JSP

JSP是什么

- Java Server Page
- 基于Java的服务器端页面
- Java规范之一

JSP存放位置

- WEB-INF之中，能够保护数据
- web目录下，作为欢迎界面index.jsp

执行原理

- 浏览器访问路径虽然以.jsp结尾，访问jsp文件，实际上底层执行的是jsp对应的Java程序
- JSP文件 -Tomcat服务器-> index_jsp.java -Tomcat服务器-> index_jsp.class
- Tomcat服务器内置了JSP翻译引擎，专门负责解析翻译JSP文件
- 实际上index_jsp类是一个Servlet，里面包含有service()方法，jsp很多东西也是解析到方法里面
- 生命周期和Servlet一样
- 第一次访问JSP会比较慢，因为要进行翻译等工作 ，之后直接调用service()方法。同时他也是在单实例多线程环境下运行的，所以我们 部署了jsp项目之后，先全部访问一遍，能够大大提升客户体验

jsp文件第一次访问的时候为什么非常慢?

- 启动JSP翻译引擎
- 需要一个翻译的过程
- 需要一个编译的过程
- 需要Servlet对象的创建过程
- init方法调用
- service方法调用.....

为什么第2+次访问JSP的时候非常快?

- 不需要重新翻译

- 不需要重新编译
- 不需要创建Servlet对象
- 直接调用Servlet对象的service方法

jsp文件在什么时候会被重新翻译?

- jsp文件被修改之后会被重新的翻译
- 怎么确定jsp文件修改了呢? Tomcat服务器会记录jsp文件的最后修改时间。

九大内置对象

内置对象	完整类名	作用范围
pageContext	javax.servlet.jsp.PageContext	页面范围【页面上下文】
request	javax.servlet.http.HttpServletRequest	请求范围
session	javax.servlet.http.HttpSession	会话范围
application	javax.servlet.ServletContext	应用范围
response	javax.servlet.http.HttpServletResponse	响应对象
out	javax.servlet.jsp.JspWriter	标准输出流
config	javax.servlet.ServletConfig	Servlet配置信息对象
exception	java.lang.Throwable	异常引用(isErrorPage="true")
page	javax.servlet.http.HttpServlet (page = this)	很少用

需要注意：九大内置对象只能在service方法里面直接用，其它方法想要使用只能在service方法里面调用并传参

JSP中的四个作用域对象/范围对象

pageContext < request < session < application

pageContext：在同一个JSP页面中共享数据，不能跨JSP页面
request： 在同一个请求中共享数据 【使用较多】
session：在同一个会话中共享数据 【使用较多】
application：所有用户共享的数据可以放到应用范围中

测试

```
<%--index.jsp--%>
<%@page contentType="text/html; charset=UTF-8"%>
<%
```

```
    pageContext.setAttribute("pageContext" , "pageContextData");
    request.setAttribute("request","requestData");
    session.setAttribute("session","sessionData");
    application.setAttribute("application","applicationData");
%>

<%=pageContext.getAttribute("pageContext") %>
<br>
<%=request.getAttribute("request") %>
<br>
<%=session.getAttribute("session") %>
<br>
<%=application.getAttribute("application") %>


<%--
<jsp:forward page="/index2.jsp"></jsp:forward>
 --%>

<%
    response.sendRedirect(request.getContextPath() + "/index2.jsp");
%>

<%--index2.jsp--%>
<%@page contentType="text/html; charset=UTF-8"%>

<%=pageContext.getAttribute("pageContext") %>
<br>
<%=request.getAttribute("request") %>
<br>
<%=session.getAttribute("session") %>
<br>
<%=application.getAttribute("application") %>

<%--
    pageContext只能在同一个JSP页面中共享数据。范围是最小的。
    通过pageContext这个页面上下文对象，可以获取当前页面中的其它对象。(对Java开发程序员没什么用)
--%>
<%=pageContext.getRequest() %>
<%=pageContext.getSession() %>
<%=pageContext.getServletContext() %>
<%=pageContext.getServletConfig() %>
```

语法解析

注释

语法：<%--注释内容--%>

位于该便签内的语句会被JSP翻译引擎忽略，不进行翻译

前端语句

- 在JSP之中，HTML,CSS,JS等被当成字符串输出，out.print("")

小脚本 scriptlet

原理：在小脚本里面的语句会被解析后，直接添加到index_jsp类的service方法里面，所以可以在里面编写Java语句，数量没有限制，会按照JSP之中编写的顺序添加到service方法里面，需要注意顺序

语法：

```
<%
    Java语句；// 是完整的Java语句，需要以分号结尾
    Java语句；
    Java语句；
    %>
```

注意事项：因为小脚本的Java语句直接翻译到service方法之中，所有不能在小脚本里面使用范围修饰符，定义成员变量，定义函数，编写静态或者动态代码块

声明 declaration

原理：放在index_jsp类的类体之中，可以直接定义成员变量、定义函数，编写静态或者动态代码块

语法：

```
<%!
    static{
        System.out.println("class is loaded!");
    }
    {
        System.out.println("新建了一个类对象！");
    }
    private int 成员;
    public void doSome(){
        System.out.println("doSome...");
    }
%>
```

表达式

原理：out.print();的简化写法，直接将里面内容放到()之中

语法：

```
<%="Hello,World!"%>    --->  out.print("Hello,World!");
<%=1+1%>               --->  out.print(1+1);
<%=request%>           --->  out.print(request);
```

