# XML

## XHTML与HTML的区别

xhml比html具有更严格的限制：

在xhtml中

1.标记名称必须小写

2.属性名称必须小写 如<img src="..">中的src

3.必须严格的嵌套 如<div><p></p></div>

4.标记必须封闭 <p></p>必须要有</p> <br/>必须要有/>

5.属性值要用双引号括起来 <table width="60">

6.属性值必须使用完整形式 <div disable="true">

7.应该区分内容标记与结构标记 不能<p><table></table></p> p是内容标记不能包含结构标记table标记

可以把?XHTML?语法规则简单的归纳如下：??????

html?、?head?、?title?、?body?元素一个都不能少，并且只能有?html?一个?root?元素；???????

不要使用简化的属性，必须使用?“name='value'”?的形式；???????

标签名和属性名都用小写字母，属性值要用引号括起来；???????

标签必须是闭合的，并且不能交叉嵌套；???????

使用?id?属性而不是?name?属性来获取元素；

# CSS

## 样式选择器

1.标记选择器 h1{}

2.类选择器 .red{}

3.id选择器 #red{}

id选择器与类选择器的区别:

(1)可以对同一个标记使用多个类选择器

<p class="one two"></p>

(2)每个标记的ID都应该不相同

(3)一个标记只能有一个ID

4."交集"选择器 div.special{...} div#special{...}

div p span.first{....}【div下的p中的类为first的span】

5."并集"选择器 h1,h2,h3{...}【h1,h2,h3都使用该样式】

特性：

css具有继承特性：如果修改一个标记的样式那么他dom树的子孙都会改变。

css具有层叠特性：行内样式>id样式>类别样式>标记样式

## 引入CSS

1.行内式

<h1 style="color:white;background-color:blue">this is h1</h1>

2.内嵌式

<style type="text/css">

h1{

color:white;

background-color:blue

}

</style>

3.导入式(先加载网页不加载css)

<style type="text/css">

@import"mystyle.css"

</style>

4.链接式(先加载css后网页比导入式更友好)

<link href="mystyle.css" rel="stylesheet" type="text/css">

# Servlet

## 符合j2ee标准的web-app的目录结构

* + WEB-INF
    - classes
    - web.xml
    - lib

## servlet生命周期

主要分为servlet创建、响应请求、终止阶段。

**Servlet何时被创建：**

　　1，默认情况下，当WEB客户第一次请求访问某个Servlet的时候，WEB容器将创建这个Servlet的实例。

　　2，当web.xml文件中如果<servlet>元素中指定了<load-on-startup>子元素时，Servlet容器在启动web服务器时，将按照顺序创建并初始化Servlet对象。

　　注意：在web.xml文件中，某些Servlet只有<serlvet>元素，没有<servlet-mapping>元素，这样我们无法通过url的方式访问这些Servlet，这种Servlet通常会在<servlet>元素中配置一个<load-on-startup>子元素，让容器在启动的时候自动加载这些Servlet并调用init()方法，完成一些全局性的初始化工作。

**Servlet响应请求阶段：**

　　对于用户到达Servlet的请求，Servlet容器会创建特定于这个请求的ServletRequest对象和ServletResponse对象，然后调用Servlet的service方法。service方法从ServletRequest对象获得客户请求信息，处理该请求，并通过ServletResponse对象向客户返回响应信息。

　　对于Tomcat来说，它会将传递过来的参数放在一个Hashtable中，该Hashtable的定义是：

|  |
| --- |
| private Hashtable<String String[]> paramHashStringArray = new Hashtable<String String[]>(); |

　　这是一个String-->String[]的键值映射。

　　HashMap线程不安全的，Hashtable线程安全。

**Servlet终止阶段：**

　　当WEB应用被终止，或Servlet容器终止运行，或Servlet容器重新装载Servlet新实例时，Servlet容器会先调用Servlet的destroy()方法，在destroy()方法中可以释放掉Servlet所占用的资源。

