一、JaveWeb 前期准备

1、软件开发领域的三大方向：

* + 桌面应用程序
  + Web应用程序
  + 嵌入式应用程序（移动端）

2、静态网页和动态网页

* 静态网页;
* 表现形式：网页中的内容固定，不会更新；
* 所需技术：HTML，CSS
* 动态网页：----与用户有交互
* 表现形式：网页中的内容通过程序动态显示，自动更新。
* 所需技术：HTML,CSS，数据库技术，至少一门高级语言（java，C#，PHP），JavaScript，XML等。主流的动态网页脚本技术（Jsp/Asp。net/PHP）

3、搭建JSP开发环境

* 搭建JavaWeb 开发环境： **JDK，Tomcat7.0，My esclipse**

安装Tomcat属于Apache开源项目，可以解析HTML常见网页，也是jsp 的容器

启动Tomcat，bin目录， startup.bin,不能关闭窗口。

* 操作步骤：

1. 安装Tomcat服务器。

* 到Apache下载Tomcat，直接解压缩就行，不用安装。
* 配置环境变量。---新建一个变量CATALINA\_HOME，将Tomcat主程序文件夹地址放进去。
* 启动服务器---bin文件目录—startup.bat，保持服务器窗口。
* 测试首页---浏览器---http://localhost:8080/

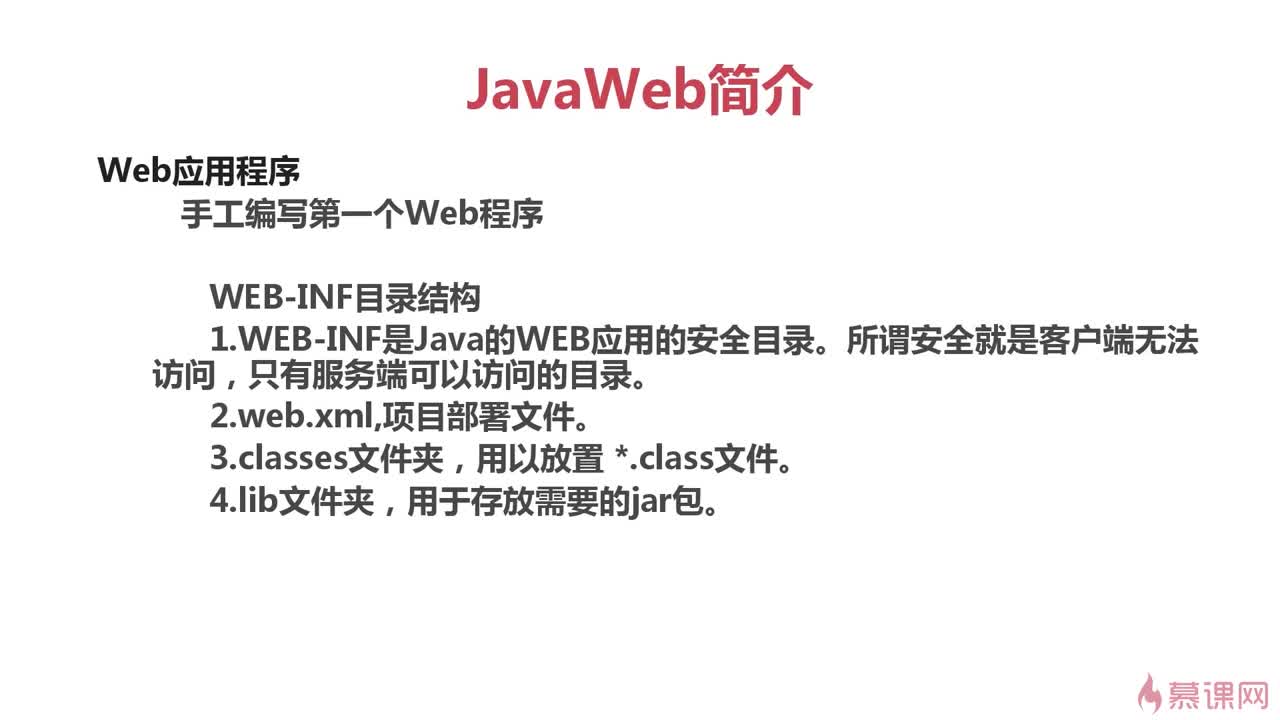
4、Tomcat服务器目录结构



5、手工编写第一个WEB程序

* + 在WebApps创建项目目录
  + 编写index.jsp
  + 创建WEB-INF目录(目录下存放classes文件夹、lib文件夹及web.xml文件)
  + 测试运行

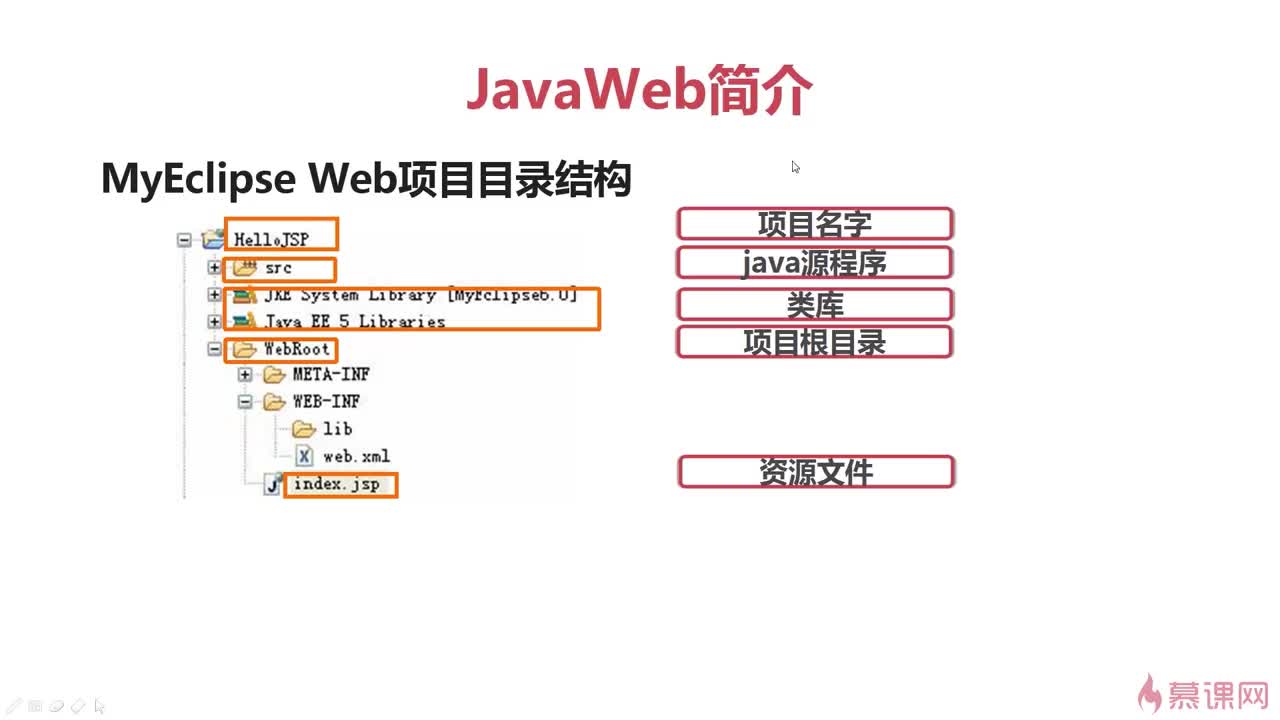
6、WEB-INF目录结构



* 修改默认首页
* 跟WEB-INF目录平行放置的index.jsp 可以作为网页的欢迎界面 由tomcat默认选定。
* 如果要改欢迎界面，就在web.xml文件里加 <welcome-file-list><welcome-file>/xxx.jsp</welcome-file></welcome-file-list>
* 

7、Myeclipse搭建环境

* 配置本地JDK
* Window—preferences—java—Installed JREs—添加本地JDK
* 配置Tomcat服务器
* Window—preferences—Myeclipse—servers—Runtime Environments—添加本地服务器，记得勾上Create
* 建立WebProject---部署到Tomcat服务器---启动服务器
* Myeclipse Web项目目录结构



* 修改Web项目虚拟路径—在Tomcat中显示的文件目录，即网页访问的路径

右键项目—properties---Myeclipse---project facets—web—web context root

8、修改Tomcat服务器默认端口号

* 修改conf/server.xml文件

<Connector port="8080" //修改这个端口号

protocol="HTTP/1.1"

connectionTimeout="20000"

redirectPort="8443"

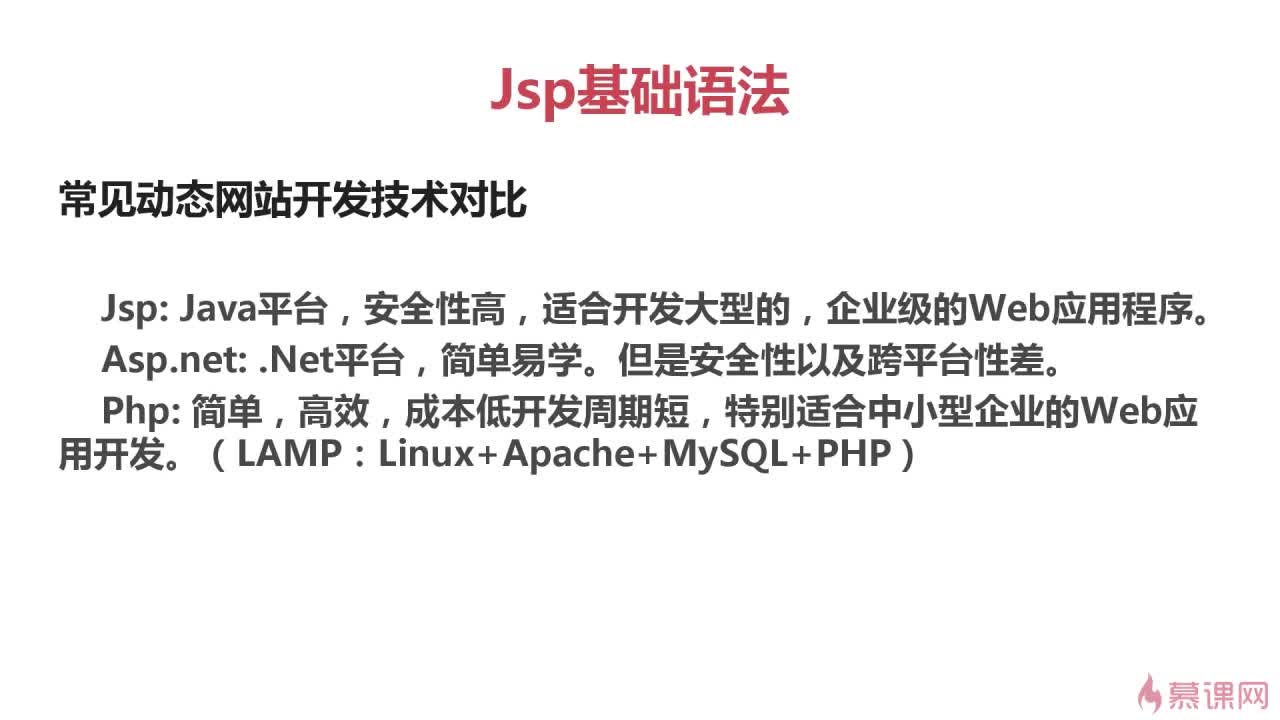
/>

二、JSP基础语法

1、JSP简介

* JSP--JAVA Server Pages:
* 其根本是一个简化的Servlet设计；
* 其实现了在JAVA中使用HTML标签；
* 是一个动态网页技术标准，也是JAVAEE的标准；
* 在服务器端执行。

