# 基础

|  |
| --- |
| WEB-INF目录下的classes和lib目录都可以存放Java的类文件, 在Servlet容器运行时, Web应用程序的类加载器将首先加载classes目录下的, 其次才是lib目录下的类.如果这两个目录下存在同名的类, 起作用的将是classes目录下的类.  WEB-INF是一个特殊的目录(所有字母都要大写). 这个目录并不属于Web应用程序可以访问的上下文路径的一部分, 对客户端来说, 这个目录是不可见的. 但该目录下的内容对于Servlet代码是可见的. |

url各段所代表的含义

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| http:// | www | . | v512.com | /bbs | /index.html |
| 协议名 | 服务器类型 |  | 域名或ip地址 | 服务器的子目录名称 | 网页名称 |

最新的html技术是:html 4.0 和xhtml 1.0

http安全版本:https

## WEB服务器 和 应用服务器

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **web服务器:**用于处理HTTP请求.   |  |  | | --- | --- | | 公司 | 服务器名称 | | Microsoft | IIS | |  | Apache(静态) |   tomcat是一个web服务器, 也是一个有优秀的Servlet/JSP容器. |
| **应用服务器:**可以提供和web应用相关的服务, 如EJB容器就是一种处理业务逻辑和事务的应用服务器.   |  |  | | --- | --- | | 公司 | 服务器名称 | |  | Tomcat(动态) | |  | Resin | | 免费 | JBoss | | BEA | WebLogic(市场占有率高) | | IBM | WebSphere | |

## 配置Tomcat里的Display Name:

|  |
| --- |
| **web.xml文件里面**  <display-name>文件上传2</display-name> |

## 中文乱码

|  |  |  |
| --- | --- | --- |
| //1. **get方式乱码**问题: 修改tomcat的server.xml | //<Connector port="80"  //protocol="HTTP/1.1"  //connectionTimeout="20000"  //redirectPort="8443"  (不好)//**URIEncoding="UTF-8"** //get提交参数, 参数URI编码类型手动设置写死  (很好)//**useBodyEncodingForURI="true"** //get提交参数, 参数URI编码类型采用页面编码类型.  />  **还有个方法:**   |  | | --- | | **1. 对中文字符串进行编码**  **用java的方法编码** <a href="2.jsp?id=<%=URLEncoder.encode("米鹏冲","UTF-8") %>">Test</a>  **或**  **用JS的方法编码**  **2. 然后对编码后的中文字符串进行解码**  <%  String id=request.getParameter("id");  if(id != null){  id = new String(id.getBytes("ISO-8859-1"),"UTF-8");  }  System.out.println(id);  %> | |
| servlet中加这句 | request.setCharacterEncoding("utf-8");  response.setCharacterEncoding("utf-8");  response.setContentType("text/html; charset=utf-8"); |

## 配置用户

|  |  |
| --- | --- |
| tomcat-users.xml | <tomcat-users>  <role rolename="manager"/>  <role rolename="admin"/>  <user username="root" password="root" roles="admin,manager"/>  </tomcat-users> |

## Tomcat自动部署

|  |
| --- |
| 方法一(不好: 这个文件只在tomcat启动时加载一次): 在conf/context.xml中配置:  <Host \*\*\*>  <Context path="/myapp" // path="/web项目名", 代表网络访问的上下文路径.  reloadable="true" //表示可以在运行时在classes与lib文件夹下自动加载类包  docBase="myapp" //表示应用程序路径  workDir="webapps/myapp/work" /> //表示缓存文件放置地点  </Host> |
| 方法二(好: 可以不用重启tomcat随时添加新项目, ):  //在conf/Catalina/localhost/king.xml //蓝色为固定写法; 红色为web项目虚目录  <?xml version='1.0' encoding='utf-8'?>  <Context docBase="D:\11111111111\workspace\cn.edu.swu\01\_jin\WebRoot"  reloadable="true"  path="/jin" />  //灰色部分好像可以不写 |
| 方法三(只在项目web.xml修改时才自动加载, 而在类修改后不会自动加载.):  context.xml: <Context reloadable="true"> |

*开发阶段,tomcat自动部署方法:*

|  |
| --- |
| 方法二(不需要重启服务器):在conf目录下建立文件结构:  conf\cataline\localhost\haha.xml  haha.xml的内容:  <?xml version="1.0" encoding="UTF-8"?>  <Context docBase="项目源文件在硬盘中的地址" path="/项目名" reloadable="true" /> |

## web.xml详解

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns="http://java.sun.com/xml/ns/j2ee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.4"  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">  <display-name>DWR (Direct Web Remoting)</display-name> <!--tomCat里面的显示名字-->  <description>A Simple Demo DWR</description> <!--暂不知道-->    <context-param> <!-- **全局servlet初始化参数**,获取方法: String name = this.getServletContext().getInitParameter("father"); -->  <param-name>father</param-name>  <param-value>金仁强</param-value>  </context-param> |
| <servlet> //注册  <servlet-name>Main</servlet-name>  <servlet-class>com.king.Main</servlet-class>  <init-param> <!-- **局部servlet初始化参数**,获取方法:String param2 = this.getServletConfig().getInitParameter("name"); 或者 String param = this.getInitParameter("param"); -->  <param-name>name</param-name>  <param-value>jinwnalin</param-value>  </init-param>  </servlet>  <servlet-mapping> //映射  <servlet-name>Main</servlet-name>  <url-pattern>/servlet/Main</url-pattern>  </servlet-mapping> |
| <servlet> //注册  <servlet-name>hehe</servlet-name>  <jsp-file>/hehe.jsp</jsp-file>  <load-on-startup>1</load-on-startup> <!-- 当 值>=0, 该Servlet会在Servlet容器加载当前web应用时创建, 值越小越早被创建;当值<0, 该Servlet在第一次访问的时候创建 -->  </servlet>  <servlet-mapping>  <servlet-name>hehe</servlet-name>  <url-pattern>/main</url-pattern>  </servlet-mapping> |
| 过滤器  <filter>  <filter-name>filterTest</filter-name>  <filter-class>com.king.FilterTest</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>utf-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>filterTest</filter-name>  <url-pattern>\*.do</url-pattern>  </filter-mapping>    实例    <filter>  <filter-name>**chartEncoding**</filter-name>  <filter-class>com.webqq.filter.CharacterEncodingFilter</filter-class>  </filter>  <filter>  <filter-name>**CheckUserLogin**</filter-name>  <filter-class>com.webqq.filter.CheckUserLoginFilter</filter-class>  </filter>  <filter>  <filter-name>**CheckAdminLogin**</filter-name>  <filter-class>com.webqq.filter.CheckAdminLoginFilter</filter-class>  </filter>      <filter-mapping>  <filter-name>**chartEncoding**</filter-name>  <url-pattern>/filterWeb/\*</url-pattern>  </filter-mapping>  <filter-mapping>  <filter-name>**chartEncoding**</filter-name>  <url-pattern>/admin/\*</url-pattern>  </filter-mapping>    <filter-mapping>  <filter-name>**chartEncoding**</filter-name>  <url-pattern>/web/\*</url-pattern>  </filter-mapping>    <filter-mapping>  <filter-name>**chartEncoding**</filter-name>  <url-pattern>/userInfo/\*</url-pattern>  </filter-mapping>    <filter-mapping>  <filter-name>**chartEncoding**</filter-name>  <url-pattern>/servlet/\*</url-pattern>  </filter-mapping>  <filter-mapping>  <filter-name>**chartEncoding**</filter-name>  <url-pattern>/home/\*</url-pattern>  </filter-mapping>            <filter-mapping>  <filter-name>**CheckUserLogin**</filter-name>  <url-pattern>/filterWeb/\*</url-pattern>  </filter-mapping>    <filter-mapping>  <filter-name>**CheckUserLogin**</filter-name>  <url-pattern>/web/\*</url-pattern>  </filter-mapping>    <filter-mapping>  <filter-name>**CheckUserLogin**</filter-name>  <url-pattern>/userInfo/\*</url-pattern>  </filter-mapping>  <filter-mapping>  <filter-name>**CheckUserLogin**</filter-name>  <url-pattern>/home/\*</url-pattern>  </filter-mapping>      <filter-mapping>  <filter-name>**CheckAdminLogin**</filter-name>  <url-pattern>/admin/\*</url-pattern>  </filter-mapping> |
| 监听器  <listener>  <listener-class>com.webqq.servlet.CoreListener</listener-class>  </listener>    <listener>  <listener-class>com.webqq.servlet.ChatRoomListener</listener-class>  </listener> |
| session有效时间  <session-config>  <session-timeout>120</session-timeout>  </session-config> |
| <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  </welcome-file-list>  </web-app> |

## web.xml中\*通配符的优先级:

|  |  |
| --- | --- |
| 优先级 最高 | /abc/\* |
|  | /\* |
|  | /abc |
| 优先级 最低 | \*.do |

## 开发阶段,显示URL列表:

|  |
| --- |
| 修改Web.xml ; 当找不到欢迎页面时, 就显示URL列表.  <servlet>  <servlet-name>default</servlet-name>  <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>  <init-param>  <param-name>debug</param-name>  <param-value>0</param-value>  </init-param>  <init-param>  <param-name>listings</param-name>  <param-value>false</param-value> //黄色修改成true  </init-param>  <load-on-startup>1</load-on-startup>  </servlet> |

## 允许跨域请求

|  |
| --- |
| <filter>     <filter-name>CorsFilter</filter-name>     <filter-class>org.apache.catalina.filters.CorsFilter</filter-class>  </filter>  <filter-mapping>     <filter-name>CorsFilter</filter-name>     <url-pattern>/\*</url-pattern>  </filter-mapping> |

## 修改网站图标, 写在网页面head里面

|  |
| --- |
| <link rel=*"shortcut icon"* href=*"by/image/purple.ico"* > <!-- 必须是ico格式图标 --> |

## JAVA获取服务器路径

|  |  |
| --- | --- |
| 在JSF环境中获取到  ServletContext | ServletContext sc = (ServletContext)FacesContext.getCurrentInstance().getExternalContext().getContext(); |
| servlet中获得项目绝对路径 | String filePath=this.getServletConfig().getServletContext().getRealPath("/"); |
| 根目录所对应的绝对路径 | request.getServletPath(); |
| 文件的绝对路径 | request.getSession().getServletContext().getRealPath(request.getRequestURI()) |
| 当前web应用的绝对路径 | servletConfig.getServletContext().getRealPath("/"); |
| ServletContext对象获得几种方式 | Javax.servlet.http.HttpSession.getServletContext()  Javax.servlet.jsp.PageContext.getServletContext()  Javax.servlet.ServletConfig.getServletContext() |
| 获取web项目的classpath | String classpath = this.getClass().getResource("/").toString(); |

# JSP

## jsp中常用代码:

|  |
| --- |
| <%  String **path** = request.getContextPath();  String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+**path**+"/";  %> |

## jsp中得到ContextPath的两种方法

|  |
| --- |
| 1.java脚本  <%=request.getContextPath()%>  2.el  ${pageContext.request.contextPath}  3.JSTL  <c:url value="/" /> |

## JSP基本元素

### 指令元素

#### page指令

除了import属性和pageEncoding属性可以出现多次外, 在同一个转换单元中同样的属性只能出现一次.

|  |  |
| --- | --- |
| language="默认为java" |  |
| info="…" | 定义jsp页面的描述信息,在页面转换后可以用Servlet.getServletInfo()获取. |
| import="…" (可以出现多次) | 导入需要的类 |
| session="默认为true" | 是否可用session对象 |
| contentType="默认值为text/html"   |  |  | | --- | --- | | text/plain | 纯文本文件 | | text/html | 纯文本的HTML页面 | | application/x-msexecl | excel文件 | | application/msword | word文件 | |  |
| pageEncoding="默认值为ISO-8859-1"(可以出现多次) |  |
| extends="…" |  |
| errorPage="…" |  |
| isErrorPage="true" |  |
| buffer="默认为8kb" | 指定缓冲区大小的最小值, 单位为kb,也可以设置成none |
| autoFlush="默认为true" | 缓冲满了是否自动刷新输出到客户端 |
| isELIgnored="默认false" | 是否忽略EL表达式 |
| isThreadSafe="true" | 是否线程安全 |

|  |
| --- |
| <%@ page  language=*"java"*  contentType=*"text/html; charset=GB18030"*  pageEncoding=*"GB18030"*  errorPage=*""*  import=*""*  isELIgnored=*"false"*  session=*"true"*  info=*""*  extends=*""*  isThreadSafe=*"true"*  isErrorPage=*"false"*  autoFlush=*"true"*  buffer=*"8kb"*  %> |
| <%@include file=*""* %> |
| <%@ taglib prefix=*"logic"* uri=*"http://struts.apache.org/tags-logic"*%> |

#### taglib指令

|  |  |
| --- | --- |
| <%@ taglib prefix=*"myTags"* uri=*"http://\*\*\*\*\*\*\*"*%>  <mytag:doSomthing>  </mytag:doSomthing> | JSP页面引入和使用标记 |

#### include指令(编译时包含)

|  |
| --- |
| <%@ include file="url"%> |
| JSP转换为servlet编译生成class文件, 反编译得到的代码如下  RequestDispatcher rd;  rd = request.getRequestDispatcher("/header.jsp").include(request.response);  rd = request.getRequestDispatcher("/main.jsp").include(request.response);  rd = request.getRequestDispatcher("/footer.jsp").include(request.response); |

### attribute指令

|  |
| --- |
| 只能用于标记文件中所声明的属性. |

#### tag指令

|  |
| --- |
|  |

#### variable指令

|  |
| --- |
|  |

### 行为元素(动作元素)(运行时执行)(20种)

|  |  |  |
| --- | --- | --- |
| JSP 1.2中原有的6种 | <jsp:include page=*"被包含资源的相对路径"* flush=*"指定在包含目标资源之前是否刷新输出缓冲区, 默认为false, 不可接受动态值"*>  <jsp:param value=*"jinwanlin"* name=*"name"*/>  <jsp:param value=*"23"* name=*"age"*/>  </jsp:include> | 动态包含 (运行时包含),可传参数 |
| <jsp:forward page=*"资源相对路径, 可以接受动态值"* > <!--请求转发到另一个JSP页面 -->  <jsp:param name=*"name"* value=*"jinwanlin"* /> <!-- name不可接受动态值, value可以接受动态值 -->  <jsp:param name=*"age"* value=*"23"* />  </jsp:forward> | 服务器端跳转, 执行到这一句, 自动跳转到这个页面  和this.getServletContext().getRequestDispatcher("/haha.jsp").forward(request, response);  功能一样. |
| <jsp:param> | 定义参数 |
| <jsp:plugin> 用于生成HTML标记<Embed>或<Object>元素(取决于浏览器的类型)从而实现Java applet或JavaBean组件的执行.  type属性: 指定要嵌入对象的类型: applet|bean ; 不可接受动态值.  code属性: 指定applet的完全限定类名, 必须以".class"结尾, 该文件必须存在于codebase属性指定的目录中. 不可接受动态值.  codebase属性: 指定包含类文件的目录的相对路径; 不可接受动态值.  name属性: 用于指定applet或bean的名称, 不可接受动态值  archive属性: 指定applet的归档文件的URI列表(用逗号分隔), 这些归档文件中包含预加载的类和其他资源, 归档文件的URI要针对applet的codebase属性加以解释, 不可接受动态值.  align属性: 指定applet或bean在显示区域中的对齐方式, 不可接受动态值. 可选的对齐方式如下:  left: 把组件放在页面左边, 后面的文本会移至组件右边.  right, 把组件放在页面右边, 后面的文本会移至组件左边.  bottom, 把组件的底部与当前行文本底部对齐.  top, 把组件的顶部与当前行顶部对齐.  texttop, 把组件的顶部与当前行文本顶部对齐.  middle,把组件的总部与当前行基线对齐.  absmiddle,把组件的中部与当前行中部对齐.  baseline,把组件的底部与当前行基线对齐.  absbottom把组件的底部与当前行底部对齐.  height属性: 以像素或百分比指定applet或bean显示区域的高度, 可以接受动态值.  hspace属性: 指定applet区域左右所需留下的空间, 单位为像素. 不可接受动态值.  vspace属性: 指定applet区域上下所需留下的空间, 单位为像素. 不可接受动态值.  jreversion属性: 指定applet或bean正常执行所需的JRE版本. 默认为1.1, 不可接受动态值.  nspluginurl属性: 指定Netscape插件URL, 不可接受动态值.  iepluginurl属性: 指定IE插件URL, 不可接受动态值. |  |
| <jsp:params>  <jsp:plugin>动作体可以包含一个<jsp:params>元素, 用来指定applet的参数. 并且由<jsp:param>元素指定具体的参数.  <jsp:params>只能用于<jsp:plugin>动作体中. |  |
| <jsp:fallback> 对于不支持<Embed>和<Object>元素的浏览器, <jsp:plugin>动作体可以包含一个<jsp:fallback>元素, 用以指定替代显示文本.  <jsp:params>只能用于<jsp:plugin>动作体中. |  |
| 存取JavaBean的3种 | <jsp:useBean  id="JavaBean的实例名称, 不可接受动态值"  scope="指定实例bean的有效范围, 不可以接受动态值.page(默认)|request|session|application"  class="bean的完全限定类名, 不可以接受动态值"  type="bean的完全限定类型名称, 即bean类的一个父类或其实现的一个接口名称"  beanName="指定bean的名字, 可以接受动态值. beanName属性必须与type属性结合使用, 不能与class属性同时使用."  > (使用这个获得一个JavaBean) | <jsp:useBean> 用于创建JavaBean的实例;  <jsp:getProperty> 和 <jsp:setProperty> 用于操作JavaBean的属性  <jsp:useBean id=*"KnowledgeTypes"* class=*"com.ismp.dao.hb.datatype.KnowledgeTypeDataType"* scope=*"page"* />  相当于<% NameBean person = (NameBean)pageContext.getAttribute("customer"); %> |
| <jsp:getProperty  name="指定要获取属性值的bean的名称, 不可接受动态值."  property="指定要获取的bean属性的名称, 不可接受动态值."  /> (获取JavaBean对象的属性) | 相当于<%= person.getFirstName()%> |
| <jsp:setProperty  name="指定要设置属性的bean的名称, 不可接受动态值"  property="指定bean的属性的名称, 也可以是一个星号(\*), 表示设置名称与请求参数匹配的所有属性. 不可接受动态值"  parame="指定请求参数的名称, 如果忽略该属性, 则参数名和属性名必须相同. 不可接受动态值."  value="指定bean属性的值, 不能与param属性同时使用. 可以接受动态值"  /> (设置JavaBean对象的属性) | 相当于<% person.setFirstName("金万林")%> |
| 用于JSP文档的6种  (用XML语法写的JSP页面) | <jsp:root version="jspVersion> 标签体 </jsp:root>  <jsp:root> 只能作为JSP文档(采用XML语法的JSP页面)的根元素.  JSP2.0以前的规范中, 使用<jsp:root>动作是将一个JSP页面标识为JSP文档的唯一方法.  从JSP2.0开始, 有其他的方法可以标识JSP文档. <jsp:root>动作是可选的, 并且很少使用. |  |
| <jsp:declaration> 用于声明变量和方法. 声明语句必须是完整的, 并且要遵循java的语法.  <jsp:declaration>  String msg = "Hello World!";  int i;  public String Hello(String name){  return "Hello " + name + "!";  }  </jsp:declaration> |  |
| <jsp:scriptlet> 指定在请求处理期间要执行的java代码段, 即脚本段. 脚本段可以产生输出, 彬将输出发送到客户端, 也可以实现流程控制.  <jsp:scriptlet>  out.write(Hello("Forrid"));  </jsp:scriptlet> |  |
| <jsp:expression> 用于指定表达式脚本元素, 在请求处理时将计算这些表达式, 计算的结果被转换为字符串, 插入到当前的输入流中.  <jsp:expression>  (new java.util.Date()).toLocaleString();  </jsp:expression> |  |
| <jsp:text> 主要用于JSP文档(采用XML语法的JSP页面).  <jsp:text> 动作体中只能包含模板文本或EL表达式, 而不能嵌入其他的JSP动作或脚本元素.  <jsp:text>  ${456 div 123}  </jsp:text> | ` |
| <jsp:output> 只能在JSP文档(采用XML语法的JSP页面) 和 采用XML语法的标记文档中使用, 用于输出XML声明和文档类型声明.  omit-xml-declaration属性: 是否忽略XML声明. 可选的包括: true|yes|false|no. 该属性不可接受动态值. 如果含有<jsp:root>元素, 默认值为yes.  doctype-root-element属性: 指定在所生成的DOCTYPE(文档类型)声明中使用XML文档根元素名, 不可接受动态值. 必须与doctype-stytem属性结合使用.  doctype-stytem属性: 指定DOCTYPE声明中使用的公共标识(Public ID), 不可接受动态值, 只有在指定doctype-stytem属性时, 才可使用doctype-public属性. |  |
| 动态生成XML的3种 | <jsp:attribute> 为JSP动作元素定义一个属性值.  <jsp:attribute>还可以在<jsp:element>中使用, 用于指定输出元素的属性.  name属性: 指定属性名.不可接受动态值.  trim属性: 指定是否忽略空白. |  |
| <jsp:body> 为一个动作元素定义动作提(即动作元素的内容), 该动作仅当动作属性由<jsp:attribute>动作定义时才是必要的. |  |
| <jsp:element> 主要在JSP文档(采用XML语法的JSP页面)中使用, 用于动态定义一个XML元素的标记.  只有一个name属性: 用于指定所生成元素的名称, 可以结合搜动态值. |  |
| 用于标记文件的2种 | <jsp:invoke> 只能在标记文件(Tag File)中使用, 用于计算指定的片段(Rragment), 并将输出增加到调用页面的输出流, 或将其保存到一个变量中. |  |
| <jsp:doBody> 只能在标记文件(Tag File)中使用, 用于计算相应的定制动作的本体, 并将结果输出到调用页面中, 或将其保存到一个变量中. |  |

