内置对象

JSP脚本中包含九个内置对象，这九个内置对象都是servlet API的实例，只是jsp默认的对其进行录了初始化。（由jsp页面对servlet的——jspservice()方法来创建这些实例），也就是说，他们已经是对象了。

九大内置对象是：application，config，Exception，out，page，pagecontext，request

,response,session

打开任意一个jsp编译而成的java文件，里面的service（）方法，

|  |
| --- |
| final javax.servlet.jsp.PageContext pageContext;  javax.servlet.http.HttpSession session = null;  final javax.servlet.ServletContext application;  final javax.servlet.ServletConfig config;  javax.servlet.jsp.JspWriter out = null;  final java.lang.Object page = this;  javax.servlet.jsp.JspWriter \_jspx\_out = null;  javax.servlet.jsp.PageContext \_jspx\_page\_context = null;  try {  response.setContentType("text/html; charset=utf-8");  pageContext = \_jspxFactory.getPageContext(this, request, response,  "error.jsp", true, 8192, true);  \_jspx\_page\_context = pageContext;  application = pageContext.getServletContext();  config = pageContext.getServletConfig();  session = pageContext.getSession();  out = pageContext.getOut();  \_jspx\_out = out; |

，因为内置对象都是在service中，因此只能在jsp脚本、表达式中使用这些内置对象。不能在声明中使用。 否则系统将提示找不到这些变量

上面的代码中并没有exception内置对象，这与页面正好相符，只有当页面的page指令的iserrorPage属性为true时，才可以使用exception对象。也就是说异常处理页面对应servet时才会初始化exception。

### 5.6.1 application 对象

先介绍web服务器的实现原理，

对于绝大多数的浏览器，需要完成三件事：

1. 向远程服务器发送请求。
2. 读取远程服务器返回的字符串数据。
3. 负责根据字符串数据渲染出一个丰富的页面。

Application 对整个web应用有效，

Session对一次回话有效，

Request 对一次请求有效

Page 对当前页面有效

Jsp中的application、session、request、pageContext 四个对象分别用于操作对应作用域中的数据。

Application通常有两个作用：

1. 在整个web应用的多个jsp、servlet之间共享数据
2. 访问web应用的配置参数

在servlet中并没有application内置对象，所以通过

|  |
| --- |
| ServletContext sc=getservletConfig().getservletContext(); |

获取ServletContext实例。

在整个web应用中的多个jsp和servlet中

|  |
| --- |
| y>  <%! int i; %>  <%  application.setAttribute("conter",String.valueOf(++i));  %>  <%=i%> |

|  |
| --- |
| <%@ page contentType="text/html;charset=utf-8" language="java" %>  <!DOCTYPE html>  <html>  <head>  <title></title>  </head>  <body>  <%=application.getAttribute("conter")%>  </body>  </html> |

访问web应用中的应用参数

|  |
| --- |
| <%  String name=application.getInitParameter("user");  String pass=application.getInitParameter("pass");  %>  <%=name%>  <%=pass%> |

Web.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app xmls:xsi="http://xmlns.jcp.org/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://java.sun.com/xml/ns/javaee"  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_3\_1.xsd" version="3.1">  <welcome-file-list>  <welcome-file>a.jsp</welcome-file>  </welcome-file-list>  <context-param>  <param-name>user</param-name>  <param-value>root</param-value>  </context-param>  <context-param>  <param-name>pass</param-name>  <param-value>123456</param-value>  </context-param>  </web-app> |

### 5.6.2 config

|  |
| --- |
| <%=config.getServletName()%> |

所有的jsp都有相同的名字jsp。

Config 对象是servletconfig的实例，该接口用来获取配置文件内容方法是getInitParameter(String param).

