# JSP运行原理：

1. 客户端访问hello.jsp

2. 服务器读取hello.jsp内容到内存

3. 服务器根据hello.jsp内容生成Servlet程序 //保存在 tomcat/work

4. Servlet编译运行

JSP翻译Servlet :

* 固定包名 ：org.apache.jsp
* 文件名:hello.jsp ---- hello\_jsp.java

# JSP中脚本元素

1. 声明 <%! %> //定义内容将会被翻译Servlet类 成员变量和方法

2. 表达式 <%= %> //用于向页面输出内容,等价于<& out.print()%>

3. 代码块<% %> //在<%%> 之间编写任何java代码 ,代码块可以和html嵌套使用

JSP第一次运行时,总要翻译Servlet ,效率低.以后访问,如果没有修改JSP不会重新翻译

# JSP指令(JSP Directive)

page指令

include指令

taglib 指令

# page指令

1. language 只能为java

2. extends JSP翻译后Servlet 继承父类，不许为Servlet

3. session 定义JSP中是否可以直接使用Session隐含对象,默认为true

在编译成servlet后,生成如下代码;

HttpSession session = null;

session = pageContext.getSession();

4. import 完成 JSP翻译后 Servlet 导包

5. buffer/autoFlush 设置out隐含对象属性

buffer 设置缓冲区大小

autoFlush 设置当缓冲区满后，自动刷出

6. isELIgnored 设置JSP是否忽略执行EL表达式 //默认值false ,不忽略

7. 通过contentType和pageEncoding 设置 JSP页面编码

pageEncoding保存到硬盘上的编码

contentType 在Servlet生成HTML传递浏览器采用编码response.setContentType()

8. 通过errorPage和isErrorPage控制JSP页面发生错误时跳转

<%@ page isErrorPage=”true”>

errorPage指定 在页面发生错误跳转到哪个页面

# include指令(静态包含某个页面)

静态包含: <% @ include file="/day/hello.jsp" %>

* 静态包含多个jsp文件,都只翻译成一个servlet文件
* include包含目标页面整个内容,被包含被荣只需要填写html到吗片段.

# taglib指令 ，在jsp页面引用标签库文件

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

uri ---- 定义标签 唯一命名空间

prefixt ---- 命名空间前缀

引用jstl时，在导入jstl.jar中META-INF/c.tld

<short-name>c</short-name> -------- 就是prefix属性

<uri>http://java.sun.com/jsp/jstl/core</uri> ----- 就是uri属性

# 管理HTTP协议会话状态 ：Cookie 和Session

# Cookie对象

浏览器一般只允许存放300个Cookie, 每个站点最多存放20个Cookie,每个Cookie的大小限制为4KB。

Cookie API:

Cookie mCookie = **new** Cookie(String name, String value);//创建cookie

// cookie,name 不允许改变,写入到客户端文件,不能更改.

mCookie.setPath("/dayo7"); //设置Cookie的路径,客户端发送mCookie.setMaxAge(60\*60); //设置cookie的有效期,不设置关闭浏览器,自动清除

mCookie.setMaxAge(0); //立即清除cookoe.例如清除商品浏览记录

mCookie.setDomain(".baidu.com"); //设置第三方cookie,浏览器不启用,存在安全问题

response.addCookie(mCookie);//写入cookie到客户端

Cookie []cookies = request.getCookies();//获取客户端携带的cookie

**//案例1 :服务器返回客户端上次访问的时间**

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

response.setCharacterEncoding("utf-8");

Cookie []cookies = request.getCookies();

**if** (cookies==**null**) {

**long** now = System.*currentTimeMillis*();

Cookie mCookie = **new** Cookie("last", now+"");

response.addCookie(mCookie);

}**else** {

**for** (Cookie cookie:cookies){

**if** (cookie.getName().equals("last")) {

Long time = Long.*parseLong*(cookie.getValue());

Date date = **new** Date(time);

DateFormat format = **new** SimpleDateFormat

("yyyy年MM月dd日 : HH时mm分ss秒");

response.getWriter().print("上次访问时"+format.format(date));

}

}

}

……//重新写入

}

**//案例2: 浏览商品记录,清除商品记录,添加商品到购物车cookie+session**

# Session 对象

* cookie是保存在客户端，Session保存在服务器端
* Session将数据存在服务器，数据更加安全 ,但是占用服务器内存资源 ，Cookie不会占用服务器资源

HttpSession httpSession = request.getSession() //获取session对象

httpSession.getAttribute("name") //获取session保存的数据

httpSession.setAttribute(String name,Object obj); //写入数据到session

// 将写回给浏览器jession id写入客户端cookie,保证数据持久化

Cookie cookie = new Cookie("JSESSIONID", httpSession.getId());

cookie.setMaxAge(60 \* 60);

# Cookie生命周期

//创建

Cookie cookie = new Cookie(name,value) ;

response.addCookie(cookie);

//销毁

会话cookie在浏览器关闭时销毁,持久cookie会在cookie过期(MaxAge)后销毁

# Session生命周期

//创建

request.getSession()

