# XMl

## 概念

eXtensible Markup Language 可扩展的标记语言。其实就是一个文件，文件的后缀为 .xml

如果说java是一门跨平台的语言，那XML就是跨平台的数据。

应用场景

1. 可以用来保存数据

2. 可以用来做配置文件：利用XML可以保存具有关系的数据的特性

3. 利用XML跨平台的特性，用来在不同的操作系统不同的开发语言之间传输数据

### 文档声明

简单声明， version : 解析这个xml的时候，使用什么版本的解析器解析

<?xml version="1.0" ?>

encoding : 解析xml中的文字的时候，使用什么编码来翻译

<?xml version="1.0" encoding="gbk" ?> encoding属性默认值为老外喜欢的iso8859-1

standalone : no - 该文档会依赖关联其他文档 ， yes-- 这是一个独立的文档

<?xml version="1.0" encoding="gbk" standalone="no" ?> 默认值为yes表示是一个独立文档。

注意：很多的解析器会忽略这个属性，但是学习知识要按标准去学，所以这个属性也要掌握。

encoding详解

在解析这个xml的时候，使用什么编码去解析。 ---解码。

老师演示了 ：下面xml内容正常点击保存拿到浏览器打开会报错：

<?xml version="1.0" encoding="utf-8"?>

<a>

<b>你好</b>

</a>

原因：windows系统默认是gbk编码来保存的：你在点击保存的时候其实用的是gbk编码保存，xml内容又是utf-8(拿到浏览器中渲染是根据xml头文件指定的编码来解析)就会报错；

解决办法：

第一：保存文件的时候使用另存为:编码选择为utf-8进行保存

第二：头文件encoding改为gbk，点击保存即可:因为windows默认就是ANSI(windows根据默认编码操作)来保存的

最后注意为了通用：建议使用UTF-8编码保存，以及encoding 都是 utf-8

### 元素(标签)

一个xml标签就是一个元素，标签分为开始标签、结束标签

标签体：在开始标签和结束标签之间的文本称为标签体。

自闭标签: 一个标签即不含标签体也不包含其他标签，把开始标签和结束标签进行合并，叫自闭标签。

格式良好的XML文档应该具有并且只能有一个根标签，其他标签都应该是这个跟标签的子孙标签。

XML 命名规则：

名称可以含字母、数字以及其他的字符 ，区分大小写，例如，<P>和<p>是两个不同的标记。

名称不能以数字或者标点符号开始

名称不能以字符 “xml”（或者 XML、Xml）开始

名称不能包含空格、中间不能包含冒号（:）

属性

一个元素可以包含多个属性，属性值要用单引号或双引号括起来，属性的命名规范，参照元素的命名规范。

注释

<!-- 这是一段注释 --> 注意：注释不能出现在文档声明之前。

### CDATA区和转义字符

转义字符

& --> &amp;

< --> &lt;

> --> &gt;

" --> &quot;

' --> &apos;

