**Java EE是十三种核心技术的综合应用：Java**

数据库连接（JDBC）

命令和目录接口（JNDI, Java Naming and Directory Interface）

企业Java Beans（EJB）

JavaScript Pages(Jsp)

Java Servlet

Java IDL/CORBA

Java 事务体系（JTA）/事务服务（JTS）

JavaMail和JavaBeans激活架构（JAF）

Java信使服务（JMS）

扩展标记语言（XML）

**Web框架：Struts、hibernate、spring**

“我亦无他，惟手熟尔”

**JSP的概述（基本语法）：JavaServer Pages**

jsp = html + java片段 + jsp标签(语法) + javascript(css)

Servlet 缺点：界面布局难

Jsp 技术：界面好做、且其以Servlet为基础

**Jsp(前台：界面) + JavaBeans(后台：逻辑层)** 的基础上再结合 **Servlet**, 就会构成 **MVC 标签**

<http://www.cnblogs.com/rocomp/p/4822419.html>

**Jsp 特点**

JSP这门技术的最大的特点在于，写jsp就像在写html，但它相比**html**而言，html只能为用户提供静态数据，而Jsp技术允许在页面中嵌套java代码，为用户提供动态数据。相比**servlet**而言，servlet很难对数据进行排版，而jsp除了可以用java代码产生动态数据的同时，也很容易对数据进行排版。

不管是JSP还是Servlet，虽然都可以用于开发动态web资源。但由于这2门技术各自的特点，在长期的软件实践中，人们逐渐把servlet作为web应用中的**控制器组件**来使用，而把JSP技术作为数据显示模板来使用。

其原因为，**程序的数据通常要美化后再输出**：让jsp既用Java代码产生动态数据，又做美化会导致页面难以维护。让servlet既产生数据，又在里面嵌套html代码美化数据，同样也会导致程序可读性差，难以维护。因此最好的办法就是根据这两门技术的特点，让它们各自负责各的，servlet只负责响应请求产生数据，并把数据通过转发技术带给jsp，数据的显示jsp来做。

**1、Jsp 标签 - jsp基本语法**

**1）指令元素**

用于从Jsp发送一个信息到容器，比如设置全局变量、文字编码、引入包等

**a. page指令**

page指令即页面指令，可以定义在整个JSP界面范围内有效的属性。

//声明是一个jsp文件，并且支持中文

<%@ page contentType="text/html;charset=gb2312"%>

**常用属性：**

language="xx"：jsp中嵌入的代码是xx，通常java

import="包.\*.类名.." 在该jsp页面引入包或者某个具体的类

session=[true|false] 是否在jsp页面给你获取session对象，默认为true

buffer=[none|8k|指定大小] 给out对象使用的缓冲区是多大，默认是8k

autoFlash=[true|false] 当buffer满后，是否自动刷新到浏览器，默认是true

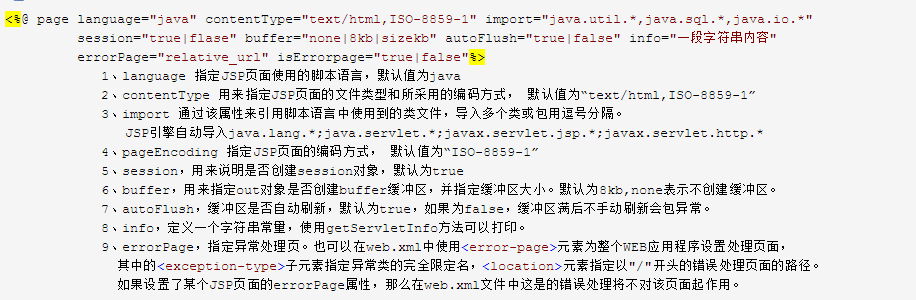
isThreadSafe=[true|false] 默认为true，表示该**jsp的线程安全**由程序员控制，false则对应的servlet将实现线程安全接口

pageEncoding="编码方式"

contentType="text/html;charset=gb2312"

errorPage=”ralativeURL”：该属性用来指定一个当前页面出现异常时所要调用的页面。如果属性值为“/”开头的路径，则将在当前应用程序的根目录下查找文件；否则，将在当前页面的目录下查找文件。

isErrorPage=[true|false]：将该属性值设为true，此时在当前页面中可以使用exception异常对象。若在其他界面中errorPage属性指定了该界面，则当页面出现异常时，会跳转到该页面，并可在该页面中通过exception对象输出错误信息。相反，如果属性值为false，则在当前页面中不能使用exception对象。属性默认值为false。