### 脚本元素

#### 声明

|  |
| --- |
| <%!  declaration  %> |

#### 表达式

|  |
| --- |
| <%=expressions%> |

#### 脚本片段

|  |
| --- |
| <%  脚本段  %> |

## 隐式对象(内置对象)

|  |  |  |
| --- | --- | --- |
| 对象名 | 对象类型 | 作用 |
| pageContext | javax.servlet.jsp.PageContext  页面上下文对象 | 当前jsp上下文对象,可以获取当前页面的其它八个对象  使用pageContext的setAttribute()和getAttribute()来传递数据.只能在同一个JSP页面获取. |
| request | javax.servlet.HttpServletRequest | 在请求处理结束后就失效.使用request的setAttribute()和getAttribute()来设置和获取属性. |
| session | javax.servlet.http.HttpSession | 会话对象,保存跨页面信息.   |  |  | | --- | --- | | setMaxInactiveInterval(int interval) | 设定当前会话的可持续时间 | | invalidate() | 使当前会话失效, 原来会话中存储的所有对象都不能访问, 即注销用户登录. | | getCreationTime() | 会话创建时间 | | getLastAccessedTime() | 最后访问时间 | |
| application | javax.servlet.HttpServletContext | 一个web应用公用 |
| response | javax.servlet.HttpServletResponse | 通过它可完成转向 |
| out | javax.servlet.jsp.JspWriter | HTTP标准输出 |
| config | javax.servlet.ServletConfig | 获取JSP页面的Servlet配置信息, 就是web.xml中的相关信息. |
| page | java.lang.Object | 当前jsp自身. 如同Java中的this. 由于page对象占用内存较多, 一般情况下, 不推荐使用page对象. |
| exception | java.lang.Throwable | 违例对象, 只能在错误处理页中使用. |

|  |  |  |
| --- | --- | --- |
| JSP隐私对象 | 输入输出对象 | [out](#_out对象) |
| [request](#_request对象(请求有效)) |
| [response](#_response对象) |
| 作用域通信对象 | [pageContext](#_pageContext对象(页内有效)) |
| [session](#_session对象(会话有效)_1) |
| [application](#_application对象(应用有效)) |
| Servlet对象 | [page](#_page对象) |
| [config](#_config对象) |
| 错误对象 | [exception](#_exception对象) |

### out对象

out对象是JspWriter类的实例,是向客户端输出内容常用的对象

字段

|  |  |
| --- | --- |
| DEFAULT\_BUFFER | 指定默认缓冲区大小 |
| NO\_BUFFER | 指示out对象不缓冲输出 |
| UNBOUNDED\_BUFFER | 不限制缓冲区大小, 用于BodyContent |

方法

|  |  |  |
| --- | --- | --- |
| void  void | print(数据类型 变量名)throws IOException  println(数据类型 变量名)throws IOException | 输出指定的数据, 其中类型的取值可以为boolean|char|byte|char[]|double|float|int|long|Object|String |
| void  void | println()throws IOException  newLine() | 输出一个换行字符"\n" |
| void  void  void  void  void | write(String str)  write(String str, int offset, int length)  write(String singleCharacter)  write(char[] charArray)  write(char[] charArray, int offset, int length) | 输出字符或字符串, 可以指定字符串的起始和结束位置. |
| void | clear() | 清除缓冲区的内容, 但不会吧数据输出到客户端. 如果缓冲区已经刷新输出, 则抛出一个IOException异常 |
| void | clearBuffer() | 清除缓冲区的当前内容, 并把数据输出到客户端. 如果缓冲区已经刷新输出, 并不抛出一个IOException异常 |
| void | flush() | 刷新缓冲区输出 |
| int | getBufferSize() | 返回缓冲区以字节数的大小，如不设缓冲区则为0 |
| int | getRemaining() | 返回缓冲区还剩余多少可用(字节数) |
| boolean | isAutoFlush() | 返回缓冲区满时，是自动刷新缓冲区; 如果自动刷新缓冲区, 则返回true; 否则返回false. |
| void | close() | 关闭输出流 |

### request对象(请求有效)

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的需求，然后做出响应。它是HttpServletRequest类的实例。

用户请求访问的当前web组件, 以及和当前web组件共享同一个用户请求的其它web组件.

如果用户请求访问的是JSP网页, 那么该JSP网页的<%@include>指令以及<forward>标记包含的其它JSP文件也能访问共享对数据.

|  |  |  |
| --- | --- | --- |
| object | getAttribute(String name) | 返回指定属性的属性值 |
| Enumeration | getAttributeNames() | 返回所有可用属性名的枚举 |
| String | getAuthType() | 返回保护Servlet的认证机制名称(包括:BASIC\_AUTH|CLIENT\_CERT\_AUTH|DIGEST\_AUTH|FORM\_AUTH); 如果Servlet未被保护, 则返回null |
| String | getCharacterEncoding() | 返回字符编码方式 |
| int | getContentLength() | 返回请求体的长度（以字节数） |
| String | getContentType() | 得到请求体的MIME类型 |
| String | getContextPath() | 返回请求的上下文路径(也就是项目名) |
| Cookie[] | getCookies() | 返回客户端所有的Cookie对象数组. |
| long | getDateHeader(String name) | 返回指定请求头部的日期; 如果请求中未包括指定的头部, 则返回-1. |
| String | getHeader(String name) | 返回指定的请求头部信息, 如果请求中未包括指定的头部, 则返回null. |
| Enumeration | getHeaderNames() | 返回请求中所有头部的名称集合. |
| Enumeration | getHeaders(String name) | 返回指定的请求头部所有的值. |
| ServletInputStream | getInputStream() | 得到请求体中一行的二进制流 |
| int | getIntHeader(String name) | 返回指定请求头部的整形值. 如果请求中未包括指定的头部, 则返回-1. |
| String | getLocalAddr() | 返回响应请求的服务器地址. |
| Locale | getLocale() | 返回当前页面的区域对象. 例如zh\_CN |
| Enumeration | getLocales() | 返回请求中所有的地域对象. |
|  | getLocalName() |  |
| int | getLocalPort() | 返回响应请求的服务器端口. |
| String | getMethod() | 返回发送请求的方式(GET|POST|PUT等) |
| String | getParameter(String name) | 返回客户端传入参数名为name的参数值 |
| Map | getParameterMap() | 返回一个Map对象, 该对象包含请求中的所有参数. |
| Enumeration | getParameterNames() | 返回可用参数名的枚举 |
| String[] | getParameterValues(String name) | 返回客户端传入参数名为name的参数值(数组类型) |
| String | getPathInfo() | 返回请求中与URI相关的所有额外路径信息. 如果不存在额外的路径信息, 则返回null. 对于一个JSP页面, 该方法总是返回null. |
| String | getPathTranslated() | 用于将getPathInfo()方法的结果翻译成相应的文件系统路径, 并予返回. 如果getPathInfo()方法返回null, 该方法也返回null. |
| String | getProtocol() | 返回请求用的协议类型和版本号, 例如HTTP/1.1 |
| String | getQueryString() | 返回请求URL中包含的查询字符串. |
| BufferedReader | getReader() | 返回请求的输入流对应的Reader对象. |
| String | getRealPath(String path) | 返回一虚拟路径的真实路径 |
| String | getRemoteAddr() | 返回发送此请求的客户端IP地址 |
| String | getRemoteHost() | 返回发送此请求的客户端主机名 |
| int | getRemotePort() | 返回发送请求的客户端端口. |
| String | getRemoteUser() | 返回发送请求的客户端验证用户名. 如果客户端未经验证, 则返回null. |
| RequestDispatcher | getRequestDispatcher(String path) | 返回一个RequestDispatcher对象, 该对象相当于位于指定路径的资源的一个包装器. |
| String | getRequestSessionId | 返回请求的Session标识ID |
| String | getRequestURI getRequestURI getRequestURI () | 返回发送请求的客户端地址, 不包括请求的查询字符串(除IP地址外的请求路径). 例如西大校园网登陆服务器: /login/login1.jsp |
| StringBuffer | getRequestURL() | 返回响应请求的服务器地址, 不包括请求的查询字符串. 例如西大校园网登陆服务器: http://202.202.96.53/login/login1.jsp |
| String | getScheme() | 返回请求用的协议名,如:HTTP|HTTPS及FTP等, 默认为HTTP |
| String | getServerName() | 返回接受请求的服务器主机名(IP地址) |
| int | getServerPort() | 返回服务器接受此请求所用的端口号 |
| String | getServletPath() | 返回请求URI中标识Servlet的路径. 对于JSP页面, 该方法将返回JSP页面的完整上下文相对路径. /login/login1.jsp |
| HttpSession | getSession(Boolean create) | 返回与请求相关的当前HttpSession对象. 参数Boolean类型的create, 指定是否创建新的HttpSession对象. |
| Principal | getUserPrincipal() | 返回一个当前已认证用户名的Principal对象. |
| boolean | isRequestedSessionIdFromCookie() | 判断会话ID是否来自Cookie(是否通过Cookie传入). |
| boolean | isRequestedSessionIdFromURL() | 判断请求的会话ID是否通过URL传入. |
| boolea | isRequestedSessionIdValid() | 判断请求的会话ID是否仍然有效. |
| boolean | isSecure() | 检查请求是否使用安全连接(如HTTPS) |
| boolean | isUserInRole(String role) | 判断已认证用户是否属于指定的逻辑角色. 传入的参数role为逻辑角色. |
| void | removeAttribute(String name) | 移除指定的属性. |
| void | setAttribute(String name,Object value) | 设置属性的属性值 |
| void | setCharacterEncoding(String name) | 设置请求的字符编码方式. |

### response对象

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。

常用字段

|  |  |  |
| --- | --- | --- |
| 202 | SC\_ACCEPTED | 请求已被接受, 但未处理完成, 该字段可通过setStatus()方法返回给客户端. |
| 502 | SC\_BAD\_GATEWAY | HTTP服务接收到一个无效的响应, 该响应来自充当代理或网关的服务器; 该字段可通过setStatus()方法返回给客户端. |
| 400 | SC\_BAD\_REQUEST | 指示通过客户端发送的请求语法不正确. 该字段可通过setStatus()方法返回给客户端. |
| 409 | SC\_CONFLICT | 指示由于与资源当前状态冲突, 请求无法完成. 该字段可通过setStatus()方法返回给客户端. |
| 100 | SC\_CONTINUE | 指示允许客户端继续请求. 该字段可通过setStatus()方法返回给客户端. |
| 201 | SC\_CREATED | 指示服务器在请求的响应中建立了一个新的资源. |
| 417 | SC\_EXPECTATION\_FAILED | 指示服务器无法满足在请求头部提供期望值. |
| 403 | SC\_FORBIDDEN | 用于指示除非拥有权限, 否则服务器拒绝提供所有请求的资源. |
| 504 | SC\_GATEWAY\_TIMEOUT | 指示服务器无法得到及时的响应, 该响应来自从当网关或代理的远程服务器. |
| 410 | SC\_GONE | 指示在服务器上的资源已经不再可用, 并且没有更新的地址. |
| 505 | SC\_HTTP\_VERSION\_NOT\_SUPPORTED | 指示服务器不支持或拒绝支持在请求消息中使用的HTTP协议版本. |
| 500 | SC\_INTERNAL\_SERVER\_ERROR | 指示一个HTTP服务器内部错误, 从而阻止服务器完成请求. |
| 411 | SC\_LENGTH\_REQUIRED | 指示服务器无法处理请求, 除非定义数据长度. |
| 405 | SC\_METHOD\_NOT\_ALLOWED | 指示对于由Request-URI标识的资源, 在Request-Line中指定的方法不允许使用. |
| 301 | SC\_MOVED\_PERMANENTLY | 指示资源已永久地移至新位置, 以后的引用应当使用新的URI. |
| 302 | SC\_MOVED\_TEMPORARILY | 指示资源已临时移至另一个位置, 以后的引用应当使用原始的URI. |
| 300 | SC\_MULTIPLE\_CHOICES | 指示被请求的资源可以在多个位置找到, 并将在返回的文档中列出来. |
| 204 | SC\_NO\_CONTENT | 指示在没有新信息的情况下, 浏览器继续显示先前的文档. |
| 203 | SC\_NON\_AUTHORITATIVE\_INFORMATION | 指示客户端呈现的元信息并非源自服务器. |
| 406 | SC\_NOT\_ACCEPTABLE | 指示所请求资源的MIME类型与发送请求总accept头部指定的类型不一致. |
| 404 | SC\_NOT\_FOUND | 指示所请求的资源未找到. |
| 501 | SC\_NOT\_IMPLEMENTED | 指示HTTP服务器不支持完成请求所需的功能. |
| 304 | SC\_NOT\_MODIFIED | 指示一个有条件的GET操作找到的资源可用,但未修正. |
| 200 | SC\_OK | 一切正常. |
| 206 | SC\_PARTIAL\_CONTENT | 仔细服务器已经完成部分GET请求. |
| 402 | SC\_PAYMENT\_REQUIRED | 指示保留使用. |
| 412 | SC\_PRECONDITION\_FAILED | 指示请求头部中给定的某些先决条件在服务器上测试失败. |
| 407 | SC\_PROXY\_AUTHENTICATION\_REQUIRED | 指示客户端必须通过代理服务器的身份认证. |
| 413 | SC\_REQUEST\_ENTITY\_TOO\_LARGE | 指示服务器拒绝处理请求.因为请求实体比服务器想要或能够处理的要大. |
| 408 | SC\_REQUEST\_TIMEOUT | 指示在服务器等待的时间内, 客户端不能生成请求. |
| 414 | SC\_REQUEST\_URI\_TOO\_LONG | 指示请求的URI比服务器想要解释的要长. |
| 416 | SC\_REQUESTED\_RANGE\_NOT\_SATISFIABLE | 指示服务器无法为请求的字节范围提供服务. |
| 205 | SC\_RESET\_CONTENT | 指示浏览器要重置文档显示. |
| 303 | SC\_SEE\_OTHER | 指示请求可在不同的URI下找到. |
| 503 | SC\_SERVICE\_UNAVAILABLE | 指示HTTP服务器因临时超载而无法处理请求. |
| 101 | SC\_SWITCHING\_PROTOCOLS | 指示服务器将按照其上的头部信息转换为一个不同的协议. |
| 307 | SC\_TEMPORARY\_REDIRECT | 指示所请求的资源临时存在于不同的URI下. |
| 401 | SC\_UNAUTHORIZED | 指示请求需要HTTP认证. |
| 415 | SC\_UNSUPPORTED\_MEDIA\_TYPE | 指示服务器拒绝为请求提供服务, 因为所请求的资源不支持请求实体的格式. |
| 305 | SC\_USE\_PROXY | 指示说请求的资源必须通过Location字段给定的代理服务器获得. |

常用方法

|  |  |  |
| --- | --- | --- |
| void | [addCookie](mk:@MSITStore:F:\（（教程））\（1_Java%20EE）\API\servletAPI.chm::/jsdk23/javax/servlet/http/HttpServletResponse.html#addCookie(javax.servlet.http.Cookie))([Cookie](mk:@MSITStore:F:\（（教程））\（1_Java%20EE）\API\servletAPI.chm::/jsdk23/javax/servlet/http/Cookie.html) cookie) | 用于为响应添加指定的Cookie对象. |
| void | [addDateHeader](mk:@MSITStore:F:\（（教程））\（1_Java%20EE）\API\servletAPI.chm::/jsdk23/javax/servlet/http/HttpServletResponse.html#addDateHeader(java.lang.String, long))(String name, long date) | 用于添加一个带有指定名称和日期值的响应头部. 参数是(指定头部名称, 指定日期值为自GMT1970年1月1日零点以来的毫秒数) |
| void | [addHeader](mk:@MSITStore:F:\（（教程））\（1_Java%20EE）\API\servletAPI.chm::/jsdk23/javax/servlet/http/HttpServletResponse.html#addHeader(java.lang.String, java.lang.String))(String name, String value) | 用于添加一个带有指定名称和字符串值的响应头部. |
| void | [addIntHeader](mk:@MSITStore:F:\（（教程））\（1_Java%20EE）\API\servletAPI.chm::/jsdk23/javax/servlet/http/HttpServletResponse.html#addIntHeader(java.lang.String, int))(String name, int value) | 用于添加一个带有指定名称和整数值的响应头部. |
| boolean | [containsHeader](mk:@MSITStore:F:\（（教程））\（1_Java%20EE）\API\servletAPI.chm::/jsdk23/javax/servlet/http/HttpServletResponse.html#containsHeader(java.lang.String))(String name) | 用于判断指定名称的响应头部是否已经设置. |
| String | [encodeRedirectUrl](mk:@MSITStore:F:\（（教程））\（1_Java%20EE）\API\servletAPI.chm::/jsdk23/javax/servlet/http/HttpServletResponse.html#encodeRedirectUrl(java.lang.String))(String url) | 用于对sendRedirect方法使用的URL进行编码. |
| String | [encodeRedirectURL](mk:@MSITStore:F:\（（教程））\（1_Java%20EE）\API\servletAPI.chm::/jsdk23/javax/servlet/http/HttpServletResponse.html#encodeRedirectURL(java.lang.String))(String url) | 用于通过包含会话ID对指定的URL进行编码. |
| void | flushBuffer() throws IOException | 用于强制把相应实体中的所有内容发送到客户端. |
| int | getBufferSize() | 用于返回相应的实际缓冲区大小(字节数) |
| String | getCharacterEncoding() | 用于返回相应的字段编码方式. |
| String | getContentType() | 用于返回相应内容的MIME类型.( MIME类型,就像: text/html;charset=GB2312 ) |
| Locale | getLocale() | 用于返回响应的区域对象. |
| ServletOutputStream | getOutputStream() | 用于返回一个可以输出二进制数据的ServletOutputStream对象 |
| PrintWriter | getWriter() | 用于返回一个可以向客户端发送字符文本的PrintWriter对象. |
| boolean | isCommitted() | 用于判断响应是否已经提交. |
| void | reset() | 清空缓冲区中的所有内容以及头部和状态吗. |
| void | resetBuffer() | 清空缓冲区中所有内容, 但保留头部和状态码. |
| void | sendError(int sc, String msg) | 向客户端发送错误响应. 参数解释:(错误状态码, 描述信息) |
| void | sendRedirect(String location) | 用于向客户端发送一个指定位置的临时重定向响应. |
| void | setBufferSize(int size) | 用于设置响应实体的首选缓冲区大小(字节数). |
| void | setCharacterEncoding(String charset) | 用于设置响应实体使用的字符编码方式. 如果已经通过setContentType()或setLocale()方法设置了字符编码, 则该方法将覆盖原编码. |
| void | setContentLength(int length) | 用于设置响应内容的长度(字节数). |
| void | setContentType(String type) | 用于设置发送到客户端响应内容的MIME类型, 参数type为指定的MIME类型 |
| void | setDateHeader(String name, long value) | 用于设置带有指定名称和日期值的响应头部. |
| void | setHeader(String name, String value) | 用于设置带有指定名称和字符串值的响应头部. |
| void | setIntHeader(String name, int value) | 用于设置带有指定名称和整数值的响应头部. |
| void | setLocale(Locale loc) | 用于设置响应的区域对象. 参数loc为指定Locale对象 |
| void | setStatus(int sc) | 设置响应的状态码. 该方法通常用于没有错误的返回. 如果有错误发生, 则应当使用sendError方法代替. |

### session对象(会话有效)

session对象指的是客户端与服务器的一次会话，从客户连到服务器的一个WebApplication开始，直到客户端与服务器断开连接为止。它是HttpSession类的实例.

|  |  |
| --- | --- |
| 解决Session消耗大量内存的两个方法: | |
| 方法一 | 运用Java web容器的Session管理工具, 对Session进行持久化管理. |
| 方法二 | 如果把共享数据保存中request范围内也能完成和存放中Session范围内同样的功能, 则优先考虑保存中request范围内. |

常用方法

|  |  |  |
| --- | --- | --- |
| long | getCreationTime() | 返回SESSION创建时间(自GMT1970年1月1日零点以来的毫秒数) |
| long | getLastAccessedTime() | 返回此SESSION被客户端最近一次请求的时间(自GMT1970年1月1日零点以来的毫秒数) |
| int | getMaxInactiveInterval() | 返回两次请求间隔多长时间(以秒为单位), 超出session有效期, 此SESSION将被取消(ms) |
| String[] | getValueNames() | 返回一个包含此SESSION中所有可用属性的数组 |
| void | invalidate() | 取消SESSION，使SESSION不可用, 就是让session过期. |
| boolean | isNew() | 返回服务器创建的一个SESSION,客户端是否已经加入 |
| void | removeValue(String name) | 删除SESSION中指定的属性 |
| void | setMaxInactiveInterval(int time) | 设置两次请求间隔多长时间此SESSION被取消(以秒为单位) |
| String | getId() | 返回SESSION创建时JSP引擎为它设的惟一ID号 |
| void | setAttribute(String name, Object obj) | 设置属性name的值. |
| Object | getAttribute(String name) | 获取属性name的值 |
| ServletContext | getServletContext() | 返回ServletContext对象 |
| void | removeAttribute(String name) | 移除名为name的属性 |
| Enumeration | getAttributeNames() | 获取所有有效范围的属性(PageContext对象没有该方法) |
| Object  void  void  Enumeration | getAttribute(String name, int Scope)  getAttribute(String, name, Object, obj, int scope)  removeAttribute(String name, int scope)  getAttributeNamesInScope(int scope) | 页面上下文对象专用, 可以访问所有范围的属性和属性名.  在PageContext中可以通过制定scope的值来设置 和 获取不同有效范围的属性.  参数scope是PageContext中定义的常量:  PAGE\_SCOPE|REQUEST\_SCOPE|SESSION\_SCOPE|APPLICATION\_SCOPE |