页面：

|  |
| --- |
| <%@ page contentType="text/html;chatset=utf-8" language="java" %>  <!DOCTYPE html>  <html>  <head>  <title>config </title>  </head>  <body>  <%=config.getServletName()%>  <%=config.getInitParameter("name")%>  <%=config.getInitParameter("age")%>  </body>  </html> |

Web.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app xmls:xsi="http://xmlns.jcp.org/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://java.sun.com/xml/ns/javaee"  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_3\_1.xsd" version="3.1">  <welcome-file-list>  <welcome-file>a.jsp</welcome-file>  </welcome-file-list>  <context-param>  <param-name>user</param-name>  <param-value>root</param-value>  </context-param>  <context-param>  <param-name>pass</param-name>  <param-value>123456</param-value>  </context-param>  <servlet>  <servlet-name>config</servlet-name>  <jsp-file>/config.jsp</jsp-file>  <init-param>  <param-name>name</param-name>  <param-value>lipw</param-value>  </init-param>  <init-param>  <param-name>age</param-name>  <param-value>600</param-value>  </init-param>  </servlet>  <servlet-mapping>  <servlet-name>config</servlet-name>  <url-pattern>/config</url-pattern>  </servlet-mapping>  </web-app> |

配置jsp也是在web.xml中被当成servlet配置，为servlet配置参数使用 init-param元素，该元素可以接收parap-name和param-vlaue两个子元素，分别制定参数名称和参数值。

也可以直接访问jsp，但会发现jsp中没有获取到web.xml中配置的值，所以为了获取初始值还是通过配置的路径访问。

### 5.6.3 exception 对象

Exception 是throwable的实例。代表jsp脚本中产生的错误和异常。是jsp页面异常机制的一部分。

Exception尽在异常处理页面才起作用。

通过查看生成的servlet.Java 文件可以看出，脚本已经处在try中了所以无需再处理异常。

Jsp的处理机制对声明不起作用。

|  |
| --- |
| <%@ page contentType="text/html;charset=utf-8" language="java" errorPage="exception1.jsp" %>  <!DOCTYPE html>  <html>  <head>  <title>exception</title>  </head>  <body>  <%  int num=6/0;  %>  </body>  </html> |

异常处理页面

|  |
| --- |
| <%@ page contentType="text;html;charset=utf-8" language="java" isErrorPage="true" %>  <!DOCTYPE html>  <html>  <head>  <title>exception1</title>  </head>  <body>  <%=exception.getClass()%>  这里是区分的地方<br />  <%=exception.getMessage()%>  </body>  </html> |

只有当异常页面的 isErrorPage属性为true的时候，才可以访问exception对象。

### 5.6.4 out对象

Out对象代表一个页面输出流，一般在输出表达式可以使用的地方out 都可以使用。

|  |
| --- |
| <%@ page contentType="text/html;charset=utf-8" language="java" %>  <!DOCTYPE html>  <html>  <head>  <title>out</title>  </head>  <body>  <% out.println("这是out输出对象。"); %>  </body>  </html> |

可以使用out的地方都可以使用表达式，且表达式更加简洁。表达式的本质就是out.println();

### 5.6.5 pageContext 对象

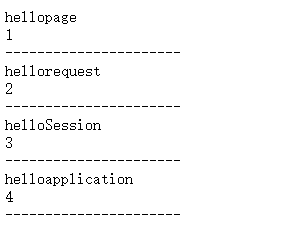
这个对象代表页面上下文，该对象主要用于访问页面的共享数据，使用pageContext可以访问page，requst，session，apppliction。范围内的内容。

