

JavaWeb之JSP

本节目标

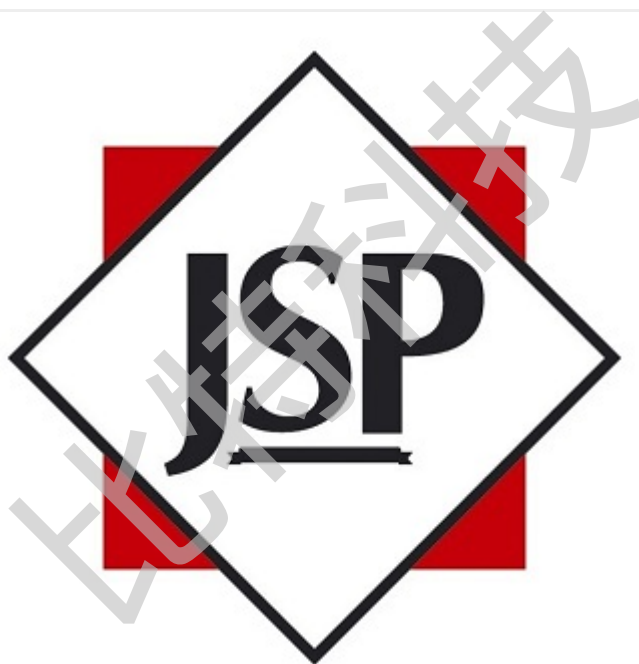
1.JSP基础语法 2.JSP内置对象 3.JSP状态管理 4.JSP指令与动作元素

1.JSP应用示例

编写一个JSP应用，完成动态的展示欢迎信息。

- 基于Maven工具创建一个JavaWeb项目
- 编写一个index.jsp页面，输出欢迎信息

2.JSP基础语法



JSP全名为：**Java Server Pages**，其根本是一个简化的Servlet设计，实现了在Java中使用HTML标签。JSP是一种动态网页技术标准，同时也是JavaEE标准。JSP与Servlet一样，是在服务器端执行的。

常见动态网页开发技术对比：

- JSP: Java平台，安全性高，适合开发大型的，企业级的Web应用程序
- Asp.net: .Net平台，简单易学。但是安全性以及跨平台性差
- PHP: 简单，高效，成本低开发周期短，特别适合中小型企业的Web应用开发

2.1 JSP页面元素

JSP页面元素构成如下图所示：



JSP指令有如下三种：

1. page指令：通常位于JSP页面的顶端，同一个页面可以有多个page指令。
2. include指令：将一个外部文件嵌入到当前JSP文件中，同时解析这个页面中的JSP语句。
3. taglib指令：使用标签库定义新的自定义标签，在JSP页面中启用定制行为。

2.2 JSP注释

JSP页面支持以下三种注释：

- HTML的注释：HTML注释在客户端可见

```
<!-- html注释 -->
```

- JSP注释：JSP注释在客户端不可见

```
<%-- JSP注释 --%>
```

- JSP脚本注释：JSP脚本注释同样也在客户端不可见

```
// 单行注释  
/**/ 多行注释
```

2.3 JSP脚本

JSP脚本指的就是在JSP页面中执行的java代码。

语法: `<% Java代码 %>`

在`<% Java代码 %>` 这一标签内的可执行java代码我们就称之为是JSP脚本。

2.4 JSP声明

JSP声明指的就是在JSP页面中定义变量或者方法

语法:

`<%! Java代码 %>`

范例: 使用JSP声明

```
<%!  
    String s = "yuisama";  
    int add(int x, int y) {  
        return x + y;  
    }  
>%  
<%  
    out.println(s);  
    out.println(add(3, 4));  
>%
```

2.5 JSP表达式

在JSP页面中执行的表达式

语法:

`<!-- 注意: JSP表达式不以分号结尾。-->`

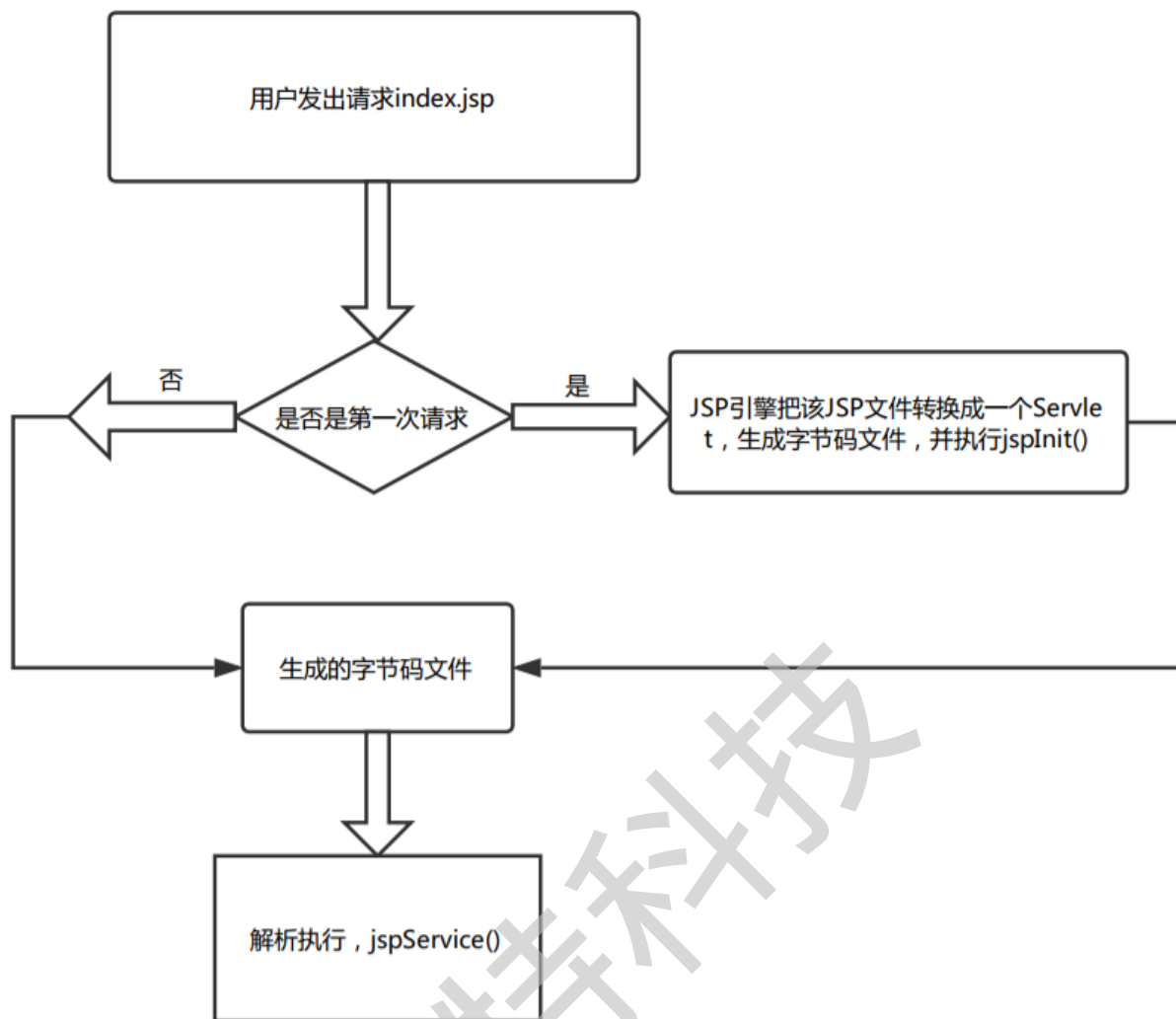
`<% =表达式 %>`

范例: 使用JSP表达式

```
<%!  
    String s = "yuisama";  
    int add(int x, int y) {  
        return x + y;  
    }  
>%  
<br>  
Hello, <%= s%> <br>  
x+y=<%= add(10,20)%> <br>
```

2.6 JSP生命周期

下面我们来对JSP页面的生命周期做一个详解, 假设现在用户要访问index.jsp, 则处理流程如下:



jspService()方法被调用来处理客户端的请求。对每一个请求，JSP引擎创建一个新的线程来处理该请求。如果有多个客户端同时请求该JSP文件，则JSP引擎会创建多个线程。每个客户端请求对应一个线程。以多线程方式执行可以大大降低对系统的资源需求，提高系统的并发量以及相应时间，但也要注意多线程带来的同步问题。由于该Servlet常驻于内存，所以响应非常快。

JSP页面只要发生修改，那么JSP引擎都会把源文件重新编译，生成最新的字节码文件。

备注：查看Tomcat的安装目录的work目录中JSP编译后的源代码

2.7 阶段项目

需求：分别用表达式和脚本打印九九乘法表

范例：使用表达式实现输出九九乘法表

```
<%@page contentType="text/html; charset=utf-8" %>
<html>
<body>
<%!
    String printMathTable() {
        String result = "";
        for (int i = 1; i <=9; i++) {
```