### application对象(应用有效)

application对象实现了用户间数据的共享，可存放全局变量。它开始于服务器的启动，直到服务器的关闭，在此期间，此对象将一直存在；这样在用户的前后连接或不同用户之间的连接中，可以对此对象的同一属性进行操作；在任何地方对此对象属性的操作，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。它是ServletContext类的实例。

|  |  |  |
| --- | --- | --- |
| Object | getAttribute(String name) | 返回给定名的属性值 |
| Enumeration | getAttributeNames() | 返回所有可用属性名的枚举 |
| ServletContext | getContext(String uripath) | 返回指定WebApplication的application对象 |
| String | getInitParameter(String parameterName) | 获取初始化参数 |
| Enumeration | getInitParameterNames() | 获取所有的初始化参数名 |
| int | getMajorVersion() | 返回服务器支持的Servlet API的最大版本号 |
| String | getMimeType(String file) | 返回指定文件的MIME类型 |
| int | getMinorVersion() | 返回服务器支持的Servlet API的最大版本号 |
| RequestDispaer | getNamedDispatcher(String name) | 返回一个RequestDispaer对象, 该对象相当于指定名称的Servlet对象的一个包装器 |
| String | getRealPath(String path) | 返回一虚拟路径的真实路径 |
| RequestDispatcher | getRequestDispatcher(String uripath) | 返回指定资源的RequestDispatcher对象 |
| URL | getResource(String path) | 返回指定资源(文件及目录)的URL路径 |
| InputStream | getResourceAsStream(String path) | 返回指定资源的输入流 |
| Set | getResourcePath(String path) | 返回web应用中所有资源路径的集合 |
| String | getServerInfo() | 返回JSP(SERVLET)引擎名及版本号 |
| Servlet | getServlet(String name) | 返回指定名的Servlet |
| String | getServletContextName() | 获取Servlet上下文名称, 该名称是在部署描述文件的<display-name>元素中定义的 |
| Enumeration | getServletNames() | 返回所有Servlet名的枚举 |
| Enumeration | getServlets() | 返回所有Servlet的枚举 |
| void | log(Exception exception,String msg) | 把指定异常的栈轨迹及错误消息写入Servlet的日志文件 |
| void | log(String msg) | 把指定消息写入Servlet的日志文件 |
| void | log(String msg,Throwable throwable) | 把栈轨迹及给出的Throwable异常的说明信息写入Servlet的日志文件 |
| void | removeAttribute(String name) | 删除一属性及其属性值 |
| void | setAttribute(String name,Object obj) | 设定属性的属性值 |

### pageContext对象(页内有效)

pageContext对象提供了对JSP页面内所有的对象及名字空间的访问, 也就是说他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，他相当于页面中所有功能的集大成者，它的本 类名也叫pageContext。

字段

|  |  |
| --- | --- |
| APPLICATION | ServletContext的名称 |

方法

|  |  |  |
| --- | --- | --- |
| Object | fndAttribute() | 搜索指定的属性 |
| Object | findAttribute(String name) | 寻找一属性,返回起属性值或NULL |
| void | forward(String relativeUrlPath) | 使当前页面重导到另一页面 |
| void | forward(String URLPath) | 将当前请求转发到指定的URL(相对路径) |
| Object | getAttribute(String name) | 取属性的值 |
| Object | getAttribute(String name)  或  getAttribute(String name, int scope) | 如果未指定范围, 默认为页面范围返回指定属性的值 |
| Object | getAttribute(String name,int scope) | 在指定范围内取属性的值 |
| Enumeration<String> | getAttributeNamesInScope(int scope) | 返回指定作用域中所有属性的名称集合 |
| Enumeration | getAttributeNamesInScope(int scope) | 返回指定范围内可用的属性名枚举 |
| int | getAttributeScope(String name) | 返回指定属性的共享范围 |
| int | getAttributeScope(String name) | 返回某属性的作用范围 |
| ErrorDate | getErrorDate() | 返回错误页面(使用page指令的isErrorPage属性)的ErrorData对象 |
| Exception | getException() | 取得页面的异常对象 |
| Exception | getException() | 返回当前页的Exception对象(exception) |
| exceptionEvaluator | getExceptionEvaluator | 返回页面的exceptionEvaluator对象,  从Servlet2.0开始, 不再推荐使用, 该方法已被JspApplicationContext.getExpressionFactory方法代替 |
| JspWriter | getOut() | 返回当前客户端响应被使用的JspWriter流(out) |
| Object | getPage() | 返回当前页的Object对象(page) |
| ServletRequest | getRequest() | 返回当前页的ServletRequest对象(request) |
| ServletResponse | getResponse() | 返回当前页的ServletResponse对象(response) |
| ServletConfig | getServletConfig() | 返回当前页的ServletConfig对象(config) |
| ServletContext | getServletContext() | 返回当前页的ServletContext对象(application) |
| HttpSession | getSession() | 返回当前页中的HttpSession对象(session) |
| VariableResolver | getVariableResolver() | 返回页面的VariableResolver对象  从Servlet2.0开始, 不再推荐使用, 该方法已被ELContext.getELRedolver方法代替. |
| void | handlePageException(Exception e) | 处理页面中未得到处理的异常 |
| void | include(String relativeUrlPath) | 在当前位置包含另一文件 |
|  | initialize() | 用于初始化pageContext对象  该方法通常诶JspFactory.getPageContext方法所调用, 用于初始化状态 |
| JspWriter | popBody() | 返回由pushBody()方法保存的out对象(out对象出栈) |
| BodyContent | pushBody() | 用于保存当前的out对象, 并返回一个新的BodyContext对象(out对象入栈) |
| void | release() | 释放pageContext所占用的资源 |
| void | removeAttribute(String name) | 删除某属性 |
| void | removeAttribute(String name,int scope) | 在指定范围删除某属性 |
| void | setAttribute(String name,Object attribute) | 设置属性及属性值 |
| void | setAttribute(String name,Object obj,int scope) | 在指定范围内设置属性及属性值 |

### page对象

page对象就是指向当前JSP页面本身，有点象类中的this指针，它是java.lang.Object类的实例

|  |  |  |
| --- | --- | --- |
| class | getClass | 返回此Object的类 |
| int | hashCode() | 返回此Object的hash码 |
| boolean | equals(Object obj) | 判断此Object是否与指定的Object对象相等 |
| void | copy(Object obj) | 把此Object拷贝到指定的Object对象中 |
| Object | clone() | 克隆此Object对象 |
| String | toString() | 把此Object对象转换成String类的对象 |
| void | notify() | 唤醒一个等待的线程 |
| void | notifyAll() | 唤醒所有等待的线程 |
| void | wait(int timeout) | 使一个线程处于等待直到timeout结束或被唤醒 |
| void | wait() | 使一个线程处于等待直到被唤醒 |
| void | enterMonitor() | 对Object加锁 |
| void | exitMonitor() | 对Object开锁 |

### config对象

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）

|  |  |  |
| --- | --- | --- |
| ServletContext | getServletContext() | 返回含有服务器相关信息的ServletContext对象 |
| String | getServletName() | 获取servlet名称 |
| Enumeration | getInitParameterNames() | 返回Servlet初始化所需所有参数名的枚举 |
| String | getInitParameter(String name) | 返回初始化参数的值 |

### exception对象

exception对象是一个例外对象，当一个页面在运行过程中发生了例外，就产生这个对象。如果一个JSP页面要应用此对象，就必须把isErrorPage设为true，否则无法编译。他实际上是java.lang.Throwable的对象

|  |  |  |
| --- | --- | --- |
| Throwable | fillInStackTrace() | 将当前异常堆栈跟踪信息填充到exception对象中 |
| Throwable | getCause() | 返回当前异常对象中的异常 |
| String | getLocalizedMessage() | 返回本地化错误提示信息 |
| String | getMessage() | 返回描述异常的消息 |
| StackTrackElement[] | getStackTrace() | 返回堆栈跟踪信息 |
| Throwable | initCause | 将指定的异常对象嵌入到当前异常对象中 |
| String | toString() | 返回关于异常的简短描述消息 |
| void | printStackTrace() | 显示异常及其栈轨迹 |
| void | setStackTrace(StackTrackElement[] stackTrace) | 设置异常对象的堆栈跟踪信息 |
| String | toString() | 返回异常对象的简短描述 |

## Cookie对象

Cookie对象创建时需要制定名称和值, Cookie对象的名称只能使用可打印的ASCII字符, 不能包含逗号, 空格, 分号, 并且不能以$开头. 在Cookie创建后, 值可以更改而名称不可以更改.

常用属性

|  |  |
| --- | --- |
| name | Cookie名称 |
| value | Cookie值 |
| domain | 只有在该域中的服务器才会发送该Cookie. 默认情况下, Cookie对象的域(domain)是发送Cookie去向的服务器. 这在对多服务器处理请求的大型环境下, 才需要设置Cookie对象的域. |
| maxAge | Cookie持续存在的时间, 单位为秒, -1表示一直有效. 如果要删除过去有效的Cookie, 只需将其有效期设置为0.  如果是整数, 在有效期内, 即使用户关闭浏览器再重新启动, Cookie依然有效.  如果是负数, 则只当浏览器运行时有效, 关闭浏览器时, Cookie就会无效. |
| path | 只有URL中包含指定的路径的服务器才发送该Cookie |
| secure | 是否仅当使用https协议才发送Cookie. https是http的加密形式. |

常用方法

|  |  |  |
| --- | --- | --- |
|  | 方法签名 | 方法功能 |
| 构造方法 | Cookie(String name, String value) | 创建新的Cookie |
| String  String | getName()  getValue() | 获取已有Cookie的信息 |
| void | setValue(String newValue) | 改变Cookie的值 |
| String  void | getDomain()  setDomain(String newPath) | 设置和获取域 |
| String  void | getPath()  setPath(String newPath) | 设置和获取路径 |
| int  void | getMaxAge()  setMaxAge(int maxAge) | 设置和获取Cookie有效期, 单位为秒. |
| void  boolean | setSecure(secureFlag)  getSecure() | 设置和获取Cookie的安全设置, 值为true和false. |
| int  void | getVersion()  setVersion(int version) | 获取和设置Cookie协议版本 |
| String  void | getComment()  setComment(String comment) | 保存Cookie对象时提示该用户Cookie的说明文字 |

## JSTL

|  |  |  |
| --- | --- | --- |
| c | <%@ taglib prefix="c" | uri="http://java.sun.com/jsp/jstl/core"%> |
| fmt | <%@ taglib prefix="fmt" | uri="http://java.sun.com/jsp/jstl/fmt"%> |
| sql | <%@ taglib prefix="sql" | uri="http://java.sun.com/jsp/jstl/sql"%> |
| XML | <%@ taglib prefix="xml" | uri="http://java.sun.com/jsp/jstl/xml"%> |
| fu | <%@ taglib prefix="fn" | uri="http://java.sun.com/jsp/jstl/functions"%> |

备注:

EL的配置: Servlet 2.4以下容器, 需要手动开启EL: <%@ page isELIgnored=”false” %>

JSTL的配置: 需要拷贝两个包到lib下: jstl.jar(接口) , standard.jar(实现), 如果只使用EL表达式, 可以不拷贝这两个jar包

### <c: >核心标记库

|  |  |  |  |
| --- | --- | --- | --- |
| 通用标记 | <**c:out**  value="uaer.name"  escapeXML="是否忽略XML标记, 默认为true"  default="默认值">  默认值  </c:out> | escapeXML意思是: 是否忽略XML标记  相当于<%= user.getName()%>   |  | | --- | | 技巧: 当日期的值为null时, 输出默认值  <fmt:formatDate var="dateVar" value="${dueDate}" />  <c:out value="${dateVar}" default="没有指定日期" /> | |
| **<c:set**  var="变量名, 不可接受动态值"  value="值的表达式, 可以接受动态值"  target="指定目标对象, 可以是一个Map对象, 也可以是一个JavaBean对象, 该bean至少有一个含有设置方法的属性. 可以接受动态值"  scope="指定变量的作用范围, 默认page|request|session|application"  property="用于指定目标对象的属性, 可以接受动态值."  />  <c:set var="变量名">  <c:out value="${customer.address.street}"  <c:out value="${customer.address.city}"  </c:set> | **java代码调用EL的值**  <c:set scope="page" var="status" value="${mri.yarnMapreduceStatus}" />  <%=ApplicationState.fromCode(pageContext.getAttribute("status").toString()).getDesc() %> |
| <**c:remore**  var="指定要移除的变量名"  scope="指定变量的作用范围, 默认page|request|session|application"/> | 删除特定范围中指定的变量. |
| <**c:catch** var="指定变量名, 不可接受动态值">  ……  </c:catch> | 捕获本体内嵌套标记抛出的任何异常 |
| 条件处理 | <**c:if**  text="仅当所指定的测试表达式的计算结果为true时, 才计算本体"  var="指定变量名, 不可接受动态值."  scope="指定变量的作用范围, 默认page|request|session|application">  </c:if> | 计算结果也可以保存为一个作用域变量. |
| <**c:choose**>  <**c:when** test="指定测试表达式,可以接受动态值, 当此表达式为true时, 输出体中的内容">…</c:when>  <c:when text="…">…</c:when>  ……  <c:otherwise>…</c:otherwise>  </c:choose> | <**c:choose**>动作只允许第一个测试表达式计算结果为true的<c:when>动作得到处理. |
| 循环和迭代 | <**c:forEach**  var="是一个变量, 用于保存当前循环取出的临时值, 不可接受动态值"  items="用于指定进行迭代的集合, 其类型为Collection|Iterator|Enumeration|Map|对象数组|基本类型数组等, 可以接受动态值"  varStatus="指定保存LoopTagStatus对象的嵌套变量名, 不可接受动态值"  begin="用于指定开始索引, 从0计起, 可以接受动态值"  end="用于指定结束索引, 可以接受动态值. 元素索引不能大于该结束索引, 从0计起. 如果end小于begin, 则不会计算<c:forEach>动作体"  step="指定迭代时索引的递增值, 默认为1, 可以接受动态值.">  ……  </c:forEach> | 遍历   |  | | --- | | <c:forEach items=*"${atts}"* var=*"item"* varStatus=*"s"*>  <c:out value=*"${item}"*/>的四种属性：  index：${s.index}</br>//从0开始计数  count：${s.count}</br>//从1开始计数  first：${s.first}</br>//第一个  last：${s.last}</br>//最后一个  </c:forEach> |  |  |  | | --- | --- | | 遍历数组: | <c:forEach var="age" items="<%= ages %>">  <c:out value="${age}" />  </c:forEach> | | 遍历List: | <c:forEach var="name" items="${names}">      <c:out value="${name}" />  </c:forEach> | | 遍历Set: | <c:forEach var="entry" items="<%= set %>">      <c:out value="${entry}" />  </c:forEach> | | 遍历Map: | <c:forEach var="entry" items="<%= map %>">  <c:out value="${entry.key}"/>  <c:out value="${entry.value}"/>  </c:forEach>  <c:out value="${map[key]}" /> | | 遍历UserBean的List: | <c:forEach var="user" items="${users}">  <c:out value="${user.username}"/>  <c:out value="${user.password}"/>  </c:forEach> |  |  | | --- | | 让表格隔行样式相同同  <tr  <c:if test="${status1.index%2 eq 0}">  class='form\_odd'  </c:if>  <c:if test="${status1.index%2 eq 1}">  class='form\_even'  </c:if>  >  <td><c:out value="${status1.count}"/></td>  </tr>  </c:forEach> | |
| <**c:forTokens**  var="指定保存当前元素的嵌套变量的名称, 不可以接受动态值."  items="用于指定要处理的字符串, 可以接受动态值, 如:红-黄-蓝"  delims="用于指定分隔符, 可以有多个分隔符, 多个分隔符挨着写,就如 ("|,")这两个分隔符组成的分隔符列表,可以接受动态值, 如: - "  varStatus="用于指定保存LoopTagStatus对象的嵌套变量名, 不可接受动态值"  begin="用于指定开始索引, 从0计起, 默认为0, 可以接受动态值."  end="用于指定结束索引, 从0计起. 如果end小于begin, 则不计算<c:forTokens>动作体. 可以接受动态值"  step="指定迭代时索引的递增值, 默认为1, 可以接受动态值.">  <c:out value="${item}/>  </c:forTookens> | 指定分隔符来拆分字符串 |
| URL操作 | <**c:url**  value="用于指定页面相对路径或上下文相对路径, 也可以是一个绝对URL. 可以接受动态值"  context="用于指定同一容器中, 一个外部web应用的上下文相对路径, 要以一个斜线'/'开头. 可以接受动态值"  var="用于指定保存URL的变量名, 不可接受动态值"  scope="指定变量的作用域范围, 如page|request|session|application等, 不可以接受动态值"> | 将一个url格式的字符串保存到一个变量中  <a href="${pageContext.request.contextPath}/user/login">登录</a> |
| <**c:import**  url="用于指定所要导入资源的URL, 可以接受动态值"  context="用于指定同一容器中, 一个外部web应用的上下文相对路径, 要以一个斜线'/'开头, 可以接受动态值."  var="指定保存资源内容的变量名称, 不可以接受动态值."  scope="指定变量的作用范围, 默认page|request|session|application, 不可接受动态值"  charEncoding="指定内容的编码方式, 默认为导入机制(协议)所提供的编码方式, 如果未提供, 则编码为ISO-8859-1. 可以接受动态值"  varReader="用于指定java.io.Reader对象的嵌套变量名, 不可以接受动态值."  > | 导入一个内部或外部资源, 类似于<jsp:import>  导入一个controller地址  <c:import url=*"/alarm/page"* /> |
| <**c:redirect**  url="用于指定页面相对路径或上下文相对路径, 也可以是一个绝对URL. 可以接受动态值."  context="用于指定同一容器中, 一个外部web应用的上下文相对路径, 要以一个斜线(/)开头, 可以接受动态值"> | 重定向 |
| 向URL增加一个请求参数 | <**c:param**  name="用于指定参数的名称, 可以接受动态值."  value="用于指定参数的值, 可以接受动态值."  > | <c:param>动作是<c:import>|<c:redirect>|<c:url>等动作的一个嵌套动作, 用于向URL增加一个请求参数. |

### <xml: > XML标记库(不常用)

|  |  |  |
| --- | --- | --- |
| XML核心标记 | <x:parse> | 用于解析一个XML文档的内容. |
| <x:out> | 用于输出XPath表达式的计算结果. |
| <x:set> | 用于将一个XPath表达式的计算结果保存到一个作用域变量中. |
| XML流程控制标记 | <x:if> | 用于表示XML条件判断标记, 仅当指定的XPath表达式计算为true时, 才计算其动作体, 计算结果可以保存为一个作用域变量. |
| <x:when> | 用于表示一个<x:choose>动作中的互斥选项之一, 仅当该动作的测试表达式为true时, 且是第一个测试为true的<x:when>动作时, 才会计算其动作体. |
| <x:choose> | 用于嵌套使用<x:when>和<x:otherwise>动作标记. |
| <x:otherwise> | 用于表示一个<x:choose>动作中的默认选项, 仅当它前面所有的<x:when>动作的测试表达式计算都为false时, 才会计算其本体. |
| <x:forEach> | 用于表示XML循环标记. |
| XML转换标记 | <x:transform> | 用于使用一个XSLT样式表转换一个源XML文档. |
| <x:param> | 用于转换(Transform)添加一个参数, 而且只能用于<x:transform>动作体中. |

### <sql: > SQL标记库(不常用)

|  |  |
| --- | --- |
| <sql:setDataSource> | 用于建立一个数据源 |
| <sql:query> | 执行sql查询(select) |
| <sql:update> | 执行sql更新(insert, update, delete) |
| <sql:param> | 为sql语句中的参数指定值. |
| <sql:dateParam> | 用于为SQL语句中的参数指定java.utilDate类型的值. |
| <sql:transaction> | 用于为嵌套数据库动作元素提供一个共享连接, 将所有的语句作为一个事务来处理. |

### <fmt: > 格式标记库

主要包含语言, 数字和日期等格式化动作.