2、常见动态网站开发



3、JSP页面元素构成



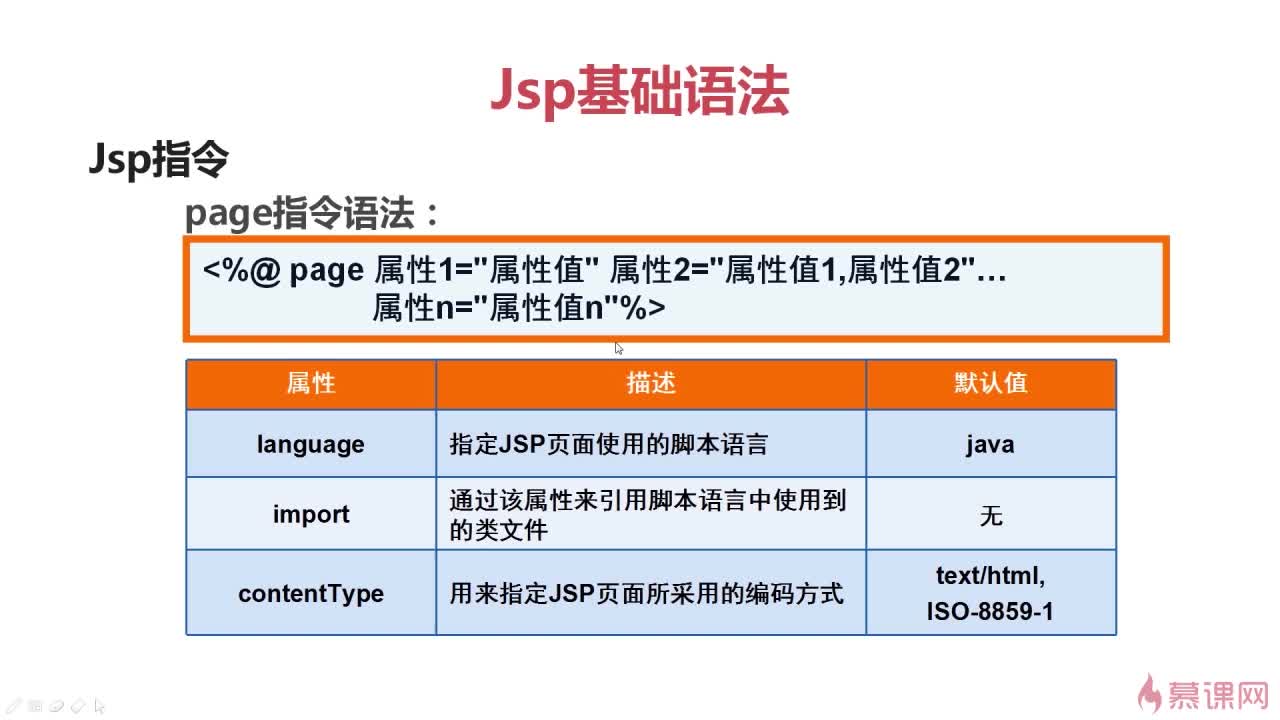
4、JSP指令

* Page：指令，通常位于JSP页面顶端，同意页面可以有多个page指令。
* Include：将一个外部文件嵌入到当前JSP文件中，同时解析这个页面的JSP语句。
* Taglib：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。
* pageEncoding是jsp文件本身的编码

contentType的charset是指服务器发送给客户端是的内容编码

contentType更常用

* Page指令：



5、JSP注释---在JSP页面的注释



6、JSP脚步

* JSP脚本是：在jsp页面中执行的java代码

语法格式为： <% java code %>---后面要加分号结束

7、JSP声明---在JSP页面中定义变量或方法

例：----百分号%后面要跟着一个感叹号！

<%!

String s = "张三";

int add(int x,int y)

