**一．Tomcat**

Web应用程序：BS架构的通过浏览器访问

静态网页：内容固定，不会更新 HTML+CSS

动态网页：内容通过程序动态显示，自动更新

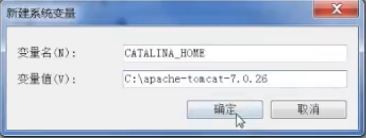
HTML+CSS+DB+高级语言+JSP/Asp.net/Php

Tomcat是一个服务器，用来装载web应用程序提供访问服务

**Tomcat环境变量配置：**

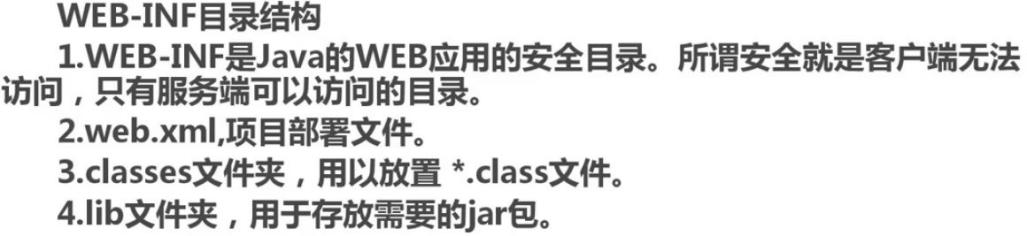
1、如果直接从bin目录下运行startup.bat 启动，是不需要配置环境变量的，而如果直接使用cmd命令则需要环境变量

2、配置了环境变量对全局生效，所以如果需要同时启动多个tomcat，则要么一个环境都不配置，要么所有环境都配置



**启动：bin - startup.bat 关闭cmd则停止服务 端口默认：8080**





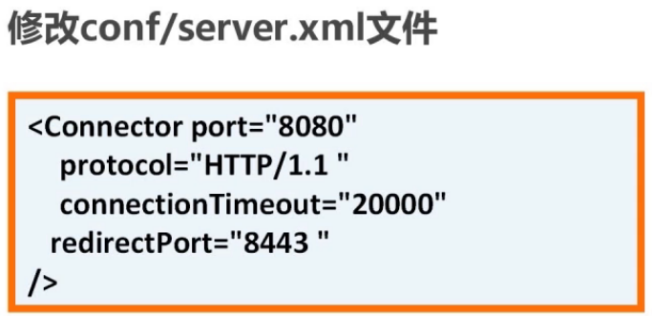
**修改项目默认欢迎页面【访问项目名即可】**

<welcome-file-list>

<welcome-file>xxx.jsp</welcome-file>

</welcome-file-list>

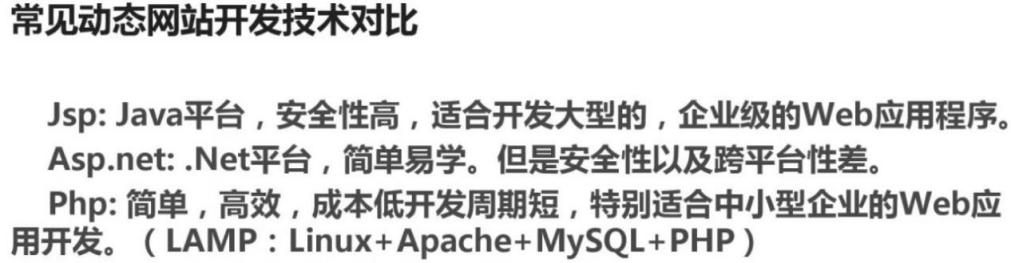
**修改Tomcat默认端口号**

**解决get方式传递参数中文乱码问题：**

在上图内附加 URIEncoding=“utf-8"