//销毁三种

1. 服务器关闭时销毁
2. session过期时销毁
3. 手动调用session.invalidate

# 设置session过期时间

* 配置web.xml

<session-config>

<session-timeout>30</session-timeout>

</session-config>

* 调用session对象

HttpSession session = request.getSession();

session.setMaxInactiveInterval(60\*60); 设置session过期时间1小时

# Servlet 三种存储

* ServletContext
* HttpServletRequest
* HttpSession

每个对象各自维护类似map集合结构，具备相同几个方法

setAttribute 存入一个属性

getAttribute 取出一个属性

removeAttribute 移除一个属性

三种数据范围对象 在哪些情况使用？

ServletContext服务器启动时创建,服务器关闭销毁.所有Servlet共享,保存全局数据

例如：数据库连接池、工程配置属性、配置文件内容 ----- 一般不建议使用

HttpSession 在request.getSession时创建,在三种情况下销毁,保存与用户相关数据

例如：系统登陆信息、购物车数据

HttpServletRequest 在客户端发起请求时,服务器创建该对象,在响应结束时,对象销毁 保存Servlet向JSP传输数据信息

例如：执行某个操作，Servlet将操作结果传递JSP （Servlet将登陆错误信息传递JSP ）、Servlet将数据库查询结果传递JSP

生命周期:ServletContext > HttpSession > HttpServletRequest

# JSP支持的三种注释

1. JSP注释 <%-- --%> 只存在JSP文件源码中,翻译成servlet是消失。

2. Java注释 /\*\* \*/文档注释 /\* \*/多行注释 // 单行注释

文档注释 /\*\* \*/ 生成javadoc 主要用来注释 包、类、成员变量、成员方法多行注释 和 单行注释 不会生成javadoc，注释代码实现逻辑 用于在方法内

Java注释 在JSP翻译为Servlet时存在，生成HTML网页源代码中不存在

3、HTML注释 <!-- -->

结论：JSP注释在翻译阶段消失，Java注释在Servlet运行阶段消失，HTML注释不消失

# JSP内置对象

request HttpServletRequest//请求对象

response HttpServletResponse//响应对象

session HttpSession //会话对象

application ServletContext //web应用全局唯一对象

config ServletConfig //初始化数据

page this //jsp生成的servlet对象this,指向Object，一般不用

pageContext PageContext //页面上下文属性。

exception Throwable 当页面是一个错误页面,用exception获得异常信息

out JspWriter

pageContext对象代表当前页面上下文

1． 向page范围存取数据 (JSP地中存储空间)

findAttribute 依次在 page 、request 、session 、 application 四个数据范围进行数据查找

EL 中 ${name} 调用 findAttribute 在四个范围中依次查找数据

2． pageContext 用来获得其它八个隐含对象

pageContext封装八个隐含对象意义：框架编写，得到PageContext对象 相当于得到 JSP九个内置对象

# out对象

out功能向浏览器输出信息，是JspWriter类型，内部使用PrintWriter实现，拥有独立缓冲区。

out.println("aaa");//后输出

response.getWriter().println("bbb");

//在out输出内容之前,out先输出到response缓冲区

# JSTL (JSP Standard Taglib Liberary) :JSP标准标签库

# JSTL+EL,简化JSP开发

<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%> 在使用标签前,引用名称空间

//<c:forEach> <c:if>

<%

request.setAttribute("name",new int[]{1,2,3,4,5,6};)

%>

<c:foreach

item="${requestScope.name}" var="i">

i的值是&{i}<br>

</c:foreach>

# jsp中六个动作元素标签

<jsp:useBean> <jsp:setProperty> <jsp:getProperty> //与 JavaBean操作相关

<jsp:include> <jsp:forward> <jsp:param >

<jsp:include> 等价于 request.getRequestDispatcher().include

<jsp:include>动态包含<%@ include file=”a.jsp” %>静态包含

<jsp:include page="文件路径" />被包含页面不需要完整html，只需要编写html片段

<jsp:forward> 等价于 request.getRequestDispatcher().forward //转发

<jsp:forward page="/b.jsp"></jsp:forward> 等价于 request.getRequestDispatcher("/b.jsp").forward(request,response);

EL 全名为Expression Language :表达式语言

1. 获取JSP四种范围数据

${applicationScope.name } <%=getServletContext().getAttribute("name") %>

${requestScope.address } <%=request.getAttribute("address") %>

${pageScope.name} ${sessionCcope.name}

//&{name} 在四个范围依次查找,如果查找属性不存在，返回是一个 "" 空串，而不是null

获得数组、List、Map、对象属性时,使用[“”]获取

2、在EL中执行算术、比较、逻辑运算

在EL 执行运算时，运算语句必须写入 ${}中

在EL 获得属性值 执行算术运算，自动类型转换，执行算术运算时，必须都是数字

${"a"+"b"} //发生数字格式化错误

empty运算符

1) 判断一个属性是否存在，通常empty运算符都是结合c:if 一起使用

2) 使用empty判断List 或者 Map是否为空 (size==0)

${user!=null?user.name:""} //三元运算符