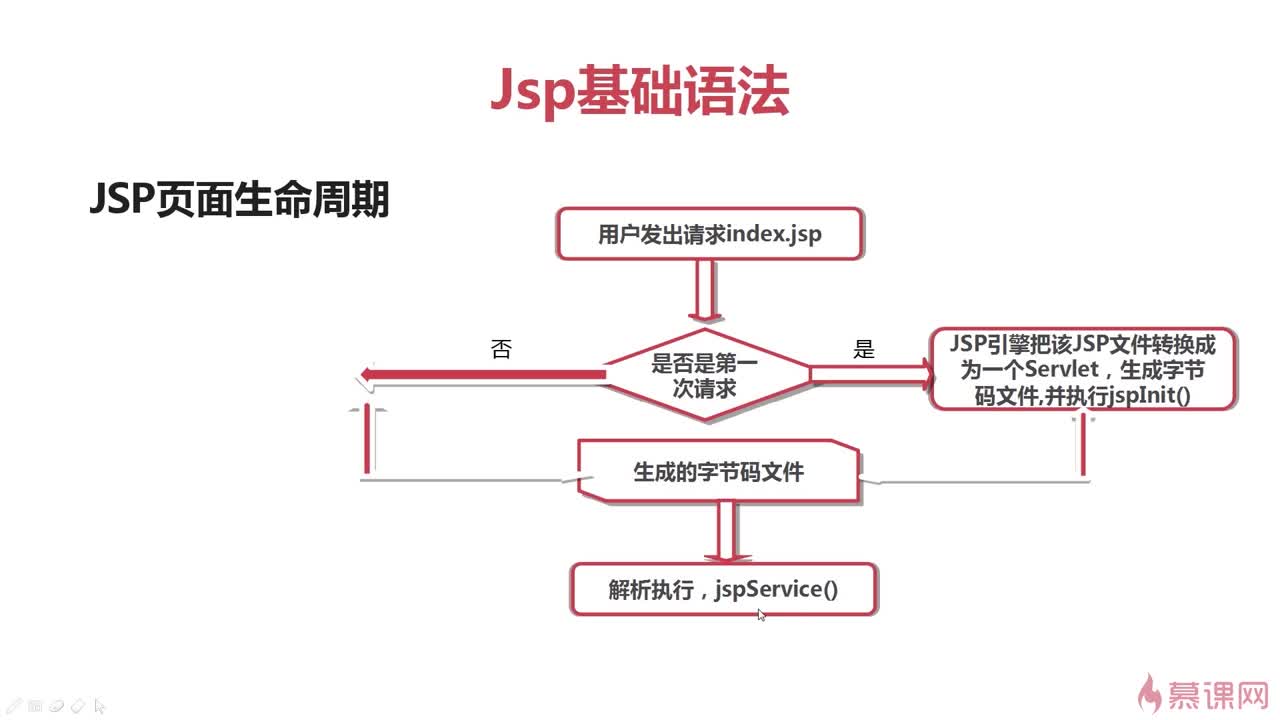
{ return x+y; } %>

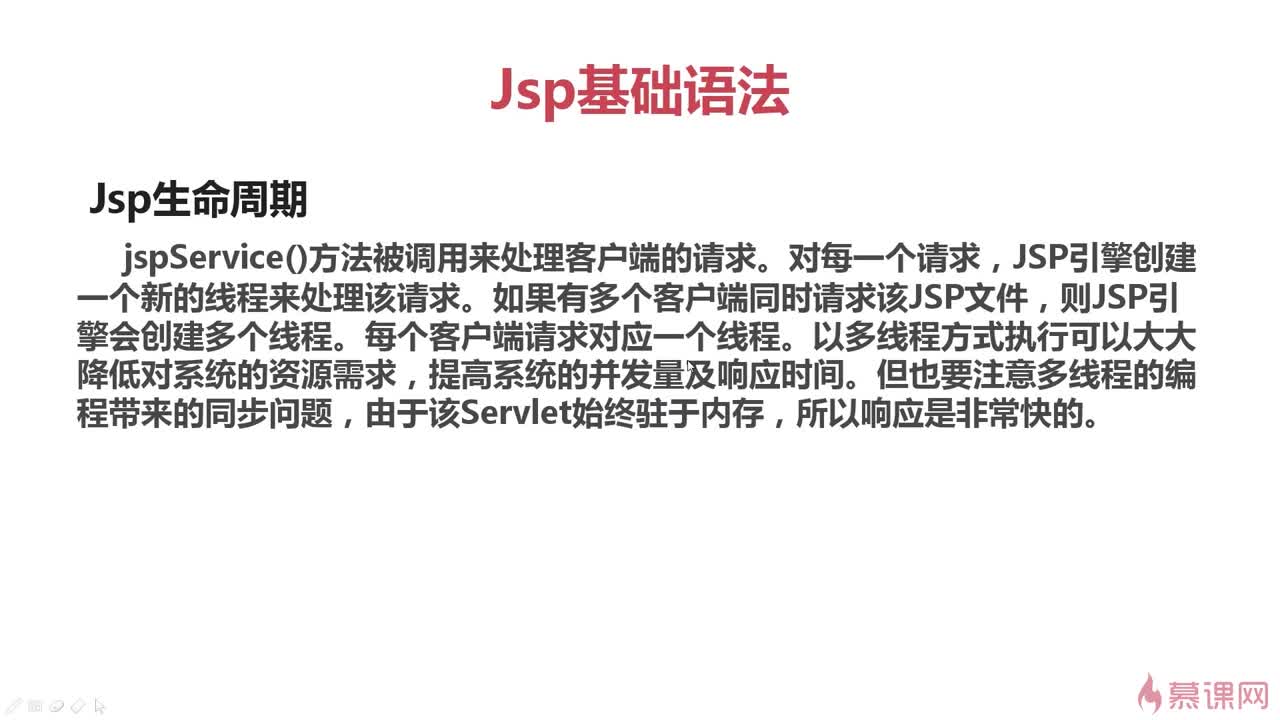
8、JSP表达式---在JSP页面中执行的表达式

语法：----后面要加等号

<%=表达式 %> **表达式不以分号结束。**

9、JSP页面生命周期





1、编译阶段：将jsp编译成servlet

2、初始化阶段：将jsp对应的servlet进行初始化，创建servlet实例，调用jspInit()方法进行初始化。jspInit()方法只执行一次。

3、执行阶段：servlet实例调用jspService()方法对请求进行相应的处理

4、销毁阶段：servlet实例调用jspDestroy()方法对该实例进行销毁

* Eclipse for JAVAEE的位置：

Tomcat下work文件夹含有jsp页面经编译生成的字节码文件

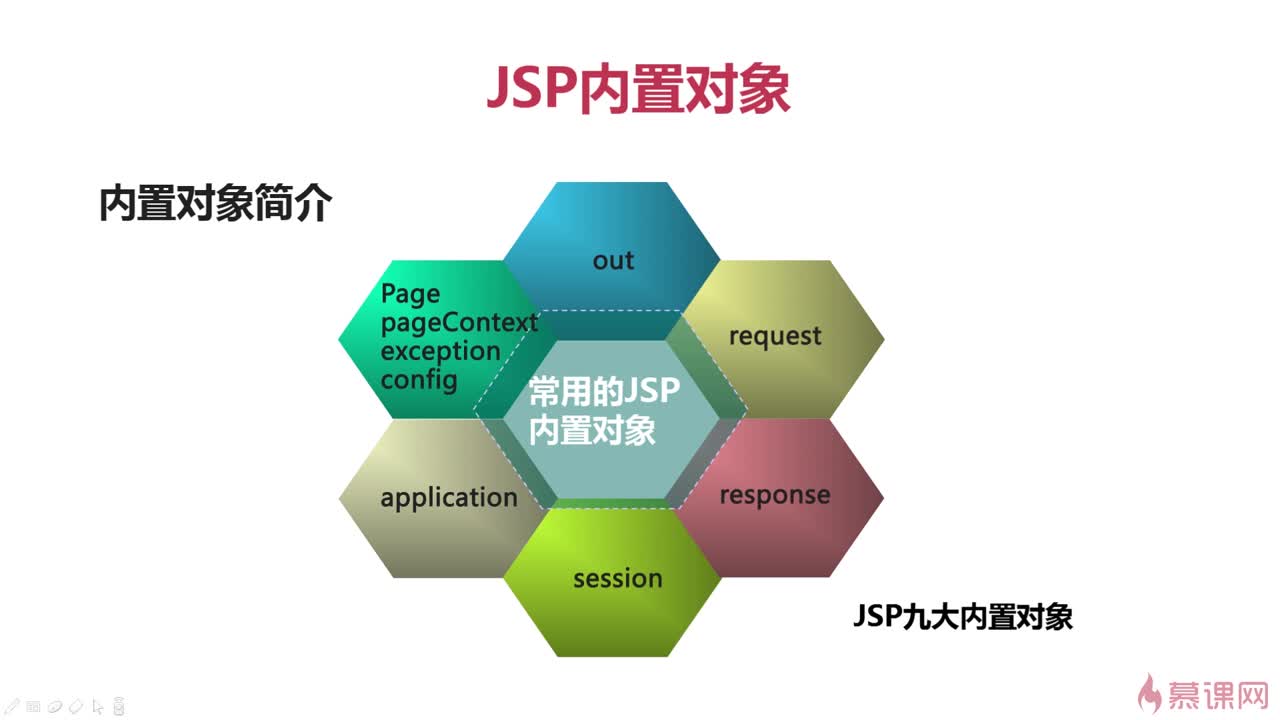
但是eclipse的项目部署在tomcat上后，项目不在webapps中，而是在默认空间中(F:\EclipseWeb\webApp);

字节码文件也不在work中，而是在eclipse的工作空间：(F:\EclipseWeb\webApp\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\work)

三、JSP内置对象

1、内置对象简介

* JSP内置对象是WEB容器创建的一组对象，不用new关键字就可以使用的内置对象。



2、WEB程序的请求响应模式

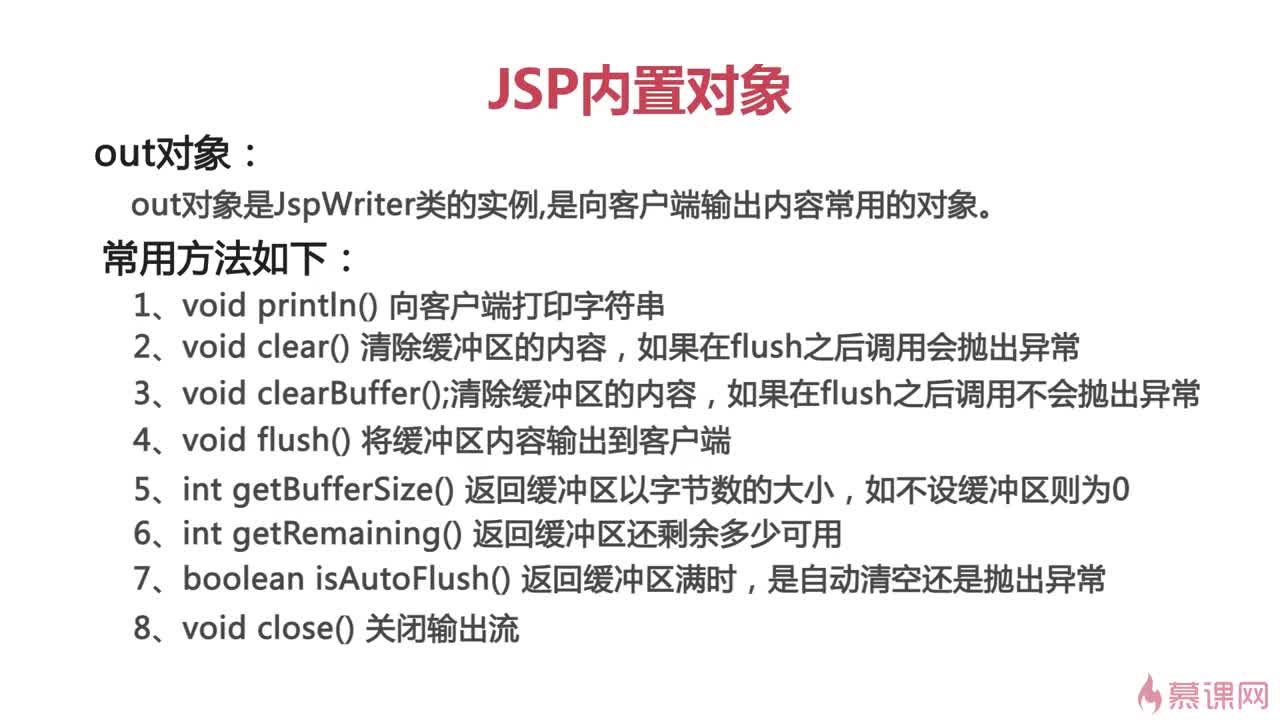
* web应用程序都是基于请求和响应的模式

用户给服务器发请求是request。

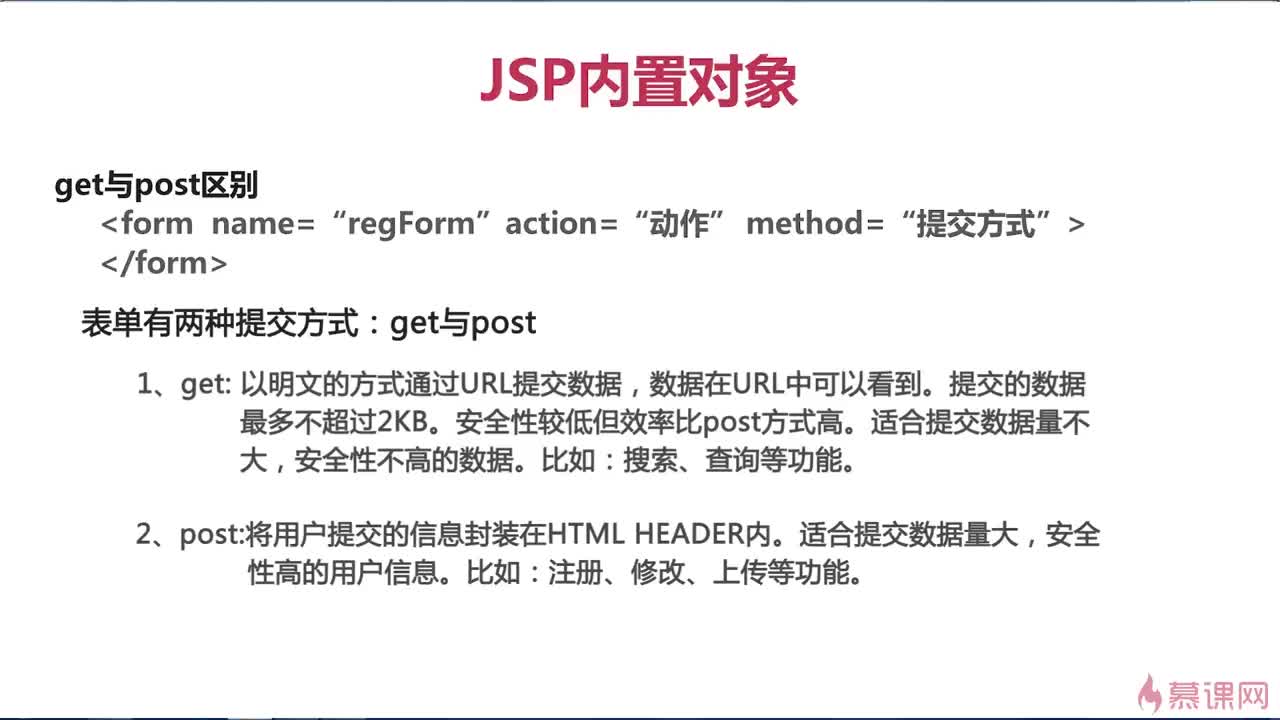
服务器向用户响应请求是response。

* web应用程序都是基于请求和响应的模式，当客户端发送了一个http请求后，服务端根据请求的内容做出相应的处理，并且返回给客户端一个响应response(大多数是一个HTML页面）