* + 只有一个对象
  + 第一次请求的时候被初始化，只一遍
  + 初始化后先调用init方法，只一遍
  + 每个请求，调用一遍service🡪service🡪doGet/doPost。以多线程的方式运行
    - **不要在servlet中设计成员变量。**
  + 卸载前调用destroy方法

## 为什么有2个init方法

## Cookie 和session的区别

注意cookie有时候禁不掉

|  |  |
| --- | --- |
| Cookie | session |
| 存储在客户端 | 存储在服务器端 |
| 两种类型   * 有声明周期 * 无声明周期 | 两种实现方式   * 依赖于cookie * url重写   response.encodeURL(request.getRequestURL().toString()) |
| 父路径不能访问子路径的cookie | 同一个session的窗口共享一个session |
| 典型应用：   * + 3个月不用再登陆   + 购物车（http://www.china-pub.com/） | 典型应用：   * 用户登陆 * 购物车也可以用session实现。· |
| 不可靠 | 可靠 |

## forward和sendredirect的区别

|  |  |
| --- | --- |
| Sendredirect | Forward |
| 是不同的request | 虽然是不同的对象，但是，可以取到上一个页面的内容 |
| send后的语句会继续执行，除非return | forward后的语句不会继续发送给客户端 |
| 速度慢 | 速度快 |
| 需要到客户端的往返，可以转到任何页面， | 服务器内部转换， |
| 地址栏有变化 | 地址栏没有变化 |
| 可以传参数，直接写在url后面 | 可以传参数 |
| /代表的是http://127.0.0.1/  <%response.sendRedirect("/test/ServletToJSP");%>  <form action="/test/ServletToJSP">  <a href="/test/ServletToJSP">test</a> | /代表的是http://127.0.0.1/test  **this**.getServletContext().getRequestDispatcher ("/servlet\_jsp/ServletUseJsp.jsp") .forward(req, resp);  并且只能以/开头 |
| 常用 | 常用 |

## jsp内置对象

out request response session application exception

# 附录

## JSP和Servlet页面中文显示为乱码的解决方式

经过网上搜索和自己的经验，JSP/Servlet页面的中文显示为乱码的问题可以细分为5类:  
1. JSP页面显示乱码。  
2. Servlet接收Form/Request传递的参数时显示为乱码  
3. JSP接收Form/Request传递的参数时显示为乱码  
4. 用<jsp:forward page="catalog2.html"></jsp:forward>时页面显示乱码  
5. 数据库存取的时候产生乱码。  
解决方式：  
1. JSP页面显示乱码的解决方式：  
**第一种方式**为在页面的开头加上：  
<%@ page language="java" contentType="text/html; charset=GBK" pageEncoding="GBK"%>  
<!--这里的 GBK可以由 gb2312代替，此处以GBK为例。下同 -->  
注：有时候如果不再页面开头加上这句，则页面中无法保存中文字符，并提示：中文字符在不能被iso-8859-1字符集mapped，这是由于默认情况下，JSP是用iso-8859-1来编码的，可以在Window->Preferences->General->Content Type选项下，在右边的窗口选择Text->Jsp，然后在下面的Default Encoding由默认的iso-8859-1改为GBK，然后点击update即可解决。  
然而这种方式会带来一些问题：由于这一句在其他文件include该文件的时候不能被继承，所以include它的文件也需要在文件开头加上这句话，此时如果用的是pageEncoding="gbk"则会出现问题。类似于org.apache.jasper.JasperException: /top.jsp(1,1) Page directive: illegal to have multiple occurrences of contentType with different values (old: text/html;charset=GBK, new: text/html;charset=gbk).  
类似地，如果两个文件一个用的是gbk，一个用的是gb2312也会出现问题。  
**另一种更好的解决方式为：**  
在项目的web.xml中添加以下片段：  
<!-- 下面的代码为解决页面乱码问题而加入 -->  
<jsp-config>   
            <jsp-property-group>   
                <description>   
                   Special property group for JSP Configuration JSP example.   
                </description>   
                <display-name>JSPConfiguration</display-name>   
                <url-pattern>\*.jsp</url-pattern>   
                <el-ignored>true</el-ignored>   
                <page-encoding>GBK</page-encoding>   
                <scripting-invalid>false</scripting-invalid>   
                <include-prelude></include-prelude>   
                <include-coda></include-coda>   
            </jsp-property-group>              
            <jsp-property-group>   
                <description>   
                   Special property group for JSP Configuration JSP example.   
                </description>   
                <display-name>JSPConfiguration</display-name>   
                <url-pattern>\*.html</url-pattern>   
                <el-ignored>true</el-ignored>   
                <page-encoding>GBK</page-encoding>   
                <scripting-invalid>false</scripting-invalid>   
                <include-prelude></include-prelude>   
                <include-coda></include-coda>   
</jsp-property-group>   
</jsp-config>   
<!--       添加的代码结束 -->  
**2. Servlet接收Form/Request传递的参数时显示为乱码的解决方式：**  
**第一种解决方式**为在用到request方法的前面加上这条语句：  
request.setCharacterEncoding("GBK");  
同样地，这也会由于页面设置中GbK或gB2312大小写不同或者采用不同的汉语字符集而发生错误。  
**另一种更好的解决方式**为：添加一个名为SetCharacterEncodingFilter的filter。  
filter的源文件为（参见apach安装目录下\webapps\jsp-examples\WEB-INF\classes\filters中的SetCharacterEncodingFilter.java文件）：  
  