**二．JSP**





**Jsp: sun公司制订的一种服务端动态页面的技术规范。**

**注意：**

a.因为直接使用Servlet虽然也可以生成动态页面，但是繁琐(out.println()),

并且维护很难(如果修改页面就必须要进入java代码中)，所以sun公司就制定了JSP规范。

b.jsp是一个以.jsp为后缀的文件，该文件会被容器转换成一个servlet文件然后才执行。

**JSP页面元素构成：**

注释；静态内容；指令元素<%@ %>；脚本元素<% %>；JSP动作<jsp: />

**Jsp指令**





**contentType中指定的MIME类型**

MIME全程为Multipurpose Internet Mail Extension，即多功能Internet邮件扩充服务；

在tomcat的安装目录下的/conf/web.xml中可以查看设置MIME类型，假如现在我们希望页面用word形式显示，设置对应的MIME类型

<mime-mapping>

     <extension>doc</extension>

     <mime-type>application/msword</mime-type>

 </mime-mapping>

****

**Jsp注释**

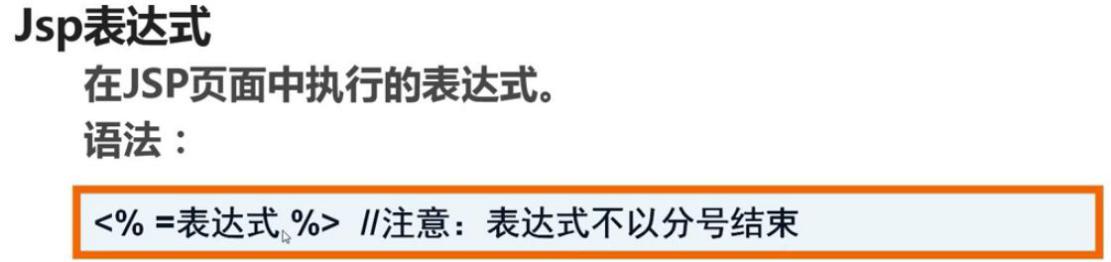


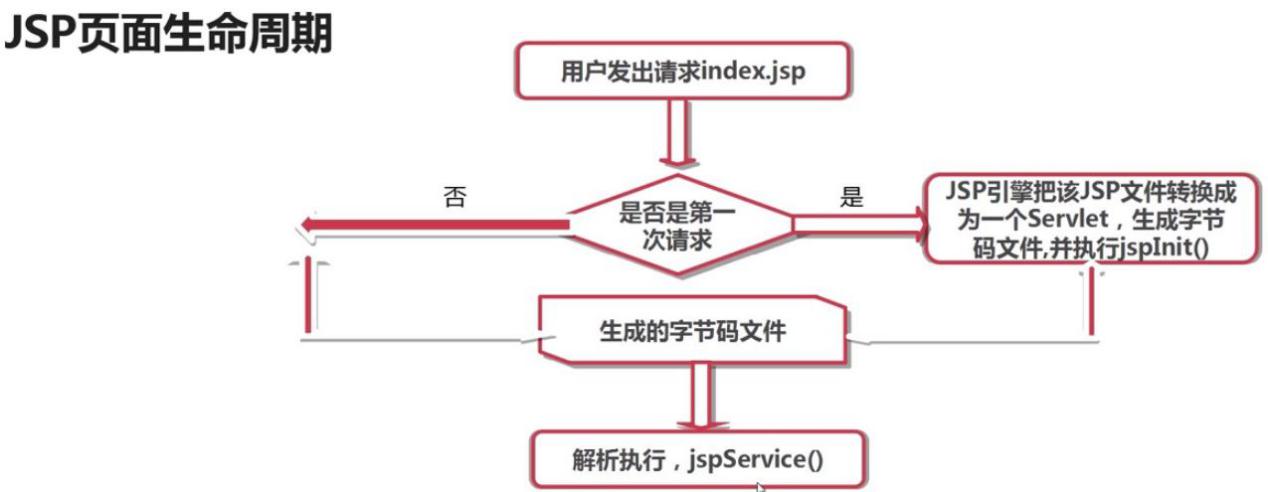


使用JSP脚本元素可以将Java代码嵌入到JSP页面里，这些Java代码将出现在由当前JSP页面生成的Servelt中，使JSP将静态内容与动态内容分离出来。脚本元素包含表达式、脚本和声明等。



注：Java代码可以是定义变量或方法方便JSP脚本调用变量或方法





**Jsp执行过程**

阶段1：

**容器将jsp文件转换成一个servlet类，生成在tomcat的work目录下**。

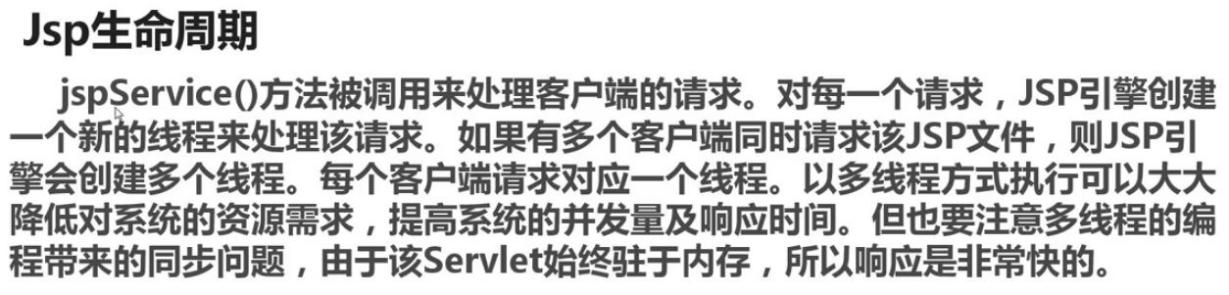
html css js-------->service方法里面，使用out.println()/out.write()输出

<% java片段 %>-------->service方法里面，照搬 java片段

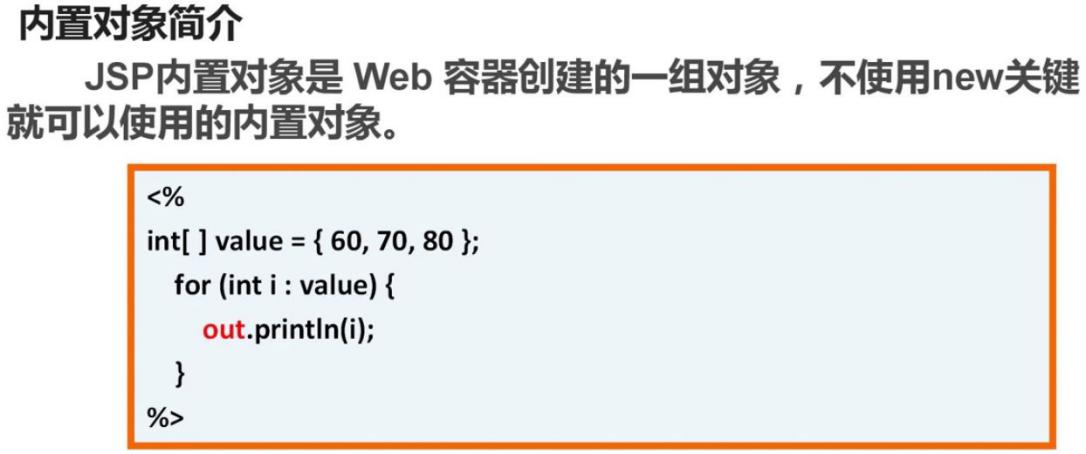
<%=java表达式 %>------>service方法里面，使用out.println()输出