[illegible]

范例：使用脚本输出九九乘法表

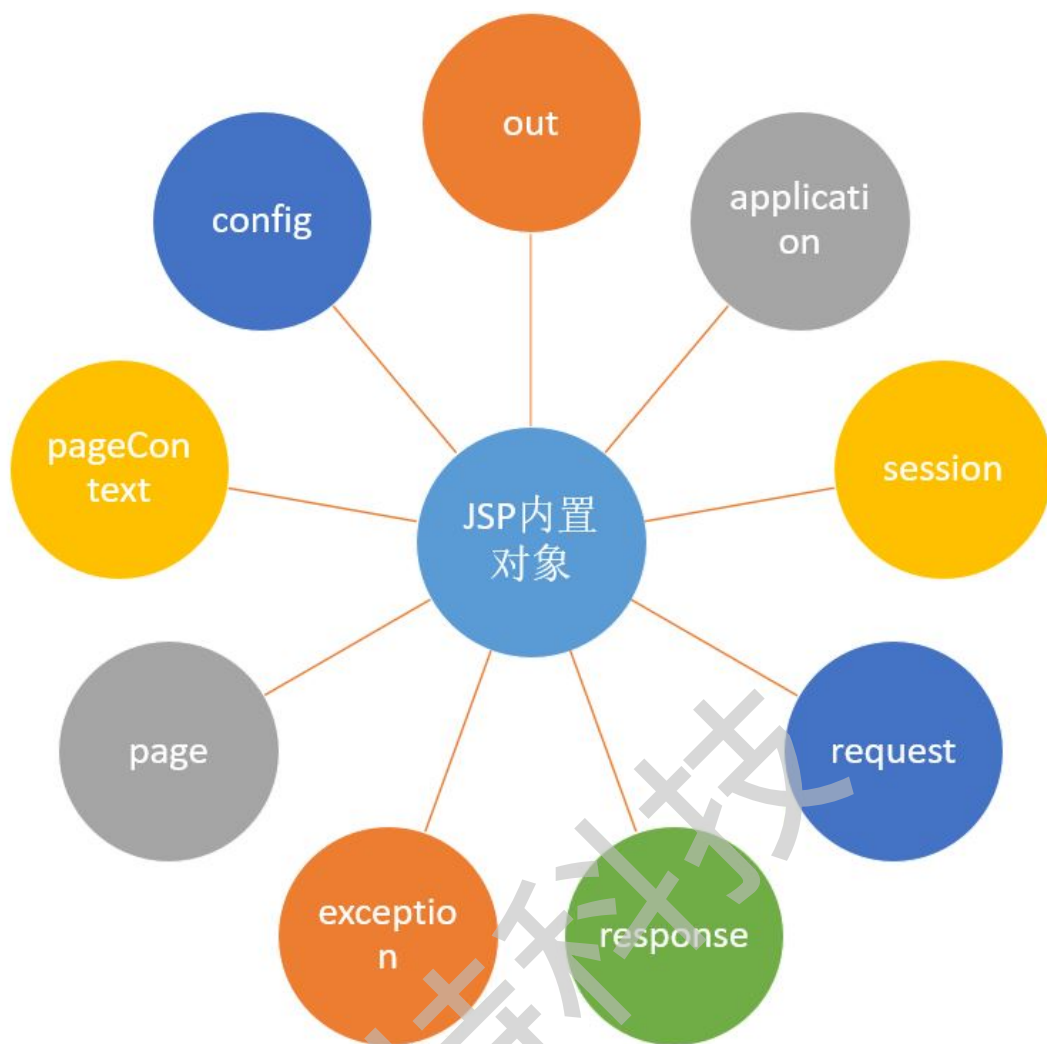
```
<%@page contentType="text/html"; charset=utf-8" %>
<html>
<body>
<%!
    void printMathTable(JspWriter out) throws Exception{
        for (int i = 1; i <=9; i++) {
            for (int j = 1; j <=i ; j++) {
                out.println(i+"*"+j+"="+i*j)+"&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;";
            }
            out.println("<br>");
        }
    }
%>
<h1>九九乘法表</h1>
<hr>
<%
    // 使用脚本调用声明的方法
    printMathTable(out);
%>
</body>
</html>
```