3、OUT对象



4、表单两种提交方式：



* 语法：

<form action="dologin.jsp" name="loginform" method="get">

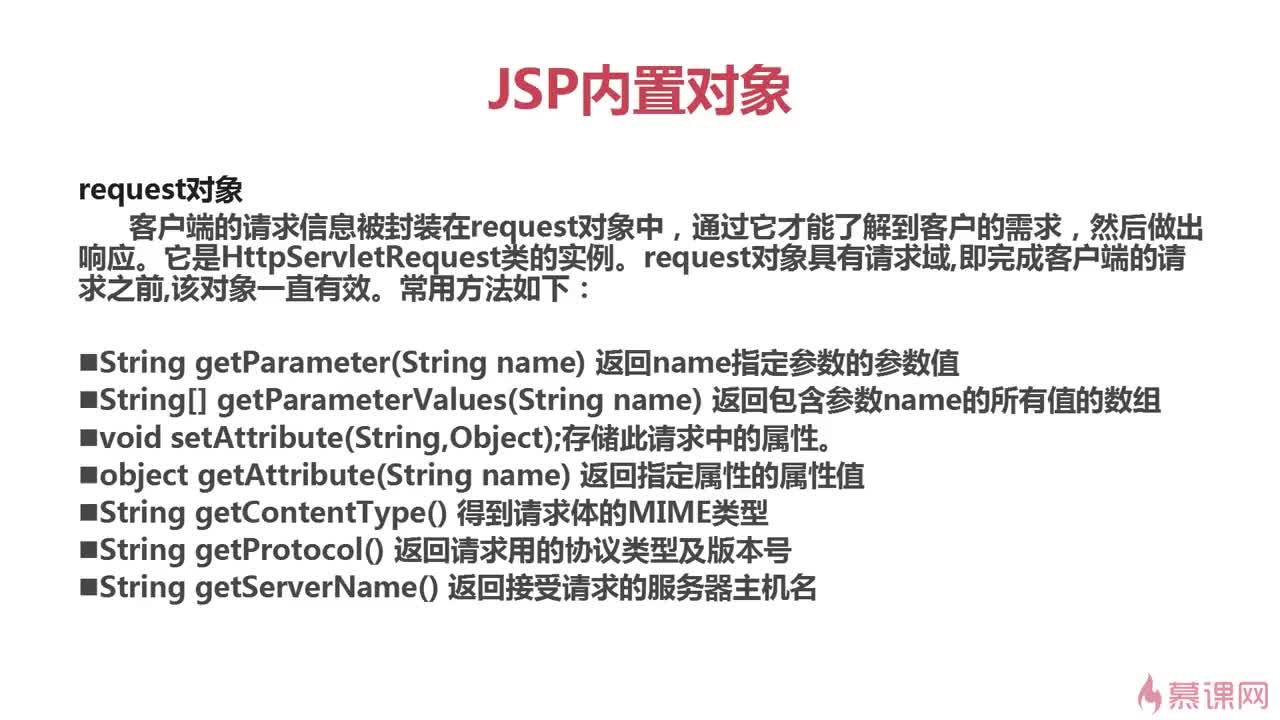
5、request对象

* URL传参中文乱码问题可通过修改配置文件解决

修改服务器conf\server.sml文件, 向其中的<Connector/>中添加URIEncoding="urf-8"即可解决

* 通过POST提交传参数的乱码问题

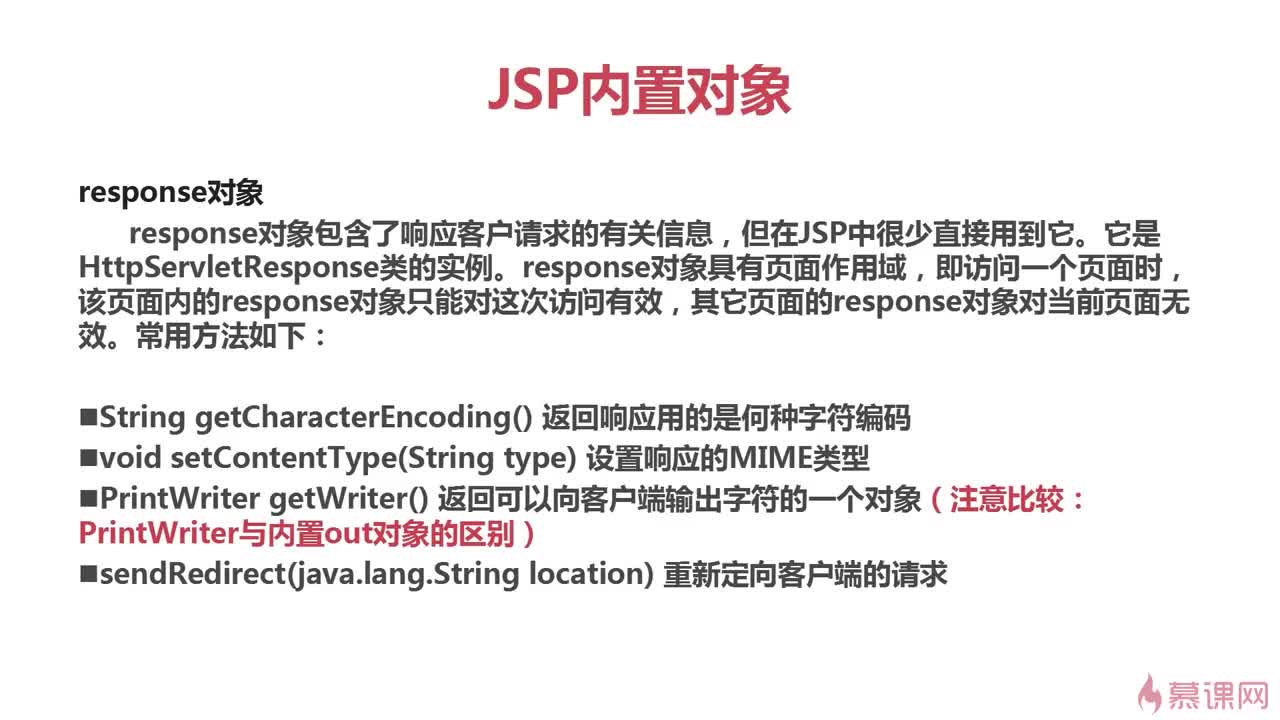
request.setCharacterEncoding("utf-8");



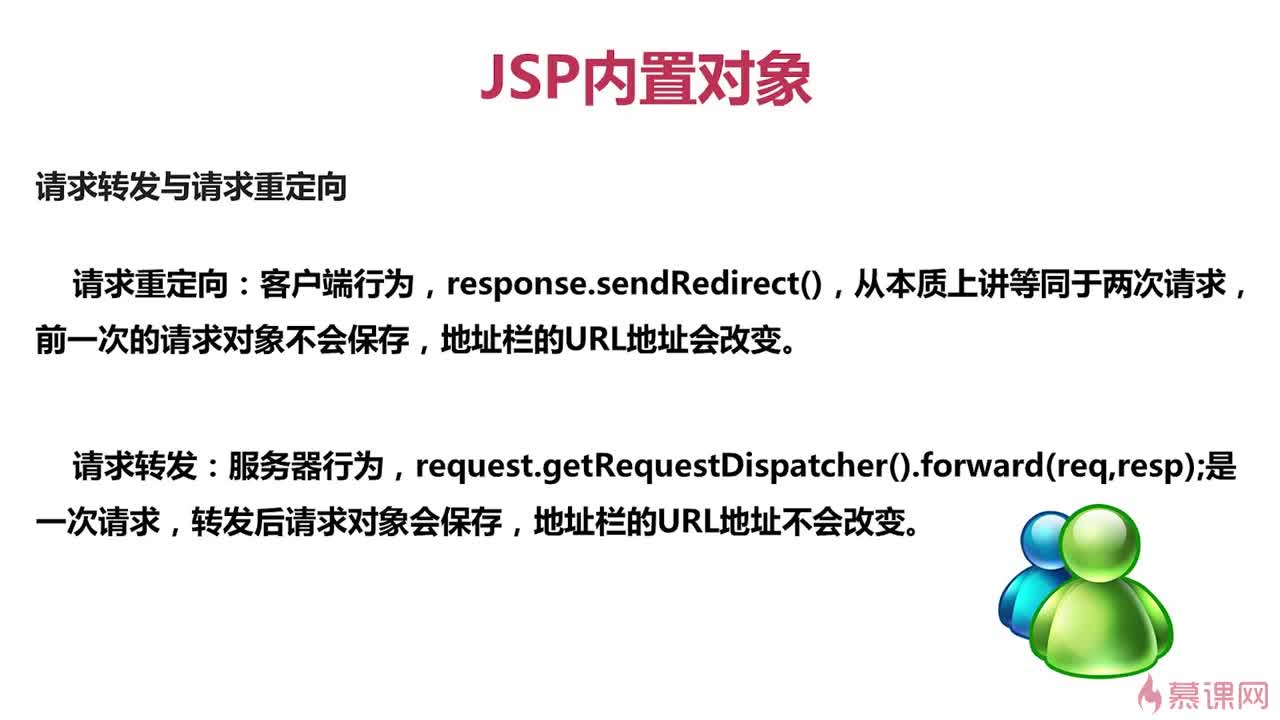


6、response对象

* response对象表示服务器端对客户端的回应。
* PrintWriter对象在内置out对象之前输出，用out对象的flush()方法强制清空缓冲区，同时强制把缓冲区的内容输出到浏览器，这样就可以在PrintWriter对象之前输出。
* response 对象具有页面作用域，即访问一个页面时，该页面的response对象只能对这次访问有效，其它页面的response对象对当前页面无效