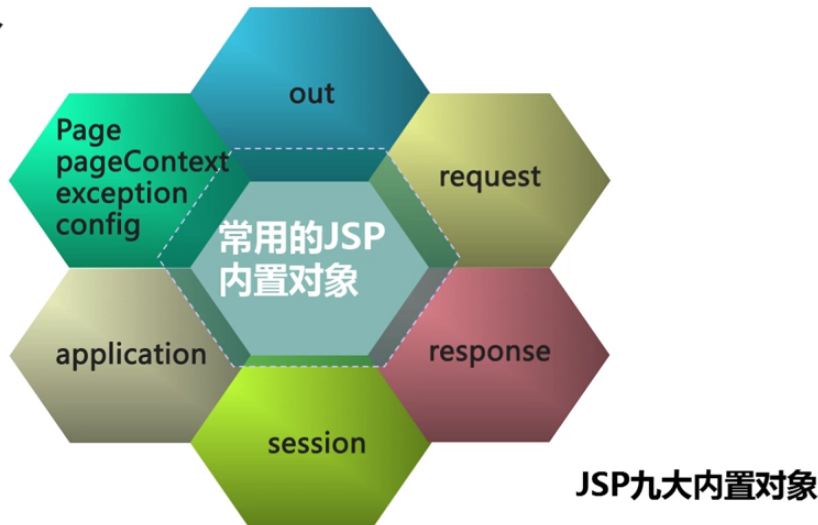
阶段2：

容器调用该servlet

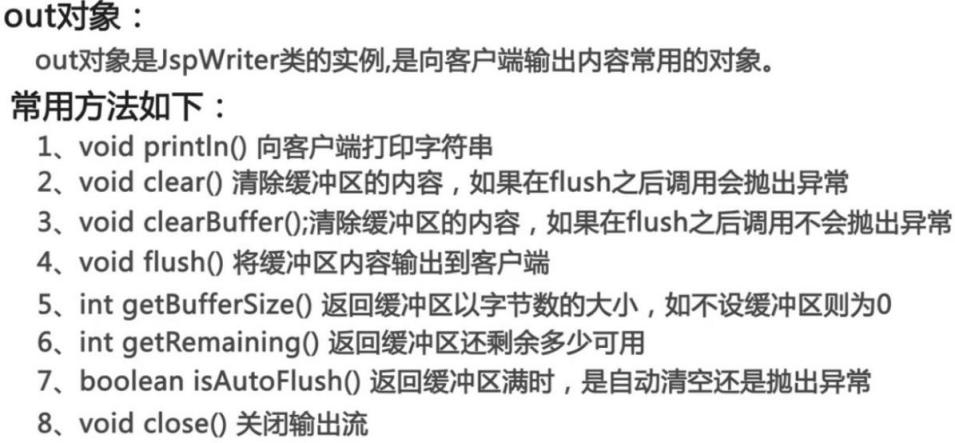


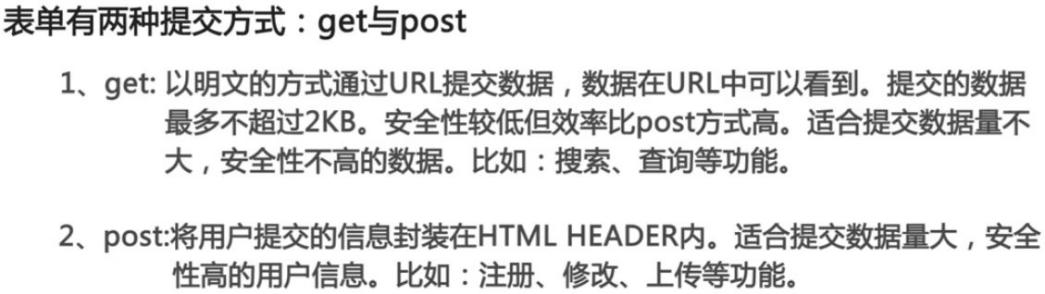


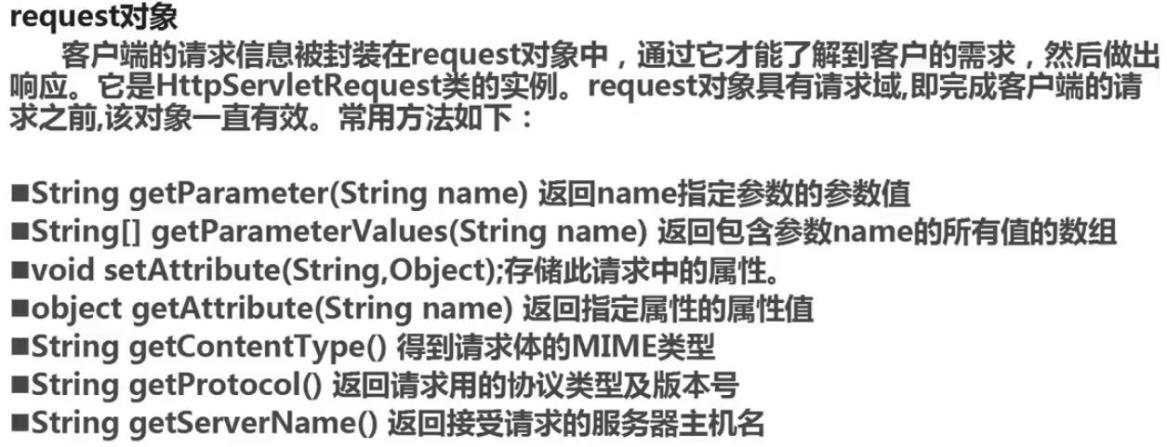




ACE PPRR SO

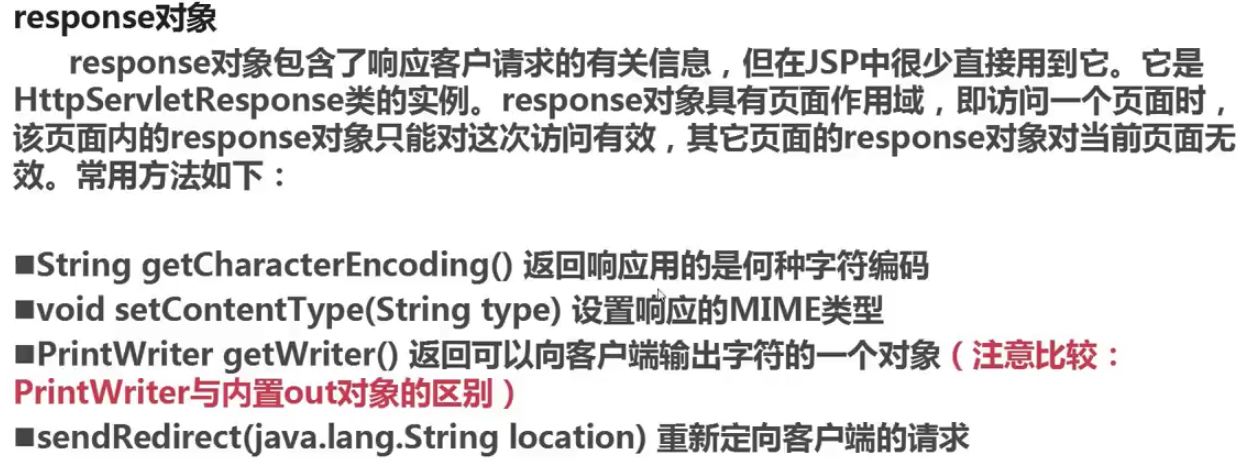








**request.setCharacterEncoding("utf-8"); //解决中文乱码问题，无法解决URL传递中文【url？Name=xxx&password=xxx】出现的乱码问题；**



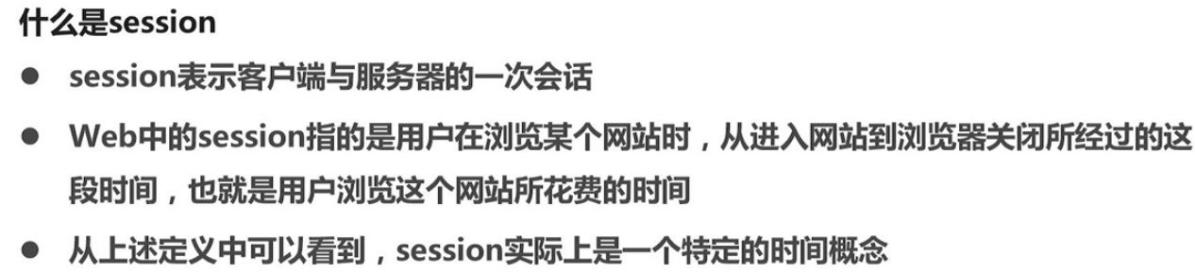
PrintWriter xxx=response.getWriter();

xxx.println();