3.JSP内置对象

3.1 内置对象简介

JSP内置对象是Web容器创建的一组对象，不使用关键字**new**就可以使用的对象，我们称之为内置对象。

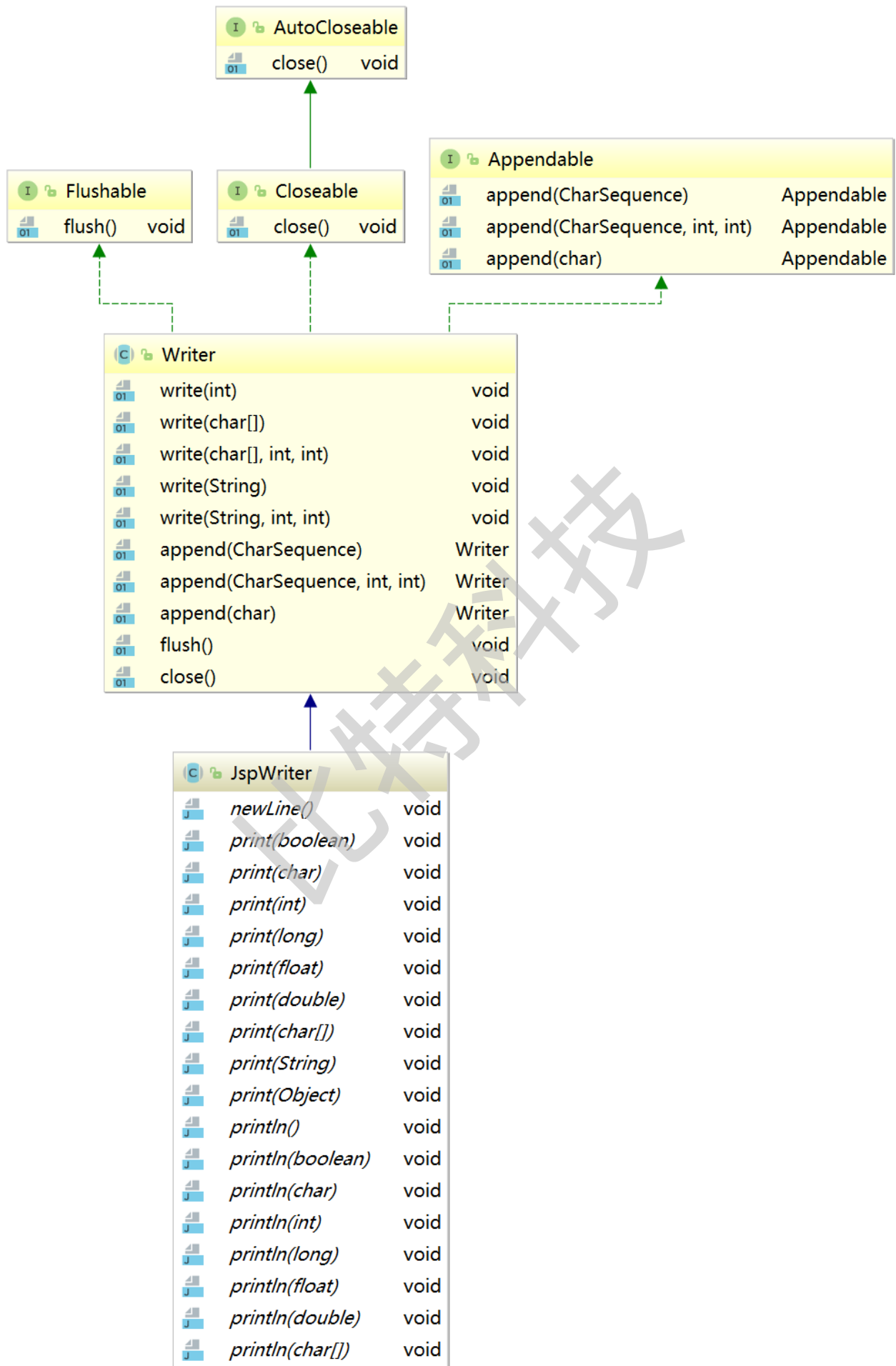
JSP九大内置对象：












3.2 out对象

在讲解out对象之前，我们先来看一下什么是缓冲区，**缓冲区(Buffer)**：所谓缓冲区就是内存的一块区域，用来保存临时数据。

out对象：是JspWriter类的实例，是客户端输出内容常用的对象。



	<code>println(String)</code>	void
	<code>println(Object)</code>	void
	<code>clear()</code>	void
	<code>clearBuffer()</code>	void
	<code>flush()</code>	void
	<code>close()</code>	void
	<code>getBufferSize()</code>	int
	<code>getRemaining()</code>	int
	<code>isAutoFlush()</code>	boolean