package com.filters;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.UnavailableException;  
public class SetCharacterEncodingFilter implements Filter {

       protected String encoding = null;  
         protected FilterConfig filterConfig = null;  
         protected boolean ignore = true;  
         public void destroy() {  
             this.encoding = null;  
             this.filterConfig = null;  
         }    
         public void doFilter(ServletRequest request, ServletResponse response,  
                              FilterChain chain)  
throws IOException, ServletException {  
             // Conditionally select and set the character encoding to be used  
             if (ignore || (request.getCharacterEncoding() == null)) {  
                 String encoding = selectEncoding(request);  
                 if (encoding != null)  
                     request.setCharacterEncoding(encoding);  
             }  
// Pass control on to the next filter  
             chain.doFilter(request, response);  
         }  
  
        public void init(FilterConfig filterConfig) throws ServletException {  
this.filterConfig = filterConfig;  
             this.encoding = filterConfig.getInitParameter("encoding");  
             String value = filterConfig.getInitParameter("ignore");  
             if (value == null)  
                 this.ignore = true;  
             else if (value.equalsIgnoreCase("true"))  
                 this.ignore = true;  
             else if (value.equalsIgnoreCase("yes"))  
                 this.ignore = true;  
             else  
                 this.ignore = false;  
         }  
           protected String selectEncoding(ServletRequest request) {  
             return (this.encoding);  
         }  
  