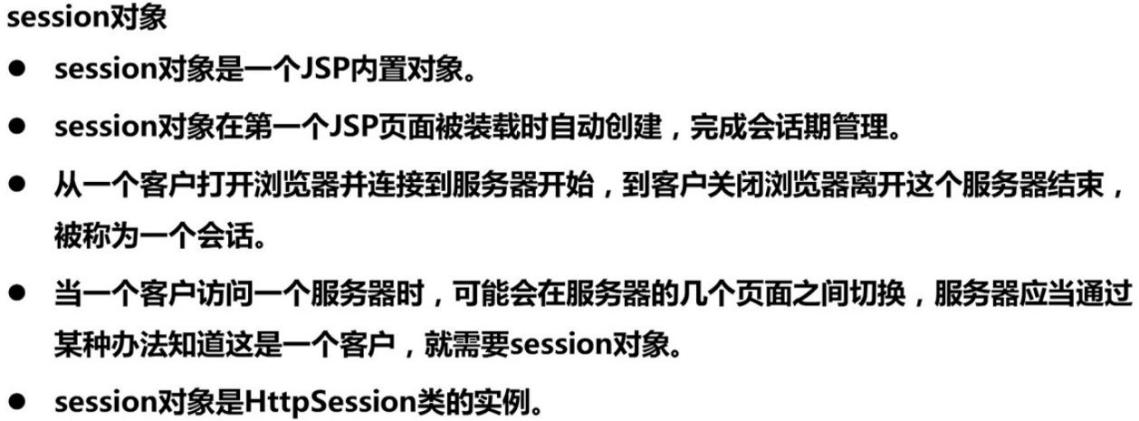
区别：默认内置对象out输出顺序靠后，可以通过out.flush解决

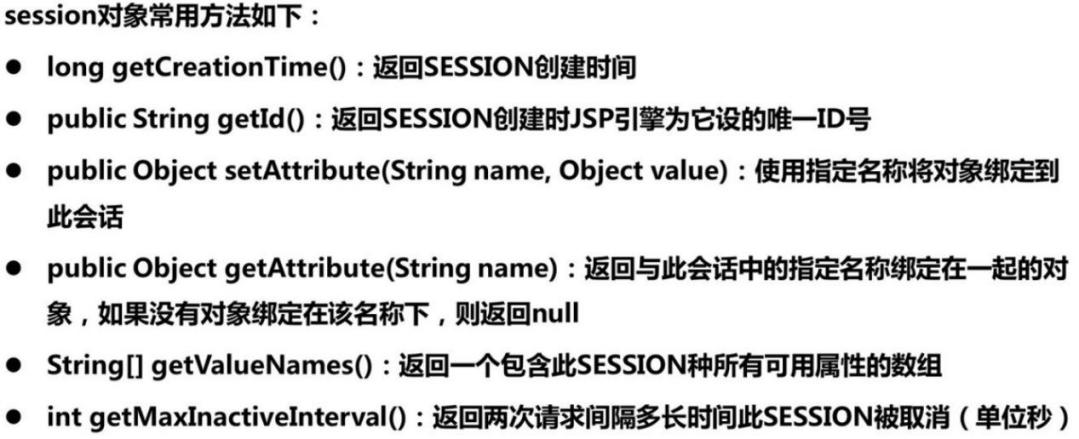


重定向之前，容器会清空response对象上缓存的所有数据



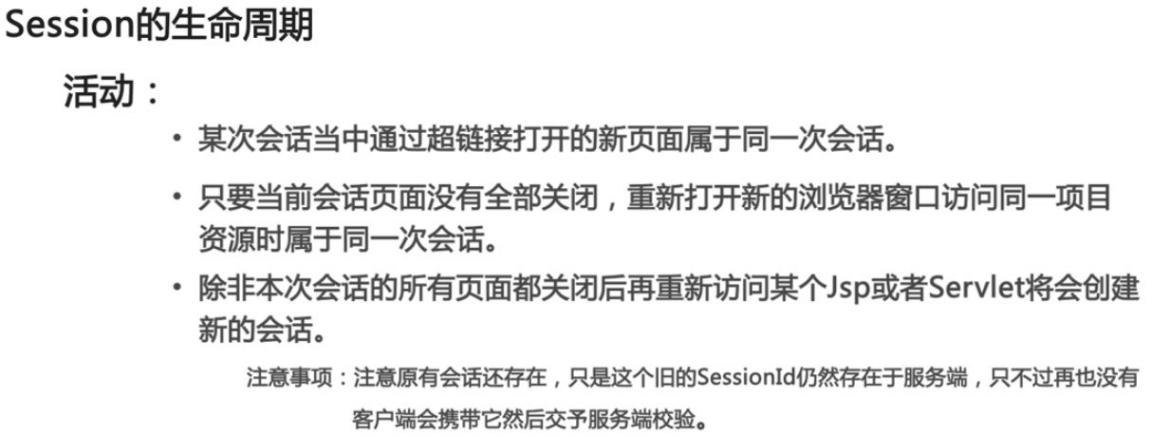






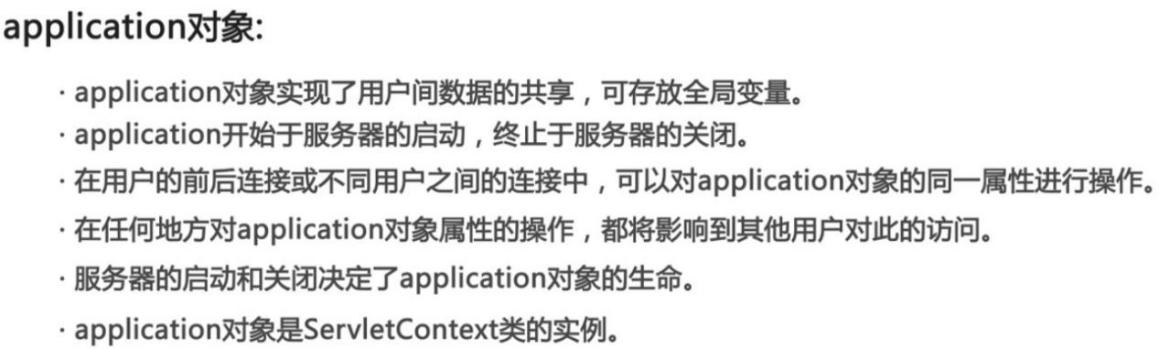
session.getAttribute("xxxxx").toString();