|  |  |  |
| --- | --- | --- |
| 语言 | <fmt:setLocal  value="指定一个本地化地域代码, 可以认为是一个Locale对象的字符串描述, 该字符串包含一个ISO-693语言代码,  其后可以包含一个ISO-3166国家代码, 之间需由字符"-"或"\_"分割, 如"en\_US", 可以接受动态值"  variant="指定特定于供应商或浏览器的变量参数代码, 例如用WIN代表Windows, 用MAC代表Macintosh,  如果是两个变量, 则用下划线分隔开, 把最重要的放在前面, 可以接受动态值 "  scope="指定本地化环境配置变量的作用范围, 可选page|request|session|application, 不可接受动态值"> | 设置本地化语言地域 |
| <fmt:requestEncoding  value="指定解释请求参数的字符编码方式, 可以接受动态值."> | 设置请求的字符编码方式. |
| <fmt:message  key="用于指定要查找的消息键, 可以接受动态值.  如果key为null, 则在页面上会产生"??????"的错误消息, 如果找不到资源文件".properties", 则在页面上会产生"???<key>???"的错误消息"  bundle="指定用于查找消息键的整个资源束的本地化上下文,可以接受动态值."  var="用于指定保存本地化消息的变量名, 不可以接受动态值."  scope="用于指定变量的作用域范围, 可选page|request|session|application, 不可接受动态值"> | 指定键映射到本地化消息,  并完成参数替换. |
| <fmt:param  value="用于指定用于参数替换的参数值, 可以接受动态值."> | 为<fmt:message>动作提供一个  用于参数替换的参数值 |
| <fmt:bundle  basename="用于指定资源束的基本文件名, 可以接受动态值."  prefix="用于为嵌套<fmt:message>动作指定消息键的前缀, 可以接受动态值."> | 用于为其动作体中的JSP元素  装在一个资源束 |
| <fmt:setBundle  basename="用于指定资源束的基本文件名, 可以接受动态值"  var="用于指定保存本地化上下文变量的名称, 不可以接受动态值."  scope="用于指定变量或者本地化上下文配置变狼的作用域范围, 可选page|request|session|application, 不可接受动态值"> | 用于装载一个资源束,  并将其保存到一个变量或配置变量中. |
| 数字  货币  百分数 | <fmt:formatNumber  value="用于指定要格式化的数值, 可以接受动态值."  type="指定格式化后的数值类型, 可选number(指定格式化为常规数字)|currency(指定格式化为货币)|percent(指定格式化为百分数)"  pattern="用于指定格式化模式, 不依赖于内置模式. 指定该模式后, 该模式属性的值应当符合java.text.DecimalFormat类约定的模式字符串. 可接受动态值."  currencyCode="用于指定ISO-4217货币编码, 可以接受动态值. 备注: CNY代表￥, USD代表＄. "  currencySymbol="指定货币符号￥或＄等, 可以接受动态值."  groupingUsed="true|false. 用于指定是否在格式化输出中包含分组分隔符(就是逗号分隔的千分位符号), 可以接受动态值."  maxIntegerDigits="用于指定格式化输出中整数部分的最大位数, 可以接受动态值."  minIntegerDigits="指定格式化输出总整数部分的最小位数, 可以接受动态值."  maxFractionDigits="指定格式化输出总小数部分的最大位数, 可以接受动态值."  var="指定保存格式化结果字符串的变量名称, 不可以接受动态值."  scope="指定变量的作用域范围, 可选page|request|session|application, 不可接受动态值"  > | 用于将一个数值格式化成某种数字类型,  如常规数字, 货币或百分数.  <fmt:formatNumber value="12" type="currency" pattern=".00元"/> 将显示 12.00元  <fmt:formatNumber value="12" type="currency" pattern=".0#元"/> 将显示 12.0元  <fmt:formatNumber value="1234567890" type="currency"/> 将显示 ￥1,234,567,890.00  (那个钱的符号和当前web服务器的 locate 设定有关)  <fmt:formatNumber value="123456.7891" pattern="#,#00.0#"/> 将显示 123,456.79  <fmt:formatNumber value="12" type="percent" />将显示 1,200%  type 可以是currency、 number、 和percent。  补充一下:  <fmt:formatNumber value="${requestScope.aaaa}" type="currency" pattern=".00元"/>  这种写法，如果 request 里面的 aaaa 这个变量是 null 的话，会显示 .00元 ，不爽，还是改成  <fmt:formatNumber value="${requestScope.aaaa}" type="currency" pattern="0.00元"/>  比较好，并不影响有值的数字显示。 |
| <fmt:parseNumber  value="指定要解析的字符串, 如果解析失败, 则抛出一个java.text.ParseException异常, 可以接受动态值."  type="指定value将被解析成的数值类型, 可以接受动态值."  pattern="用于定制格式化模式, 例如:pattern="￥##0.00" "  parseLocale="用于指定解析操作期间所用的本地化地域代码, 如zh\_CN 或 en\_US. 可以接受动态值."  integerOnly="true|false. 用于判断是否仅解析给定值的整数部分, 可以接受动态值."  var="用于指定保存解析结果的变量名称, 可以接受动态值."  scope="用于指定变量的作用域范围, 可选page|request|session|application, 不可接受动态值"  > | 对某种数字类型(如常规数字, 货币 或 百分数)  的字符串进行解析. |
| 日期  时间 | <fmt:formatDate  value="用于指定要格式化的日期和时间, 可以接受动态值."  type="用于指定要格式化日期和时间的哪一部分, 可选time|date|both, 可以接受动态值."  dateStyle="用于为日期指定预定义的格式化样式, 可选default|short|medium|long|full, 仅适用于type为date或both的情况, 可以接受动态值."  timeStyle="用于为时间制定预定义的格式化样式, 可选default|short|medium|long|full, 仅适用于type为time或both的情况, 可以接受动态值."  pattern="用于为日期和时间定值格式化样式, 不依赖于内置样式,  指定该模式后, 该模式属性的值应当符合java.text.SimpleDateFormat类约定的模式字符串. 可接受动态值." 例如:pattern="yyyy-MM-dd"  timeZone="指定格式化时间所用的时区, 可以接受动态值."  var="指定格式化结果字符串的变量名称, 可以接受动态值."  scope="指定变量的作用域范围, 可选page|request|session|application, 不可接受动态值"> | 使用给定的样式或模式对日期和时间进行格式化  <fmt:formatDate value="${question.endDate}" type=*"date"*/>  <fmt:formatDate value="${productInstance.startDate}" pattern="yyyy-MM-dd"/>  <fmt:formatDate value="${topic.createDate}" pattern="yyyy-MM-dd HH:mm:ss" /> |
| <fmt:parseDate  value="用于指定要解析的日期和时间字符串, 可以接受动态值."  type="用于指定value将被解析成日期和时间的哪一部分, 可选time|date|both, 可以接受动态值"  dateStyle="用于为日期指定预定义的格式化样式, 从而确定如何对日期部分进行解析, 仅适用于type为date或both的情况, 可以接受动态值."  timeStyle="用于为时间指定预定义的格式化样式, 从而确定如何对时间部分进行解析, 仅适合于type为time或both的情况, 可以接受动态值."  pattern="用于定制格式化模式, 从而确定日期和时间字符串将如何被解析. 例如:MM/dd/yyyy HH:mm:ss "  timeZone="用于指定解析所有的时区, 可以接受动态值. 例如: timeZone="GMT+01:00" "  parseLocale="指定解析操作期间所用的本地化地域代码, 可以接受动态值."  var="指定保存解析结果的变量名称, 不可以接受动态值."  scope="指定变量的作用域范围, 可选page|request|session|application, 不可接受动态值"> | 对日期和时间类型的字符串进行解析 |
| <fmt:setTimeZone  value="用于指定时区, 可以认为是一个时区标识字符串, 该字符串可以是缩写(如PST|CST),  时区全名(如America/Los\_Angeles) 或 自定义标识(如GMT-08:00), 可以接受动态值"  var="指定保存java.util.TimeZone类型的时区变量名称, 不可以接受动态值."  scope="用于指定变量或时区配置变量作用域范围, 可选page|request|session|application, 不可接受动态值"> | 用于将指定时区保存到时区配置变量中 |
| <fmt:timeZone  value="用于指定时区, 可以认为是一个时区标识字符串, 该字符串可以是缩写(如PST|CST),  时区全名(如America/Los\_Angeles) 或 自定义标识(如GMT-08:00), 可以接受动态值"> | 用于为其动作体中嵌套的<fmt:formatDate>动作  或<fmt:parseDate>动作指定时区. |

### <fn: > 函数标记

#### 标准函数

JSTL的函数跟别的标签不一样, 它只有放在EL表达式中使用. 用于处理字符串的.

使用格式: ${fn:contains("jinwanlin", "jin")} ; 返回值为true

|  |  |
| --- | --- |
| 获取字符串, list, set长度 | length |
| 改变字符串的大小写 | toLowerCase, toUpperCase |
| 获取字符串的子串 | substring, substringAfter, substringBefore |
| 去除字符串两头的空格 | trim |
| 替换字符串中的字符 | replace |
| 检查字符串是否包含另外的字符串 | indexOf, startsWith, endsWith, contains, containsIgnoreCase |
| 分割字符串, 合并字符串 | split, join |
| 字符串中XML标记的转义处理 | escapeXml |

方法详细描述

|  |  |
| --- | --- |
| fn:contains(inputString, subString)  inputString="用于指定输入的字符串, 其类型为String, 如果inputString参数为null, 则将被作为空字符串来处理."  subString="指定测试子字符串, 类型为String, 如果subString参数为null, 则将被作为空字符串来处理." | 用于判断输入字符串是否包含指定子字符串.  如果inputString字符串包含subString字符串, 则返回true, 否则返回false.  注意: 大小写是敏感的. |
| fn:containslgnoreCase(inputString, subString)  inputString="用于指定输入的字符串, 其类型为String, 如果inputString参数为null, 则将被作为空字符串来处理."  subString="指定测试子字符串, 类型为String, 如果subString参数为null, 则将被作为空字符串来处理." | 用于判断输入字符串是否包含指定子字符串.  与fn:contains()不同的是fn:containsIgnoreCase()函数将忽略大小写. |
| fn:startsWith(inputString, prefex)  inputString="用于指定输入的字符串, 其类型为String, 如果inputString参数为null, 则将被作为空字符串来处理."  prefex="指定前缀, 其类型为String, 如果inputString参数为null, 则将被作为空字符串来处理." | 用于判断输入字符串是否以指定的前缀开始.  如果inputString字符串是以prefix字符串开始的, 则返回true, 否则返回false. |
| fn:endsWith(inputString, suffix)  inputString="用于指定输入的字符串, 其类型为String, 如果inputString参数为null, 则将被作为空字符串来处理."  suffix="指定后缀, 其类型为String, 如果inputString参数为null, 则将被作为空字符串来处理." | 用于判断输入字符串是否以指定的后缀结束.  如果inputString字符串是以suffix字符串结束的, 则返回true, 否则返回false. |
| fn:escapeXml(inputString)  inputString="指定输入的字符串, 其类型为String, 如果inputString参数为null, 则将被作为空字符串来处理." | 用于将输入字符串中的<, >, &, ', " 等字符转换为相应的XML字符实体码.返回转换后的字符串. |
| fn:indexOf(string, subString)  string="用于指定用于计算的字符串, 其类型为String, 如果inputString参数为null, 则将被作为空字符串来处理."  subString="用于指定子字符串, 其类型为String, 如果subString参数为null, 则将被作为空字符串来处理." | 用于计算一个字符串中, 指定子字符串第一次出现的索引. (大小写是敏感的)  如果string中包含subString字符串, 则返回subString字符串在string字符串中第一次出现的位置.  否则返回-1. 如果subString字符串为null, 则返回0. |
| fn:split(string, delimiters)  string="指定要被分离的字符串, 其类型为String, 如果inputString参数为null, 则将被作为空字符串来处理."  delimiters="指定分隔符列表字符串, 其类型为String, 如果inputString参数为null, 则将被作为空字符串来处理." | 用于将一个字符串分离成一个子字符串数组.  返回分离后的字符串数组 |
| fn:join(array, separator)  array="指定要连接的数组, 其类型为String[], 如果array参数为null, 则将被作为空字符串数组来处理."  separator="用来指定用于连接的字符串, 其类型为String, 如果separator参数为null, 则将被作为空字符串来处理." | 用于将数组中的全部元素连接成一个字符串.  返回连接后的字符串 |
| fn:replace(inputString, beforeString, afterString)  inputString="指定输入的字符串, 其类型为String, 如果inputString参数为null, 则将被作为空字符串来处理."  beforeString="指定要替换的字符串, 其类型为String, 如果beforeString参数为null, 则将被作为空字符串来处理."  afterString="指定替换后的字符串, 其类型为String, 如果afterString参数为null, 则将被作为空字符串来处理." | 用于将字符串中的某些字符串替换成指定字符串.  返回一个字符串, 该字符串是将inputString字符串中所有的beforeString字符串  替换成afterString字符串的结果. |
| fn:trim(String)  String="指定原始的字符串, 其类型为String, 如果inputString参数为null, 则将被作为空字符串来处理." | 用于移除一个字符串两端的空格.  返回删除前后空格之后的字符串. |
| fn:substring(inputString, beginIndex, endIndex)  inputString="指定输入的字符串, 其类型为String, 如果inputString参数为null, 则将被作为空字符串来处理."  beginIndex="用于指定要获取的子字符串的起始位置, 其类型为int.  如果beginIndex 大于输入字符串的长度, 则返回空字符串, 如果beginIndex小于0, 则将被设置为0.  endIndex="用于指示要获取的子字符串的结束位置, 其类型为int.  如果endIndex大于输入字符串的长度或小于0, 则将被设置为输入字符串的长度;  如果endIndex小于beginIndex, 则返回空字符串" | 用于获取一个字符串的子字符串.  返回inputString字符串中, 从beginIndex字符串到endIndex字符串之间的子字符串.  **实例:**标题, 截取字符串, 并显示省略号:  ${fn:substring(adminNotice.title, 0, 10)}  ${fn:length(adminNotice.title) > 10 ? "..." : ""} |
| fn:substringAfter(inputString, subString)  inputString="用于指定输入的字符串, 其类型为String, 如果inputString参数为null, 则将被作为空字符串来处理."  subString="用于指定子字符串, 其类型为String, 如果subString参数为null, 则将被作为空字符串来处理." | 用于获取一个字符串中指定子字符串之后的子字符串.  返回inputString字符串中subString字符串之后的子字符串. |
| fn:substringBefore(inputString, subString)  inputString="指定输入的字符串, 其类型为String, 如果inputString参数为null, 则将被作为空字符串来处理.""  subString="用于指定子字符串, 其类型为String, 如果subString参数为null, 则将被作为空字符串来处理." | 用于获取一个字符串中指定子字符串之前的子字符串.  返回inputString字符串中subString字符串之前的子字符串 |
| fn:toLowerCase(inputString)  inputString="指定输入的字符串, 其类型为String, 如果inputString参数为null, 则将被作为空字符串来处理."" | 用于将字符串中的所有字符串转换为小写字符.  返回将inputString转换为小写字符之后的字符串. |
| fn:toUpperCase(inputString)  inputString="指定输入的字符串, 其类型为String, 如果inputString参数为null, 则将被作为空字符串来处理."" | 用于将字符串中的所有字符串转换为大写字符.  返回将inputString转换为大写字符之后的字符串. |
| fn:length(input)  input="指定输入的对象, 可以是一个集合或字符串, 其类型为Object,  如果input参数为null, 则将被作为空集合对象或空字符串处理." | 获取一个集合中的元素数或一个字符串中的字符数.  返回: input对象的大小(整数类型) (集合的元素数 或 字符串的字符数)  获取list或set的长度  ${fn:length(list)}  ${fn:length(set)} |

#### 自定义函数

|  |
| --- |
| 一个EL自定义函数实际上是映射到java类的一个static方法. 具体的映射在标记库描述文件(Tag Library Descriptor, TLD)中定义.  在一个TLD中, 自定义函数名必须是唯一的.  要点:  1. 类和方法 必须是public static的  2. 编写tld文件(参见standard.javr/META-INF/fn.tld), 放在WEB-IFN的任何子目录中都可以, 一般放在WEB-IFN下面.  3. 在JSP中引入 和 使用 |
| <?xml version="1.0" encoding="UTF-8" ?>  <taglib xmlns=<http://java.sun.com/xml/ns/j2ee>  xmlns:xsi=<http://www.w3.org/2001/XMLSchema-instance>  xsi:schemaLocation="  http://java.sun.com/xml/ns/j2ee  http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary\_2\_0.xsd"  version="2.0">  <description>JSTL 1.1 functions library</description> <!-- 描述 -->  <display-name>JSTL functions</display-name> <!-- 描述-->  <tlib-version>1.1</tlib-version> <!-- 版本号, 保持这个 -->  <short-name>fn</short-name> <!-- 前缀, 需要自己起名 -->  <uri>[http://java.sun.com/jsp/jstl/functions</uri](http://java.sun.com/jsp/jstl/functions%3c/uri)> <!-- 命名空间, 需要自定义 -->  <function>  <description>函数描述</description>  <name>length</name> <!-- 标签中使用的函数名 -->  <function-class>org.apache.taglibs.standard.functions.Functions</function-class> <!-- 函数所在全类名 -->  <function-signature>int length(java.lang.Object)</function-signature> <!-- 返回值类型 函数名(参数类型) 注意: 基本类型简写 , 非基本类型写全类名 -->  <example>You have ${fn:length(shoppingCart.products)} in your shopping cart. </example>  </function>  </taglib> |
| 在web.xml中加入配置  <taglib>  <taglib-uri>/myGreeting</taglib-uri> <-- /myGreeting 是在JSP中引入自定义函数标签的属性uri的值-->  <taglib-location>/WEB-INF/taglib.tld</taglib-location>  </taglib> |

简单自定义函数

复杂自定义函数

标记属性中使用自定义函数

函数的嵌套

## EL(表达式语言Expression Language)

### 保留字

|  |  |  |  |
| --- | --- | --- | --- |
| And | Or | No | Instanceof |
| eq | ne | le | ge |
| lt | gt | empty | null |
| true | false | div | mod |

### 1. 字符串和数字之间类型自动转换

|  |  |
| --- | --- |
| Object转换为String | Object为数值时, 把其值直接转换为字符串 |
| Object==null, 返回长度为0的字符串: "" |
| 如果Object.toString()产生异常, 出错; 否则返回Object.toString() |
| Object转换为数值 | 如果Object为boolean类型, 出错; |
| 如果Object==null, 返回0; |
| 如果Object=="", 返回0; |
| 如果Object为字符串, 且字符串可以转换为数值, 返回数值; 否则出错. |

### 2. 空值代替异常

### 3. 算术运算符, 关系运算符, 逻辑运算符, empty运算符

|  |  |  |  |
| --- | --- | --- | --- |
| 算术运算符 | + |  |  |
| - |  |  |
| \* |  |  |
| / | div |  |
| % | mod |  |
| 关系运算符 | == | eq | equal |
| != | ne | not equal |
| < | lt | less-than |
| > | gt | greater-than |
| <= | le | less-or-equal |
| >= | ge | greater-or-equal |
| 逻辑运算符 | && | and |  |
| || | or |  |
| ! | not |  |
| empty运算符 | {empty 变量} |  | 如果操作数为null, 返回true; 或者操作数本身是一个空容器, 空数组, 或长度为0的字符串等, 返回true 例子:  ${**empty** bYNote}  ${**empty** bYNote ? "" : "checked='checked'"} |

|  |  |  |
| --- | --- | --- |
| 运算类型 | 缩写 | 英文 |
| 相等 | eq | equal |
| 不相等 | ne、neq | not equal |
| 大于 | gt | greater than |
| 小于 | lt | less than or equal |
| 大于等于 | gte、ge | great than or equal |
| 小于等于 | lte、le |  |
| 非 | not |  |
| 求模 | mod |  |

### 4. 支持条件输出

|  |
| --- |
| ${test ? expression1 : expression2}  例子: ${**empty** bYNote ? "" : "checked='checked'"} //如果bYNote为空, 输出"", 否则输出checked='checked' |

### 5. 对请求参数, cookie, 其它请求数据的简洁访问.

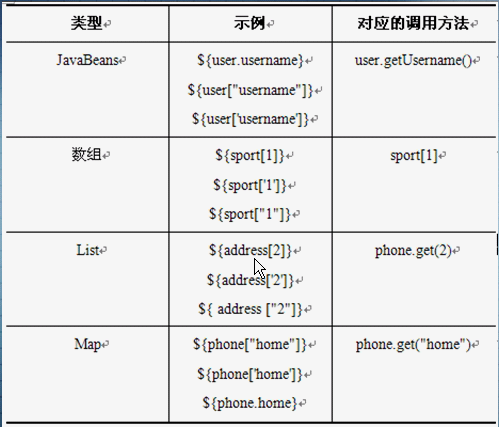
|  |  |  |  |
| --- | --- | --- | --- |
| EL隐含对象 | pageContext | 页面上下文对象,可获取JSP另外八大隐含对象 | ${pageContext.session.id} 相当于 <%=session.getId()%> |
| param | map对象,包含页面所有参数(名称--值) | ${param.custId} 相当于 <%=request.getParameter("custID")%> |
| paramValues | map对象,包含页面所有参数(名称—多个值) | <c:forEach items="${paramValues.phone}" begin="1" **var**="curPhone">  <c:out value="${curPhone}"  </c:forEach> |
| header | map对象,包含每个header(名称—值) | 注意: 如果属性名中包含非字母和数字字符, 只能使用[]访问, 例如: ${header["User-Agent"]} |
| headerValues | map对象,包含每个header(名称—多个值) | 很少使用 |
| cookie | map对象,包含每个Cookie(名称—值) | ${cookie.userCookie.name}: ${cookie.userCookie.value} |
| initParam | map对象,包含servlet上下文初始参数(参数名—值) | ${initParam.defaultColor.value} 相当于 <%=application.getInitParameter("defaultColor");  ${initParam["javax.servlet.jsp.jstl.fmt.localizationContext"]} |
| pageScope | map对象, 以变量名作为map的关键字 |  |
| requestScope | map对象, 以变量名作为map的关键字 |  |
| sessionScope | map对象, 以变量名作为map的关键字 |  |
| applicationScope | map对象, 以变量名作为map的关键字 |  |

### 6. ${name}默认依次从pageScope, requestScope, sessionScope, applicationScope范围内查找变量, 若没找到则返回空; 也可指定查找范围: ${requestScope.name}

### 7. 访问运算符: . 和 []

|  |  |
| --- | --- |
| .  点运算符 | 通常用于访问对象的属性. 例如: ${user.sex}  当被访问的属性本身也是对象时, 可以递归使用点运算符. 例如: ${user.birthday.year} |
| []  方括号运算符 | 通常用于检索数组或集合的元素   |  |  | | --- | --- | | ${email[1]} | 返回email数组的第二个元素 | | ${employee["id"]} | 对于实现java.util.Map接口的集合, 方括号运算符使用关联的键检索存储在映射中的值. | |
| 异同 | 方括号运算符与点运算符在某种程度上可以实现互换, 例如: ${user.sex} 等价于 ${user["sex"]}  当要检索的对象的属性名称中包含一些特殊字符(如"." 或"-")时, 通常使用方括号运算符. |

### 8. EL对集合元素(JavaBean/数组/List/Map)的简单访问



### 用EL获得List的size长度

|  |
| --- |
| 首先在页面前端处：  <%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions"%>  然后在你需要统计的地方写入：  ${fn:length(list)} |

## 自定义标记

## 自定义标签

|  |
| --- |
| 1. 继承SimpleTagSupport, 覆写doTag()把自定义标签代码写在里面. 2. 新建xml格式的文件, 起名:\*.tld; 拷贝一个tag头.   <tag>  <name>\*\*</name>  <tag-clas>\*\*</tag-class>  <body-content>empty</body-content>  </tag>   1. 在jsp中导入tag的URI |

### TagSupport类

|  |
| --- |
|  |

### BodyTagSupport类

### SimpleTagSupport类

### DynamicAttributes接口

### TryCatchFinally接口

### 标记库描述符

## JSP出错处理

### 编译时出错

返回HTTP协议错误代码500, 表示web服务器错误.