}  
同时在web.xml中添加一下片段：  
<!-- 为解决乱码问题而添加 -->    
          <filter>   
             <filter-name>SetCharacterEncoding</filter-name>   
             <filter-class>com.filters.SetCharacterEncodingFilter</filter-class>   
             <init-param>   
                 <param-name>encoding</param-name>   
                 <param-value>GBK</param-value>   
             </init-param>   
          </filter>   
         <filter-mapping>   
             <filter-name>SetCharacterEncoding</filter-name>   
             <url-pattern>/\*</url-pattern>   
         </filter-mapping>  
<!-- 添加代码结束 -->    
**3. JSP接收Form/Request传递的参数时显示为乱码**  
当我们按照第二种乱码的解决方式修改了web.xml并添加了filter之后，有时候并不一定就对乱码问题高枕无忧了，有时候我们会奇怪的发现Sevlet接收Form/Request传递的参数可以正常显示了，但是jsp页面接受Form/Request传递的参数却仍然显示为乱码。这是为什么呢？  
对于我遇到的情况而言，我发现是由于我在用Form发送信息的页面采用了这样的html:  
<form action="getParam.jsp" >  
姓名<input type="text" name ="UserName"> <br>  
选出你喜欢吃的水果：  
<input type ="checkbox" name = "checkbox1" value = "苹果"> 苹果  
<input type ="checkbox" name = "checkbox1" value = "西瓜"> 西瓜  
<input type ="checkbox" name = "checkbox1" value = "桃子"> 桃子  
<input type ="checkbox" name = "checkbox1" value = "葡萄"> 葡萄  
<input type = "submit" value = "提交">  
</form>  
也就是说没有指定form的method属性。而问题就发生在此，Form的默认mothod属性为get.  
而get是通过在发送请求的url后面加？然后加参数和值来传递数据的的，编码格式为ASCII.这就要求我们在传递的数据中有非ASCII字符或是超过了100个字符，那么你必须使用method="post"，否则就会出现乱码。  
所以解决方式为：第二种乱码的解决方式+在发送页面的Form中指定method为post.  
**4. 用<jsp:forward page="catalog2.html"></jsp:forward>时页面显示乱码的解决方式**  
此时实际上乱码的原因和产生其他几种乱码的原因不同，它的原因在于我们用eclipse编辑要forward的html或者jsp文件时，采用的编码是可以产生中文乱码的编码而不是GBK或者GB2312.所以解决方式就是把eclipse编辑器的编码方式改为GBK或者GB2312.  
具体操作方式见：上文红色字体部分。  
**5. 数据库存取的时候产生乱码的解决方式**（由于尚未遇到此问题，不知道说的是什么，故此部分完全采用[枫之阁](http://treetree42.bokee.com/)的文字。如有侵权，请及时联系我删除之，谢谢。）  
当然，在写数据库时，也要保正数据库的编码与其它一致：  
我们可以在系统盘windows目录下的my.ini文件，在文件中插入一行default-character-set=GBK,但上面说了这么多，大家也应该明白些了吧，改动太多的默认设置不是我的风格，因此上，这一行还是不要加的好。   
但不管怎么样，我们还是要创建一个基于中文编码的数据库，当然，用客户端登录的时候，某些客户用自动把字体编码转换成中文编码。在这里，我想说一下在DOS下创建中文编码数据库的方法：   
在进入数据库的时候，用mysql --default-character-set=gbk -u root -p 这句话进入mysql，然后创建数据库，如：create database admin;这样创建起来的数据库就是基于中文编码的了。  
用连接数据库的时候，读出的数据也可能是乱码，解决这个问题的方法非常简单，只要在你建立数据库连接的时候把URL设置成下面这个样子就可以了：URL= jdbc:mysql://localhost:3306/my\_database?useUnicode=true&characterEncoding=GBK   
好了，说了这么多，总结一句话结束吧，把各种地方的编码统一起来，那么，所在的乱码问题就都解决了！

## 拷贝javeWeb项目时可能出现的问题

1.直接拷贝javeweb项目时它的web context-root 路径可能没改：

解决办法：

首选项->myeclipse->web->把web context-root改为当前项目名

2.jdk问题

a.引用的jre包出现问题

解决办法：把当前的jre包从项目中移除 在添加当前可用的jre包

b.项目的jdk编译级别问题

解决办法：prefence->java->complier

c.tomcat的jdk版本问题

## 四个有用的过滤器

### 使浏览器不缓存页面的过滤器

import javax.servlet.\*;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

public class ForceNoCacheFilter implements Filter {

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

{

((HttpServletResponse) response).setHeader("Cache-Control","no-cache");

((HttpServletResponse) response).setHeader("Pragma","no-cache");

((HttpServletResponse) response).setDateHeader ("Expires", -1);

filterChain.doFilter(request, response);

}

public void destroy()