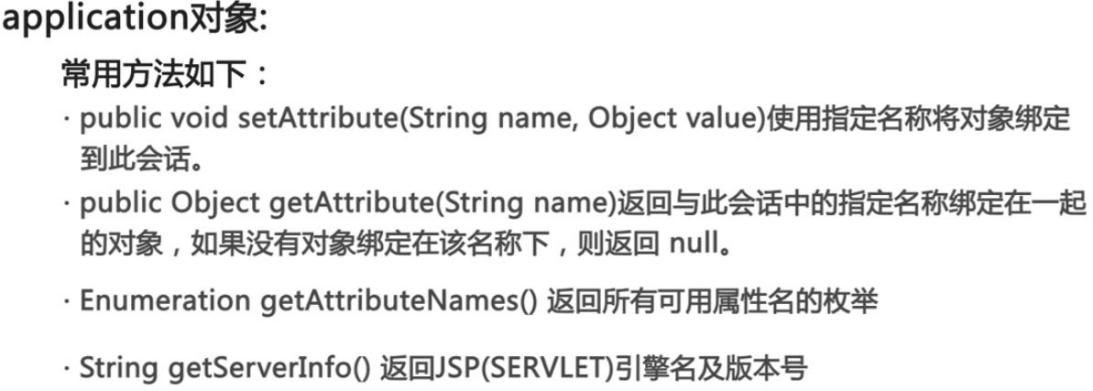


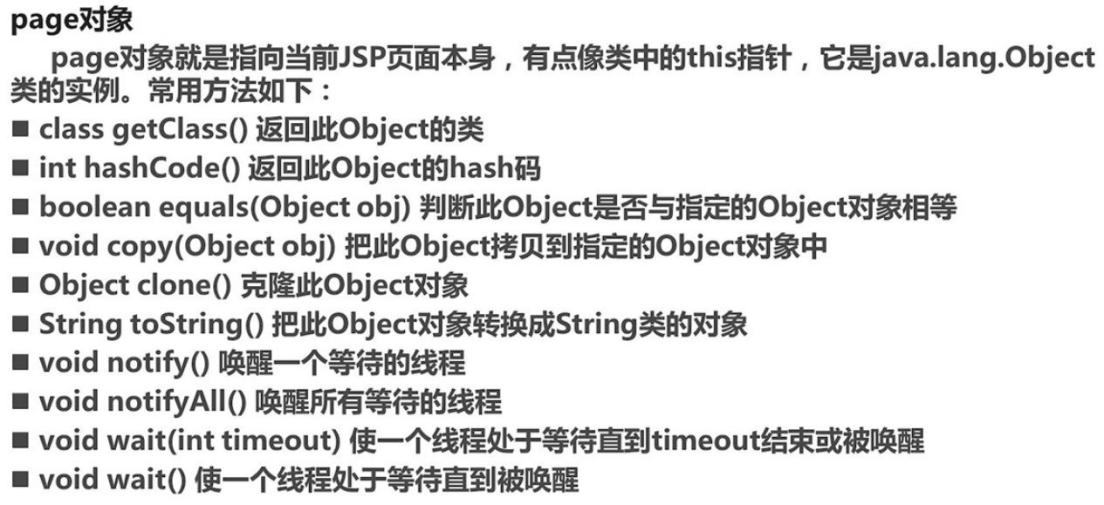


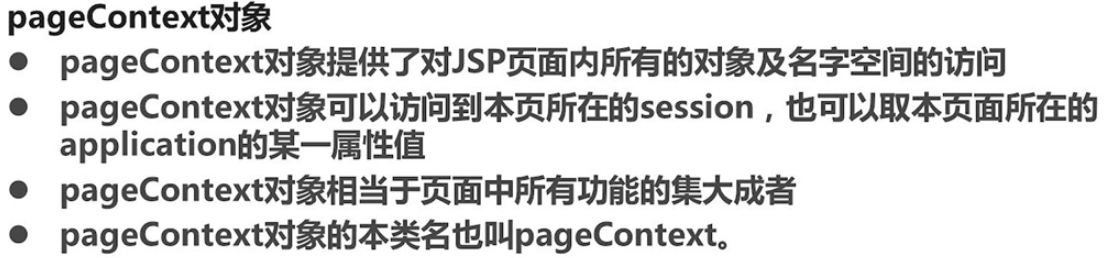


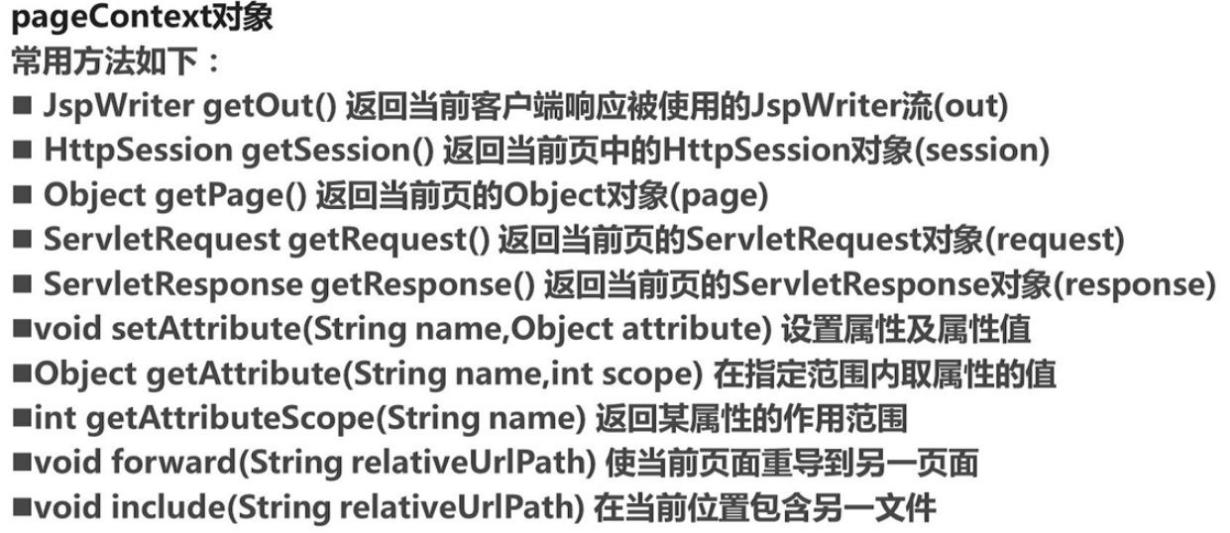
0表示无超时限制



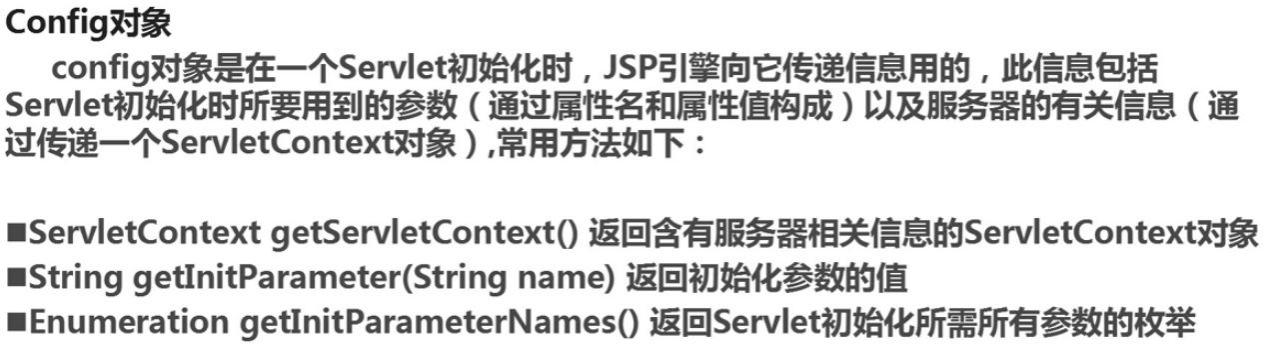


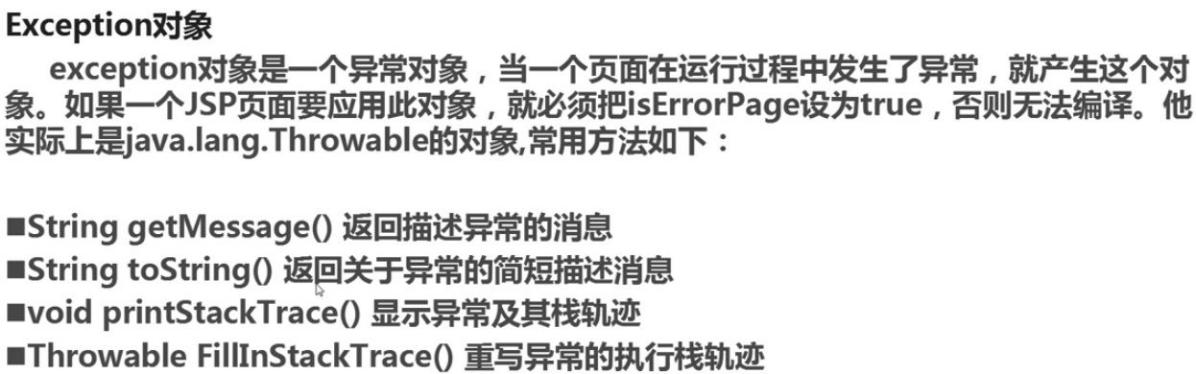






用户名是：<%=pageContext.getSession().getAttribute("username") %><br>