指令

作用：引导JSP的翻译引擎如何进行JSP翻译

语法：<%@指令名 属性名=属性值 属性名=属性值 属性名=属性值%>

page指令常用的属性

- contentType　设置JSP的响应内容类型，同时在响应的内容类型后面也可以指定响应的字符编码方式
- pageEncoding　　设置JSP响应时的字符编码方式(可以在contentType之中设置)
- import　　组织导入
- session　　设置当前JSP页面中是否可以直接使用session内置对象
- errorPage　　错误页面
- isErrorPage　　是否是错误页面
- isELIgnored　　是否忽略EL表达式【后期讲】

contentType、pageEncoding、import

```
<%@page contentType="text/html" %>
<%@page pageEncoding="UTF-8" %>
<%--
    上面两条语句等价于
    <%@page contentType="text/html;charset=UTF-8" %>
--%>
<%@page import="java.util.Date" %>
<%--
    可以在一次import之中导入多个类，类与类之间使用逗号,分隔
--%>
<%@page import="java.util.*,java.text.SimpleDateFormat" %>

<%
    Date nowTime = new Date();
%>

<%=nowTime %>
<br>
<%
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
%>
```

```
<%=sdf.format(nowTime) %>
```

errorPage

```
<%@page contentType="text/html; charset=UTF-8" errorPage="/error.jsp"%>

<%--
    关于page指令的errorPage属性:
        当前JSP页面出错之后，要跳转的页面路径，需要使用该属性指定。
--%>
<%
    String s = null;
    s.toString();
%>
```

isErrorPage

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

<html>
    <head>
        <title>错误页面</title>
    </head>
    <body>
        <img src="/prj-jsp-05/images/error.gif" />
    </body>
</html>

<%--
    关于page指令中的isErrorPage属性:
        - isErrorPage = "false" 表示内置对象exception无法使用【缺省情况下是false】
        - isErrorPage = "true"  表示内置对象exception可以使用
 --%>
<%-- 使用内置对象exception打印异常堆栈追踪信息 --%>
<%-- exception引用指向了抛出的异常 --%>
<%
    exception.printStackTrace();
%>
```

session

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

```
<%=session %>
 --%>

<%--
    关于page指令中的session属性：
        - session="true"
            * 表示在当前JSP中可以直接使用内置对象session
            * 程序执行的时候获取当前的session会话对象，若获取不到则新建session对象

        - session="false"
            * 表示在当前JSP中不能直接使用内置对象session
            * 但是有一些业务可能要求在当前JSP页面中获取当前的session对象，没有获取到则不新建session对象，此时需要编写以下程序

        - 若session这个属性没有指定，默认值就是session="true"

--%>
<%@page contentType="text/html; charset=UTF-8" session="false"%>
<%
    HttpSession session = request.getSession(false);
%>
<%=session%>
```

include指令

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

<html>
    <head>
        <title>include指令</title>
    </head>
    <body>
<%--
关于include指令：
    1、a.jsp可以将b.jsp包含进来，当然被包含的资源不一定是jsp，也可能是其它的网络资源
    2、include作用：
        在网页中有一些主体框架，例如：网页头、网页脚，这些都是固定不变的，
        我们可以将网页头、网页脚等固定不变的单独编写到某个JSP文件中，
        在需要页面使用include指令包含进来。
        优点：
            代码量少了
            便于维护【修改一个文件就可以作用于所有的页面】
    3、在一个jsp中可以使用多个include指令
    4、include实现原理：
        4.1 编译期包含
        4.2 a.jsp包含b.jsp，底层共生成一个java源文件，一个class字节码文件,翻译期包含/编译期包含/静态联编

    5、静态联编的时候，多个jsp中可以共享同一个局部变量，同时变量不能重名。
    因为最终翻译之后service方法只有一个。
--%>
        <%@include file="/index2.jsp" %>
```

```
        <%=i%>
    </body>
</html>
```

动作

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

<%--
    关于JSP中的动作:
        语法格式: <jsp:动作名   属性名=属性值    属性名=属性值....></jsp:动作名>
--%>
<%
    request.setAttribute("username","wangwu");
%>

<%-- 转发是一次请求 --%>
<%--
<jsp:forward page="/index2.jsp"></jsp:forward>
 --%>

<%-- 以上JSP的动作可以编写为对应的java程序进行实现 --%>
<%--
<%
    request.getRequestDispatcher("/index2.jsp").forward(request,response);
%>
 --%>

<%-- 编写java程序完成重定向 --%>
<%
    response.sendRedirect(request.getContextPath() + "/index2.jsp");
%>

<%--
    JSP主要是完成页面的展示，最好在JSP文件中少的编写java源代码:
        以后会通过EL表达式 + JSTL标签库来代替JSP中的java源代码
        当然，使用某些动作也能代替java源代码
 --%>
```

include动作

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

<html>
    <head>
        <title>include动作</title>
```

```
    </head>
    <body>
<%--
    关于JSP中的include动作：
        1、a.jsp包含b.jsp，底层会分别生成两个java源文件，两个class字节码文件
        2、编译阶段并没有包含，编译阶段是两个独立的class字节码文件，生成两个Servlet，两个独立的service方法
        3、使用include动作属于运行阶段包含， 实际上是在运行阶段a中的service方法调用了b中的service方法，达到了包含效果
        4、a.jsp包含b.jsp，若两个jsp文件中有重名的变量，只能使用动态包含。其余都可以使用静态包含。
        5、include动作完成的动态包含，被称为动态联编。
--%>
        <jsp:include page="/b.jsp"></jsp:include>
    </body>
</html>
```