**举例：**

<%@ page contentType="text/html;charset=gb2312"%>

这个指令的作用就相当于response.setContentType("text/html;charset=gb2312");

但是这个指令和 <%@ page pageEncoding="gb2312"%>的区别是：

* pageEncoding是jsp文件本身的编码
* contentType的charset是指服务器发送给客户端时的内容编码

JSP要经过两次的“编码”，第一阶段会用**pageEncoding**，第二阶段会用utf-8至utf-8，第三阶段就是由Tomcat出来的网页， 用的是**contentType**。

* 第一阶段是jsp编译成.java，它会根据pageEncoding的设定读取jsp，结果是由指定的编码方案翻译成统一的UTF-8 JAVA源码（即.java），如果pageEncoding设定错了，或没有设定，出来的就是中文乱码；
* 第二阶段是由JAVAC的JAVA源码至java byteCode的编译，不论JSP编写时候用的是什么编码方案，经过这个阶段的结果全部是UTF-8的encoding的java源码；
* JAVAC用UTF-8的encoding读取java源码，编译成UTF-8 encoding的二进制码（即.class），这是JVM对常数字串在二进制码（java encoding）内表达的规范。
* 第三阶段是Tomcat（或其的application container）载入和执行阶段二的来的JAVA二进制码，输出的结果，也就是在客户端见到的，这时隐藏在阶段一和阶段二的参数contentType就发挥了功效

**b. include指令**

该指令用于当前的JSP界面中，在当前使用该指令的位置嵌入其他的文件，如果被包含的文件中有可执行代码，则显示代码执行的结果。引入一个文件，通常是jsp文件，**jsp引擎将两个jsp文件翻译成一个servlet文件**

<%@ include file="filename（文件的绝对路径或相对路径）"%>

**c. taglib指令**

这个指令允许Jsp页面使用自定义标签，使用格式如下：

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

**uri属性：**该属性指定了标签描述符，该描述符是一个对**标签描述文件(\*.tld)的映射**。在tld标签描述文件中定义了该标签库中的各个标签名称，并为每个标签制定一个处理类。

**prefix属性：**该属性指定一个在页面中使用由uri属性**指定的标签库的前缀**。前缀不能命名为jsp、jspx、java、javax、sun、servlet和sunw。

如：

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:set var="name" value="hello">

即通过<c:set>标签将hello值赋给了遍历name。

**2）脚本元素**

在一定程度上可以理解为java片段

**a. 脚本程序script**

<% java程序片段 %>

**b. JSP表达式**

**注意：表达式中不能有分号**

<%= 变量或可以返回值的方法或Java表达式 %>

<%=java表达式%>

例如：

<% String name="login.jsp" %>

<form action="<%=name%>"></form>

生成如下HTML代码：

<form action="login.jsp"></form>

**c. declaration声明标识：定义变量**

<%!变量声明%>

比如：<%! int count = 10%>

<%!函数声明%>

比如：

<%! int why(int num){ //定义servlet成员函数

int result = 0;

for(int i=0;i<num;i++){

esult += i;

}

}

%>

**注：**

a. 在**java片段定义变量**时，该变量则是**局部变量**

b. 而在声明中定义的变量，则该变量是jsp对应的servlet成员变量，即**全局变量**

c. **jsp本质是一个类函数**

**3）动作元素**

动作元素是使用xml语法写的，是jsp规定的一系列标准动作，在容器处理jsp时，或当容器遇到动作元素时，就执行相应的操作

**a. <jsp:useBean>、<jsp:setProperty>、<jsp:getProperty>**

* <jsp:useBean>：创建一个javabean实例，并且通过属性的设置可以将实例存储到JSP中的指定范围内
* **<jsp:setProperty>**：给一个javabean实例设置初始值
* **<jsp:getProperty>：**取得一个javabean实例的成员变量