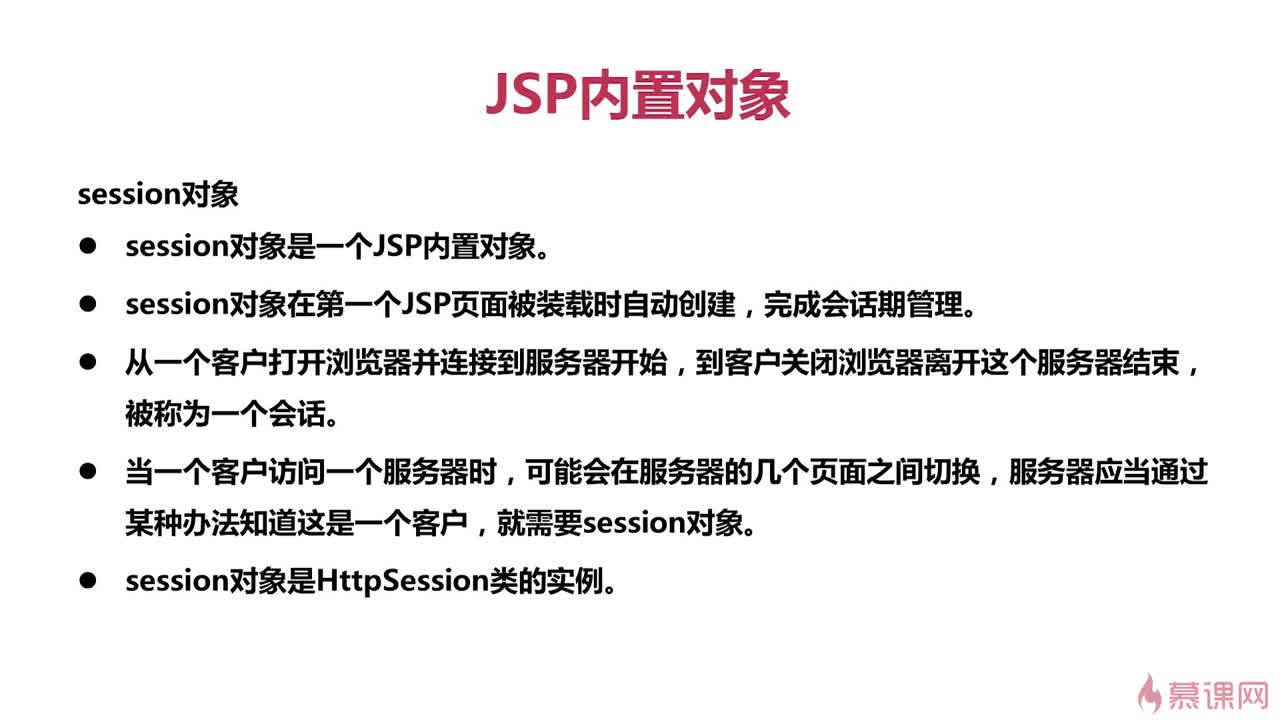
7、请求重定向和请求转发

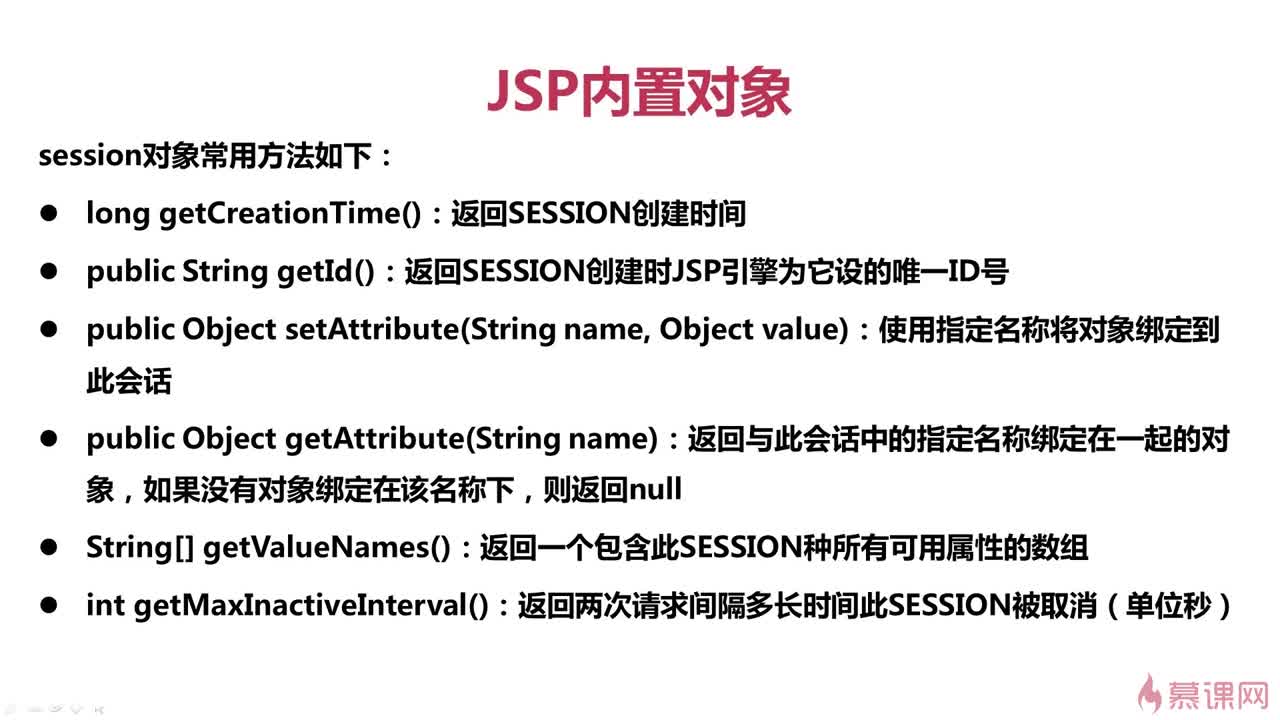


8、session对象

* session表示客户端与服务端的一次会话，指的是用户在浏览某个网站时，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。**是一个时间概念，也是一个过程的概念。**
* session用于区分不同的客户端，保存在服务端的内存中，**和用户一一对应**。
* session在客户端浏览器第一次访问服务器时创建，然后保存在服务器端，当每次浏览器访问服务器时，服务器会查看这份档案，然后明确是谁在访问，session内部应该是有一个id，以区分不同的用户的。

9、session对象常用方法





* 时间格式化语法：

<%

SimpleDateFormat sdf = new SimpleDateFormat("yyyy年MM月dd日 HH:mm:ss");

Date d = new Date(session.getCreationTime());