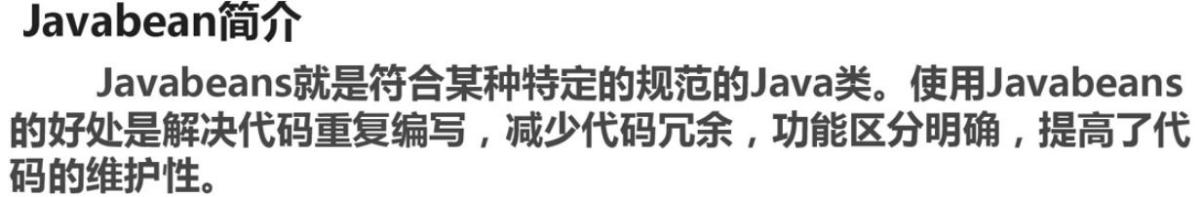


正常执行页面

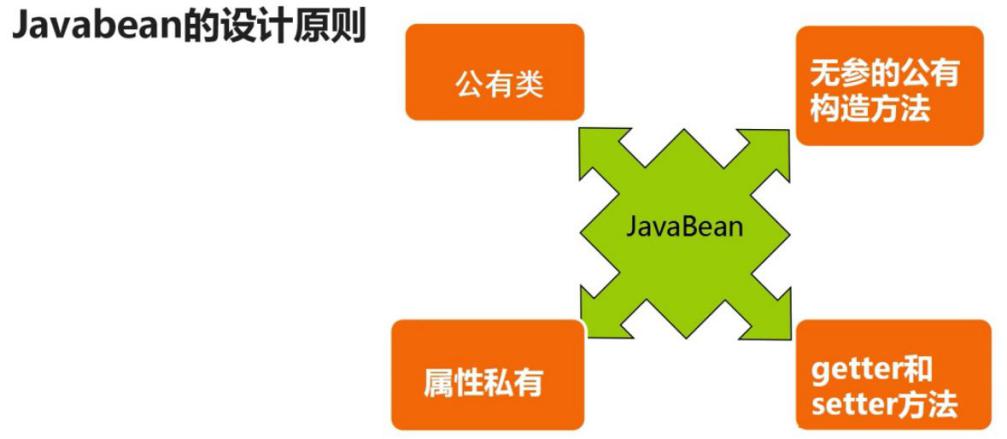
<%@page language="java" import="java.util.\*" contentType="text/html; charset=utf-8" **errorPage="exception.jsp"**%>

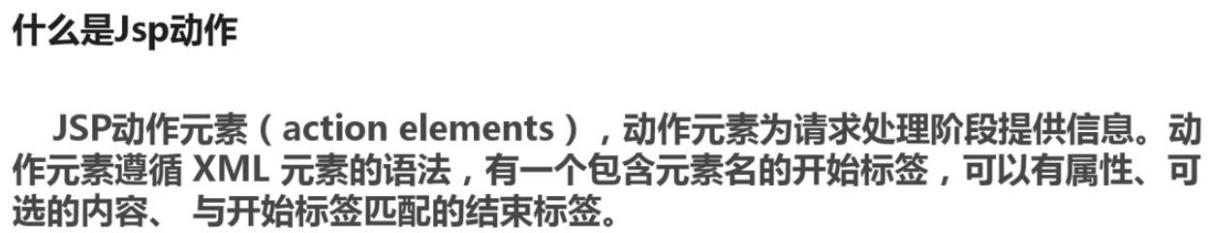
输出错误信息页面

<%@page language="java" import="java.util.\*" contentType="text/html; charset=utf-8" **isErrorPage="true"** %>