这三个标签是一起用来操作bean对象的，<jsp:useBean>是用来初始化bean对象的，<jsp:setProperty>标签是用来设置bean对象中的属性值，<jsp:getProperty>标签是用来获取bean对象中的属性值的，例子：



同时我们定义了一个Bean对象：com.weijia.domain.Person，其中有一个name属性(一定要有get/set方法才叫属性)。对于标签<jsp:useBean>他可以指定在哪个域中创建这个bean对象，他的规则是首先在这个域中查找有没有这个对象，没有就创建，有就直接拿来使用。

对于<jsp:setProperty>标签，可以直接使用value属性设置属性的值，这里面他内部是有一个**类型转换**的，可以将字符串值转化成8中基本类型的值，其他对象类型的转化是会报错的，比如我们现在Person类中有一个属性birthday是Date型的，那么我们这里就不能直接写成这样：



这样系统会报类型转化错误，

当然我们可以直接使用脚本表达式给这个属性值传递一个Date类型的对象，这样就不会又错了：



**b. <jsp:param>**

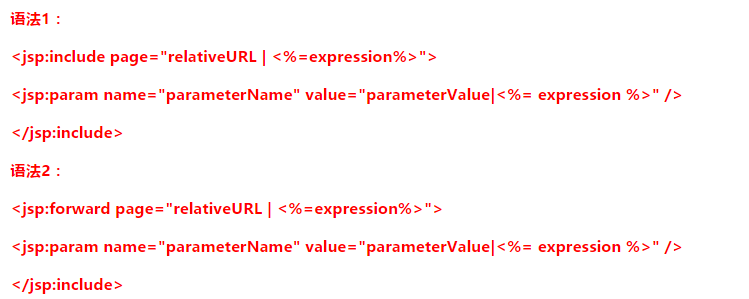
//给一个jsp设置参数，常常与<jsp:include>结合使用

<jsp:include file="info.jsp">

<jsp:param name="parameter1" value="parameterValue"/>

<jsp:param name="parameter2" value="parameterValue"/>

</jsp:include>



<jsp:param>标签的name属性用于指定参数名，value属性用于指定参数值。在<jsp:include>和<jsp:forward>标签中可以使用多个<jsp:param>标签来传递多个参数。

**c. <jsp:include>**

//动态引入另外一个文件

<jsp:include page="xx.jsp"></jsp:include> 动态引入

<%@ include file="" %> 静态引入

**相同点：**把一个文件引入到另一个文件

**区别：**静态引入 --> 把两个jsp文件翻译成一个servlet，所以被引入文件不要包含<body><html>

动态引入 --> 把两个jsp文件分别翻译，所以被引入的jsp可包含有<body><html>

**d. <jsp:plugin>**

//使用插件，可以再页面中插入Java小应用程序或者javaBean，在客户端运行

**e. <jsp:forward>**

//转发

属性：page="url" --> 直接跳转到url页面

在开发jsp过程中，通常把jsp放入WEB-INF目录，防止用户直接访问这些jsp文件

在WebRoot下存在一个入口页面，主要是转发即<jsp:forward page="url" />

**f. <jsp:fallback>**

//是**<jsp:plugin>**的子标识，当使用**<jsp:plugin>**标识加载Java小应用程序或者javaBean失败时，可通过**<jsp:fallback>**标识向用户输出提示信息

**4）JSP九大内置对象**

每个JSP 页面在第一次被访问时，WEB容器都会把请求交给JSP引擎（即一个Java程序）去处理。JSP引擎先将JSP翻译成一个\_jspServlet(实质上也是一个servlet) ，然后按照servlet的调用方式进行调用。由于JSP第一次访问时会翻译成servlet，所以第一次访问通常会比较慢，但第二次访问，JSP引擎如果发现JSP没有变化，就不再翻译，而是直接调用，所以程序的执行效率不会受到影响。

