Tomcat与javaweb技术详解

常见的HTML转移字符和实际字符的对应关系

单引号:(‘)--🡪&#039;

双引号:(“)-🡪&quot;

小于:(<)---🡪&lt;

大于:(>)---🡪&gt;

与:(&)------🡪&amp;

空格:( )-----🡪&nbsp;

<URL:Uniform> Resource Locator统一资源定位器->应用层协议://主机ip地址或域名/资源所在路径/文件名

HTTP:处在应用层，建立在TCP/IP协议的基础上，默认端口80

应用层(客户端)---🡪HTTP🡨---应用层(服务器端)

传输层-------------🡪 TCP 🡨---传输层

互联网层----------🡪 IP 🡨---互联网层

网络接口层

物理层

HTTP协议是保证各种编程语言都能顺利交流的关键

HTTP请求由三部分组成

1. 请求方法、URI(Uniform Resource Identifier统一资源标识符)和HTTP协议的版本

请求方法:

get:想要获得服务器某个文档

post:post+get，在获取某个文档时还要提交很多数据给服务器

head:交流一些内部信息

put:提交一个文档给服务器

delete:客户端请求删除远程服务器的一个文档

1. 请求头(Request Header)

“\r\n”：回车符和行结束符

1. 请求正文(Request Content)

HTTP响应也由3部分构成

1. HTTP协议的版本、状态代码和描述

状态代码

200：响应成功

400：错误的请求。客户发送的HTTP请求不正确

404：文件不存在。在服务器上没有客户要求访问的文档

405：服务器不支持客户的请求方法

500：服务器内部错误

1. 响应头(Response Header)
2. 响应正文(Response Content):包含html直接代码

MIME(Multipurpose Internet Mail Extension):多用途网络邮件扩展协议，用于规范正文

Servlet

Servlet规范把能够发布和运行JavaWeb应用的Web服务器称为Servlet容器，它最主要的特征是动态执行

ServletRequest和ServletResponse都是Tomcat创建的，在Response结果发送给客户后销毁

JavaWeb应用中的Servlet实现类中的程序代码，而Tomcat就是Servlet容器，能访问Servlet，并且能作为一个服务器

浏览器--->Tomcat服务器--->ServletRequest--->Servlet--->ServletResponse--->Tomcat--->浏览器

Servlet在请求是.jsp的时候使传参变得非常的简单

<servlet-name> <servlet-class> <init-param>

<load-on-startup>如果是负数，则访问时才加载，如果是正数，则表达第几加载

<servlet-mapping>用来映射uri

//Servlet转发代码target=”/hello.jsp”;

ServletContext context = getServletContext();

RequestDispatcher dispatcher =context.getRequestDispatcher(target);

dispatcher.forward(request, response);

servlet的url都是要自己带/的，目录一般是不需要自己带/的，参照dos的cd

<%@ taglib uri="/mytaglib" prefix="mm" %>

<Host name="www.baidu.com" appBase="C:\mycompany" unpackWARs="true" autoDeploy="true">

<Alias>mycompany1.com</Alias> 别名

<Alias>mycompany1</Alias>

</Host>

Abstract Class GenericServlet实现了Servlet ServletConfig(getServletContext()方法就在这个类中) Serializable

GenericServlet通过init(ServletConfig config);在容器实例化他的时候获得了ServletConfig对象，以后的一切就容易了，继承了GenericServlet可以在servlet中直接使用getServletContext()方法，获得了ServletContext对象则获得了一切…..getRequestDispatcher(String)就是他的杰作

HTTPServlet是GenericServlet的子类，将service方法解析成http特有的doGet,doPost

ServletContextListener能够监听ServletContext对象的生命周期，实际上就是监听Web应用的生命周期

contextInitialized(ServletContextEvent sce);

contextDestroyed(ServletContextEvent sce);

创建一个监听器时，implements ServletContextListener接口，覆写如上两个方法，在web.xml注册这个Listener即可

禁止浏览器缓存网页

response.addHeader("pargma",”no-cache”);只适用http/1.0

response.setHeader(“Cache-Control”,”no-cache”);

repsonse.setHeader(“Expires”,”0”);保存时间为0

下载文件表头设置

response.setContentType("application/force-download");

response.setHeader("Content-Length",String.valueOf(length));

response.setHeader("Content-Disposition", "attachment;filename=\""+filename +"\" ");

上传文件

Content-Type:multipart/form-data;boundary=-------

Cookie是服务器在客户端硬盘上存放的信息，是服务器产生的，用来跟踪客户状态，http协议规定，位于http请求头和响应头中

Cookie cookie = new Cookie(String str,String str);

cookie.getName();cookie.getValue();

response.addCookie(cookie);

Cookie cookies[] = request.getCookies();

setMaxAge(int expiry);设置cookie的保存时间，默认为-1，即使是-1，也会保存到浏览器关闭！设为0即删除此cookie

getMaxAge();获得cookie的保存时间

setPath("/");表示tomcat下的所有应用享受此cookie("/app1/")，只有app1享受

setDomain(".cat.tom");表示给[www.cat.com服务器访问此cookie](http://www.cat.com服务器访问此cookie)的权限

转发和包含

ServletContext.getRequestDispatcher(String path);path(/uri)绝对路径，带"/"，配置的uri-pattern，如果直接请求jsp或html则相对helloapp目录开始/\*.jsp

request.getRequestDispatcher(String path);path相对路径，不带"/"，指于当前servlet的位置关系

转发过程中，源组件的response正文被清空，不能提交，就是不能用close(),flushBuffer()方法（如果用了这些方法，相当于没用forward，另一个组件抛出异常）只能提交目标组件的response，但是源组件的forward后面的 代码也会被执行