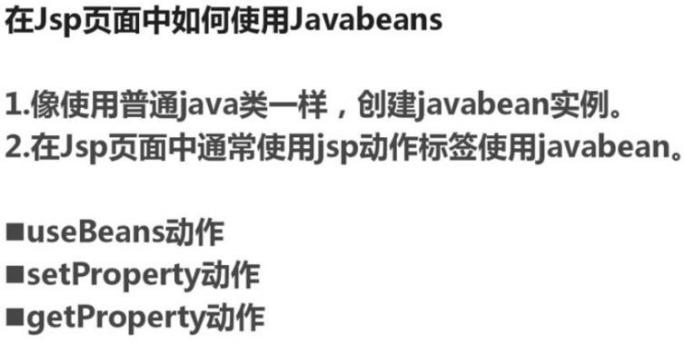


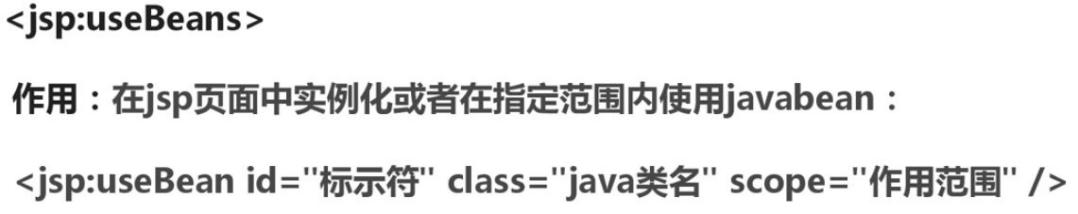
注：其实就是一个普通的java类，该类符合了某种特定的规范，规范可以理解为javabean的设计原则，描述一个对象