### 运行时出错

如果异常没被捕获, 就会将将客户端请求及出现的异常转向到由JSP页面定义的出错处理页面, 如果JSP没有指定出错页面, 会使用默认的处理, 即返回HTTP协议错误代码500.

错误处理页中会隐含一个对象变量exception, 使用该变量可以获取原来页面的异常信息. <%=exception %>

一般不使用错误处理页面来显示错误, 因为错误给客户, 客户也看不懂.

# servlet

|  |
| --- |
| Servlet生命周期: |

## Servlet接口

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
| void | init() | 初始化Servlet, 在Servlet实例化之后才被调用 |
| void | service(ServletRequest request, ServletResponse response) | 处理客户端请求, 在init方法成功完成后才被调用. |
| void | destroy() | 销毁Servlet, 只当service方法中的所有线程均已退出或超时时才能被调用. |
| ServletConfig | getServletConfig() | 获取ServletConfig对象, 该对象包含该Servlet的初始化和启动参数. |
| String | getServletInfo() | 获取Servlet相关信息, 该字符串是无格式的文本, 并且不支持任何标记. |

## GenericServlet类

GenericServlet是一个普通的, 协议无关的Servlet, 实现了Servlet和ServletConfig等接口.

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
| 构造函数 | GenericServlet() | 该构造函数什么也不做, Servlet所有的初始化都是有init方法完成的. |
| void | int() | 初始化Servlet |
| abstract void | service(ServletRequest request, ServletResponse response) | 处理客户端请求. |
| void | destroy() | 销毁Servlet |
| String | getInitParameter(String name) | 返回初始化参数的值 |
| Enumeration | getInitParameterNames() | 获取所有初始化的参数名 |
| ServletConfig | getServletConfig() | 返回ServletConfig对象, 该对象包含该Servlet的初始化参数, 上下文和实例名 |
| ServletContext | getServletContext() | 返回该Servlet的上下文, 通过ServletContext对象可以设置和访问属性. |
| String | getServletInfo() | 获取Servlet相关信息, 该字符串是无格式的文本, 并且不支持任何标记. |
| String | getServletName() | 获取Servlet的名称 |
| void | log(String msg) | 将制定信息写入Servlet日志文件中 |

## HttpServlet抽象类

扩展了GenericServlet类, 用于创建一个适用于web站点并支持HTTP协议的Servlet. 一个HttpServlet的子类必须至少重载doGet()或doPost()方法中的一个.

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
| 构造函数 | HttpServlet() | 什么都不做, 因为是一个抽象类 |
| void | init() | 初始化Servlet |
| void | service() | 处理客户端请求 |
|  | destory() | 销毁Servlet |
|  | doGet() |  |
|  | doPost() |  |
|  | doHead() |  |
|  | doPut() |  |
|  | doDelete() |  |
|  | doTrace() |  |
|  | doOptions() |  |
|  | getInitParameter() | 返回初始化参数的值 |
|  | getInitParameterNames() | 返回所有初始化参数名称的集合 |
|  | getLastModified() | 返回HttpServletRequest对象的最后访问时间 |

## ServletRequest接口

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
| Object | getAttribute(String name) | 返回指定属性的值, 如果该属性不存在, 则返回null |
| Enumeration | getAttributeNames() | 返回请求中所有属性的名称集合, 如果没有属性名, 则返回一个空枚举对象 |
| String | getCharacterEncoding() | 返回请求的字符编码方式, 如果请求未指定字符编码方式,则返回null. |
| int | getContentLength() | 返回内容的长度(字节数), 如果内容长度未知, 则返回-1. |
| String | getContentType() | 返回内容的MIME类型, 如果内容类型未知,则返回null. |
| ServletInputStream | getInputStream() | 返回请求输入流, 使用ServletInputStream对象将请求体内容作为二进制数据返回.  ServletInputStream sis = request.getInputStream();  **byte** b[] = **new** **byte**[1024];  **int** len = sis.readLine(b, 0, 1024);    response.setContentType("text/html;charset=UTF-8");  PrintWriter out = response.getWriter();    out.println("内容类型: ");  out.println(request.getContentType());  out.println("<br />");  out.println("内容长度: ");  out.println(request.getContentLength());  out.println("<hr />");    out.println("内容如下: ");  **for**(**int** i=0; i<len; i++){  out.print(b[i] + "");  }  out.close(); |
| String | getLocalAddr() | 获取服务器IP地址 |
| Locale | getLocale() | 返回首选的区域对象, 如zh\_CN |
| Enumeration | getLocales() | 获取所有的区域对象, 如zh\_CN |
| String | getLocalName() | 获取服务器名称 |
| int | getLocalPort() | 获取服务器端口号 |
| String | getParameter(String name) | 返回请求参数的值 |
| Map | getParameterMap() | 返回请求参数的参数映射(String类型 参数名--String数组类型 参数值) |
| Enumeration | getParameterNames() | 返回String类型枚举, 包含请求参数中所有的参数名称 |
| String[] | getParameterValues(String name) | 返回指定参数的所有值的集合, 如果指定的参数不存在, 则该方法返回null. 如果参数只有一个值,则数组长度为1 |
| String | getProtocol() | 返回请求所使用的协议名称和版本. 如"HTTP/1.1" |
| BufferedReader | getReader() | 返回包含请求体的BufferedReader对象 |
|  | getRealPath | Java Servlet 2.1 API以来, 不再推荐使用, 已被ServletContext.getRealPath()String取代 |
| String | getRemoteAddr() | 返回发送请求的客户端IP地址 |
| String | getRemoteHost() | 返回发送请求的客户端主机名 |
| int | getRemotePort() | 获取客户端端口号 |
| RequestDispatcher | getRequestDispatcher() | 返回RequestDispatcher对象, 该对象用作给定路径上的资源包装器. 如果Servlet无法返回一个RequestDispatcher对象, 则该方法返回null |
| String | getScheme() | 返回构成请求的协议机制, 协议机制包括HTTP, FTP, HTTPS |
| String | getServletName() | 返回服务器的名称 |
| int | getServletPort() | 返回服务器的端口号 |
| boolean | isSecure() | 判断请求是否使用安全链接, 例如HTTPS |
| void | removeAttribute(String name) | 移除一个指定的属性 |
| void | setAttribute(String name, Object obj) | 将一个属性存储到请求中, 该方法常与RequestDispatcher()方法协作使用. |
| void | setCharacterEncoding(String env) | 设置请求的字符编码方式. 该方法必须在读取请求参数 或 使用getReader()读取输入之前进行调用, 否则无效. |

## ServletResponse接口

ServletResponse接口定义了一个对象, 该对象用于帮助Servlet给客户端发送响应. Servlet容器创建一个ServletResponse对象, 并将其作为参数传递给Servlet的Service方法.

在响应体中药传递二进制数据, 可以使用有getOutputStream()方法返回的ServletOutputStream对象;要传送字符数据, 可以使用由getWriter()方法返回的PrintWriter对象.

响应体所支持的字符集, 可以使用setCharacterEncoding()方法或setContentType()方法显示指定, 也可以使用setLocale()方法隐式指定.

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
| void | flushBuffer() | 用于强制将缓冲区的内容写到客户端, 该方法用于自动提交响应. |
| int | getBufferSize() | 用于返回响应所使用的缓冲区实际大小, 如果没有使用缓冲, 则该方法返回0. |
| String | getCharacterEncoding() | 用于返回响应的字符编码方式, 如果没有指定字符编码方式, 则返回ISO-8859-1 |
| String | getContentType() | 返回响应内容的MIME类型, 在响应被提交之前, 必须通过setContentType()方法指定内容类型, 如果没有指定内容类型, 则该方法返回null. |
| Locale | getLocale() | 返回响应的区域类型. |
| ServletOutputStream | getOutputStream() | 返回一个适合在响应中写入二进制数据的ServletOutputStream对象; Servlet并不会对二进制数据进行编码, 在ServletOutputStream对象上调用flush()方法将提交响应. |
| PrintWriter | getWriter() | 返回一个能够向客户端发送字符文本的PrintWriter对象. 在PrintWriter对象上调用flush()方法将提交响应. |
| boolean | isCommitted() | 用于判断响应是否已经被提交. 已提交的响应将拥有状态码和头部信息. |
| void | reset() | 用于清空缓冲区数据, 包括状态码和头部信息 |
| void | resetBuffer() | 用于清空缓冲区数据, 不包括状态码 或 头部信息, 如果响应已被提交, 则抛出异常. |
| void | setBufferSize(int size) | 用于为响应设置首选缓冲区大小 |
| void | setCharacterEncoding | 该方法可以被重复调用, 用以改变字符编码方式, 如果在调用getWriter()方法之后 或 响应已被提交之后调用, 则该方法不会产生任何效果. |
| void | setContentLength(int len) | 用于设置响应内容的长度. |
| void | setContentType(String type) | 用于设置响应内容的类型, 该方法可以被重复调用, 用以改变内容类型和字符编码, 如果在响应已被提交之后调用, 则该方法不会产生任何效果. |
| void | setLocale(Locale loc) | 用于设置相应的地域代码. 该方法可以被重复调用, 用以改变地域和字符编码, 如果在响应已被提交之后调用, 则该方法不会产生任何效果. |

## HttpServletRequest接口

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
|  | getAttribute() | 返回指定的属性值 |
|  | getAttrubuteNames() | 获取属性名集合 |
| String | getAuthType() | 返回保护Servlet认证机制名称, 用于保护Servlet的认证机制名称包括BASIC\_AUTH, CLIENT\_CERT\_AUTH, DIGEST\_AUTH, FORM\_AUTH或一个依赖于容器的字符串, 如果Servlet未受保护, 则返回null. |
| String | getContextPath() | 获取上下文路径 |
| Cookie[] | getCookies() | 返回客户端发送的Cookie对象数组. 如果客户端没有发送Cookie, 则该方法将返回null. |
| long | getDateHeader(String name) | 返回指定请求头部的Date类型的值, 返回一个长整形的值, 从GMT 1970年1月1日以来的毫秒数. 如果请求总未包含指定的头部, 则返回-1. |
| String | getHeader(String name) | 返回指定的请求头部的值, 如果请求头部未包含指定的头部, 则返回null. |
| Enumeration | getHeaderNames() | 返回请求中所有头部名称的集合 |
| Enumeration | getHeaders(String name') | 返回指定的请求头部所有的值. |
| int | getInputStream() | 返回请求的输入流. |
| int | getIntHeader(String name) | 返回指定请求头部的整型值. 如果请求中未包含指定的头部, 则返回-1. |
| String | getMethod() | 返回发送请求的方式(GET, POST, PUT等). |
| String | getPathInfo() | 返回请求中与URI相关的所有额外的路径的信息. |
| String | getPathTranslated() | 用于将getPathInfo()方法的结果翻译成响应的文件系统路径. 并予返回. |
| String | getQueryString() | 返回请求URI中包含的查询字符串.  例如URI为: http://202.202.96.53/login/login1.jsp?username=yyy777&password=777yyy 将返回username=yyy777&password=777yyy |
| String | getRemoteUser() | 用于返回发送请求的客户端验证用户名. 如果客户端未经验证, 返回null. |
| String | getRequestedSessionId() | 返回请求的会话标识. |
| String | getRequestURI() | 返回一个字符串, 包含URI中从协议名到查询字符串之间的部分, 不包括查询字符串参数. |
| StringBuffer | getRequestURL() | 用于返回重构的请求URL, 该URL包括协议, 服务器名, 端口号, 和 服务器路径, 但不包括查询字符串参数. |
| String | getServletPath() | 返回请求URL中标识Servlet的路径. 该Servlet路径以"/"开始, 并包括正在调用的Servlet的名称或路径, 如果使用"/\*"模式, 则该方法将返回一个空字符串. |
| HttpSession | getSession(boolean create) | 返回请求关联的当前HttpSession对象.  对于getSession()方法, 如果请求没有会话, 该方法自动创建并返回新会话.  对于getSession(boolean)方法, 如果请求没有会话, 则需判断create值, 如果create=true, 则该方法创建并返回新会话, 否则返回null. |
| Principal | getUserPrincipal() | 返回一个包括当前已认证用户名的Principal对象. 如果用户未被认证, 则该方法返回null. |
| boolean | isRequestedSessionIdFromCookie() | 判断请求会话标识是否作为Cookie传入. |
| boolean | isRequestedSessionIdFromURL() | 判断请求的会话ID是否作为请求URL的一部分传入. |
| boolean | isRequestedSessionIdValid() | 判断会话ID是否仍然有效. 如果客户端没有指定任何会话标识, 则该方法返回false. |
| boolean | isSecure() | 判断请求是否使用安全链接. 例如HTTPS. |
| boolean | isUserInRole(String role) | 判断已认证用户是否属于指定的逻辑角色. |

## HttpServletResponse接口

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
| void | addCookie(Cookie cookie) | 为响应添加指定的Cookie对象. |
| void | addDateHeader(String name, long value) | 用于添加一个带有指定名称和日期值的响应头部.  long created = (new Date()).getTime();  response.addDateHeader("DateCreated", created); |
| void | addHeader(String name, String value) | 添加一个带有指定名称和字符串值的响应头部. |
| void | addIntHeader(String name, int value) | 添加一个带有指定名称和整数值的响应头部. |
| boolean | containsHeader(String name) | 判断指定名称的响应头部是否已经设置. |
| Stirng | encodeRedirectURL(String url) | 用于对sendRedirect()方法使用的URL进行编码. 如果不需要编码, 则返回未改动的URL |
| String | encodeURL(String url) | 用于通过包含会话ID 对指定的URL进行编码. 如果不需要编码, 则返回未改动的URL. |
| void | sendError(int sc, String msg) //指定状态码和描述信息  sendError(int sc)//指定状态码 | 如果响应已被提交, 则该方法将抛出一个IllegalStateException异常. |
| void | sendRedirect(String location) | 用于向客户端发送一个指定位置的临时重定向响应. |
| void | setDateHeader(String name, long value) | 用于设置带有指定名称和日期的响应头部. 如果头部已经设置, 则新值将重写原值. |
| void | setHeader(String name, String value) | 用于设置带有指定名称和字符串值的响应头部. |
| void | setIntHeader(String name, int value) | 用于设置带有指定名称和整数值的响应头部. |
| void | setStatus(int sc) | 用于设置响应的状态码.  该方法通常用于设置没有错误的返回(例如状态码:SC\_OK 或 SC\_MOVED\_TEMPORARILY)  如果有错误发生, 则应当使用sendError方法代替. |

## ServletContext接口

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
| Object | getAttribute(String name) | 返回给定名称的Servlet属性的值, 如果指定的属性不存在则返回null. |
| Enumeration | getAttrubuteNames() | 获取所有属性名 |
| ServletContext | getContext(String uripath) | 根据服务器上指定的URL返回ServletContext对象. |
| String | getInitParameter(String name) | 返回指定上下文初始化参数的值. |
| Enumeration | getInitParameterNames() | 返回上下文所有初始化参数的名称集合. 如果没有初始化参数, 则该方法返回一个空枚举. |
| int | getMajorVersion() | 返回Servlet容器支持的Java Servlet API的主版本号. |
| int | getMinorVersion() | 返回Servlet容器支持的Java Servlet API的副版本号. |
| String | getMimeType(String file) | 返回指定文件的MIME类型, 如果MIME类型未知则返回null. |
| RequestDispatcher | getRequestDispatcher(String name) | 返回一个RequestDispatcher对象, 该对象相当于指定名称的Servlet对象的一个包装器. |
| String | getRealPath(String path) | 返回指定虚拟路径所对应的物理路径. |
| RequestDispatcher | getRequestDispatcher(String path) | 返回一个RequestDispatcher对象, 该对象相当于位于指定上下文相对路径资源的一个包装器. |
| URL | gerResource(String path) | 返回映射到指定上下文相对路径的资源URL, 此方法允许Servlet容器使用任何来源的资源(本地或远程的文件系统, 数据库或者war文件)对Servlet容器可用. |
| InputStream | getResourceAsStream(String path) | 返回映射到指定上下文相对路径资源的InputStream对象. |
| Set | getResourcePaths(String apth) | 返回web应用中所有资源路径的集合, 指定路径, 以"/"开头, 比如"/EWB-INF/" |
| String | getServletInfo() | 返回当前服务器版本信息 |
|  | getServlet() | 原先定义为返回ServletContext中指定名称的Servlet, 从Servlet 2.1 API开始, 不再推荐使用该方法. 而且没有代替的方法. 将来的Servlet API中将永久删除该方法. |
| String | getServletContextName() | 返回应用部署描述文件中<display-name>元素所定义的Servlet上下文名称. |
|  | getServletNames() | 获取所有Servlet名, 从Servlet 2.1 API开始, 不再推荐使用该方法. 而且没有代替的方法. 将来的Servlet API中将永久删除该方法. |
|  | getServlets() | 获取所有Servlet, 从Servlet 2.1 API开始, 不再推荐使用该方法. 而且没有代替的方法. 将来的Servlet API中将永久删除该方法. |
| void | log(Exceptin ex, String msg)或  log(String msg)或  log(String msg, Throwable t) | 用于指定信息写入Servlet日志文件中.  其中log(Exceptin ex, String msg)已被log(String msg, Throwable t)所代替 |
| void | removeAttribute(String name) | 用于从Servlet上下文中删除指定的属性. |
| void | setAttrubute(String name, Object value) | 用于在Servlet上下文总设置指定属性的值.  如果指定的属性已经存在, 则该方法将使用新的值替换原有的属性值.  如果指定的属性值为null, 则效果与removeAttribute()方法相同. |

## 监听器

### ServletContextListener接口

### ServletContextEvent类

### ServletContextAttributeListener接口

### ServletContextAttributeEvent类

### HttpSessionListener接口

### HttpSessionActivationListener接口

### HttpSessionEvent类

### HttpSessionAttributeListener接口

### HttpSessionBindingListener接口

### HttpSessionBindingEvent类

### ServletRequestListener接口

### ServletRequestEvent类

### ServletRequestAttributeListener接口

### ServletRequestAttributeEvent类

### HttpServlet类

HttpServlet类是一个抽象类, 扩展了GenericServlet类.