JSP引擎在调用JSP对应的**\_jspServlet**时，会传递或创建9个与web开发相关的对象供\_jspServlet使用。JSP技术的设计者为便于开发人员在编写JSP页面时获得这些**web对象**的引用，特意定义了9个相应的变量，开发人员在JSP页面中通过这些变量就可以快速获得这9大对象的引用。

上面其实很多对象我们都接触过了，之前介绍servlet的时候都介绍过了，关于那个exception是个异常对象，只有当我们的Jsp页面中抛出异常的时候，才会有这个对象的产生，否则是不会传递这个对象的，至于page对象，这个很简单就是当前对象，即jsp翻译后的servlet对象，那么下面就来详细解释一下out对象和pageContext对象了。

**表单的两种提交方式：get与post：**

* 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，安全性低的数据。比如：搜索、查询等功能
* post：将用户提交的信息**封装在HTML header**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。

**a. out - JspWriter**

out对象在JSP页面中无需创建就可以使用，它的作用是用来**向客户端输出**。

其中<%=…%>与out.print()功能是相同的！它们都是向客户端输出，例如：

<%=s1%> 等同于 <%out.print(s1);%>

<%=”hello”%>等同于<%out.print(“hello”);%>，也等同于直接在页面中写hello一样。

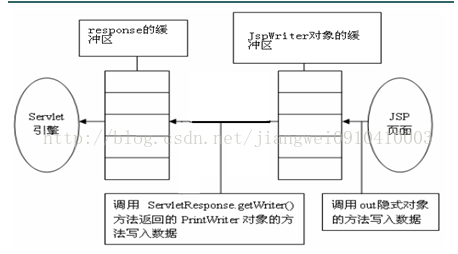
//向客户端输出数据：字节流

out.println("");

out对象是通过调用**pageContext对象的getOut方法**返回的，其作用和用法与**ServletResponse.getWriter方法返回的PrintWriter对象非常相似**。JSP页面中的out隐式对象的类型为**JspWriter**，JspWriter相当于一种带缓存功能的PrintWriter，设置JSP页面的page指令的buffer属性可以调整它的缓存大小，甚至可以关闭它的缓存。

只有向out对象中写入了内容，且满足如下任何一个条件时，out对象才去调用ServletResponse.getWriter方法，并通过该方法返回的**PrintWriter对象**将out对象的缓冲区中的内容真正写入到Servlet引擎提供的缓冲区中：

* 设置page指令的buffer属性关闭了out对象的缓存功能
* out对象的缓冲区已满
* 整个JSP页面结束



**注：**out.print("aaa"); response.getWriter().write("bbb"); 同时输出数据时，浏览器会先打印出bbb，再打印aaa。原因就是out对象是JspWriter，是带有缓冲的。

**b. request - HttpServletRequest**

//接收客户端的http请求,类似于servlet中的 HttpServletRequest

客户端的请求信息被封装在request对象中，通过它才能了解到客户端的需求，然后做出响应。它是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：**

String getParameter(String name)：返回指定参数的参数值

String[] getParameterValues()：返回指定参数的所有值

void setAttribute()：存储此请求中的属性

object getAttribute()：返回指定属性的属性值

String getContentType()：得到请求体的MIME类型

String getProtocol()：返回请求用的协议及版本号

String getServerName()：返回接受请求的服务器主机名

int getServerPort()：返回服务器接受此请求的端口号

String getCharacterEncoding()：返回字符编码方式

void setContentEncoding()：设置请求的字符编码方式

int getContentLength()：返回请求体的长度（以字节数）

String getRemoteAddr()：返回发送此请求的客户端IP地址

String getRealPath(String path)：返回一个虚拟路径的真实路径或相对路径的绝对路径

StringgetContextPath()：返回上下文路径，即项目的根目录

**中文乱码问题：**request.setCharacterEncoding("UTF-8");

**URL中中文乱码问题：**Tomcat的/conf/server.xml 的添加<Connector URIEncoding="UTF-8">属性。

**c. response - HttpServletResponse**

//封装jsp产生的回应,类似于servlet中的 HttpServletResponse

response对象包含了响应客户请求的有关信息，但在JSP中很少直接用到它。它是HttpServletResponse类的实例。**response对象具有页面作用域**，及访问一个页面时，该页面内的response对象只能对这次访问有效，其他的response对象对当前页面无效。