Powered by yFiles

常用方法如下：

1. void `println()` 向客户端打印字符串
2. void `clear()` 清除缓冲区的内容，如果在`flush`之后调用会抛异常。
3. void `clearBuffer()`：清除缓存区内容，在`flush`之后调用不会抛异常。
4. void `flush()`：将缓冲区内容输出到客户端。
5. int `getBufferSize()`：返回缓冲区字节大小，如果没有缓冲区，则为0
6. int `getRemaining()`：返回缓冲区还剩余多少字节可用
7. boolean `isAutoFlush()`：返回缓冲区满时，是自动清空还是抛出异常。
8. void `close()`：关闭输出流。

范例：使用out对象进行输出

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
</head>
<body>
<%
    out.println("<h2>咏菊</h2>");
    out.println("待到秋来九月八<br>");
    out.println("我花开时百花杀<br>");
    out.println("冲天香阵透长安<br>");
    out.println("满城尽带黄金甲<br>");
%>
缓冲区大小：<%=out.getBufferSize()%>byte<br>
缓冲区剩余大小：<%=out.getRemaining()%>byte<br>
是否自动清空缓冲区：<%=out.isAutoFlush()%><br>
</body>
</html>
```

3.3 get与post区别

```
<form name="regForm" action="动作" method="提交方式">

</form>
```

表单有两种提交方式：get和post

- 1.**get**: 以明文的方式通过URL提交数据, 数据在URL中可以看到。提交的数据最多不超过2KB。安全性较低但效率比post方式高。适合提交数据量不大、安全性不高的属性。比如搜索、查询等功能。
- 2.**post**: 将用户提交的信息封装在HTTP Body内。适合提交数据量大, 安全性高的用户信息。比如: 注册、修改、上传等功能。

范例: 观察get与post区别

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>login</title>
</head>
<body>
    <h1>用户登录</h1>
    <hr>
    <form action="dologin.jsp" name="loginForm" method="get">
        <table>
            <tr>
                <td>用户名: </td>
                <td><input type="text" name="userName"></td>
            </tr>
            <tr>
                <td>密码: </td>
                <td><input type="password" name="password"></td>
            </tr>
            <tr>
                <td colspan="2"><input type="submit" value="登录"></td>
            </tr>
        </table>
    </form>
</body>
</html>
```

3.4 request对象

客户端的请求信息被封装在request对象中, 通过这个对象才能了解到客户的需求, 然后做出响应。request是HttpServletRequest类的实例。request对象具有请求域, 即完成客户端的请求之前, 该对象一直有效。常用方法参见API: `javax.servlet.http.HttpServletRequest`

范例: 观察getParameter与getParameterValues方法使用:

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>login</title>
</head>
<body>
    <h1>用户登录</h1>
    <hr>
    <form action="dologin.jsp" name="loginForm" method="post">
        <table>
            <tr>
```

```

        <td>用户名: </td>
        <td><input type="text" name="userName"></td>
    </tr>
    <tr>
        <td>密码: </td>
        <td><input type="password" name="password"></td>
    </tr>
    <tr>
        <td>掌握语言: </td>
        <td>
            <input type="checkbox" name="skill" value="C语言">C语言
            <input type="checkbox" name="skill" value="Java">Java
            <input type="checkbox" name="skill" value="C++">C++
            <input type="checkbox" name="skill" value="Python">Python
        </td>
    </tr>
    <tr>
        <td colspan="3"><input type="submit" value="登录"></td>
    </tr>
</table>
</form>
</body>
</html>

```

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>request内置对象</title>
</head>
<body>
<h1>request内置对象</h1>
用户名: <%=request.getParameter("userName")%>
爱好: <%
    String[] skills = request.getParameterValues("skill") ;
    for(String s:skills){
        out.println(s+"&nbsp;&nbsp;&nbsp;");
    }
%>
<hr>
</body>
</html>

```

注意：表单提交时默认采用的编码是ISO-8859-1字符集，显示中文会乱码。

范例：使用setCharacterEncoding设置请求的字符编码

```

<%
    // 解决中文乱码问题,注意此字符集需要和提交页面字符集保持一致
    request.setCharacterEncoding("UTF-8");
%>

```

以上是通过post方法提交表单来传参，实际也可通过get方法，在URL中获取参数。

范例：获取get方法中URL带的参数

```
<a href="dologin.jsp?userName=yuisama&skill=Java C Python">测试get方法传参</a>
```