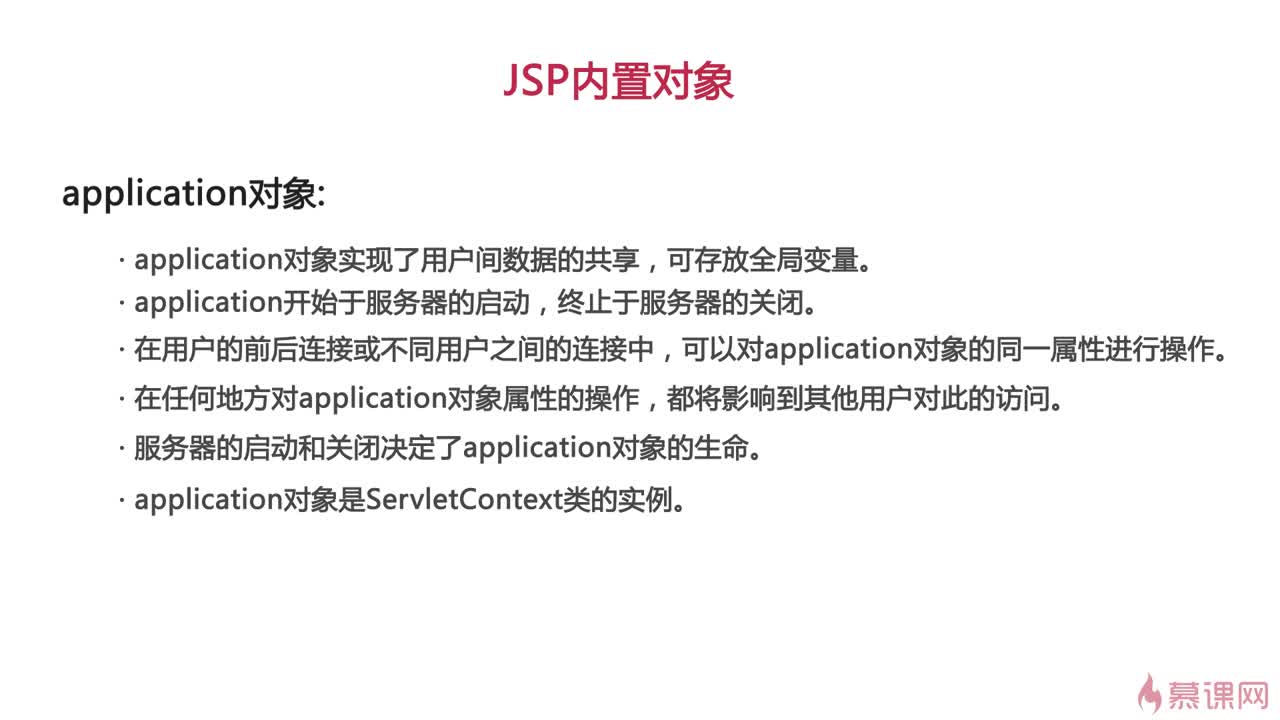
%>

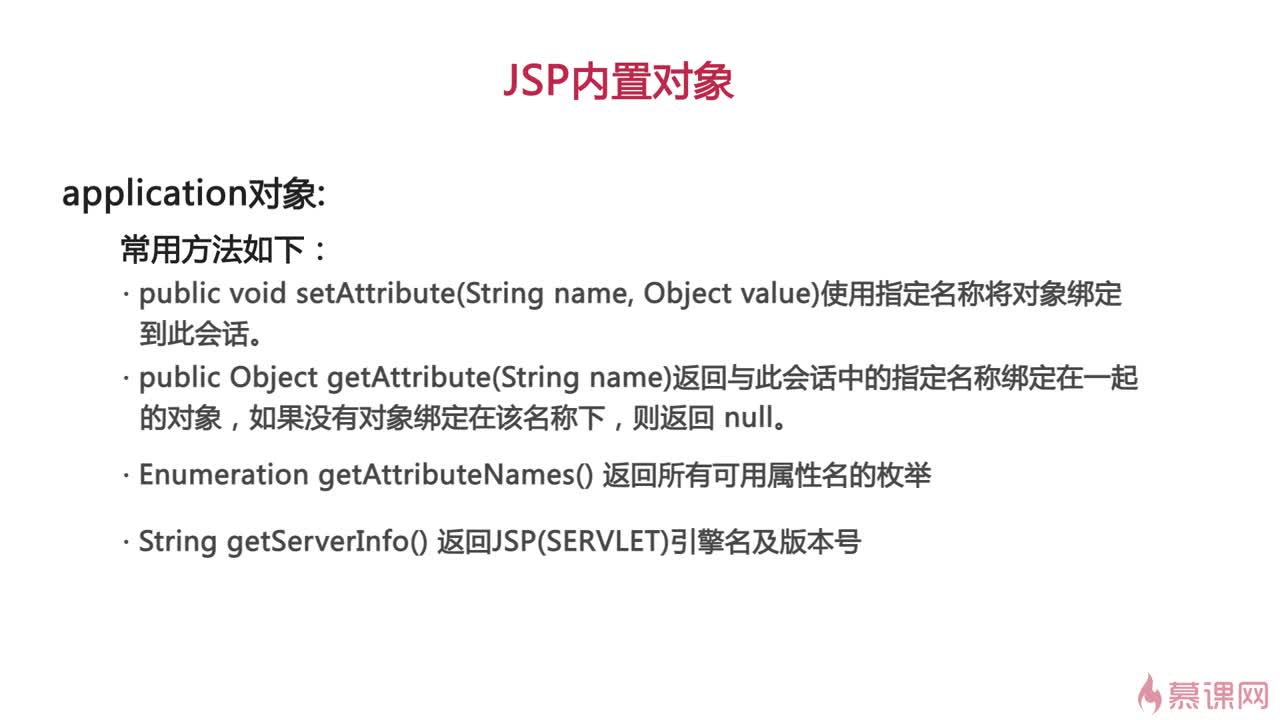
Session创建时间：<%= sdf.format(d) %>

10、Session的生命周期

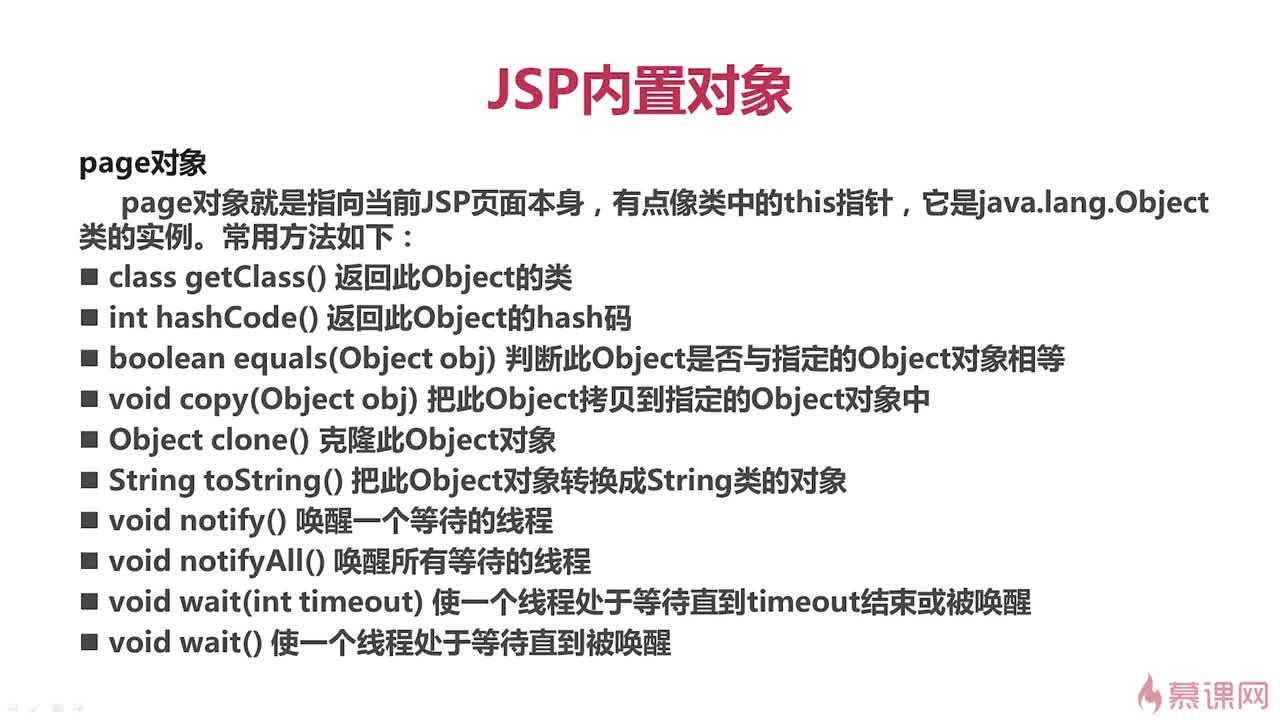
* 创建：首次访问一个jsp或servlet时，服务器会为当前会话设一个sessionId，客户端向服务器发送请求时，将此sessionId携带过去，服务端对sessionId进行校验。
* 活动：在会话中通过超链接打开新的页面属于同一会话；只要当前会话的页面没有全部关闭，重新打开浏览器的新窗口访问同一项目资源仍属于同一会话；除非本次会话的所有页面被关闭，再重新访问jsp或servlet时会创建新的会话。（原有会话依然存在，只是不能再访问。）
* 销毁：
* session.invalidate()方法；
* 超时:
* session.setMaxInactiveInterval(time)秒;
* 在WEB.XM文档中设置，单位是分钟。
* 关闭（重启）服务器。

11、application对象



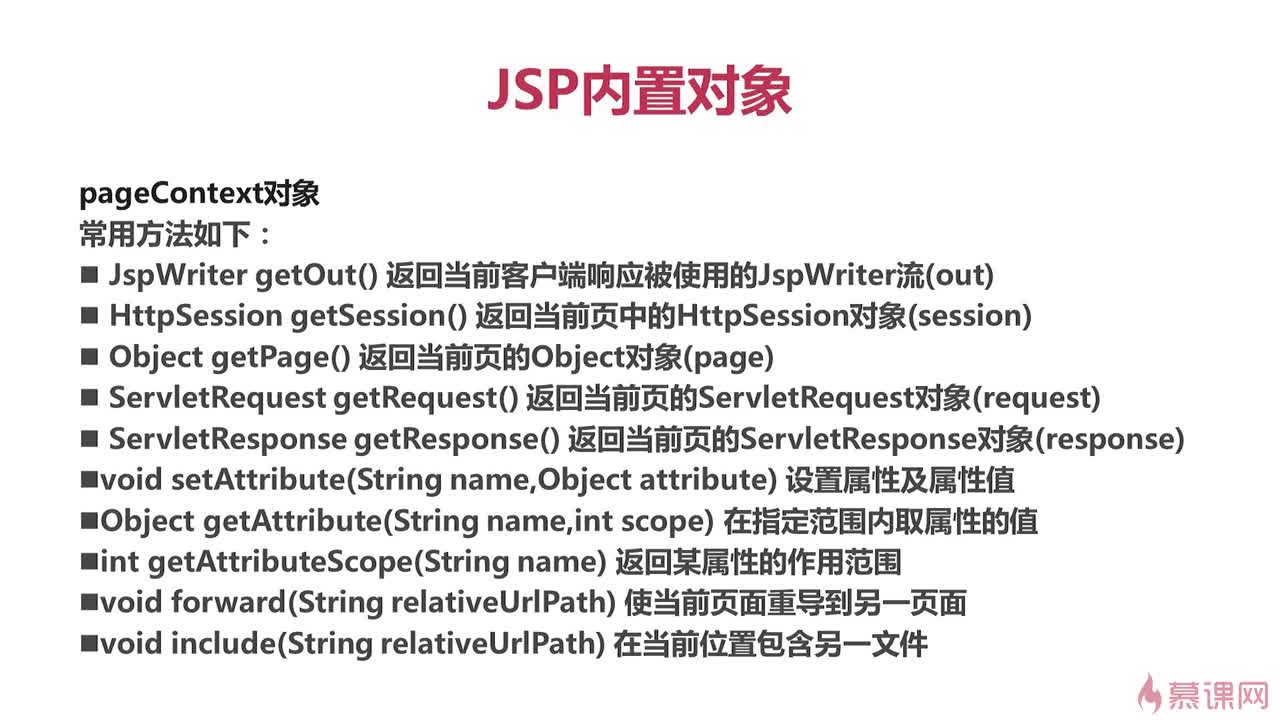


12、Page对象

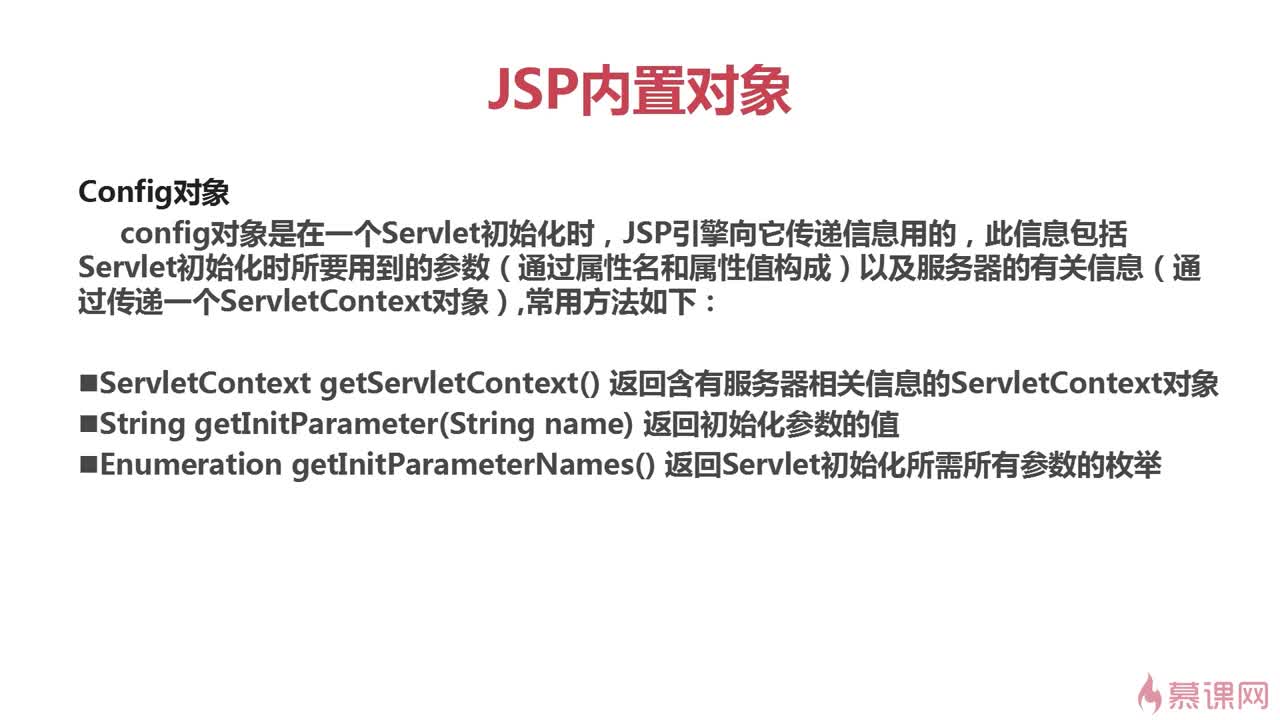


13、pageContext对象

* 提供了对JSP页面内所有的对象及名字空间的访问。
* 可以访问到本页所在的Session，也可以取本页面所在的application的某一属性值。
* 相当于页面中所有功能的集大成者。
* pageContext对象的本类名叫做pageContext。