**常用方法：**

String getCharacterEncoding()：返回响应用的是何种字符编码

void setContentType()：设置响应的MIME类型，一般为"text/html, charset=UTF-8"

PrintWriter getWriter()：返回可以向客户端输出字符的一个对象("注意比较：PrintWriter与内置out对象的区别，PrintWriter对象的其输出总是提前于内置out对象，或者在out中手动flush")

sendRedirect(loaction)：重新定向客户端的请求

* **请求重定向redirect：**是客户端行为，response.sendRedirect(),从本质上将**等同于两次请求**，前一次的请求对象不会保存，地址栏的URL地址会改变。
* **请求转发forward：**是服务器行为，request.getRequestDispatcher().forward();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。

**d. session - HttpSession**

//用于保存用户信息，跟踪用户行为; 被**浏览器独占**的某一内存空间

* session表示客户端与服务器的**一次会话**，Web中的session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。
* 在服务器的内存中，为不同的用户保存着不同的session。
* session对象是一个JSP内置的对象。
* session对象在第一个JSP页面被装载是自动创建，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，肯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。

**常用方法：**

long getCreationTime();返回session创建时间

String getId();返回session创建时JSP引擎为它设的**唯一ID号**

Object setAttribute();使用指定名称将对象绑定到此会话

Object getAttribute();返回与此会话中的指定名称绑定在一起的对象，没有返回null

String[] getValueNames();返回一个包含此session中所有可用属性的数组

int getMaxInactiveInterval();返回两次强求间隔多长时间此session被取消（单位秒）

void setMaxInactiveInterval();设置session存活时间。

**session的生命周期：**

* **创建：**当客户端第一次访问某个jsp或者Servlet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都关闭后，在重新访问某个jsp或者Servlet将会创建新的会话(但此时原有会话还存在，这个就的SessionId仍然存在于服务端，只不过再也没有客户端会携带它然后教育服务端校验，直到该会话超时。)。
* **销毁：**有三种销毁方式

1、调用了session.invalidate()方法

2、Session**过期（超时）**可以在web.xml中配置Session超时时间<session-config><session-timeout>1单位是分钟

3、服务器重新启动

**e. application - servletContext**

//多个用户共享该对象，可以做计数器，相当于servlet的ServletContext

* 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之间的连接中，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，都将影响到其他用户对此的访问。
* 服务器的启动和关闭决定了application对象的生命。
* application对象是ServletContext类的实例。

//常用方法

* void setAttribute()：使用指定名称将对象绑定到此会话
* Object getAttribute()：返回与此会话中的指定名称绑定在一起的对象，如果没有，返回null
* Enumeration getAttributeNames()：返回所有**可用属性名的枚举**
* String getServerInfo()：返回JSP(Servlet)引擎名及版本号

**f. pageContext**

//代表jsp页面的上下文

声明周期比较短，属性值只在本页面使用

pageContext对象是JSP技术中最重要的一个对象，它代表JSP页面的运行环境，这个对象不仅封装了对其它8大隐式对象的引用，它自身还是一个域对象(之前我们介绍了三个域对象：ServletContext,Session,Request)，这个**域对象的生命周期最短，作用域最小**，他的作用域就是当前的jsp页面，当然它可以用来保存数据。并且，这个对象还封装了web开发中经常涉及到的一些常用操作，例如引入和跳转其它资源、检索其它域对象中的属性等。

* getException方法返回exception隐式对象
* getPage方法返回page隐式对象
* getRequest方法返回request隐式对象
* getResponse方法返回response隐式对象
* getServletConfig方法返回config隐式对象
* getServletContext方法返回application隐式对象
* getSession方法返回session隐式对象
* getOut方法返回out隐式对象