注意：上述通过URL传参无法通过request.setCharacterEncoding方式解决中文乱码问题，此时要想解决此类乱码问题，需要修改tomcat配置文件server.xml

范例：修改Tomcat配置文件解决URL传参的中文乱码问题

打开server.xml 在此行代码下方增加

```
<!--URI编码-->
<Connector port="80" protocol="HTTP/1.1"
    connectionTimeout="20000"
    redirectPort="8443"
    URIEncoding="UTF-8"
/>
```

URIEncoding="UTF-8"，就可以设置URL传参的编码来解决中文乱码问题。

范例：使用setAttribute与getAttribute设置与取出属性

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>request内置对象</title>
</head>
<body>
<h1>request内置对象</h1>
<%
    // 解决中文乱码问题,注意此字符集需要和提交页面字符集保持一致
    request.setCharacterEncoding("UTF-8");
    // 使用setAttribute设置属性
    request.setAttribute("password", "test");
%>
用户名: <%=request.getParameter("userName")%>
爱好: <%
    String[] skills = request.getParameterValues("skill") ;
    for(String s:skills){
        out.println(s+"&nbsp;&nbsp;&nbsp;");
    }
%> <br>
密码: <%=request.getParameter("password")%>
<hr>
</body>
</html>
```

范例：request对象的其他常用方法

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

```

<html>
<head>
    <title>request内置对象</title>
</head>
<body>
<h1>request内置对象</h1>
<%
    // 解决中文乱码问题,注意此字符集需要和提交页面字符集保持一致
    request.setCharacterEncoding("UTF-8");
    // 使用setAttribute设置属性
    request.setAttribute("password","test");
%>
用户名: <%=request.getParameter("userName")%>
爱好: <%
    String[] skills = request.getParameterValues("skill") ;
    for(String s:skills){
        out.println(s+"&nbsp;&nbsp;&nbsp;");
    }
%> <br>
密码 : <%=request.getAttribute("password")%><br>
请求体的MIME类型 : <%=request.getContentType()%><br>
协议类型以及版本号 : <%=request.getProtocol()%><br>
服务器主机名 : <%=request.getServerName()%><br>
服务器端口号 : <%=request.getServerPort()%><br>
请求体大小 : <%=request.getContentLength()%><br>
请求客户端的IP地址 : <%=request.getRemoteAddr()%><br>
请求的真实路径 : <%=request.getRealPath("doLogin.jsp")%><br>
请求的上下文路径 : <%=request.getContextPath()%><br>
<hr>
</body>
</html>

```

3.5 response对象

response对象包含了响应客户请求的有关信息，但在JSP中很少直接使用到它。它是HttpServletResponse类的实例。response对象具有页面作用域，即访问一个页面时，该页面内的response对象只能对这次访问有效，其他页面的response对象对当前页面无效。常用方法参见API：`javax.servlet.http.HttpServletResponse`

范例：response对象的常用方法使用

```

<%@ page import="java.io.PrintWriter" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%
    // 设置响应的MIME类型
    response.setContentType("text/html; charset=UTF-8");
    out.println("<h1>response内置对象</h1>");
    out.println("<hr>");
    // out.flush();
    // 获得输出流对象
    PrintWriter out = response.getWriter();
    out.println("此行语句是由输出流打印");
%>

```

response对象获得的PrintWriter输出流打印的内容总是优先于内置out对象打印的内容，如果要想解决此问题，可以采用out.flush()方法清空out对象的缓冲区

3.6 转发与重定向

请求重定向：

客户端行为，response.sendRedirect()，从本质上讲相当于两次请求，前一次请求对象不会保存，地址栏的URL地址会改变。

请求转发：

服务器行为，request.getRequestDispatcher().forward(req, resp)；本质上是一次请求，转发后请求对象会保存，地址栏的URL地址不会改变。

范例：观察请求重定向与请求转发

```
// 请求重定向
response.sendRedirect("dologin.jsp");
// 请求转发
request.getRequestDispatcher("dologin.jsp").forward(request, response);
```

3.7 session对象

- 什么是session?
 - session表示客户端与服务器的一个会话
 - Web中的session指的是用户浏览某个网站时，从进入网站到浏览器关闭所经过的这段时间，也就是用户浏览这个网站所花费的时间
 - session实际上是一个特定的时间概念
 - 在服务器的内存中保存着不同用户的session
- session对象
 - session对象是一个JSP的内置对象
 - session对象在第一个JSP页面被装载时自动创建，完成会话期管理。
 - 从一个客户打开浏览器并连接到服务器开始，到客户关闭浏览器离开这个服务器结束，被称为一个会话。
 - 当一个客户访问一个服务器时，可能会在服务器的几个页面之间切换，服务器应当通过某种办法知道这是一个客户，就需要session对象。
- session对象是HttpSession类的实例

session对象常用方法参见API：`javax.servlet.http.HttpSession`

范例：观察session对象的常用操作

session_page1.jsp