有getAttribute（String namr）访问page范围内的name属性。

getAttribute（String name,int scope） 获取对应范围的内容。

相对应的有两个set方法。

|  |
| --- |
| <%@ page contentType="text/html;charset=utf-8" language="java" %>  <!DOCTYPE html>  <html>  <head>  <title>pageContext</title>  </head>  <body>  <%  pageContext.setAttribute("page","hellopage");  pageContext.setAttribute("request","hellorequest",pageContext.REQUEST\_SCOPE);  pageContext.setAttribute("session","helloSession",pageContext.SESSION\_SCOPE);  pageContext.setAttribute("application","helloapplication",pageContext.APPLICATION\_SCOPE);  %>  <%= pageContext.getAttribute("page")%><br />  <%= pageContext.getAttributesScope("page")%><br />  ----------------------<br/>  <%= pageContext.getAttribute("request",pageContext.REQUEST\_SCOPE)%><br />  <%= pageContext.getAttributesScope("request")%><br />  ----------------------<br/>  <%= pageContext.getAttribute("session",pageContext.SESSION\_SCOPE)%><br />  <%= pageContext.getAttributesScope("session")%><br />  ----------------------<br/>  <%= pageContext.getAttribute("application",pageContext.APPLICATION\_SCOPE)%><br />  <%= pageContext.getAttributesScope("application")%><br />  ----------------------<br/>  </body>  </html> |



1：对应page范围

2：对应request范围

3：对应session范围

4：对应application范围

### 5.6.6 request对象

每个request对象封装者一次用户请求，并且所有的请求参数都被封装在request里面。

Request可以代表本次请求的范围，所以还可以用来操作request范围内的属性。

#### 获取请求头和请求参数

String getParameter(String psrsmName) : 获取paramName请求参数的值。

Map getParameterMap():获取所有请求参数和参数值所组成的Map对象

Enumeration getParameterNames(): 获取所有请求参数所组成的Enumeation 对象。

String[] getParameter values(String name) :paramName 请求参数的值，当请求参数值有多个时，该方法将返回多个值所组成的数组。

HttpServletRequest ：提供了如下方法来访问请求头

String getHeader(String name) : 根据指定请求头的值。

Java.util.Enumeration<String> getHeaderNames():获取所有请求头的名称。

Java.util.Enumeration<Sting> getHeaders(String name) : 获取指定头的多个值。

Int getHeader(String name) : 获取指定请求头的值，并将该值转为整数值。

请求头和请求参数都是从用户端发从的，但请求头通常又浏览器自动添加。

Get方式请求：参数名和值跟在url之后。大小一般不超过2kb,

Post方式请求：请求的参数和值一般放在HTML HEADER 中传输，安全性高。

SetCharactoterEnCoding(“gbk”) ; 设置request编码使用放入字符集。

### 5.6.7 response 对象

Response 代表服务端响应客户端响应。大部分的时候不使用response，而使用out，

Out代表页面输出流。

Out是jspWriter的实例，jspWriter是writer的子类，writer是字符流，无法输出非字符串内容。

若输出一个PDF，out则不可以达到目的，必须使用response。

#### response响应生成费字符响应

|  |
| --- |
| <%@ page contentType="image/png" language="java" %>  <%@ page import="java.awt.image.\*,javax.imageio.\*,java.io.\*,java.awt.\*" %>  <%  // 创建BufferedImage对象  BufferedImage image=new BufferedImage(340,160,BufferedImage.TYPE\_INT\_RGB);  // 以image对象获取Graphics对象  Graphics g=image.getGraphics();  // 使用Graphics对象画画，所画的图像将会出现在image对象中  g.fillRect(0,0,400,400);  g.setColor(new Color(255,0,0));  g.fillArc(20,20,100,100,30,120);  g.setColor(new Color(0,255,0));  g.fillArc(20,20,100,100,150,120);  g.setColor(new Color(0,0,255));  g.fillArc(20,20,100,100,270,120);  g.setColor(new Color(0,0,0));  g.setFont(new Font("Arial Black",Font.PLAIN,16));  g.drawString("red:climb",200,60);  g.drawString("green:climb",200,100);  g.drawString("blue:climb",200,140);  g.dispose();  ImageIO.write(image,"png",response.getOutputStream());  %> |

也可以在别的页面使用image标签来显示这个图片页面

|  |
| --- |
| <Img arc=”response.jsp”> |

#### 重定向

重定向是response的另一个用，与foward不同，重定向会丢失所有的请求参数和request范围的属性，因为重定向将生成二次请求。与前一次请求不在同一个request中。

|  |
| --- |
| <%@ page contentType="text/html;charset=utf-8" language="java" %>  <!DOCTYPE html>  <html>  <head>  <title>redirect</title>  </head>  <body>  <% response.sendRedirect("redirect1.jsp"); %>  </body>  </html> |