代表各个域的常量

* PageContext.APPLICATION\_SCOPE
* PageContext.SESSION\_SCOPE
* PageContext.REQUEST\_SCOPE
* PageContext.PAGE\_SCOPE

下面这个方法是非常重要的，因为这个方法是在所有的域中查找数据，查找顺序是：pageContext->request->session->ServletContext，如果查找不到相应的数据的话，就返回一个空字符串，这个方法和之后要说到的el表达式的功能是一样的：

**findAttribute方法 （\*重点，查找各个域中的属性）**

**#获取数据**

1）默认情况下，从page域获取

pageContext.getAttribute("name")

2）可以从四个域中获取数据

pageContext.getAttribute("name",域范围常量)

域范围常量:

PageContext.PAGE\_SCOPE

PageContext.REQUEST\_SCOPE

PageContext..SESSION\_SCOPE

PageContext.APPLICATION\_SCOPE

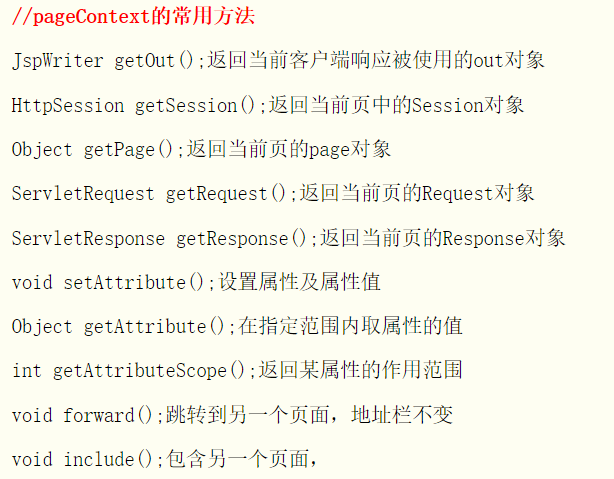
3）自动在四个域中搜索数据

pageContext.findAttribute("name");

顺序： page域 -> request域 -> session域- > context域（application域）

**4个域对象：**

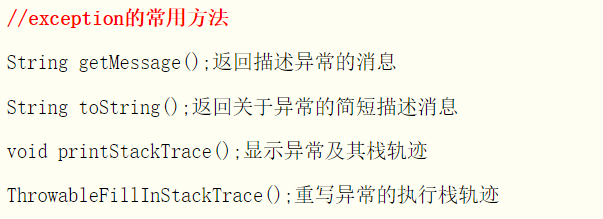
* pageContext（称之为page域）
* request（称之为request域）
* session（称之为session域）
* servletContext（称之为application域）



**g. exception**

//代表运行时的一个异常

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，就必须报isErrorPage设为true，否则无法编译。它实际上是java.lang.Throwable的对象。页面要想在有异常时得到处理，就要指定异常处理页面<% page errorPage="exception.jsp"%>