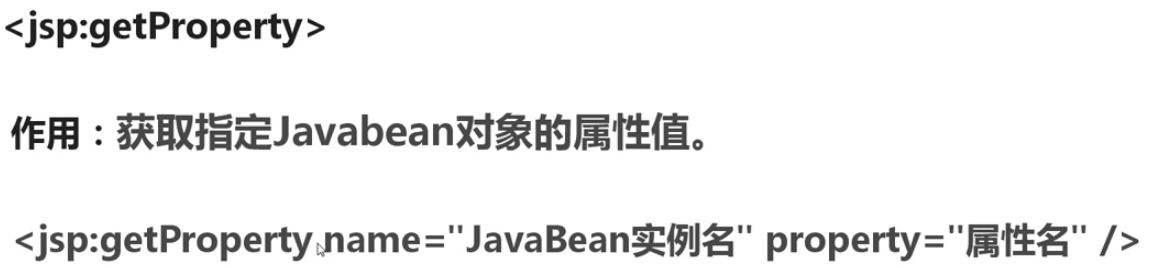






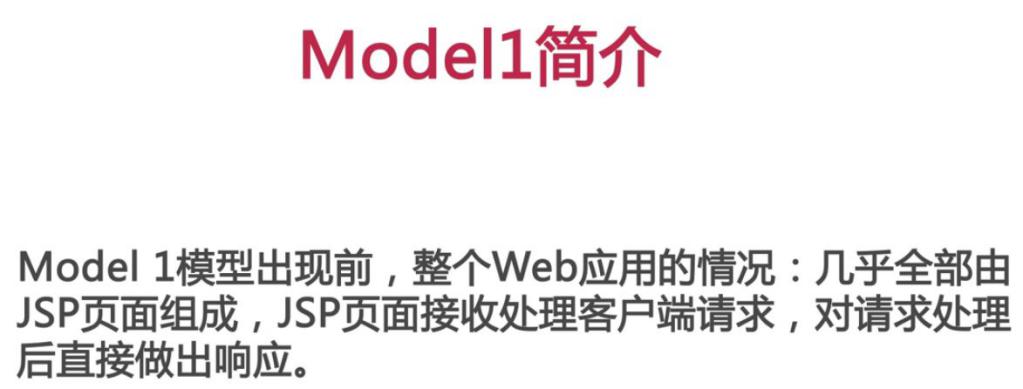




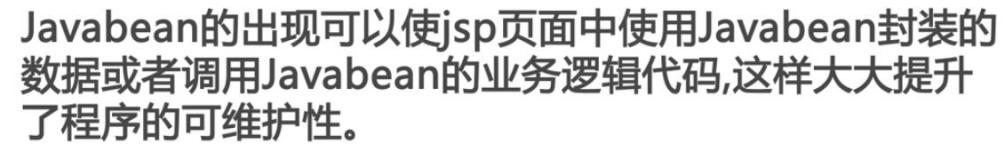


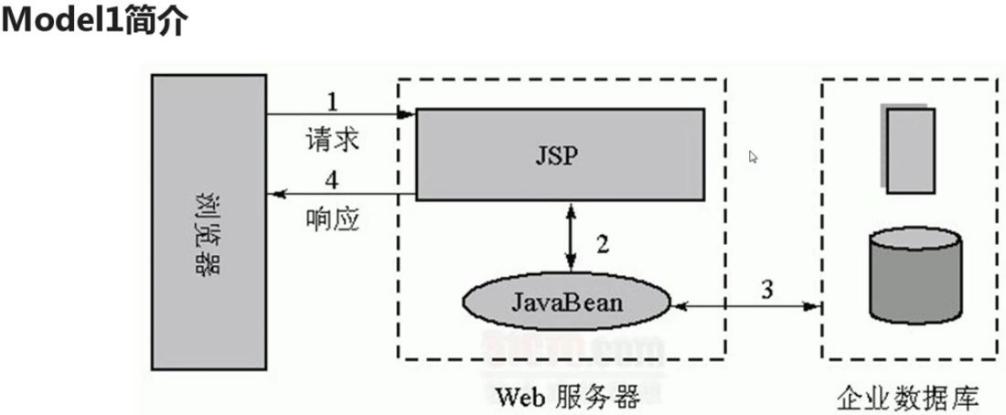


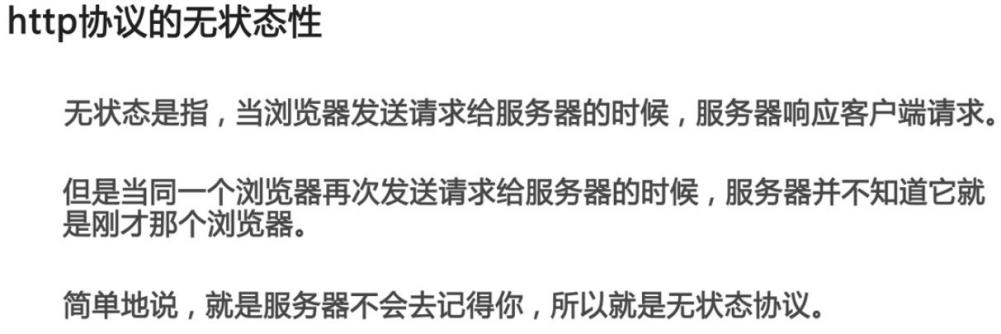






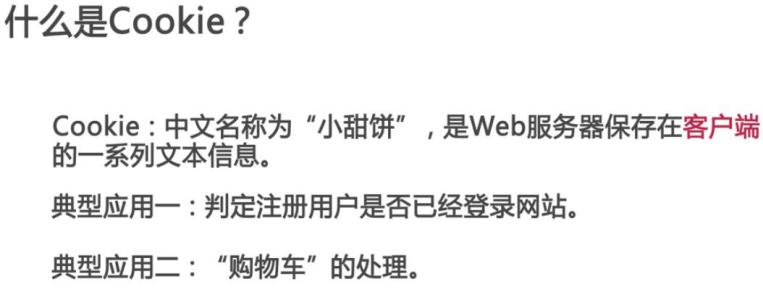


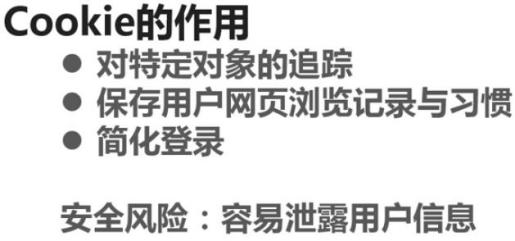






Session属于服务器端 Cookie属于客户端



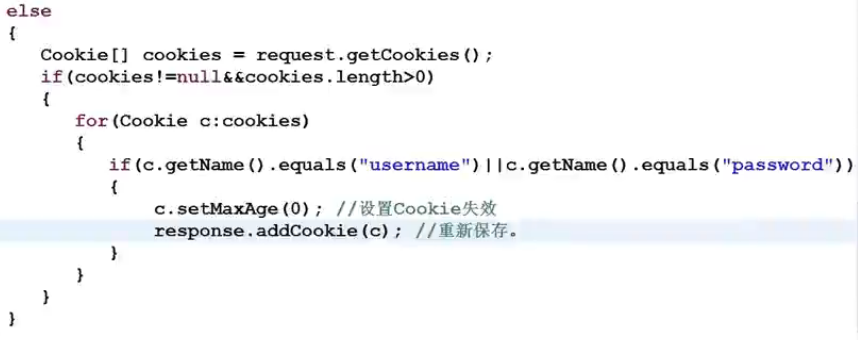




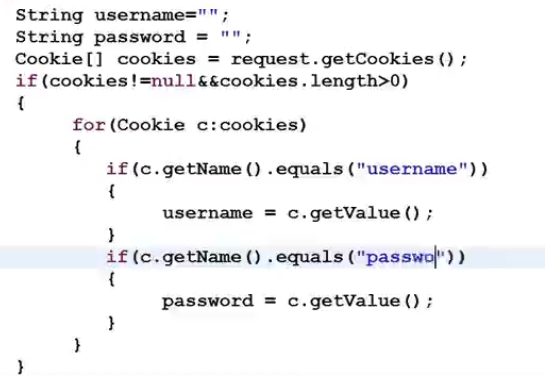


**设置cookie**





**获取cookie**





**编码问题**

ookie只能存放合法的ascii字符，

如果是中文，需要去转换成合法的ascii字符。

String URLEncoder.encode(String str,String charset)

String URLDecoder.decode(String str,String charset)

**cookie的限制**

a.不安全，不要将敏感的数据以cookie的方式保存在浏览器上，

如果一定要保存就需要加密

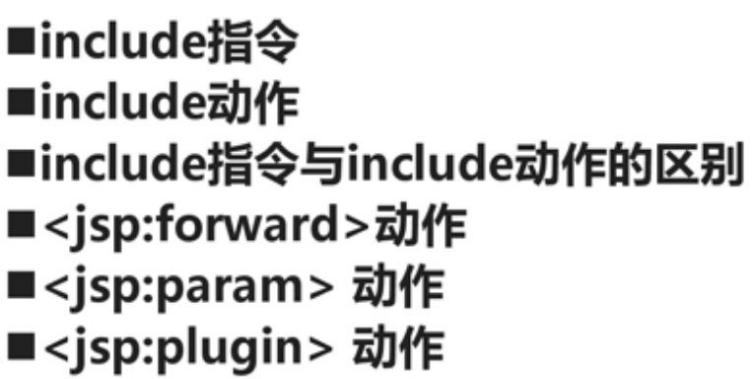
b.可以禁止cookie(可以被用户禁止)

c.大小有限制(大约4k)

d.cookie的数量也是有限制的

e.只能保存字符串

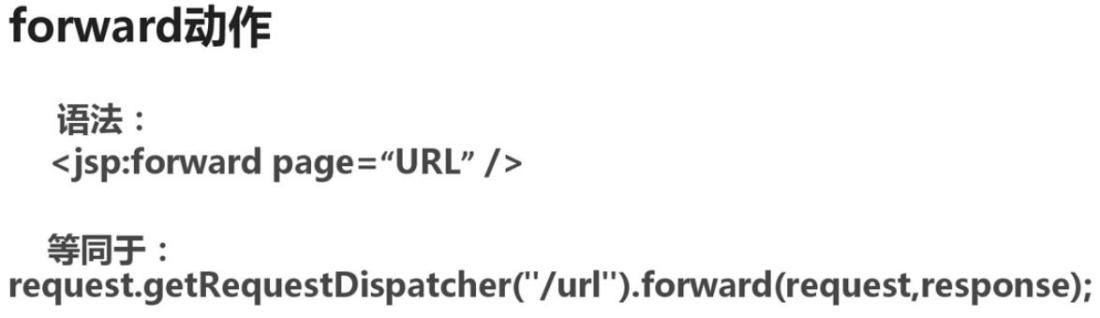
注意：在保存cookie时，不管有没有中文，最好都编码处理

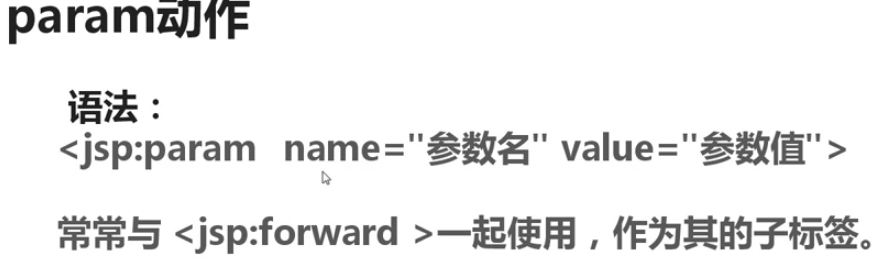




**从效果上看，二者并没有区别，开发过程尽量使用jsp动作**







对于表单提交的数据进行中间层处理，添加/修改/删除表单数据然后再跳转