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
| 构造函数 | HttpServlet() | 该函数什么也不做, 因为该类是一个抽象类 |
| void | init() | 用于初始化Servlet |
| void | Service() | 处理客户端请求. |
| void | destroy() | 用于销毁Servlet |
| void | doGet(HttpServletRequest request, HttpServletResponse response) | 支持HTTP的GET请求, 同时支持HTTP的HEAD请求. HEAD请求是仅请求头部的GET请求. |
| void | doPost(HttpServletRequest request, HttpServletResponse response) | 支持HTTP的POST请求 |
| void | doHead(HttpServletRequest request, HttpServletResponse response) | doHead()方法由service()方法调用,用于处理HTTP的HEAD请求. 该方法不向客户端返回任何数据, 仅仅返回包含内容长度的头部信息. |
| void | doPut(HttpServletRequest request, HttpServletResponse response) | 该方法允许客户端将文件放置在服务器上, 类似于通过FTP发送文件. |
| void | doDelete(HttpServletRequest request, HttpServletResponse response) | doDelete方法是由service方法调用的, 用于处理HTTP的DELETE请求. |
| void | doTrace(HttpServletRequest request, HttpServletResponse response) | doTrace方法由service方法调用, 用于处理HTTP的TRACE请求. |
| void | doOptions(HttpServletRequest request, HttpServletResponse response) | doOptions方法由service方法调用, 用于处理HTTP的TRACE请求. |
| String | getInitParameter(String name) | 返回初始化参数的值 |
| Enumeration | getInitParameterNames() | 获取所有初始化的参数名 |
| long | getLastModified(HttpServletRequest request) | 返回HttpServletRequest对象的最后修改时间(自GMT 1970年1月1日以来的毫秒数). 默认情况下该方法返回-1, 标识最后修改时间未知. |

### ServletRequest接口

该对象提供了客户端请求信息

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
| Object | getAttribute(String name) | 返回指定属性的值 |
| Enumeration | getAttributeNames() | 获取所有属性名 |
| String | getCharacterEncoding() | 获取字符编码方式, 如果请求未指定编码方式, 则返回null. |
| int | getContentLength() | 获取内容长度(字节数), 如果内容长度未知, 则返回-1. |
| String | getContentType() | 获取内容的MIME类型 |
| ServletInputStream | getInputStream() | 获取输入流, 将请求体内容作为二进制数据返回. |
| String | getLocalAddr() | 返回服务器IP地址 |
| Locale | getLocale() | 返回首选的区域对象 |
| Enumeration | getLocales() | 返回所有区域对象的集合. |
| String | getLocalName() | 返回接收请求的服务器名称 |
| int | getLocalPOrt() | 获取服务器端口 |
| String | getParameter(String name) | 返回请求参数的值 |
| Map | getParameterMap() | 获取参数映射(键作参数名----值作参数值), 键的参数类型为String, 值的参数类型是String数组 |
| String[] | getParameterNames() | 返回请求参数中所有的名称集合 |
| String[] | getParameterValues(String name) | 返回指定参数的所有参数值的集合, 如果指定的参数不存在, 则返回null; 如果参数只有一个值, 则数组长度为1 |
| String | getProtocol() | 返回请求所使用协议的名称和版本. |
| BufferedReader | getReader() | 返回包含请求体的BufferReader对象 |
| String | getRealPath(String) | Java Servlet 2.1 API以来, 不再推荐使用, 该方法已被ServletContext.getRealPath(String)方法所代替 |
| String | getRemoteAddr() | 返回发送请求的客户端IP地址 |
| String | getRemoteHost() | 返回发送请求的客户端主机名 |
| int | getRemotePort() | 返回发送请求的客户端端口号 |
| RequestDispatcher | getRequestDispatcher(String path) | 返回RequestDispatcher对象, 该对象作为给定路径上的资源包装器, 可用于转发一个请求到指定资源, 或将指定资源包含在响应中. |
| String | getScheme() | 返回构成请求的机制名称, 该方法返回的机制包括HTTP|FTP|HTTPS等. |
| String | getServerName() | 返回服务器名称 |
| int | getServerPort() | 返回服务器的端口号 |
| boolean | isSecure() | 判断请求是否使用安全链接, 例如HTTPS. |
| void | removeAttribute(String name) | 用于移除一个属性 |
| void | setAttribute(String name, Object obj) | 用于将一个属性存储到请求中 |
| void | setCharacterEncoding(String evn) | 用于设置请求的支付编码方式, 该方法必须在读取请求参数或使用getReader()读取输入之前进行调用, 否则没有效果. |

### ServletResponse接口

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### HttpServletRequest接口

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### HttpServletResponse接口

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### ServletContext接口

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### 实例

#### 继承GenericServlet的类:

|  |
| --- |
| /\*\*  \* Servlet容器只认识以下三个方法  \* init(ServletConfig config)  \* service(ServletRequest arg0, ServletResponse arg1)  \* destroy()  \*/  public class Test extends GenericServlet {  //必须覆盖此方法  @Override  public void service(ServletRequest arg0, ServletResponse arg1) throws ServletException, IOException {  System.*out*.println("service(ServletRequest arg0, ServletResponse arg1)");  }  @Override  public void destroy() {  System.*out*.println("destroy()");  }    //需要覆盖的是这个无参方法, 而不是下面有参的那个.  @Override  public void init() throws ServletException {  super.init();  System.*out*.println("init()");  }  //Servlet自动调用这个方法, 最好不要覆盖此方法  @Override  public void init(ServletConfig config) throws ServletException {  super.init(config);//这个一定要写上, 因为容易忘掉, 所以最好不要覆盖此方法.  System.*out*.println("init(ServletConfig config)");  this.init(); //手动调用无参的那个方法  }  } |

## 过滤器

|  |
| --- |
| web.xml中  <filter> <!—注册-->  <filter-name>filterTest</filter-name>  <filter-class>com.king.FilterTest</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>utf-8</param-value>  </init-param>  </filter>  <filter-mapping><!—映射-->  <filter-name>filterTest</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |
| 继承javax.servlet.Filter  用于所有广义的提交（包括get和post）  public static String filterHtml(String value){  value=value.replaceAll("&", "&amp;");  value=value.replaceAll("<", "&lt;");  value=value.replaceAll(">", "&gt;");  value=value.replaceAll(" ", "&nbsp;");  value=value.replaceAll("'", "&#39;");  value=value.replaceAll("\"", "&quot;");  value=value.replaceAll("\n", "<br />");  return value;  } |

|  |
| --- |
| public class FirstFilter implements javax.servlet.Filter {  //只在Servlet容器加载当前web应用时执行一次  public void init(FilterConfig arg0) throws ServletException {  System.*out*.println("FirstFilter.init(FilterConfig arg0)");  }  public void destroy() {  System.*out*.println("FirstFilter.destroy()");  }  public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain) throws IOException, ServletException {  System.*out*.println("FirstFilter.doFilter(ServletRequest arg0, ServletResponse arg1, FilterChain arg2)");  //例: 用户没有登录, 则跳转到登录页面  if(表达式){  request.getRequestDispatcher("login.jsp").forward(request, response);  }  filterChain.doFilter(request, response);//加上这条语句后, 执行完filter后继续向下执行.  }  } |
| <filter>  <filter-name>FirstFilter</filter-name>  <filter-class>filter.FirstFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>FirstFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

|  |  |
| --- | --- |
| 以下是几个实用的过滤器:   |  | | --- | | 用户认证过滤器  日志和审核过滤器  图像转换过滤器  数据压缩过滤器  加密过滤器  符号过滤器  触发资源访问事件过滤器  XSLT过滤器  MIME类型链过滤器 | |
| 过滤器实现了javax.servletFilter接口.  如果要改变请求(ServletRequest, HttpServletRequest)和响应(ServletResponse, HttpServletResponse)的特性, 则必须继承请求和响应的包装器类(Wrapper), 并重载其方法. |
| 过滤器可以作如下选择  1. 以常规的方式调用资源;  2. 利用修改过的请求信息调用资源;  3. 调用资源, 但在发送响应道客户机前对其进行修改;  4. 阻止该资源调用, 代之以转到其他的资源, 返回一个特定的状态代码或生成替换输出. |

### filter接口

filter可以使用FilterConfig对象以获取过滤器的初始化参数.

还可以引用ServletContext对象, 例如用于装载需要的资源.

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
| void | destroy() | 销毁过滤器 |
| void | doFilter() | 执行过滤操作 |
| void | init(FilterConfig config) | 初始化过滤器 |

### FilterConfig接口

FilterConfig接口定义了一个过滤器配置对象, 在初始化阶段, Servlet容器使用该对象向过滤器传递信息.

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
| String | getFilterName() | 返回配置文件中<filter-name>元素所定义的过滤器名称 |
| String | getInitParameter(String name) | 返回指定的初始化参数值, 如果指定的参数不存在, 则返回null. |
| Enumeration | getInitParameterNmes() | 返回所有初始化参数的名称集合, 如果过滤器没有初始化参数则返回空枚举. |
| ServletContext | getServletContext() | 返回ServletContext的一个引用 |

### FilterChain接口

定义了一个过滤器链对象, 该对象由Servlet容器提供, 用以给定一个过滤请求的符号链接.过滤器使用FilterChain以调用链上的下一个过滤器, 如果是链上的最后一个过滤器, 则调用位于链末的资源.

doFilter()方法用于调用过滤器链上的下一个过滤器.

public void doFilter(ServletRequest request, ServletResponse response)throws IOException, ServletException

### ServletRequestWrapper类

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
|  | ServletRequestWrapper() |  |
|  | getAttribute() |  |
|  | getAttributeNames() |  |
|  | getCharacterEncoding() |  |
|  | getContentLength() |  |
|  | getContentType() |  |
|  | getInputStream() |  |
|  | getLocalAddr() |  |
|  | getLocale() |  |
|  | getLocales() |  |
|  | getLocalName() |  |
|  | getLocalPort() |  |
|  | getParameter() |  |
|  | getParameterMap() |  |
|  | getParameterNames() |  |
|  | getParameterValues() |  |
|  | getProtocol() |  |
|  | getReader() |  |
|  | getRealPath() |  |
|  | getRemoteAddr() |  |
|  | getRemoteHost() |  |
|  | getRemotePort() |  |
|  | getRequest() |  |
|  | getRequestDispatcher() |  |
|  | getScheme() |  |
|  | getServerName() |  |
|  | getServerPort() |  |
|  | isSecure() |  |
|  | removeAttrubute() |  |
|  | setAttribute() |  |
|  | setCharacterEncoding() |  |
|  | setRequest() |  |

### ServletResponseWrapper类

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
| 构造函数 | ServletResponseWrapper() |  |
|  | flushBuffer() |  |
|  | getBufferSize() |  |
|  | getCharacterEncoding() |  |
|  | getContentType() |  |
|  | getLocale() |  |
|  | getOutputStream() |  |
|  | getResponse() |  |
|  | getWriter() |  |
|  | isCommitted() |  |
|  | reset() |  |
|  | resetBuffer() |  |
|  | setBufferSize() |  |
|  | setCharacterEncoding() |  |
|  | setContentLength() |  |
|  | setContentType() |  |
|  | setLocale() |  |
|  | setResponse() |  |

### HttpServletRequestWrapper类

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
| 构造函数 | HttpServletRequestWrapper() |  |
|  | getAuthType() |  |
|  | getContextPath() |  |
|  | getCookies() |  |
|  | getDateHeader() |  |
|  | getHeader() |  |
|  | getHeaderNames() |  |
|  | getHeaders() |  |
|  | getIntHeader() |  |
|  | getMethod() |  |
|  | getPathInfo() |  |
|  | getPathTranslated() |  |
|  | getQueryString() |  |
|  | getRemoteUser() |  |
|  | getRequestedSessionId() |  |
|  | getRequestUTI() |  |
|  | getRequestUTL() |  |
|  | getServletPath() |  |
|  | getSession() |  |
|  | getUserPrincipal() |  |
|  | isRequestedSessionIdFromCookie() |  |
|  | isRequestedSessionIdFromURL() |  |
|  | isRequestedSessionIdValid() |  |
|  | isUserInRole() |  |

### HttpServletResponseWrapper类

|  |  |  |
| --- | --- | --- |
| 返回值 | 方法名 | 方法功能 |
| 构造函数 | HttpServletResponseWrapper |  |
|  | addCookie() |  |
|  | addDateHeader() |  |
|  | addHeader() |  |
|  | addIntHeader() |  |
|  | containsHeader() |  |
|  | encodeRedirectURL() |  |
|  | encodeURL() |  |
|  | sendError() |  |
|  | sendRedirect() |  |
|  | setDateHeader() |  |
|  | setHeader() |  |
|  | setIntHeader() |  |
|  | setStatus() |  |

### 实例

#### 登陆filter

|  |
| --- |
| **package** com.ismp.web.filter;  **import** java.io.IOException;  **import** javax.servlet.Filter;  **import** javax.servlet.FilterChain;  **import** javax.servlet.FilterConfig;  **import** javax.servlet.ServletException;  **import** javax.servlet.ServletRequest;  **import** javax.servlet.ServletResponse;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** javax.servlet.http.HttpSession;  **public** **class** CopartnerAuthFilter **implements** Filter {  **public** **void** destroy() {}  **public** **void** doFilter(ServletRequest request, ServletResponse response, FilterChain chain) **throws** IOException, ServletException{  HttpServletRequest httpRequest = (HttpServletRequest) request;  HttpSession session = httpRequest.getSession();    **boolean** isLoggedIn = session.getAttribute("session\_id") != **null** && session.getId().equals(session.getAttribute("session\_id"));  **boolean** isLoginRequest = httpRequest.getRequestURL().toString().endsWith("session/login.do")  || httpRequest.getRequestURL().toString().endsWith("session/logout.do");    **boolean** isProcessRequest = httpRequest.getRequestURL().toString().contains("/navigation/"); //流程介绍  **boolean** isReferRequest = httpRequest.getRequestURL().toString().contains("/refer/");//咨询建议  **boolean** isKnowledgeRequest = httpRequest.getRequestURL().toString().contains("/knowledge/");//知识库    **if** (!isLoggedIn && !isLoginRequest && !isProcessRequest && !isReferRequest && !isKnowledgeRequest) {  ((HttpServletResponse) response).sendRedirect(httpRequest.getContextPath() + "/session/login.do");  } **else** {  chain.doFilter(request, response);  }  }  **public** **void** init(FilterConfig arg0) **throws** ServletException {}  } |

#### EncodingFilter.java解决中文乱码

|  |
| --- |
| **package** com.ismp.web.filter;  **import** java.io.IOException;  **import** javax.servlet.Filter;  **import** javax.servlet.FilterChain;  **import** javax.servlet.FilterConfig;  **import** javax.servlet.ServletException;  **import** javax.servlet.ServletRequest;  **import** javax.servlet.ServletResponse;  **public** **class** EncodingFilter **implements** Filter {  **protected** String encoding = **null**;  **public** **void** destroy() {  **this**.encoding = **null**;  }  **public** **void** doFilter(ServletRequest request, ServletResponse response, FilterChain chain) **throws** IOException,  ServletException {  **if** (encoding != **null**) {  request.setCharacterEncoding(encoding);  }  //处理提示消息乱码(开始)  String message = ServletRequestUtils.*getStringParameter*(request, "message");  **if** (message!=**null** && !message.equals("")) {  message = URLDecoder.*decode*(message, "UTF-8");  request.setAttribute("message", message);  }  //处理提示消息乱码(结束)  chain.doFilter(request, response);  }  **public** **void** init(FilterConfig filterConfig) **throws** ServletException {  **this**.encoding = filterConfig.getInitParameter("encoding");  }  } |
| **web.xml**  <filter>  <filter-name>EncodingFilter</filter-name>  <filter-class>com.ismp.web.filter.EncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>EncodingFilter</filter-name>  <url-pattern>\*.do</url-pattern>  </filter-mapping> |

# 代码片段摘要

取出所有表单内容

|  |
| --- |
| Enumeration paramNames = request.getParameterNames();//取出所有传来的表单名字, 放在枚举变量里.  while(paramNames.hasMoreElements()){  String paramName = (String)paramNames.nextElement();//取出一个paraName名字  out.print(“<tr><td>" + paramName + “\n<td>");  String[] paramValues = request.getParameterValues(paramName); //取出第一个paraName的名字对应的所有表单值, 放在集合里面  if(paramValues.length == 1){ //如果一个paraName对应的paramValues==1,  String paramValue = paramValues[0]; //取出paramValue  if(paramValue.length() == 0)//如果取出来的paramValue==0,代表此表单名对应的值为空  out.println(“<i>No Value</i>");  else//如果取出来的paramValue==1,代表此表单名对应的值的个数为1  out.println(paramValue);  }else{//如果一个paraName对应的paramValues > 1,  out.println(“<ul>");  for(int i=0; i<paramValues.length; i++){//循环遍历所有paramValue  out.println(“<li>" + paramValues[i]);  }  out.println(“</ul>");  }  } |
| /\*\***(另一种输出风格)**  \* 打印页面提交的所有name和value  \* **@param** request  \*/  @SuppressWarnings("unchecked")  **public** **void** printParentAndValue(HttpServletRequest request){  Enumeration paramNames = request.getParameterNames();//取出所有传来的表单名字, 放在枚举变量里.  **while**(paramNames.hasMoreElements()){  String paramName = (String)paramNames.nextElement();//取出一个paraName名字  System.*out*.println(paramName);  String[] paramValues = request.getParameterValues(paramName); //取出第一个paraName的名字对应的所有表单值, 放在集合里面  **if**(paramValues.length == 1){ //如果一个paraName对应的paramValues==1,  String paramValue = paramValues[0]; //取出paramValue  **if**(paramValue.length() == 0)//如果取出来的paramValue==0,代表此表单名对应的值为空  System.*out*.println("No Value");  **else**//如果取出来的paramValue==1,代表此表单名对应的值的个数为1  System.*out*.println(paramValue);  }**else**{//如果一个paraName对应的paramValues > 1,  **for**(**int** i=0; i<paramValues.length; i++){//循环遍历所有paramValue  System.*out*.print(paramValues[i] + ", ");  }  }  }  System.*out*.println("----------------");  } |

web中"/"的区别:

|  |
| --- |
| 服务器使用的"/"代表WEB-INF下(也就是项目下面)web站点根目录  客户端使用的"/"代表webapps下(也就是项目上层)web应用根目录 |

客户端跳转 和 服务器端跳转:

|  |  |
| --- | --- |
| **客户端跳转**, 重定向, sendRedirect | **服务器端跳转**, 请求转发, forward 等于<jsp:forward page="uri" /> |
| 慢 | 快 |
| 后面的代码继续执行,除非后面写return | 后面的代码不继续执行 |
| 不同的request | 虽然不是同一个对象,但可以取到上一个页面的参数. |
| 可以传参数(只能直接写在url后面) | 可以传参数, request对象传参数. |
| response.sendRedirect(request.getContextPath() + "/success.jsp");  还是  response.sendRedirect(“/success.jsp"); // 根(/)代表web站点的根目录啊? | this.getServletContext().getRequestDispatcher("/haha.jsp").forward(request, response);  或者  request.getRequestDispatcher("/haha.jsp").forward(request, response);  或者  String forwardPage = "/main.jsp";// 根(/)代表当前web应用程序根目录  RequestDispatcher dispatcher = null;  dispatcher = request.getRequestDispatcher(forwardPage);  dispatcher.forward(request, response); |

动态包含 和 静态包含:

|  |
| --- |
| 静态(编译时)包含:<%@ include file="body.jsp"%> //编译只生成一个class文件,不可用rul传参数. |
| 动态(运行时)包含:<jsp:include page="body.jsp" flush="true"/> //编译生成多个class文件, 运行时组合在一起. 可以用url传参数. |

## 过滤多行文本框里的内容

|  |
| --- |
| **Java版**  **public** static String filterHtml(String value){  value = value.replaceAll("&", "&amp;");  value = value.replaceAll("<", "&lt;");  value = value.replaceAll(">", "&gt;");  value = value.replaceAll(" ", "&nbsp;");  value = value.replaceAll("'", "&#39;");  value = value.replaceAll("\"", "&quot;");  value = value.replaceAll("\n", "<\br>");  **return** value;  } |
| **JS版**  /\*\*  \* (已修改好)  \* 过滤HTML文本  \*/  **function** htmlFilter(htmlText){  **var** htmlText;  htmlText = htmlText.replace(/&/g, "&amp;");  htmlText = htmlText.replace(/</g, "&lt;");  htmlText = htmlText.replace(/>/g, "&gt;");  htmlText = htmlText.replace(/ /g, "&nbsp;");  htmlText = htmlText.replace(/'/g, "&#39;");  htmlText = htmlText.replace(/\"/g, "&quot;");  htmlText = htmlText.replace(/\n/gi, "<\br>");  **return** htmlText;  } |

## 获取所有头信息

|  |
| --- |
| **public** **void** printContent(HttpServletRequest request, PrintWriter out){  Enumeration e = request.getHeaderNames();  String name = "";  **while**(e.hasMoreElements()){  headerName = (String)e.nextElement();  out.print(name);  out.println(":");  headerValue = request.getHeader(name));  out.println(headerValue);  out.println("<br>");  }  } |

*cookie的* 创建 保存 读取 删除

|  |  |
| --- | --- |
| 创建cookie  保存cookie | String username = request.getParameter("username");  Cookie cookie1 = new Cookie("username", username); //创建一个cookie  cookie1.setMaxAge(60\*60); //设置cookie有效期, 还有一个cookie1.getMaxAge();  cookie1.setPath("/"); //设置路径  cookie1.setDomain(".v512.com");//设置域名, 域名的值以点开始  response.addCookie(cookie1); //保存cookie 到用户的硬盘上 |
| 读取cookie  删除cookie | Cookie[] cookies = request.getCookies(); //取出客户端传来的所有cookie  Cookie cookie = null;  if(cookies != null){  for(int i=0; i<cookies.length; i++){ //遍历cookies  cookie = cookies[i];  if(cookie.getName().equals("bookName")){  String bookName = cookie.getValue();  }  cookie.setValue("newValue"); //修改sesion的值  cookie.setMaxAge(0); //删除cookie  }  } |

## 实例: 用户登录, 记住登陆状态Cookie

|  |
| --- |
| **登录JSP**  <form action=*"*${path}*/common/LoginServlet?method=login"* method=*"post"*>  <input type=*"text"* name=*"username"* value=*"admin"*><br/>  <input type=*"password"* name=*"password"* value=*"123456"*><br/>  <label><input type=*"checkbox"* name=*"autoLogin"* value=*"1"*>记住我的登录状态</label>  <input type=*"submit"* value=*"提交"* />  </form> |
| **登录Servlet**  **public** **void** login(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  String username = request.getParameter("username");  String password = request.getParameter("password");  String autoLogin = request.getParameter("autoLogin");    request.setAttribute("autoLogin", autoLogin);    **if**(**null**!=username && **null**!=password && username.equals("admin") && password.equals("123456")){//判断用户名密码是否正确    HttpSession session = request.getSession();  session.setAttribute("username", username);  **if**(**null**!=autoLogin && autoLogin.equals("1")){//是否记住登陆  Cookie cookie = **new** Cookie("username", (String)session.getAttribute("username"));  cookie.setMaxAge(20);  cookie.setPath("/");  response.addCookie(cookie);  }    request.getRequestDispatcher("/WEB-INF/main.jsp").forward(request, response);  }**else** {  response.sendRedirect(request.getContextPath() + "/index.jsp");  }  } |
| **注销Servlet**  **public** **void** logOut(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  HttpSession session = request.getSession();  session.setAttribute("username", **null**);    Cookie cookie = **new** Cookie("username", **null**);  cookie.setMaxAge(0);  cookie.setPath("/");  response.addCookie(cookie);    response.sendRedirect(request.getContextPath() + "/index.jsp");  } |
| **检查用户登录Filter**  **public** **void** doFilter(ServletRequest request, ServletResponse response, FilterChain chain) **throws** IOException, ServletException {    String username = **null**;    HttpSession session = ((HttpServletRequest)request).getSession();  username = (String)session.getAttribute("username");    **if**(**null**==username || username.equals("")){//如果session中找不到用户, 就去Cookie中找  username = CookieUtil.*getCookieValue*((HttpServletRequest) request, "username");  **if**(**null**!=username && !username.equals("")){//cookie中找到用户, 则把用户设置如session  session.setAttribute("username", username);  }  }      **if**(**null**!=username && !username.equals("")){  //继续跳转  chain.doFilter(request, response);  }**else** {  request.getRequestDispatcher("/index.jsp").forward(request, response);  }  } |
| **CookieUtil.java** |

Servlet中使用的对象:

|  |  |
| --- | --- |
| 1. out | PrintWriter out = response.getWriter();  JspWriter out = pageContext.getOut(); |
| 1. request | HttpServletRequest request = ServletActionContext.getRequest();  HttpServletRequest request = (HttpServletRequest)pageContext.getRequest(); |
| 1. response | HttpServletResponse response = ServletActionContext.getResponse(); |
| 1. session | HttpSession session = request.getSession();  HttpSession session = request.getSession(true); |
| 1. application   (Servlet上下文) | ServletContext application = this.getServletContext();  ServletContext application = ServletActionContext.getServletContext(); |
| 1. config   (web.xml中的映射/配置) | ServletConfig servletConfig = this.getServletConfig(); |
| 1. page(this) | Object page = this;(极少使用) |
| 1. pageContext | PageContext pageContext = this.getJspContext(); |
| 1. exception |  |

web工程对象:ServletContext

|  |
| --- |
| ServletContext servletContext = this.getServletContext();  String webFilePath = servletContext.getRealPath("/WEB-INF/web.xml"); |

访问出错代码:

|  |  |
| --- | --- |
| 200 | 正常访问 |
| 403 | 没有访问权限 |
| 404 | 客户端请求的网页不存在 |
| 400 | URL 地址请求错误 |
| 500 | 服务器内部错误，或是程序有问题。 |

# JavaBeans

任何java类, 如果实现了java.io.Serializable接口并且提供默认构造方法(没有参数的构造方法), 都可以称为JavaBean.