**h. page** – **this**

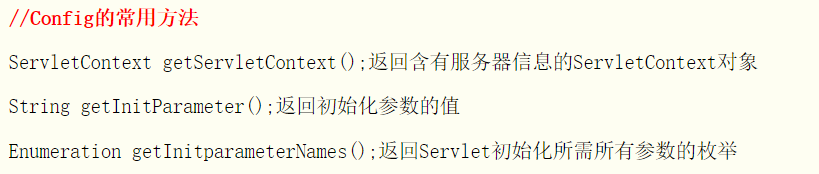
//代表jsp这个实例本身（使用较少），类似于servlet中的this

page对象就是指向当前jsp页面本身，有点像类中的this指针，它是java.lang.Object类的实例。常用方法就是Object类的成员方法。

**i. config** - **servletConfig**

//代表jsp对应的servlet的配置，可以得到web.xml中的参数，类似servlet的ServletConfig

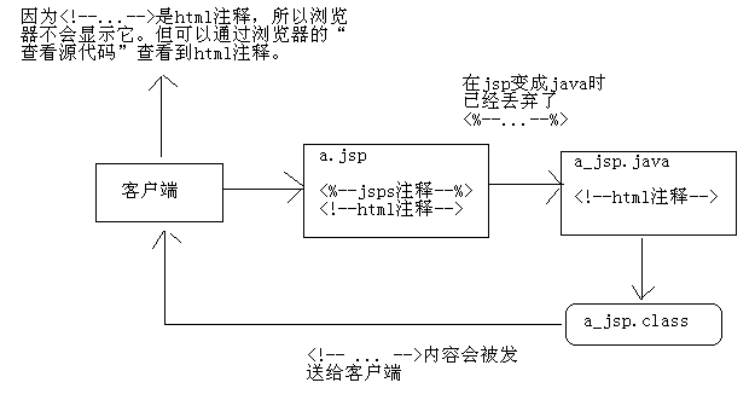
Config对象是在一个**Servlet初始化**时，JSP引擎向它传递信息用的，此信息包括Servlet初始化时所要用到的参数（通过属性名和属性值构成）以及服务器的有关信息（通过传递一个ServletContext对象）



**5）注释**

a. <!-- 注释内容 --> html注释：在jsp编译时，可能会出现中文乱码

b. <%-- 注释内容 --%> jsp专用注释



**6）JSP四大域对象**

**四个域对象：**

pageContext page域

request request域

session session域

application context域

**1）域对象作用：**

保存数据和获取数据，用于数据共享。

**2）域对象方法：**

setAttribute("name",Object) 保存数据

getAttribute("name") 获取数据

removeAttribute("name") 清除数据

**3）域对象作用范围：**

page域： 只能在当前jsp页面中使用。作用于当前页面，当前页面可见。（**当前页面**）

request域： 只能在**同一个请求中使用**。作用于请求，**转发间请求共享**。（**转发**）

session域： 只能在同一个会话（session对象）中使用。作用于整个Web应用，单个用户之间共享（**私有的**）

context域： 只能在同一个web应用中使用。作用于整个Web应用，多个用户之间共享（**全局的**）

用户注册时：先在客户端进行**验证**（jsp正则验证），然后在服务器端进行**二次验证**（java正则表达式验证）

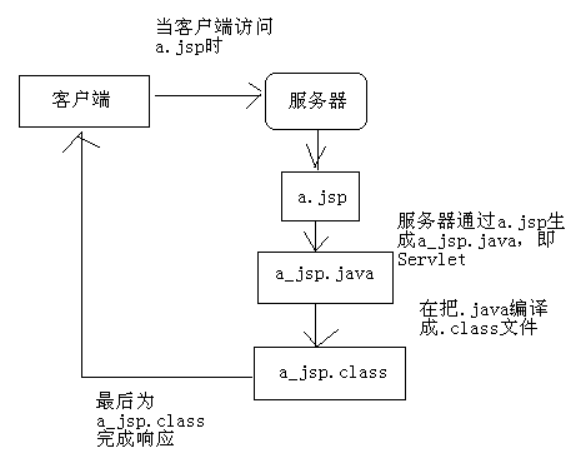
用户登录验证分析时：应采用二次验证法，严格分析

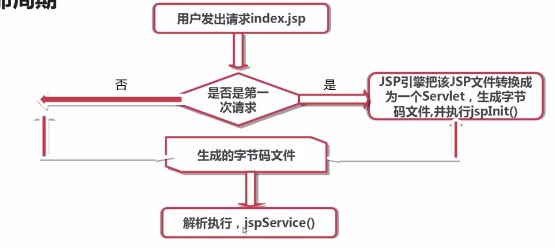
**2、JSP的运行原理**

访问jsp过程：如果是第一次访问.jsp文件会被服务器翻译成一个对应的java(servlet)文件，然后，再被翻译成.class文件,并加载到内存中，如果是以后访问jsp，那就直接调用内存中的jsp实例。所以第一次访问jsp慢，后面访问jsp的速度就会变快

Jsp与servlet的关系：a. Jsp是以servlet为基础；b. Jsp与servlet对应

JSP是一种**特殊的Servlet**，当JSP页面首次被访问时，容器（Tomcat）会先把JSP编译成Servlet，然后再去执行Servlet。所以JSP其实就是一个Servlet！