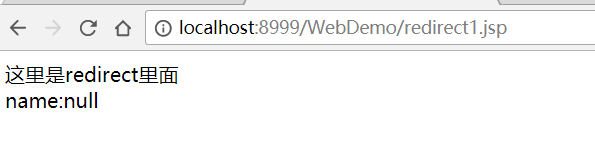
跳转页面：

|  |
| --- |
| <%@ page contentType="text/html;charset=utf-8" language="java" %>  <!DOCTYPE html>  <html>  <head>  <title>redirect</title>  </head>  <body>  这里是redirect里面  <br />  name:<%= request.getParameter("name") %>  </body>  </html> |

地址栏：

|  |
| --- |
| http://localhost:8999/WebDemo/redirect.jsp?name=pppppp |

结果：



发现地址栏中的url改变了，变成重定向的目标地址。

使用重定向会丢失所有请求参数，跟在地址栏输入新的地址回车同一个效果，即发送了第二次请求。

#### 增加cookie

Cookie 通常用于网站保存用户的某些信息，比如客户的用户名和客户的喜好等。

Cookie与session的不同：session会随着浏览器的关闭而消失，但cookie一直存放在客户端，除非超出生命周期。

增加cookie也是response内置对象完成的。

方法：void addCookie(Cookie cookie)

步骤：

1. 创建cookie实例，Cookie的构造器为cookie（String name,String value）
2. 设置cookie的生命周期，即多长时间有效。
3. 向客户端写cookie

添加cookie

|  |
| --- |
| <%@ page contentType="text/html;charset=utf-8" language="java" %>  <!DOCTYPE html>  <html>  <head>  <title>cookie</title>  </head>  <body>  <%  String name = request.getParameter("name");  out.println("========"+name);  Cookie cookie=new Cookie("name",name);  cookie.setMaxAge(60);  response.addCookie(cookie);  out.println("+++++="+name);  %>  </body>  </html> |

获取cookie

|  |
| --- |
| <%@ page contentType="text/html;charset=utf-8" language="java" %>  <!DOCTYPE html>  <html>  <head>  <title>cookie</title>  </head>  <body>  <%  Cookie[] cookies=request.getCookies();  out.println("========");  for(Cookie c:cookies){  out.println("-----");  if(c.getName().equals("name")){  out.println("\*\*\*\*"+c.getValue());  }  }  %>  </body>  </html> |

请求时：

|  |
| --- |
| http://localhost:8999/WebDemo/cookie.jsp?name=hhhh |

使用cookie必须设置生命周期，否则会随着浏览器的关闭而消失。

默认情况下cookie不允许出现中文字符。

所以先借助 java.net.URlEncoder对中文字符串进行编码，将编码后的值存入cookie，取得时候先取值，再使用java.net.URLDeecoder进行解码。

|  |
| --- |
| <%@ page contentType="text/html;charset=utf-8" language="java" %>  <!DOCTYPE html>  <html>  <head>  <title>cookie</title>  </head>  <body>  <%  Cookie cookie=new Cookie("chname",java.net.URLEncoder.encode("孙悟空","gbk"));  cookie.setMaxAge(60);  response.addCookie(cookie);  Cookie[] cookies=request.getCookies();  for(Cookie c:cookies){  out.println("-----");  if(c.getName().equals("chname")){  out.println("\*\*\*\*"+ java.net.URLDecoder.decode(c.getValue()));  }  }  %>  </body>  </html> |

## 5.6.8 session对象

Session也是常用的对象，代表从浏览器连接服务器开始到，浏览器断开服务器结束。

Session 是httpsession的实例，

有两个常用的方法，

setAttributr（Stirng name）

getAttribute（sting name）

Session 里面的内容必须是可序列化的，可以是任何可序列化对象。

序列化就是把Java对象储存在某一地方（硬盘、网络），以便于传输。