```
<%@ page import="java.text.SimpleDateFormat" %>
<%@ page import="java.util.Date" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
```


创建:

当客户端第一次访问某个jsp或者Servlet的时候,服务器会为当前会话创建一个SessionId,每次客户端向服务端发送请求时,都会将此SessionId携带过去,服务端会对此SessionId进行校验。

活动:

- 某次会话当中通过超链接打开的新页面属于同一次会话。
- 只要当前会话页面没有全部关闭,重新打开新的浏览器窗口访问同一项目资源时属于同一次会话。
- 除非本次会话的所有页面关闭后再重新访问某个jsp或者Servlet,才会创建新的会话。
- **注意!!!!!!:** 原有会话还存在,这个旧的SessionId仍然存在于服务器端,只不过再也没有客户端会携带他然后交予服务端校验。

销毁:

Session的销毁只有三种方式:

- 调用了session.invalidate()方法
- Session过期(超时)
- 服务器重新启动

范例:观察session生命周期的活动与销毁阶段

Tomcat默认session超时时间为30分钟

设置Session超时时间有两种方式:

1. 调用session.setMaxInactiveInterval(时间) // 单位是秒
2. 在项目的web.xml中配置(以分为单位)

```
<session-config>
<session-timeout>
    10
</session-timeout>
</session-config>
```

3.8 application对象

- application对象
 - application对象实现了用户间数据的共享,可存放全局变量。
 - application开始于服务器的启动,终止于服务器的关闭。
 - 在用户的前后连接或不同用户的连接中,可以对application对象的同一属性进行操作。
 - 在任何地方对application对象属性的操作,都将影响其他用户对此的访问。
 - application对象是ServletContext类的实例。
- application对象的常用方法参见API: `javax.servlet.ServletContext`

范例:使用application对象

```
<%@ page import="java.util.Enumeration" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<meta charset="UTF-8">
```

```

</html>
<body>
  <h1>application内置对象</h1>
  <%
    application.setAttribute("city","Xi'An");
    application.setAttribute("Company","Bitekeji");
    application.setAttribute("postmain",710000);
  %>
  所在城市为:<%=application.getAttribute("city")%><br>
  application中的属性有:
  <%
    Enumeration<String> enumeration = application.getAttributeNames();
    while (enumeration.hasMoreElements()) {
      out.println(enumeration.nextElement()+"&nbsp;&nbsp;&nbsp;");
    }
  %><br>
  JSP(Servlet)引擎名与版本号: <%=application.getServerInfo()%><br>
</body>

```

3.9 page对象

page对象就是指向当前JSP页面本身，有点像SE中的this指针，它是java.lang.Object类的实例。常用方法与Object类中的常用方法完全一致。

范例：使用page的toString方法

```

<%@ page import="java.util.Enumeration" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<meta charset="UTF-8">
</html>
<body>
  <h1>page内置对象</h1>
  page对象的toString(): <%=page.toString()%>
</body>

```

3.10 pageContext对象

- pageContext对象
 - pageContext对象提供了对JSP页面内所有的对象及名字空间的访问。
 - pageContext对象可以访问到本页面所在的session，application的某一属性值。
- pageContext常用方法参见API: `javax.servlet.jsp.PageContext` 位于 `jsp-api`包中。

范例：使用pageContext的几个常用方法

```

<%@ page import="java.util.Enumeration" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<meta charset="UTF-8">
</html>
<body>

```



```

<h1>page内置对象</h1>
<%
    session.setAttribute("userName","zhangxiao");
%>
用户名为:<%=pageContext.getSession().getAttribute("userName")%>
<hr>
<%
//          // 跳转到新的页面
//          pageContext.forward("login.jsp");
//          pageContext.include("include.jsp");
%>
</body>

```

include.jsp内容:

```

<%@ page import="java.util.Date" %>
<%@ page import="java.text.SimpleDateFormat" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%
    Date date = new Date();
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy年MM月dd日");
    String str = sdf.format(date);
    out.println(str+"<br>");
%>

```

3.11 exception对象

exception对象是一个异常对象，当一个页面在运行过程中发生了异常，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置为true，否则无法编译。exception对象实际上是java.lang.Throwable的对象。

常用方法如下:

- String getMessage()返回描述异常的信息。
- String toString()返回关于异常的简短描述信息
- void printStackTrace()显示异常及其堆栈
- Throwable FillInStackTrace()重写异常的执行堆栈

首先创建错误页面，指定errorPage。