# 使用FileUpload控件上传图片并自动生成缩略图、自动生成带文字和图片的水印图

<http://www.kesion.com/zzcd/aspnet/jq/3047.html>

# 文件上传

|  |
| --- |
| **开始JSP上传文件之前你要准备以下几个东西:**  **1. commons-FileUpload 1.2 包**  **下载地址:http://jakarta.apache.org/commons/fileupload/**  **或者**[**http://labs.renren.com/apache-mirror/commons/fileupload/binaries/commons-fileupload-1.2.1-bin.zip**](http://labs.renren.com/apache-mirror/commons/fileupload/binaries/commons-fileupload-1.2.1-bin.zip)  **2. commons-IO 1.3.1 包**  **下载地址:http://jakarta.apache.org/commons/io/**  **或者**[**http://apache.freelamp.com/commons/io/binaries/commons-io-1.4-bin.zip**](http://apache.freelamp.com/commons/io/binaries/commons-io-1.4-bin.zip)  **3. Commons-BeanUtils 1.7 包**  **下载地址:http://jakarta.apache.org/commons/beanutils/**  **或者**[**http://apache.freelamp.com/commons/beanutils/binaries/commons-beanutils-1.8.3-bin.zip**](http://apache.freelamp.com/commons/beanutils/binaries/commons-beanutils-1.8.3-bin.zip)  **把jar文件解压出来, 放入lib里面**    **有了上面这些东西我们就可以开始了**  **1. 新建一个叫upload的WEB项目(我用的是Lomboz3.2开发环境)**  **2. 把上面下载下来的包分别解压并拷贝\*.jar的文件到上面那个项目的WEB-INF/lib目录中**  **3.接下来我们要准备一份如下内容的upload.jsp文件,用来选择要上传的文件,** |
| **1. JSP页面**  <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%  String path = request.getContextPath();  String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";  %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <base href=*"*<%=basePath%>*"*>    <title>My JSP 'index.jsp' starting page</title>  <meta http-equiv=*"pragma"* content=*"no-cache"*>  <meta http-equiv=*"cache-control"* content=*"no-cache"*>  <meta http-equiv=*"expires"* content=*"0"*>  <meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>  <meta http-equiv=*"description"* content=*"This is my page"*>  <!--  <link rel="stylesheet" type="text/css" href="styles.css">  -->  </head>    <body>  <%=path %><br/>  <%=basePath %>  <form name=*"upform"* action=*"*<%=path%>*/servlet/UploadServlet"* method=*"POST"* enctype=*"multipart/form-data"*>  <input type =*"file"* name=*"file1"* id=*"file1"*/><br/>  <input type =*"file"* name=*"file2"* id=*"file2"*/><br/>  <input type =*"file"* name=*"file3"* id=*"file3"*/><br/>  <input type=*"submit"* value=*"Submit"* /><br/>  <input type=*"reset"* />  </form>  </body>  </html> |
| **Servlet**  **package** servlet;  **import** java.io.BufferedInputStream;  **import** java.io.BufferedOutputStream;  **import** java.io.File;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.apache.commons.fileupload.FileItemIterator;  **import** org.apache.commons.fileupload.FileItemStream;  **import** org.apache.commons.fileupload.disk.DiskFileItemFactory;  **import** org.apache.commons.fileupload.servlet.ServletFileUpload;  **import** org.apache.commons.fileupload.util.Streams;      **public** **class** UploadServlet **extends** javax.servlet.http.HttpServlet **implements** javax.servlet.Servlet {  File tmpDir = **null**;//初始化上传文件的临时存放目录  File saveDir = **null**;//初始化上传文件后的保存目录    **public** UploadServlet() {  **super**();  }    **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doPost(request,response);  }    **protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  /\* 对上传文件夹和临时文件夹进行初始化  \*/  **super**.init();  String tmpPath = request.getContextPath()+"/tmpdir/";    String savePath = **null**;  **try** {  savePath = getWebRoot()+"/tmpdir/";  System.*out*.println(savePath);  } **catch** (IllegalAccessException e2) {  e2.printStackTrace();  }  tmpDir = **new** File(tmpPath);  saveDir = **new** File(savePath);  **if**(!tmpDir.isDirectory())  tmpDir.mkdir();  **if**(!saveDir.isDirectory())  saveDir.mkdir();          **try**{  request.getContextPath();  **if**(ServletFileUpload.*isMultipartContent*(request)){  DiskFileItemFactory dff = **new** DiskFileItemFactory();//创建该对象  dff.setRepository(tmpDir);//指定上传文件的临时目录  dff.setSizeThreshold(1024000);//指定在内存中缓存数据大小,单位为byte  ServletFileUpload sfu = **new** ServletFileUpload(dff);//创建该对象  sfu.setFileSizeMax(5000000);//指定单个上传文件的最大尺寸  sfu.setSizeMax(10000000);//指定一次上传多个文件的总尺寸  FileItemIterator fii = sfu.getItemIterator(request);//解析request 请求,并返回FileItemIterator集合  **while**(fii.hasNext()){  FileItemStream fis = fii.next();//从集合中获得一个文件流  **if**(!fis.isFormField() && fis.getName().length()>0){//过滤掉表单中非文件域  String fileName = fis.getName().substring(fis.getName().lastIndexOf("\\")+1);//获得上传文件的文件名  System.*out*.println(fileName);  BufferedInputStream in = **new** BufferedInputStream(fis.openStream());//获得文件输入流  System.*out*.println(saveDir+fileName);  // BufferedOutputStream out = new BufferedOutputStream(new FileOutputStream(new File(saveDir+Long.toString(new Date().getTime()))));//获得文件输出流  BufferedOutputStream out = **new** BufferedOutputStream(**new** FileOutputStream(**new** File(saveDir+"/"+fileName)));//获得文件输出流  Streams.*copy*(in, out, **true**);//开始把文件写到你指定的上传文件夹    response.getWriter().println("<img src='http://"+request.getLocalAddr()+":"+request.getLocalPort()+tmpPath+fileName+"' />");//终于成功了,还不到你的上传文件中看看,你要的东西都到齐了吗  }  }  }  }**catch**(Exception e){  e.printStackTrace();  }  }          **public** String getWebClassesPath() {  String path = getClass().getProtectionDomain().getCodeSource()  .getLocation().getPath();  **return** path;    }  **public** String getWebInfPath() **throws** IllegalAccessException{  String path = getWebClassesPath();  **if** (path.indexOf("WEB-INF") > 0) {  path = path.substring(0, path.indexOf("WEB-INF")+8);  } **else** {  **throw** **new** IllegalAccessException("路径获取错误");  }  **return** path;  }  **public** String getWebRoot() **throws** IllegalAccessException{  String path = getWebClassesPath();  **if** (path.indexOf("WEB-INF") > 0) {  path = path.substring(0, path.indexOf("WEB-INF/classes"));  } **else** {  **throw** **new** IllegalAccessException("路径获取错误");  }  **return** path;  }  } |
| **web.xml**  <servlet>  <servlet-name>UploadServlet</servlet-name>  <servlet-class>servlet.UploadServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>UploadServlet</servlet-name>  <url-pattern>/servlet/UploadServlet</url-pattern>  </servlet-mapping> |

# 登陆图片验证码

## 1. JSP页面

|  |
| --- |
| <img id=*"validateImg"* style=*"cursor:hand;"* name=*"validates"* border=*"0"* height=*"23"* width=*"80"*  src=*"../validateCodeServlet"*  onclick="**this**.src='../validateCodeServlet?id=' + **new** Date().getTime()"  /> |

## 2. web.xml

|  |
| --- |
| <!-- 验证码-->  <servlet>  <servlet-name>ValidateCodeServlet</servlet-name>  <servlet-class>  com.ismp.web.servlet.ValidateCodeServlet  </servlet-class>  <init-param>  <param-name>width</param-name>  <param-value>75</param-value>  </init-param>  <init-param>  <param-name>height</param-name>  <param-value>20</param-value>  </init-param>  <init-param>  <param-name>codeCount</param-name>  <param-value>4</param-value>  </init-param>  <load-on-startup>3</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>ValidateCodeServlet</servlet-name>  <url-pattern>/validateCodeServlet</url-pattern>  </servlet-mapping> |

## 3. 生成图片的servlet

|  |
| --- |
| **package** com.ismp.web.servlet;  **import** java.awt.Color;  **import** java.awt.Font;  **import** java.awt.Graphics2D;  **import** java.awt.image.BufferedImage;  **import** java.io.IOException;  **import** java.util.Random;  **import** javax.imageio.ImageIO;  **import** javax.servlet.ServletException;  **import** javax.servlet.ServletOutputStream;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** javax.servlet.http.HttpSession;  /\*\*  \* **@author** mpc  \* **@version** 3.0  \* <p>  \* 登陆验证码  \* </p>  \*  \*/  **public** **class** ValidateCodeServlet **extends** HttpServlet {  **private** **static** **final** **long** *serialVersionUID* = 1L;  /\*\* 验证码图片的宽度。 \*/  **private** **int** width = 60;  /\*\* 验证码图片的高度。 \*/  **private** **int** height = 20;  /\*\* 验证码字符个数 \*/  **private** **int** codeCount = 4;  **private** **int** x = 0;  /\*\* 字体高度 \*/  **private** **int** fontHeight;  **private** **int** codeY;  **char**[] codeSequence = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'J', 'K', 'M', 'N', 'P', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '2', '3', '4', '5', '6', '7', '8', '9' };  /\*\*  \* 初始化验证图片属性  \*/  **public** **void** init() **throws** ServletException {  // 从web.XML中获取初始信息  // 宽度  String strWidth = **this**.getInitParameter("width");  // 高度  String strHeight = **this**.getInitParameter("height");  // 字符个数  String strCodeCount = **this**.getInitParameter("codeCount");  // 将配置的信息转换成数值  **try** {  **if** (strWidth != **null** && strWidth.length() != 0) {  width = Integer.*parseInt*(strWidth);  }  **if** (strHeight != **null** && strHeight.length() != 0) {  height = Integer.*parseInt*(strHeight);  }  **if** (strCodeCount != **null** && strCodeCount.length() != 0) {  codeCount = Integer.*parseInt*(strCodeCount);  }  } **catch** (NumberFormatException e) {  }  x = width / (codeCount + 1);  fontHeight = height - 2;  codeY = height - 4;  }  **public** **void** doget(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException {  **try** {  service(req, resp);  } **catch** (IOException e) {  e.printStackTrace();  }  }  **protected** **void** service(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, java.io.IOException {  // 定义图像buffer  BufferedImage buffImg = **new** BufferedImage(width, height, BufferedImage.*TYPE\_INT\_RGB*);  Graphics2D g = buffImg.createGraphics();  // 创建一个随机数生成器类  Random random = **new** Random();  // 将图像填充为白色  // g.setColor(Color.WHITE);  // 灰色  g.setColor(Color.*lightGray*);  g.fillRect(0, 0, width, height);  // 创建字体，字体的大小应该根据图片的高度来定。  Font font = **new** Font("Fixedsys", Font.*BOLD*, fontHeight);  // 设置字体。  g.setFont(font);  // 画边框。  // g.setColor(Color.BLACK);  g.drawRect(0, 0, width - 1, height - 1);  // 随机产生160条干扰线，使图象中的认证码不易被其它程序探测到。  g.setColor(Color.*BLACK*);  **for** (**int** i = 0; i < 10; i++) {  **int** x = random.nextInt(width);  **int** y = random.nextInt(height);  **int** xl = random.nextInt(12);  **int** yl = random.nextInt(12);  g.drawLine(x, y, x + xl, y + yl);  }  // randomCode用于保存随机产生的验证码，以便用户登录后进行验证。  StringBuffer randomCode = **new** StringBuffer();  **int** red = 0, green = 0, blue = 0;  // 随机产生codeCount数字的验证码。  **for** (**int** i = 0; i < codeCount; i++) {  // 得到随机产生的验证码数字。  String strRand = String.*valueOf*(codeSequence[random.nextInt(30)]);  // 产生随机的颜色分量来构造颜色值，这样输出的每位数字的颜色值都将不同。  red = random.nextInt(255);  green = random.nextInt(150);  blue = random.nextInt(255);  // 用随机产生的颜色将验证码绘制到图像中。  g.setColor(**new** Color(red, green, blue));  g.drawString(strRand, (i + 1) \* x, codeY);  // 将产生的四个随机数组合在一起。  randomCode.append(strRand);  }  // 将四位数字的验证码保存到Session中。  HttpSession session = req.getSession();  session.setAttribute("validateCode", randomCode.toString());  // 禁止图像缓存。  resp.setHeader("Pragma", "no-cache");  resp.setHeader("Cache-Control", "no-cache");  resp.setDateHeader("EXPires", 0);  resp.setContentType("image/jpeg");  // 将图像输出到Servlet输出流中。  ServletOutputStream sos = resp.getOutputStream();  ImageIO.*write*(buffImg, "jpeg", sos);  sos.close();  }  } |

## 4. session登陆校验

|  |
| --- |
| //验证码  Object validateCodeObj = session.getAttribute("validateCode");  String validateCode= **null**;  **if** (validateCodeObj != **null**) {  validateCode = validateCodeObj.toString();  }  String validation = ServletRequestUtils.*getStringParameter*(request, "validation");  **if** (validateCode != **null** && validation != **null**) {  **if** (!validateCode.equalsIgnoreCase(validation.trim())) {  **return** **new** ModelAndView("/login", "notice", "验证码错误！");  }  } |

# JDBC

## connection对象

## Statement对象

## PreparedStatement对象

## CallableStatement对象

## ResultSet对象

## 分页

### 一次查出来再分(记录少)

|  |
| --- |
|  |

### 每次查10页(记录多)

### 查一页显示一页 (记录多)

|  |  |
| --- | --- |
| MySQL | select \* from book limit 起始条数, 需要查几条; |
| Oracle | 使用子查询 |

|  |
| --- |
| /\*\*  **\* 保存分页查询条件到request中**  \* **@param** request  \*/  @SuppressWarnings("unchecked")  **public** **void** handlePageConditions(HttpServletRequest request){  String requestURL = request.getRequestURL().toString();  //设置网页地址  String pageURL = requestURL.substring(requestURL.lastIndexOf("/")+1);//基础路径  request.setAttribute("pageURL", pageURL);  //设置page  Object pageObj = request.getParameter("page");  **if** (pageObj == **null**) {  pageObj = "1";  }  request.setAttribute("page", pageObj.toString());//当前页  //设置size  Object sizeObj = request.getParameter("size");  **if** (sizeObj == **null**) {  sizeObj = "10";  }  request.setAttribute("size", sizeObj.toString());//当前页    StringBuffer sb = **new** StringBuffer();  Enumeration paramNames = request.getParameterNames();//取出所有传来的表单名字, 放在枚举变量里.  **while**(paramNames.hasMoreElements()){  String paramName = (String)paramNames.nextElement();//取出一个paraName名字  **if** (paramName.equals("page") || paramName.equals("size")) {  **continue**;  }  sb.append("&");  sb.append(paramName);  sb.append("=");  String[] paramValues = request.getParameterValues(paramName); //取出第一个paraName的名字对应的所有表单值, 放在集合里面  **if**(paramValues.length == 1){ //如果一个paraName对应的paramValues==1,  String paramValue = paramValues[0]; //取出paramValue  sb.append(paramValue);  }  // else{//如果一个paraName对应的paramValues > 1,  // for(int i=0; i<paramValues.length; i++){//循环遍历所有paramValue  // System.out.print(paramValues[i] + ", ");  // }  // }  }    request.setAttribute("conditions", sb.toString());  } |

## Tomcat连接池 + JNDI

|  |
| --- |
| Tomcat连接池 + JNDI  --------------context.xml------------------------  <Context reloadable="true">  <WatchedResource>WEB-INF/web.xml</WatchedResource>  <Resource name='jdbc/mysql\_ds" //连接池名: 任意. JNDI通过这个名字找到一个type类型的对象, 通过这个对象可获取链接, 进而使用连接池.  auth="Container" //验证方式  type="javax.sql.DataSource" //数据源类型(好像必须是这个), 全类名  maxActive="100" maxldle="30" maxWait="10000"  username="root" password="root"  driverClassName=" com.mysql.jdbc.Driver"  url="jdbc:mysql://localhost/library" />  </Context> |
| package dao;  import java.sql.Connection;  import java.sql.SQLException;  import java.util.List;  import java.util.Map;  import javax.naming.Context;  import javax.naming.InitialContext;  import javax.naming.NamingException;  import javax.sql.DataSource;  import org.apache.commons.dbutils.QueryRunner;  import org.apache.commons.dbutils.handlers.MapListHandler;  public class Tomcat\_JNDI\_2 {  public void query() {    String sql = "insert intu s\_tudent(name, age, add, bir) values(?, ?, ?, ?)";  String param[] = {"jin", "23", "宜宾", "1985-08-06"};//sql参数都放在一个数组里面.  try{  Context context = new InitialContext();//JNDI里面的方法,  DataSource ds = (DataSource)context.lookup("java:/comp/env/jdbc/MySQL");//JNDI里的context通过lookup方法获得一个数据源对象    //1.可以获取连接conn, 但不推介  Connection conn = ds.getConnection(); //数据源对象调用getConnection方法获得一个数据库连接    //2.不用获取数据库链接了, 使用JNDI的QueryRunner对象qr 添加/删除/修改/查询 数据.  QueryRunner qr = new QueryRunner(ds);            //添加  int result = qr.update(sql, param);//传入sql语句和参数数组, 自动关闭数据库连接, 都看不见conn对象    //查询(一): 使用:MapListHandler()把每条记录转换成一个map对象,map对象存入List类型的results中  List results = (List)qr.query(sql, new MapListHandler());    //遍历results(里面存的map对象)  Map map = null;  for (int i = 0; i < results.size(); i++) {  map = (Map) results.get(i);  System.*out*.print("id:" + map.get("id") + "\t");  System.*out*.print("name:" + map.get("name") + "\t");  System.*out*.print("sex:" + map.get("sex") + "\t");  System.*out*.print("bir:" + map.get("bir") + "\t");  System.*out*.println("add:" + map.get("add"));  }    // //查询(二): 使用BeanListHandler(JavaBeanName.class)把每条记录转换成一个JavaBean对象, JavaBean对象存入List类型的results中; JavaBean的字段名要和数据库字段名相同  // List results = (List)qr.query(sql, new BeanListHandler(Student.class));  //  // //遍历results(里面存的JavaBean对象)  // Student student = null;  // for (int i = 0; i < results.size(); i++) {  // student = (student) results.get(i);    // System.out.print("id:" + student.getId() + "\t");  // System.out.print("name:" + map.getName() + "\t");  // System.out.print("sex:" + map.getSex() + "\t");  // System.out.print("bir:" + map.getBir() + "\t");  // System.out.println("add:" + map.getAdd());  // }  }catch(NamingException e){  e.printStackTrace();  }catch(SQLException e){  e.printStackTrace();  }  }  } |