{

}

public void init(FilterConfig filterConfig) throws ServletException

{

}

}

### 检测用户是否登陆的过滤器

import javax.servlet.\*;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import java.util.List;

import java.util.ArrayList;

import java.util.StringTokenizer;

import java.io.IOException;

public class CheckLoginFilter

implements Filter

{

protected FilterConfig filterConfig = null;

private String redirectURL = null;

private List notCheckURLList = new ArrayList();

private String sessionKey = null;

public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException

{

HttpServletRequest request = (HttpServletRequest) servletRequest;

HttpServletResponse response = (HttpServletResponse) servletResponse;

HttpSession session = request.getSession();

if(sessionKey == null)

{

filterChain.doFilter(request, response);

return;

}

if((!checkRequestURIIntNotFilterList(request)) && session.getAttribute(sessionKey) == null)

{

response.sendRedirect(request.getContextPath() + redirectURL);

return;

}

filterChain.doFilter(servletRequest, servletResponse);

}

public void destroy()

{

notCheckURLList.clear();

}

private boolean checkRequestURIIntNotFilterList(HttpServletRequest request)

{

String uri = request.getServletPath() + (request.getPathInfo() == null ? "" : request.getPathInfo());

return notCheckURLList.contains(uri);

}

public void init(FilterConfig filterConfig) throws ServletException

{

this.filterConfig = filterConfig;

redirectURL = filterConfig.getInitParameter("redirectURL");

sessionKey = filterConfig.getInitParameter("checkSessionKey");

String notCheckURLListStr = filterConfig.getInitParameter("notCheckURLList");

if(notCheckURLListStr != null)

{

StringTokenizer st = new StringTokenizer(notCheckURLListStr, ";");

notCheckURLList.clear();

while(st.hasMoreTokens())

{

notCheckURLList.add(st.nextToken());

}

}

java源动力,java开源社区,http://www.web-java.com

}

}

### 字符编码的过滤器

import javax.servlet.\*;

import java.io.IOException;

public class CharacterEncodingFilter

implements Filter

{

protected FilterConfig filterConfig = null;

protected String encoding = "";

public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException

{

if(encoding != null)

servletRequest.setCharacterEncoding(encoding);

filterChain.doFilter(servletRequest, servletResponse);

}

public void destroy()

{

filterConfig = null;

encoding = null;

}

public void init(FilterConfig filterConfig) throws ServletException

{

this.filterConfig = filterConfig;

this.encoding = filterConfig.getInitParameter("encoding");

}

}

### 资源保护过滤器

package catalog.view.util;

import javax.servlet.Filter;

import javax.servlet.FilterConfig;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import java.io.IOException;

import java.util.Iterator;

import java.util.Set;

import java.util.HashSet;

//

import org.apache.commons.logging.Log;

import org.apache.commons.logging.LogFactory;

public class SecurityFilter implements Filter {

//the login page uri

private static final String LOGIN\_PAGE\_URI = "login.jsf";

//the logger object

private Log logger = LogFactory.getLog(this.getClass());

//a set of restricted resources

private Set restrictedResources;

public void init(FilterConfig filterConfig) throws ServletException {

this.restrictedResources = new HashSet();

this.restrictedResources.add("/createProduct.jsf");

this.restrictedResources.add("/editProduct.jsf");

this.restrictedResources.add("/productList.jsf");

}

public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)

throws IOException, ServletException {

this.logger.debug("doFilter");

String contextPath = ((HttpServletRequest)req).getContextPath();

String requestUri = ((HttpServletRequest)req).getRequestURI();

this.logger.debug("contextPath = " + contextPath);

this.logger.debug("requestUri = " + requestUri);

if (this.contains(requestUri, contextPath) && !this.authorize((HttpServletRequest)req)) {

this.logger.debug("authorization failed");

((HttpServletRequest)req).getRequestDispatcher(LOGIN\_PAGE\_URI).forward(req, res);

}

else {

this.logger.debug("authorization succeeded");