```

<%@ page import="java.util.Enumeration" %>
<%@ page contentType="text/html; charset=UTF-8" language="java"
errorPage="exception.jsp" %>
<html>
<meta charset="UTF-8">
</html>
<body>
<h1>测试exception对象</h1>
<%
    // 抛出异常
    System.out.println(10/0);
%>
</body>

```

然后在指定的错误页面将isErrorPage置为true

```
<%@ page contentType="text/html; charset=UTF-8" language="java" isErrorPage="true" %>
<html>
<meta charset="UTF-8">
</html>
<body>
<h1>exception内置对象</h1>
异常消息为:<%=exception.getMessage()%><br>
异常消息的字符串表示为<%=exception.toString()%><br>
</body>
```

4. JSP指令与动作元素

4.1 include

include指令语法:

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

先创建一个要包含的页面date.jsp

```
<%@ page import="java.util.Date" %>
<%@ page import="java.text.SimpleDateFormat" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%
    Date date = new Date();
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy年MM月dd日");
    String str = sdf.format(date);
    out.println(str);
%>
```

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<meta charset="UTF-8">
</html>
<body>
<h1>include指令</h1>
<hr>
<%@include file="date.jsp"%>
</body>
```

include动作语法:

```
<jsp:include page="URL" flush="true|false"/>
```

其中page为要包含的页面, flush表示被包含的页面是否要从缓冲区读取

范例:使用include动作

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<meta charset="UTF-8">
</html>
<body>
<h1>include动作</h1>
<hr>
<jsp:include page="date.jsp" flush="false"/>
</body>

```

4.2 forward动作

语法:

```
<jsp:forward page="URL"/>
```

等同于

```
request.getRequestDispatcher("/url").forward(request, response);
```

创建一个用户登录页面

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>login</title>
</head>
<body>
<h1>用户登录</h1>
<hr>
<form action="/forward_action.jsp" method="post">
    <table>
        <tr>
            <td>用户名: </td>
            <td><input type="text" name="userName"></td>
        </tr>
        <tr>
            <td>密码: </td>
            <td><input type="password" name="password"></td>
        </tr>
        <tr>
            <td colspan="3"><input type="submit" value="登录"></td>
        </tr>
    </table>
</form>
</body>
</html>

```

forward_action.jsp如下:

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>login</title>
</head>
<body>
<h1>用户登录</h1>
<hr>
<jsp:forward page="user.jsp"/>
</body>
</html>
```

user.jsp如下:

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
</head>
<body>
<h1>用户信息</h1>
<hr>
<%
    String userName = "";
    String password = "";
    if (request.getParameter("userName")!=null) {
        userName = request.getParameter("userName");
    }
    if (request.getParameter("password")!=null) {
        password = request.getParameter("password");
    }
%>
用户名: <%=userName%><br>
密码: <%=password%><br>

</body>
</html>
```

4.3 param动作

语法:

```
<jsp:param name="参数名" value="参数值">
```

常常与[jsp:forward](#)一起使用, 作为其子标签; 修改上述forward_action.jsp, 添加param动作如下:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>param动作</title>
</head>
<body>
<hr>
    <% request.setCharacterEncoding("UTF-8"); %>
    <jsp:forward page="user.jsp">
        <jsp:param name="email" value="zhangxiao@bitekeji.com"/>
    </jsp:forward>
</body>
</html>
```

修改user.jsp来获取param动作中新增的参数，如下：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
</head>
<body>
<h1>用户信息</h1>
<hr>
<%
    request.setCharacterEncoding("utf-8");
    String userName = "";
    String password = "";
    String email = "";
    if (request.getParameter("userName")!=null) {
        userName = request.getParameter("userName");
    }
    if (request.getParameter("password")!=null) {
        password = request.getParameter("password");
    }
    if (request.getParameter("email")!=null) {
        email = request.getParameter("email");
    }
%>
用户名:<%=userName%><br>
密码:<%=password%><br>
电子邮箱:<%=email%><br>
</body>
</html>
```

总结

知识块	知识点	分类	掌握程度
JSP	1.JSP基本语法 2.JSP内置对象	实战型	掌握
JSP指令和动作	1. include, forward, param	实战型	掌握

比特科技