# 附录1 服务器状态码

|  |
| --- |
| 1xx-信息提示 这些状态代码表示临时的响应。客户端在收到常规响应之前，应准备接收一个或多个1xx响应。  100-继续。  101-切换协议。  2xx-成功 这类状态代码表明服务器成功地接受了客户端请求。  200-确定。客户端请求已成功。  201-已创建。  202-已接受。  203-非权威性信息。  204-无内容。  205-重置内容。  206-部分内容。  3xx-重定向 客户端浏览器必须采取更多操作来实现请求。例如，浏览器可能不得不请求服务器上的不同的页面，或通过代理服务器重复该请求。  301-对象已永久移走，即永久重定向。  302-对象已临时移动。  304-未修改。  307-临时重定向。  4xx-客户端错误 发生错误，客户端似乎有问题。例如，客户端请求不存在的页面，客户端未提供有效的身份验证信息。400-错误的请求。  401-访问被拒绝。IIS定义了许多不同的401错误，它们指明更为具体的错误原因。这些具体的错误代码在浏览器中显示，但不在IIS日志中显示：  401.1-登录失败。  401.2-服务器配置导致登录失败。  401.3-由于ACL对资源的限制而未获得授权。  401.4-筛选器授权失败。  401.5-ISAPI/CGI应用程序授权失败。  401.7–访问被Web服务器上的URL授权策略拒绝。这个错误代码为IIS6.0所专用。  403-禁止访问：IIS定义了许多不同的403错误，它们指明更为具体的错误原因：  403.1-执行访问被禁止。  403.2-读访问被禁止。  403.3-写访问被禁止。  403.4-要求SSL。  403.5-要求SSL128。  403.6-IP地址被拒绝。  403.7-要求客户端证书。  403.8-站点访问被拒绝。  403.9-用户数过多。  403.10-配置无效。  403.11-密码更改。  403.12-拒绝访问映射表。  403.13-客户端证书被吊销。  403.14-拒绝目录列表。  403.15-超出客户端访问许可。  403.16-客户端证书不受信任或无效。  403.17-客户端证书已过期或尚未生效。  403.18-在当前的应用程序池中不能执行所请求的URL。这个错误代码为IIS6.0所专用。  403.19-不能为这个应用程序池中的客户端执行CGI。这个错误代码为IIS6.0所专用。  403.20-Passport登录失败。这个错误代码为IIS6.0所专用。  404-未找到。  404.0-（无）–没有找到文件或目录。  404.1-无法在所请求的端口上访问Web站点。  404.2-Web服务扩展锁定策略阻止本请求。  404.3-MIME映射策略阻止本请求。  405-用来访问本页面的HTTP谓词不被允许（方法不被允许）  406-客户端浏览器不接受所请求页面的MIME类型。  407-要求进行代理身份验证。  412-前提条件失败。  413–请求实体太大。  414-请求URI太长。  415–不支持的媒体类型。  416–所请求的范围无法满足。  417–执行失败。  423–锁定的错误。  5xx-服务器错误 服务器由于遇到错误而不能完成该请求。  500-内部服务器错误。  500.12-应用程序正忙于在Web服务器上重新启动。  500.13-Web服务器太忙。  500.15-不允许直接请求Global.asa。  500.16–UNC授权凭据不正确。这个错误代码为IIS6.0所专用。  500.18–URL授权存储不能打开。这个错误代码为IIS6.0所专用。  500.100-内部ASP错误。  501-页眉值指定了未实现的配置。  502-Web服务器用作网关或代理服务器时收到了无效响应。  502.1-CGI应用程序超时。  502.2-CGI应用程序出错。application.  503-服务不可用。这个错误代码为IIS6.0所专用。  504-网关超时。  505-HTTP版本不受支持。 |

# 嵌入flash视频

|  |
| --- |
| <iframe src="${article.video}" width="970" height="545" frameborder="0" scrolling="no" style="width: 100px; height: 80px;"></iframe> |

# displayTable相关

|  |
| --- |
| **displaytag\_zh.properties(unicode转码后的)**  paging.banner.placement=bottom  paging.banner.item\_name=\u6761\u8BB0\u5F55  paging.banner.items\_name=\u6761\u8BB0\u5F55  basic.msg.empty\_list=\u6CA1\u6709\u76F8\u5173\u4FE1\u606F\!  paging.banner.onepage=<span class\="pagelinks">\u7B2C **{0}** \u9875</span></span>  paging.banner.first=<span class\="pagelinks">\u3010\u9996\u9875/\u4E0A\u4E00\u9875\u3011 **{0}**\u3010<a href\="**{3}**">\u4E0B\u4E00\u9875</a>/<a href\="**{4}**">\u5C3E\u9875</a>\u3011\u8DF3\u8F6C\u5230\u7B2C<input name\="tz" id\="tz" size\="1" type\="text" style\="width\: 23px" />\u9875<input type\="button" onclick\="firstPage\_Go();" style\="width\:35px;" value\="\u8DF3\u8F6C"/><input id\="hd" name\="hd" type\="hidden" value\="**{1}**"/></span>  paging.banner.last=<span class\="pagelinks">\u3010<a href\="**{1}**">\u9996\u9875</a>/<a href\="**{2}**">\u4E0A\u4E00\u9875</a>\u3011 **{0}** \u3010\u4E0B\u4E00\u9875/\u5C3E\u9875\u3011\u8DF3\u8F6C\u5230\u7B2C<input name\="tz" id\="tz" size\="1" type\="text" style\="width\: 23px" />\u9875<input type\="button" onclick\="OtherPage\_Go();" style\="width\:35px;" value\="\u8DF3\u8F6C"/><input id\="hd" name\="hd" type\="hidden" value\="**{1}**"/></span>  paging.banner.no\_items\_found=<span class\="pagebanner">\u51710\u6761\u8BB0\u5F55 \u5F53\u524D\u4E3A\uFF1A</span>  paging.banner.one\_item\_found=<span class\="pagebanner">\u51711\u6761\u8BB0\u5F55 \u5F53\u524D\u4E3A\uFF1A</span>  paging.banner.all\_items\_found=<span class\="pagebanner">\u5171\uFF1A**{0}** **{1}** \u5F53\u524D\u4E3A\uFF1A</span>  paging.banner.page.selected=<strong>**{0}**</strong>  paging.banner.page.link=<a href\="**{1}**" title\="\u8F6C\u81F3\u7B2C **{0}**\u9875">**{0}**</a>  paging.banner.full=<span class\="pagelinks">\u3010<a href\="**{1}**">\u9996\u9875</a>/<a href\="**{2}**">\u4E0A\u4E00\u9875</a>\u3011 **{0}** \u3010<a href\="**{3}**">\u4E0B\u4E00\u9875</a>/<a href\="**{4}**">\u5C3E\u9875</a>\u3011\u8DF3\u8F6C\u5230\u7B2C<input name\="tz" id\="tz" size\="1" type\="text" style\="width\: 23px" />\u9875<input type\="button" onclick\="OtherPage\_Go();" style\="width\:35px;" value\="\u8DF3\u8F6C"/><input id\="hd" name\="hd" type\="hidden" value\="**{1}**"/></span>  paging.banner.some\_items\_found=<span class\="pagebanner"> \u5171\u8BA1 **{0}** **{1}**, \u663E\u793A **{2}** \u5230 **{3}** \u6761 </span> |
| 跳转到\_\_页(JS代码)  <script type=*"text/javascript"*>  **function** fucCheck(INDEX) {  **var** i, j, strTemp;  strTemp = "0123456789";  **for** (i = 0; i < INDEX.length; i++) {  j = strTemp.indexOf(INDEX.charAt(i));  **if** (j == -1) {  //说明有字符不合法  **return** **false**;  }  }  //说明合法  **return** **true**;  }  /\*\*当前数据的总页数，从Action中传值过来\*/  **var** total = ${total}+14/15; /\*\*这个total是servlet.setAttribute的变量名, 用来存储记录总条数, 所以本JS只能是内部样式, 不能是外部样式\*/  **function** firstPage\_Go() {  **var** page = document.getElementById("tz").value;  **if** (!fucCheck(page)) {  alert("输入数字非法，请重新输入！");  **return** **false**;  }  **if** (page > total) {  alert("超出了最大页数，请重新输入！" + total);  } **else** **if** (page < 1) {  alert("页数不能小于1！");  } **else** {  window.location = document.getElementById("hd").value.replace("p=", "p=" + document.getElementById("tz").value);  }  }  /\*\*\*/  **function** OtherPage\_Go() {  **var** page = document.getElementById("tz").value;  **if** (!fucCheck(page)) {  alert("输入数字非法，请重新输入！");  **return** **false**;  }  **if** (page > total) {  alert("超出了最大页数，请重新输入！");  } **else** **if** (page < 1) {  alert("页数不能小于1！");  } **else** {  window.location = document.getElementById("hd").value.replace("p=1", "p=" + document.getElementById("tz").value);  }  }  /\*\*回车事件\*/  **function** preparaSubmit(evt) {  **var** nKeyCode = window.event ? event.keyCode: evt.which;  **if** (nKeyCode == 13) {  go2();  }  }  /\*\*给页码输入框, 绑定回车事件\*/  $(**function**(){  $("#tz").keyup(**function**(event){  **var** myEvent = event || window.event; //获取当前用户按下的键值, 解决不同浏览器获取事件对象的差异  **var** kcode = myEvent.keyCode;  **if** (kcode == 13) {  **if**(document.getElementById("hd").value.search("p=1") == -1){  firstPage\_Go();  }**else**{  OtherPage\_Go();  }  }  });  })  </script> |

servlet中用到的各种地址

|  |
| --- |
| request.getContextPath() = /hboys  request.getRequestURI() = /hboys/  request.getLocalAddr() = 127.0.0.1  request.getRemoteAddr() = 127.0.0.1  request.getServerName() = 127.0.0.1  request.getRequestURL() = http://127.0.0.1:8081/hboys/  request.getPathInfo() = null  request.getPathTranslated() = null  request.getScheme() = http  request.getServletPath() = /index.jsp  request.getClass() = class org.apache.catalina.connector.RequestFacade  request.getHeaderNames() = org.apache.tomcat.util.http.NamesEnumerator@1ad0839  request.getLocale() = zh\_CN  request.getLocales() = org.apache.catalina.util.Enumerator@f6f1b6  request.getParameterMap() = {}  request.getUserPrincipal() = null  request.getParameterNames() = java.util.Hashtable$EmptyEnumerator@1629e71  request.getRealPath("newsPub") = D:\workspace\hboys\WebRoot\newsPub |

# 防盗链

## 过滤器实现防盗链

|  |
| --- |
| **web.xml**  <filter>  <filter-name>imageRedirectFilter</filter-name>  <filter-class>filter.ImageRedirectFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>imageRedirectFilter</filter-name>  <url-pattern>/img/\*</url-pattern>  </filter-mapping> |
| **filter**  **public** **void** doFilter(ServletRequest req, ServletResponse res, FilterChain chain) **throws** IOException, ServletException {  HttpServletRequest request = (HttpServletRequest) req;  HttpServletResponse response = (HttpServletResponse) res;  // 禁止缓存  response.setHeader("Cache-Control", "no-store");  response.setHeader("Pragrma", "no-cache");  response.setDateHeader("Expires", 0);  // 链接来源地址  String referer = request.getHeader("referer");  System.*out*.println("链接来源地址:" + referer);  //提取［域名］和 ［web应用名称］  String applicationName = "";  **if** (referer!=**null**) {  String[] url = referer.split("/");  **if** (url.length >= 4) {  applicationName = "/" + url[3];  // System.out.println("上一个页面, 域名或IP地址:" + url[2]);  // System.out.println("上一个页面, WEB 应用名称:" + url[3]);;  // System.out.println("--------------------------");  }  }    request.getContextPath();      **if** (referer == **null** || applicationName.equals(request.getContextPath())) {  chain.doFilter(request, response);  }**else** {  //如果 链接地址来自其他网站，则返回错误图片  request.getRequestDispatcher("/img/error.gif").forward(request, response);    }    } |

## JS实现页面防盗链

|  |
| --- |
| <html>  <head>  <title>页面防盗链</title>  <script language=*"javascript"* type=*"text/javascript"*>  <!--    **var** currentURL = document.URL; //当前文档的URL  **var** frontURL = document.referrer; //上一个文档的URL  //alert("链接来源地址:" + frontURL);    //如果上一个文档的URL为空，则是直接打开当前文档，则不存在盗链的问题。否则有可能是盗链。  **if** (frontURL != "") {    **var** currentURLs = currentURL.split("/");//通过分割，将当前文档的URL各部分存放在currentURLs数组中。    **var** frontURLs = frontURL.split("/");//通过分割，将上一个文档的URL各部分存放在frontURLs数组中。    //两个数组的第3个元素都为URL的域名部分  //比较两个数组的第3个元素，如果域名相同，则不是盗链，否则就是盗链  **if**(currentURLs.length>=4 && frontURLs.length>=4){  // alert(currentURLs[3] + "<==>" + frontURLs[3]);  **if** (currentURLs[3] == frontURLs[3]) {  document.write("链接来源地址:" + frontURL + "<br />");  document.write("网页不是盗链<br />");  } **else** {  document.write("链接来源地址:" + frontURL + "<br />");  document.write("网页是盗链<br />");  //可以使用以下代码跳转到网站的首页  //history.location = "http://" + currentURLs[2];  }  }  } **else** {  document.write("链接来源地址:" + frontURL + "<br />");  document.write("您是直接打开该文档的，不存在盗链问题<br />");  }  -->  </script>  </head>  <body>  <img src=*"http://127.0.0.1:8080/Anti-hotlinking\_1/img/pos.jpeg"* width=*"50"* height=*"50"*/>    </body>  </html> |

# listener

|  |
| --- |
| **web.xml中的配置**  <listener>  <listener-class>com.broadin.telegather.web.InitProjectServletContextListener</listener-class>  </listener> |
| **listener类**  /\*  \* @(#) Aa.java 2011-2-17  \* Project :博广通联数据采集系统  \* Copyright: 2010 broadin Inc. All rights reserved.  \*/  package com.broadin.telegather.web;  import javax.servlet.ServletContextEvent;  import javax.servlet.ServletContextListener;  import com.broadin.telegather.ApplicationConstant;  public class InitProjectServletContextListener implements ServletContextListener {  public void contextDestroyed(ServletContextEvent arg0) {  }  public void contextInitialized(ServletContextEvent event) {  // event.getServletContext().setAttribute("contextPath",  // "/"+event.getServletContext().getServletContextName());  event.getServletContext().setAttribute("contextPath","/telegather-webapp");  //Ext 皮肤样式  event.getServletContext().setAttribute("xtheme","none.css");  event.getServletContext().setAttribute("title", "后台管理");  event.getServletContext().setAttribute("PAGE\_SIZE",ApplicationConstant.PAGE\_SIZE);  }  } |

# servlet读取properties文件

|  |
| --- |
| <!-- 初始化用的servlet, 应用启动时执行 -->  <servlet>  <servlet-name>initPrameter</servlet-name>  <servlet-class>com.ismp.web.servlet.InitPrameter</servlet-class>  <init-param>  <param-name>propertiesFile</param-name>  <param-value>ismp-customer-webapp.properties</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>initPrameter</servlet-name>  <url-pattern>/initPrameter</url-pattern>  </servlet-mapping> |
| /\*  \* @(#) 2010-11-14  \* Project :新疆商务领航  \* Copyright: 2010 XJGZ Inc. All rights reserved.  \*/  package com.ismp.web.servlet;  import java.io.BufferedInputStream;  import java.io.FileInputStream;  import java.io.InputStream;  import java.util.Properties;  import javax.servlet.ServletConfig;  import javax.servlet.ServletContext;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  /\*\*  \* web应用启动时, 把一些基本信息从properties文件里读取后, 放到Application里面  \*  \* @author jinwanlin  \* @version 3.0 Revise History:  \*  \*/  @SuppressWarnings("serial")  public class InitPrameter extends HttpServlet {  public void init() throws ServletException {  }  @Override  public void init(ServletConfig config) throws ServletException {  ServletContext application = config.getServletContext();  String propertiesFile = config.getInitParameter("propertiesFile");  String propertiesFileFullPath = application.getRealPath("/WEB-INF/classes/"+propertiesFile);    Properties props = new Properties();  try {  InputStream in = new BufferedInputStream(new FileInputStream(propertiesFileFullPath));  props.load(in);    application.setAttribute("managerHome", props.getProperty("com.ismp.web.managerUrl"));    //读取所有配置  // Enumeration en = props.propertyNames();  // while (en.hasMoreElements()) {  // String key = (String) en.nextElement();  // String Property = props.getProperty(key);  // System.out.println(key + Property);  // }  in.close();  } catch (Exception e) {  e.printStackTrace();  }finally{    }  }  } |

# log4j配置

|  |
| --- |
| # Configure logging  log4j.rootLogger=INFO, stdout  log4j.logger.com.ismp=DEBUG, appfile  log4j.appender.stdout=org.apache.log4j.ConsoleAppender  log4j.appender.stdout.layout=org.apache.log4j.PatternLayout  log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n |

# getServletConfig() 区别getServletContext()

|  |
| --- |
| **getServl**e**tConfig()**    在servlet初始化时，容器传递进来一个ServletConfig对象并保存在servlet实例中，该对象允许访问两项内容：初始化参数和 ServletContext对象，前者通常由容器在文件中指定，允许在运行时向sevrlet传递有关调度信息，比如说getServletConfig().getInitParameter("debug")后者为servlet提供有关容器的信息。此方法可以让servlet在任何时候获得该对象及配置信息。  **getServletContext()**  一个servlet可以使用getServletContext（）方法得到web应用的servletContext  即而使用getServletContext的一些方法来获得一些值  比如说getServletContext().getRealPath("/")来获得系统绝对路径  getServletContext().getResource("WEB-INF/config.xml")来获得xml文件的内容 |

# 保存数组在内存, 生命周期类似Application(例子: 长沙国安--过滤字典)

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.Collections;  **import** java.util.List;  /\*\*  \* 常量类  \*  \* **@author** zhangweidong  \* **@version** 3.0 Revise History:  \*  \*/  **public** **class** ApplicationConstant {  /\*\* 过滤字典的关键字List \*/  **public** **static** List<String> *keywordList* = Collections.*synchronizedList*(**new** ArrayList<String>());    /\*\*  \* 添加关键字  \*  \* **@param** keyword  \*/  **public** **static** **void** putKeyword(String keyword) {  **if** (!*keywordList*.contains(keyword)) {  *keywordList*.add(keyword);  }  }  /\*\*  \* 批量添加列表  \*  \* **@param** keyword  \*/  **public** **static** **void** putKeyword(List<String> keyword) {  **if** (!*keywordList*.contains(keyword)) {  *keywordList*.addAll(keyword);  }  }  /\*\*  \* 移除关键字  \*  \* **@param** keyword  \*/  **public** **static** **boolean** removeKeyword(String keyword) {  **synchronized** (*keywordList*) {  **if** (*keywordList*.contains(keyword)) {  *keywordList*.remove(keyword);  **return** **true**;  }**else**{  **return** **false**;  }  }  }  } |

# 字符串过滤(例子: 长沙国安--过滤字典)

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.Collections;  **import** java.util.List;  **import** com.broadin.core.service.AbstractBaseService;  **import** com.broadin.oms.dao.dict.IFilterDictDao;  **import** com.broadin.oms.model.dict.FilterDict;  **import** com.broadin.oms.util.PageResult;  /\*\*  \* 系统过滤字典service  \*  \* **@author** zhangweidong  \* **@version** 3.0 Revise History:  \*  \*/  **public** **class** FilterDictServiceImpl **extends** AbstractBaseService<FilterDict> **implements** IFilterDictService {  /\*\*  \* 条件查询过滤字典列表  \*  \* **@param** filterDict  \* **@param** pageIndex  \* **@param** pageSize  \* **@return**  \*/  @SuppressWarnings("unchecked")  **public** PageResult list(FilterDict filterDict, **int** pageIndex, **int** pageSize){  IFilterDictDao dao = (IFilterDictDao) getGenericDao();    //当pageIndex <= 0 或者 pageSize <=0, 认为是不分页  **if**(pageIndex<=0 || pageSize<=0){  pageIndex = 0;  pageSize = 0;  }    List list = dao.list(filterDict, pageIndex, pageSize);  **int** total = dao.count(filterDict);  **return** **new** PageResult(list, total);  }    /\*\*  \* 数据库中是否有 指定关键字的数据  \*  \* **@param** filterDict  \* **@param** pageIndex  \* **@param** pageSize  \* **@return**  \*/  **public** **boolean** hasFilterDictByKeyWords(**final** String keywords){  IFilterDictDao dao = (IFilterDictDao) getGenericDao();  **return** dao.hasFilterDictByKeyWords(keywords);  }    /\*\*  \* 查找 是否有符合要求的字符串  \*  \* **@return**  \*/  @SuppressWarnings("unchecked")  **public** List<String> searchWord(String content, List arr) {  **if**(arr.isEmpty()){  **return** Collections.*EMPTY\_LIST*;  }    Node rootNode = **new** Node('R');  *createTree*(arr, rootNode);    List<String> words = **new** ArrayList<String>();  List<String> word = **new** ArrayList<String>();  **char**[] chars = content.toCharArray();  Node node = rootNode;  **int** a = 0;  **while** (a < chars.length) {  node = *findNode*(node, chars[a]);  **if** (node == **null**) {  node = rootNode;  a = a - word.size();  word.clear();  } **else** **if** (node.flag == 1) {  word.add(String.*valueOf*(chars[a]));  StringBuffer sb = **new** StringBuffer();  **for** (String str : word) {  sb.append(str);  }  words.add(sb.toString());  a = a - word.size() + 1;  word.clear();  node = rootNode;  } **else** {  word.add(String.*valueOf*(chars[a]));  }  a++;  }  **if** (words != **null** && words.size() > 0) {  **return** words;  }  **return** **null**;  }  /\*\*  \* 创建树  \*  \*/  @SuppressWarnings("unchecked")  **private** **static** **void** createTree(List arr, Node rootNode) {    **for** (**int** i = 0; i < arr.size(); i++) {  String str = arr.get(i).toString();  **char**[] chars = str.toCharArray();  **if** (chars.length > 0)  *insertNode*(rootNode, chars, 0);  }  }  **private** **static** **void** insertNode(Node node, **char**[] cs, **int** index) {  Node n = *findNode*(node, cs[index]);  **if** (n == **null**) {  n = **new** Node(cs[index]);  node.nodes.add(n);  }  **if** (index == (cs.length - 1))  n.flag = 1;  index++;  **if** (index < cs.length)  *insertNode*(n, cs, index);  }  **private** **static** Node findNode(Node node, **char** c) {  List<Node> nodes = node.nodes;  Node rn = **null**;  **for** (Node n : nodes) {  **if** (n.c == c) {  rn = n;  **break**;  }  }  **return** rn;  }  **private** **static** **class** Node {  **public** **char** c;  **public** **int** flag; // 1：表示终结，0：延续  **public** List<Node> nodes = **new** ArrayList<Node>();  **public** Node(**char** c) {  **this**.c = c;  **this**.flag = 0;  }  }  } |