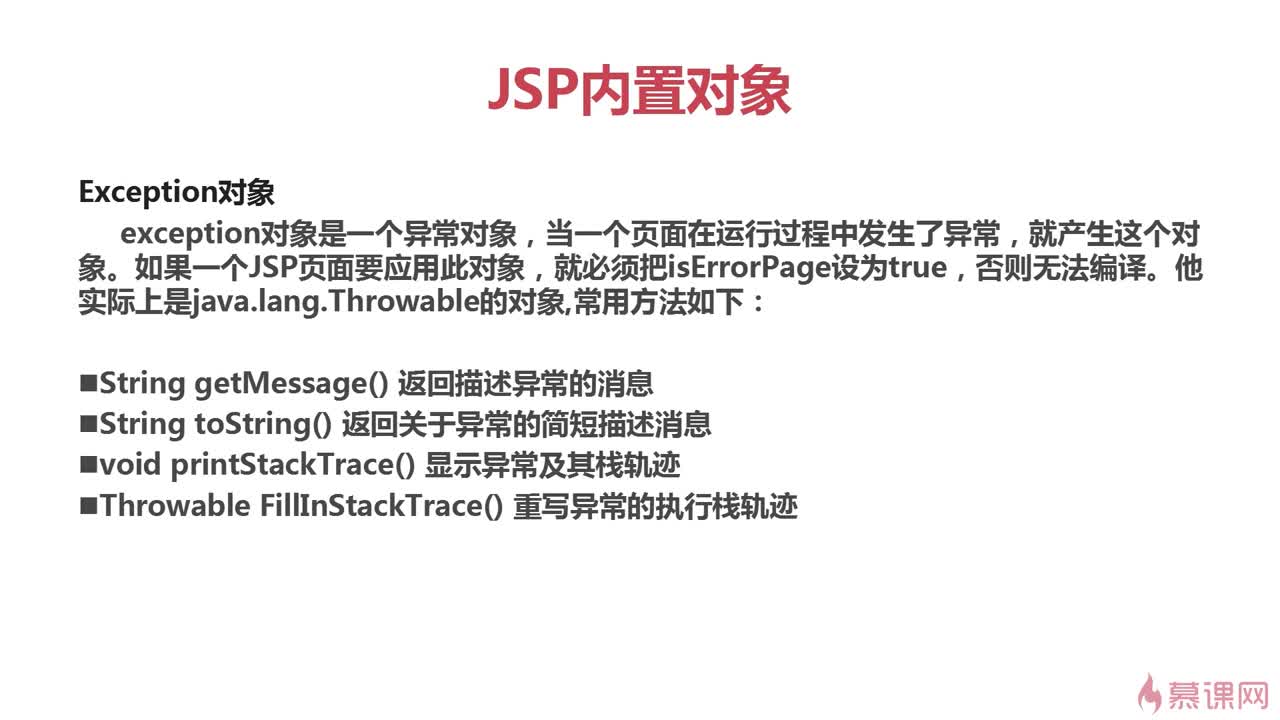


14、config对象



15、exception对象

* 产生异常页面的page指令中要添加 errorPage=”exception”
* 处理异常页面的page指令中要添加 isErrorPage=”true”



五、JavaBean

1、简介

* 就是符合某种特定规范的java类。好处：解决代码重复编写，减少代码冗余，功能区分明确，提高了代码的维护性。
* JavaBean设计原则---里面不含有其他方法
* 公有类
* 无参数的公有构造方法
* 属性私有
* 对每个属性都有getter和setter方法

2、JSP动作元素（action element）---是一个标签

* JSP动作元素为请求处理阶段提供信息。动作元素遵循XML语法，有一个包含元素名的开始标签，可以有属性，可选的内容、与开始标签匹配的结束标签



3、创建JavaBean---普通的方式

* 像使用普通java类一样

1、创建一个web project项目。

2、在src文件下，创建一个包，在包中创建一个类，满足设计原则即可

3、在index.jsp页面中通过import导入之前创建的类（import="包名.类名"）

4、通过使用new创建Javabean实例(创建对象)

5、使用set方法赋值

6、使用get方法取值

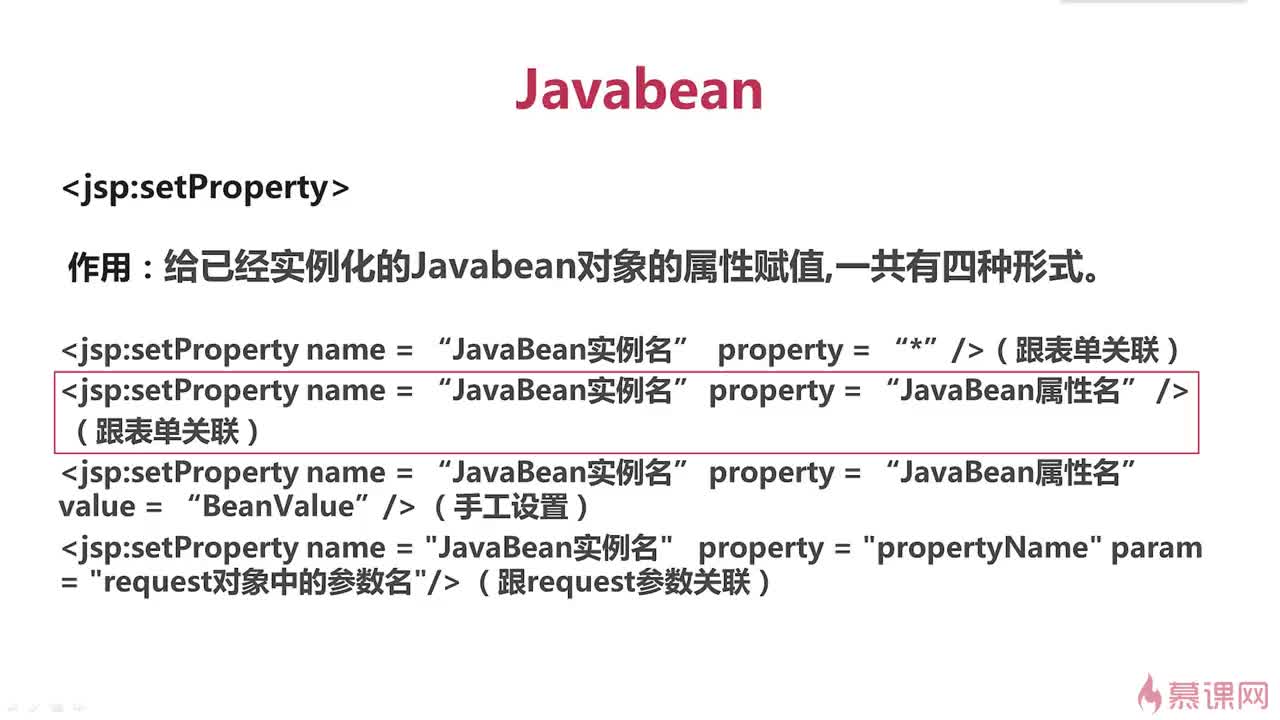
4、JSP标签使用JavaBean-----useBean

* 作用：在jsp页面中实例化或者在指定范围内使用javabean。

<jsp:useBean id="标识符" class="java类名" scope="作用范围"/>

5、JavaBean--<jsp:setProperty>

* 需要表单提交属性：<jsp:setProperty name="Javabean实例名" property="\*"/>： 与之前页面提交的表单的【所有属性进行匹配】，若有与javabean类属性名相同的，则进行赋值。比如表坛提交了username="xxx"，而javabean里有username属性，则对其进行赋值
* 需要表单提交属性<jsp:setPropery name="Javabean实例名" property="javabean属性名" />：对【指定的属性进行匹配】---部分属性
* 手动设置属性：<jsp:setProperty name = "Javabean实例名" property="Javabean属性名" value="BeanValue" />
* 与request参数有关,通过url传参 ：<jsp:setProperty name="Javabean实例名" property="propertyName" param="request对象中的参数名" />



* + 根据表单自动匹配所有的属性

<jsp:setProperty name="myUsers" property="\*"/>

* + 根据表单自动匹配相应的属性（部分匹配）

<jsp:setProperty name="myUsers" property="username"/>

* + 与表单无关，通过手工赋值给属性

<jsp:setProperty name="myUsers" property="username" value=“Lisa”/>

* + 通过url传参数给属性赋值

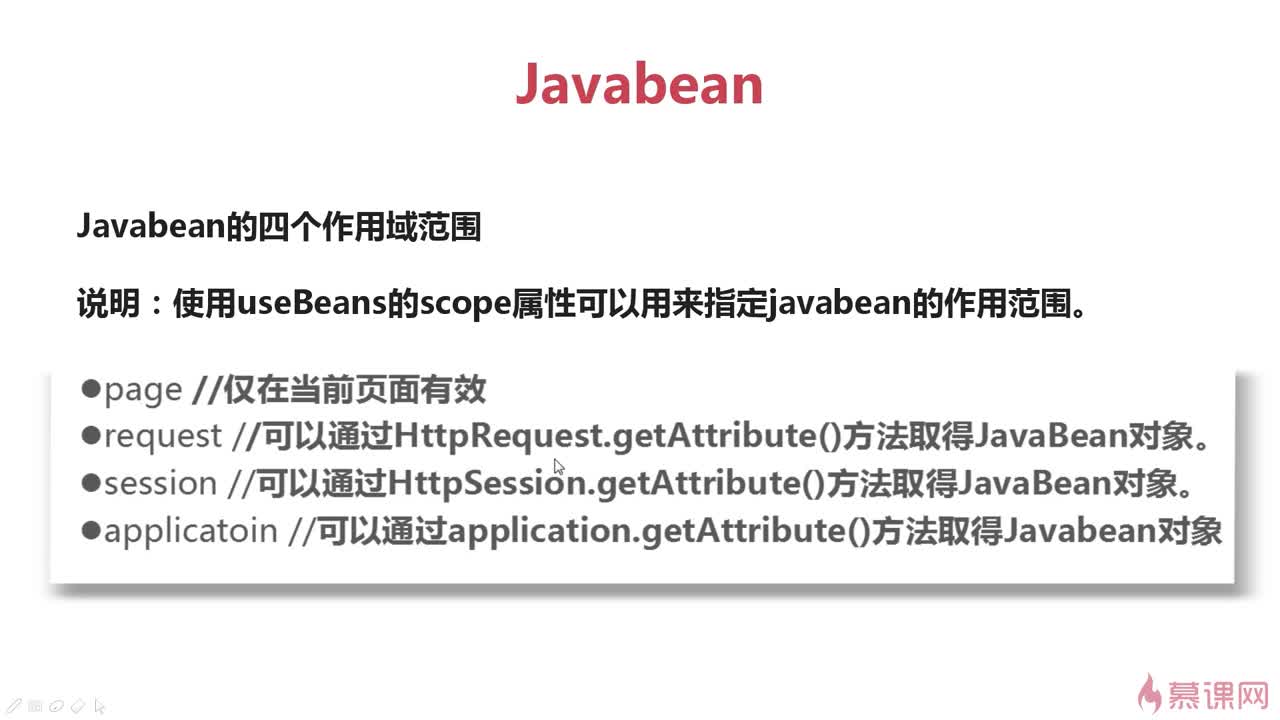
<jsp:setProperty name="myUsers" property="username"/>