用户第一次请求Tomcat会将jsp文件编程成一个**Servlet的java文件**，并将java文件编译成class文件，加载到内存，生成文件在Tomcat的work目录的对应项目文件夹。如果jsp页面发生了修改，JSP引擎会对其进行重新编译并加载到内存，以方便用户请求。注意，用户第一次请求一个jsp页面时，首先被执行的方法是构造方法。

\_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资源的需求，提高系统的并发量及响应时间。但也要主要**多线程的编程**带来的同步问题，由于该Servlet始终驻于内存，所以响应是非常快的。

JSP引擎在调用JSP对应的\_jspServlet时，会传递或创建9个与web开发相关的对象共\_jspServlet使用。JSP技术的设计者为便于开发人员在编写JSP页面是获得这些web对象的引用，特意定义了9个相应的变量，开发人员在JSP页面中通过这些变量就可以快速获得这9大对象的引用。

**3、其他**

一个jsp实例：

<%@ page contentType="text/html;charset=gb2312" %>

<html>

<body>

<h1>jsp</h1>

<%

out.println("why!welcome");

%>

</body>

</html>

jsp版计算器(详情请看实例)：

两页面：interface.jsp(界面)和result.jsp(计算)

知识点：a. 如何接受jsp页面提交的数据，并处理

b. jsp中如何使用java片段

c. jsp中如何使用JavaScript完成对数据合法性的校验

java ee 在开发中有几个重要的开发模式

Model1模式、model2模式、MVC模式

model1模式可分为两类：

model1（纯jsp模式）

model1（jsp + java classes）

import="java.util.\*,import java.sql.\*" //引包

MVC模式：界面和模型完全分离

M: java class

V: 视图

C: 控制器

改进方案：

1、增加控制器servlet: 即将loginCl.jsp 用servlet代替，发挥servlet充当控制中转的优势

2、在控制器 servlet中去调用模型model去完成用户验证，并准备要显示的用户信息数据

解决用户名为中文名的方式：

1、如：//得到用户名和密码

String username = request.getParameter("username");

//解决用户名为中文的三种方式

username = new String(username.getBytes("iso-8859-1"),"gb2312");

根据上述方法建类调用方法

2、使用过滤器来解决

3、通过配置 server.xml 文件（但不稳定）

<Connector>中加入 URIEncoding = "gbk"

3、JSP中如何使用JavaScript（JS）和CSS技术

4、美化管理系统界面，完善页面

常用软件：得到延伸配置paycolor; 设置网页logo

5、时尚购物网

a. 需求分析:

1）系统描述（系统可以做什么）

2）角色

前端客户：普通用户

系统管理员：可细化为：商品和公告管理者、订单管理者、用户管理者

3）用例识别

在角色的基础上，可以识别出与角色相关的用例

模块设计：列出相关功能列表

需求分析-->用例图(流程图，uml.rose)-->模块功能设计(将各模块功能细化)

b. 系统结构分析

设计模式：MVC，采用架构

项目中的M、V、C

WEB服务器：Tomcat

数据库：MySQL

c. 数据库设计

d. 网站总体设计

界面设计方式：

1）div + css布局

2）table + css布局

3）混合布局：两种分时合用

网站数据库初步设计：

1）先初步设计，然后完善：步步为营

2）

设计方式：模范某网站分析 --> 创新

做项目的步骤：

1、界面（模仿）

2、数据库（表的设计）[界面（功能）--> 猜表]

用户表和商品表

3、建立web工程：Shopping

4、主页面(index.jsp):界面使用DW，界面先是静态的，而后使用数据库分析

在DW中，建立HTML --> 然后在MyEclipse中建立jsp

开发界面时，使用table + css 方式布局

在table + css布局中，常用技巧是行中加入表格，做精确的布局

CSS是层叠样式表：控制网页字体、颜色、大小、背景、边框等

JavaScript返回原页面：window.open(url)，默认open函数是打开一个新页面，如果在后面加"\_self"

解决中文乱码：request.setCharacterEncoding(编码方式); //编码方式：utf-8、gbk、gb2312