包含的话，则是所有目标组件的response都会包含到源组件中，如果其他组件是html，则将文档的内容直接添加到响应中，很明显其他组件不能用close()和flushBuffer方法

重定向response.sendRedirect(String location);只存在HttpServletResponse中

原响应不发送，所以不能用close()，只发送302响应，浏览器收到马上请求新的url，原组件之后的代码也会执行

如果location开头为"/"，表示继续请求当前服务器的其他应用 /helloapp/output1，也可以是完整的<http://...URL>，重定向最关键在在于，不在共享request和response

并发问题，放在service中的则是局部变量，放在service则变成实例变量了

synchronized(this)

对象池，堆栈排列是解决并发的两个方法SingleThreadModel是对象池

JSP

jsp是结合了html和servlet生成html优点，摒弃缺点所产生的，本质是servlet

jsp编译后放在tomcat的work目录下

JSP语法

jsp指令<%@ 指令名 属性=”值”%>page指令、include指令、taglib指令

<%@ page import=”java.io.\*,java.util.Hashtable” %>

<%@ page contentType=”text/html;charset=GB2312”%>等于response.setContentType(“text/html;…”);

<%@ page language=”java”%>

<%@ page method=”doPost”%>还包括doGet和service

<%@ page session=”true”%>如果为false，则不适用session

<%@ page errorPage=”error\_url”%>发生异常时，转发到error\_url页面

<%@ page isErrorPage=”true”%>此页面是否为异常处理页面

include指令

<%@ include file=”目标组件的绝对url或相对url” %>静态包含，可包含html或jsp

jsp表达式<%= %>

jsp注释<%-- --%>

jsp跟servlet一样，有jspInit()方法和jspDestroy()方法，可以在页面中进行覆写,\_jspService()是主要的

jsp标签

<jsp:forward page=”转发的目标组件的绝对URL或相对URL” />，转发后剩下的代码不会被执行

<jsp:forward page=”target.jsp”><jsp:param name=”” value=””></jsp:forward>

<jsp:include file=”被包含组件的URL的绝对URL或相对URL”>静态包含，相当于文件组合，数据共享

<jsp:include page=” 被包含组件的URL的绝对URL或相对URL” flush=”true”>动态包含，数据不共享

<%@ page errorPage=”errorpage.jsp” %>发生错误时发送给哪个jsp

<%@ page isErrorPage=”true”%>错误处理页必须的声明

pageContext.setAttribute(“username”,”tom”,PageContext.REQUEST\_SCOPE);

pageContext方法

getPage();获得当前页面的Servlet实例

getRequest();

getResponse();

getServletConfig();

getServletContext();

getSession();

getOut();返回一个JspWriter对象

EL表达式是用来简化JSP表达式的<%=%>

MVC设计模式

Model-View-Controller模型-视图-控制器

MVC是一种设计模式，强制性把应用程序的数据展示、数据处理和流程控制分开

视图：视图向用户显示相关的数据，并能接受用户的输入数据，但是它并不进行任何实际的业务处理。视图可以向模型查询业务状态，但不能改变模型。视图还能接受模型发出的数据更新事件，从而对用户界面进行同步更新。

模型：模型是应用程序的主体部分。模型表示业务数据和业务逻辑，一个模型能为多个视图提供数据。由于同一个模型可以被多个视图重用，所以提高了模型的可重用性。

控制器：控制器负责应用的流程控制。所谓流程控制，这里是指接受用户的输入并调用相应的模型和视图去完成用户的需求。当Web用户单击Web页面中的【提交】按钮来发送HTML表单时，控制器会接受请求并调用响应的模型组件去处理请求，然后再调用相应的视图来显示模型返回的数据。

MVC处理过程：首先控制器接受用户的请求，并决定应该调动哪个模型来进行处理；然后模型根据客户请求进行相应的业务逻辑处理，并返回数据；最后控制器调用相应的视图来格式化模型返回的数据，并通过视图呈现给用户。

JSP Mode1(jsp即是servlet)

全部由JSP来处理，从而使得产品设计的弹性力度很小，很难满足用户的变化性需求。

1. 网页设计和java编程强耦合在一起，使得程序员必须要精通网页设计和Java编程。
2. 众多jsp逻辑错综复杂
3. 调试困难：html标记、Java代码和JavaScript代码都集中在一个网页中
4. 维护性和可读性差

JSP Model2

Mode1即是JSP+javabean(jsp代码中就必须嵌入控制器)

Model2是JSP+servlet控制器+javabean(EJB也是模型层)，请求将直接发给控制器，而不是JSP

struts

struts.xml放在WEB-INF/classes中，WEB-INF/lib中放着所有的jar文件

Struts架构：模型由实现业务逻辑的javabean或EJB组件构成，控制器由FilterDispatcher和Action来实现，视图由一组JSP(含struts标签)文件构成。

FilterDispatcher是Struts框架的核心组件，实现javax.servlet.Filter接口，负责接收HTTP请求信息，并根据配置文件struts.xml配置信息，把请求转发给适当的Action对象(如果Action对象不存在，将会被创建)

Action：在大型业务中，Action充当客户请求和业务逻辑处理之间的适配器(Adaptor)，其功能是将数据展示与业务逻辑分开，Action类侧重于控制应用程序流程

execute()是Action的主要方法，其Sting返回值代表下一步负责处理请求的目标组件的逻辑名字

url=mypack/Login.action 先去struts.xml中匹配name=mypack的包，然后匹配真实路径/mypack，再去<package>中寻找name=Login的action，然后匹配真实的package.class

<s:text>标签是用来读取properties文件的