<jsp:setProperty name="myUsers" property="password" param=“mypass”/>

6、<jsp:getProperty/>

作用：获取指定javabean对象的属性值。

〈jsp:getProperty name="javabean实例名" property="javabean属性名"〉



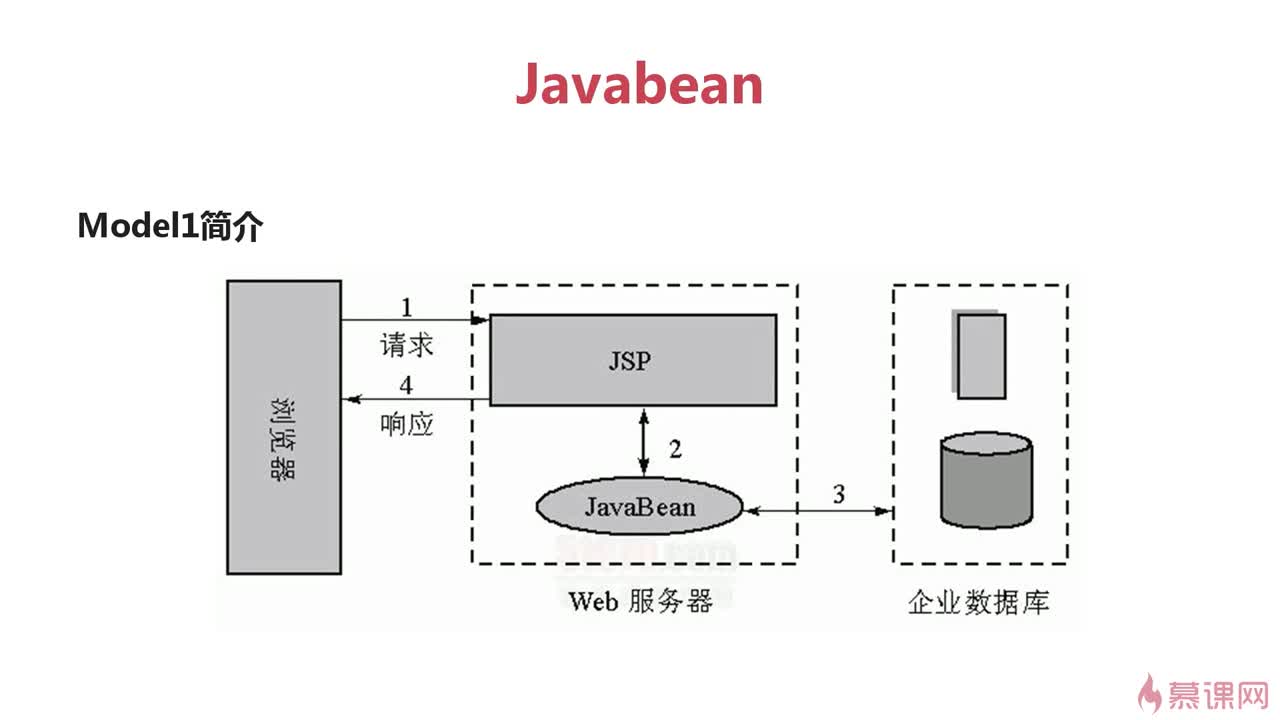
7、JavaBean四个作用域范围

语法：

<jsp:useBean name=”myusers” class=”com.po.User” scope=”page”/>

8、Model 1

* Model1分为三层：
* 界面层：由JSP页面进行与客户端浏览器的交互，可以直接调用JavaBeans
* 业务逻辑层：由JavaBeans进行处理业务逻辑，封装数据，并与数据库层进行交互
* 数据库层：底层数据库，可被javaBeans访问
* 避免了JSP页面直接进行业务逻辑处理所带来的维护性，扩展性差的弱点.体现了分层设计的思想



六、JSP状态管理

1、http协议的无状态性

* 无状态是指，当浏览器发送请求给服务器的时候，服务器响应客户端请求。但是当同一个浏览器再次发送请求给服务器的时候，服务器并不知道它就是刚才那个浏览器

2、保存状态的机制

* session(在服务器保存的内置对象) 和 cookie(保存在客户端的文本记录)
* 简介
* cookie是一种保存在客户端的用户信息的文本记录。
* 经典用例：判断注册用户是否已经登录成功。

用户的“购物车”之类需要在不同的页面之间切换并且保持状态和信息。

* cookie的作用是什么？
* cookie可以实现用户的跟踪
* cookie可以记录用户的访问记录和习惯
* cookie可以实现简化登录
* 安全风险：用户的信息可能会被泄漏。

3、JSP中创建Cookie

* 创建cookie对象

Cookie cookie = new Cookie(String key,Object value);

* 添加cookie对象

response.addCookie(cookie);

* 获取cookie,返回值是一个数组

Cookie[] cookies = request.getCookies();

* 常用的方法
* void setMaxAge(int age) 设置cookie的有效期，时间以秒为单位
* void setValue(String value) 创建cookie后，对cookie进行赋值
* String getValue() 获取cookie的值
* int getMaxAge() 获取cookie的有效时间，时间以秒为单位
* String getName() 获取cookie的名字

4、解决Cookie无法保存中文字符串的问题

* java.net包下的URLEncoder类来进行编码

public static encode(String string, String enco);

* 2.URLDecoder类进行解码

URLDecoder.decode(String string, String enco);

* 3.如果在登录界面把username和password的值设为了null，那么在登陆框就会出现null

最好是设为""

* 4.查找Cookie时,是按照k值进行查找的

-----------------------------------

使用URLEncoder（在java.net包下）解决无法在Cookie当中保存中文字符串问题：

String username = URLEncoder.encode(request.getParameter("username"),"utf-8");//编码，防止中文乱码

String password = URLEncoder.encode(request.getParameter("password"),"utf-8");//编码，防止中文乱码

5、session 与 cookie 对比

* session和cookie的区别：

1.session保存在服务器端，而cookie保存在客户端

2.session保存重要的信息，而cookie中保存不重要的信息

3.session可以随着会话的结束而销毁会话，但是cookie可以长期保存

4.session保存的是Object类型的，但是cookie中保存的是String类型的

共同点：都可以设置生命周期，都可以保存用户信息



七、指令与动作

1、include指令简介

* jsp中的指令是用来设置jsp页面的相关属性，如编码方式和脚本语言等。
  + page指令是为容器提供当前页面的使用说明，一个页面可以包含多个page指令
  + include 指令是包含指定文件，如jsp,Html等
  + taglib自定义标签是指引入一个自定义标签的集合的定义，包括库路经、自定义标签等
* 举例说明如下：
  + 1.<%@ page language="java" import="java.text.\* contentType="text.html;charset=utf-8" %>
  + 2.<%@ include file="date.jsp" %>在另一个文件中包含了date.jsp文件的内容
  + 3. 目前还没有找到例子。
* include指令

语法：<%@ include file="URL"%>

2、include动作指令

* include 动作的格式：

<jsp:include page="date.jsp" flush="true" />

include动作元素是一个包含了date.jsp页面的页面，是否从缓冲区中获取页面，为true

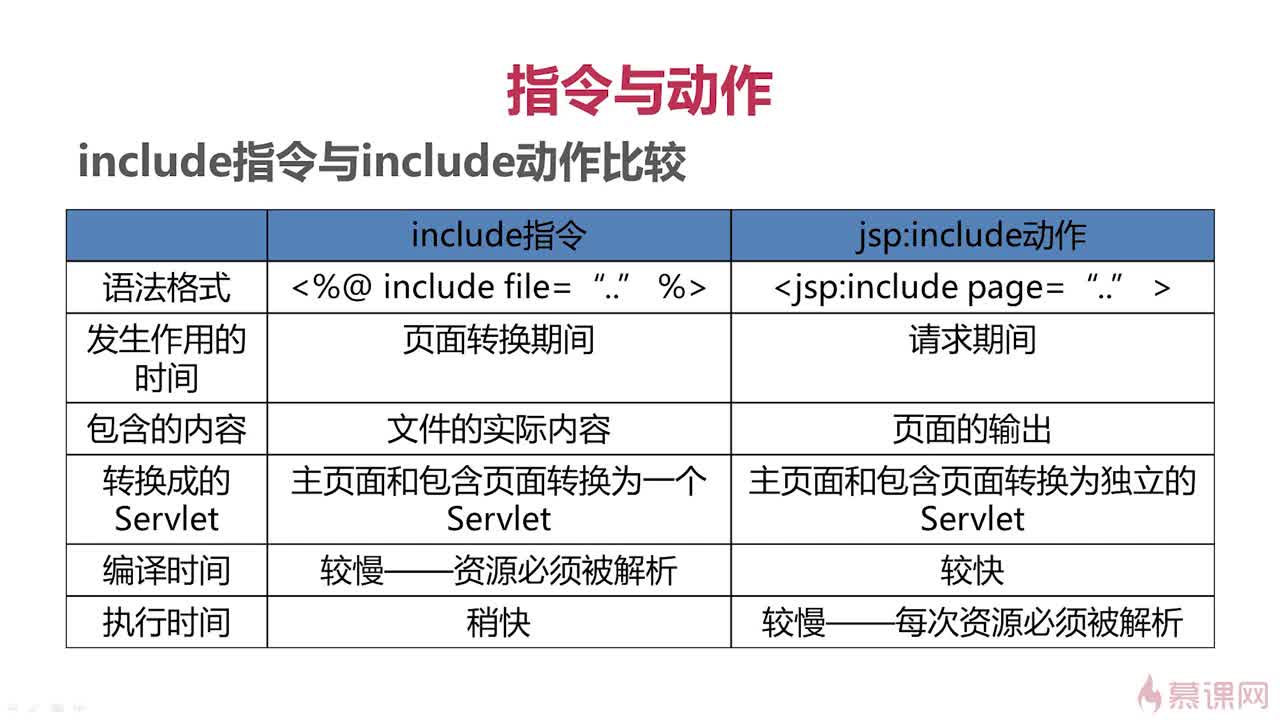
3、include指令和include动作比较

* include指令生成一个字节码文件。

include动作生成两个字节码文件。

* 页面内容经常变化适合用include动作。

页面内容不经常变化适合用include指令。



4、forward动作

* forward动作实现服务器内部跳转

格式：<jsp:forward page="url"/>

相当于：

request.getRequestDispatcher("url").forward(request,response)—请求转发，URL地址不发生变化

5、param动作

* 使用param动作指令可以对表单传过来的参数进行修改

<jsp:forward page="">

<jsp:param value="" name="">

</jsp:forward>

两个动作一般是配合使用的

6、其他动作元素

* <jsp:element>动作元素、<jsp:attribute>动作元素和<jsp:body>动作元素

用于动态生成XML元素

xml元素包含标签、属性和内容

<jsp:element name="标签名"> 生成xml标签

<jsp:attribute name="属性名">属性值</jsp:attribute> 生成xml标签属性

<jsp:body>内容</jsp:body> 生成xml标签内容

* 示例:

<jsp:element name="xmlElement">

<jsp:attribute name="xmlElementAttr">

xmlElementAttrVal

</jsp:attribute>

<jsp:body>

xmlContent

</jsp:body>

</jsp:element>

* 动态产生的结果为:

<xmlElement xmlElementAttr="xmlElementAttrVal">

xmlContent

</xmlElement>