被CDATA区扩起来的内容，将会被浏览器当作文本来处理。

<des><![CDATA[<a href="http://www.baidu.com">我爱黑马训练营</a>]]></des>

CDATA区和转义字符的区别

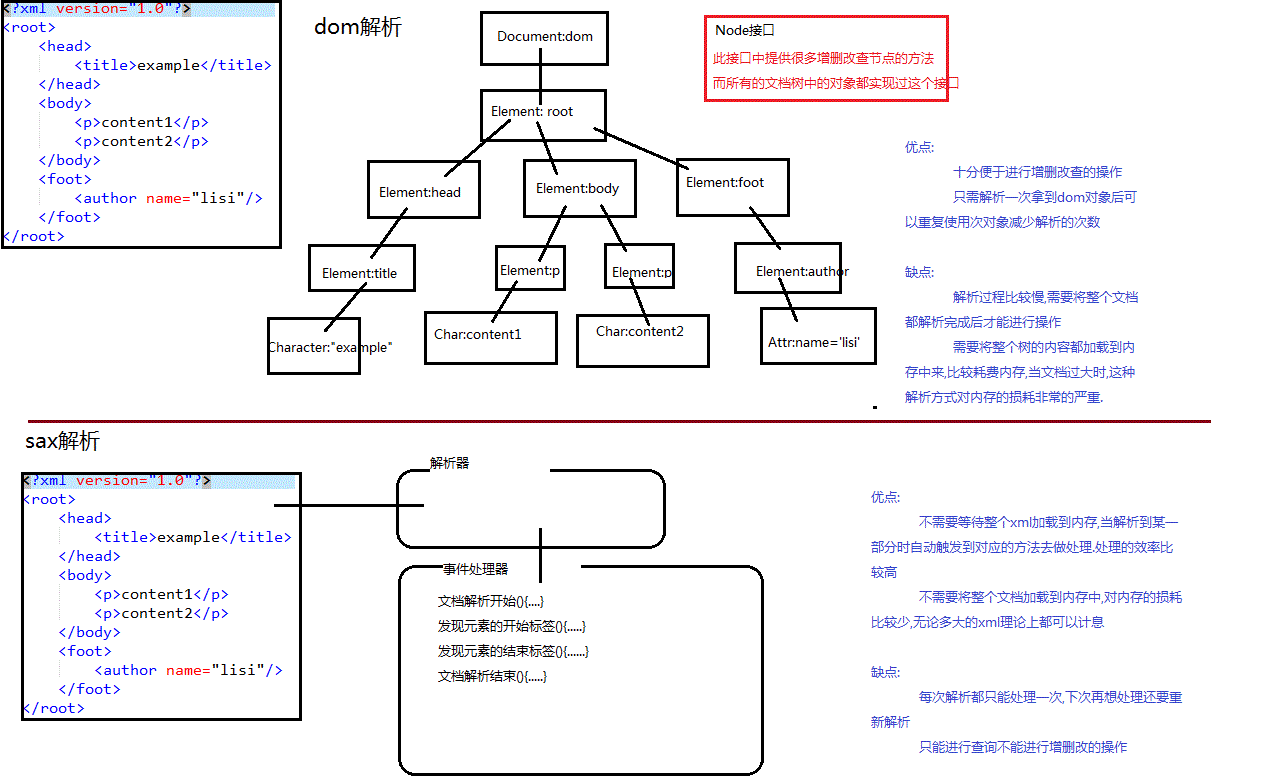
(1)CDATA区可以成段的进行转义，而转义字符一次只能转义一个字符

(2)CDATA区转义的字符可以保存数据本来的格式只是通知解析器按文本去处理。转义字符改变了数据本身的内容，利用其他字符替代了转义字符。

请思考，如果要转义的内容是一段js程序的话，如果用转义字符合适不合适？

## 解析XML

常用的有两种Dom和SAX：jaxp既有dom方式也有sax方式,并且这套解析api已经加入到j2se的规范中



### DOM

DOM4J是第三方的，基于Dom进行优化的。

需要导入包：dom4j-1.6.1.jar和hibernate包中的jaxen-1.1-beta-7.jar

<书架>

<书 id="1">

<书名>倚天屠龙记</书名>

<作者>金庸</作者>

<价格>100</价格>

</书>

<书 id="2">

<书名>陆小凤</书名>

<作者>古龙</作者>

<价格>99</价格>

</书>

</书架>

整体简单流程

//第一步 创建对象 通过流read读取全部xml内容放入内存中

SAXReader sr = new SAXReader();

Document d=sr.read(Demo1.class.getClassLoader().getResourceAsStream(”a.xml”));

//第二步 获得根节点对象Element对象

Element root=d.getRootElement();

//第三步 获得根节点中的下一个标签<书>：用根节点的对象root获取

获取元素有3种方式 ①获取root下指定的标签：root.element("书");

②获取全部：list获取root.elements();迭代：root.elementIterator()

③获取重名的名字全部：list获取root.elements(“书”);迭代：root.elementIterator(“书”)

List<Element> list=root.elements();

Element shu=list.get(1);

Element shu2=root.element("书");

//第四步 获取属性得到对象Attribute

Attribute id=shu.attribute("id");

String idName=id.getName();//打印：id 就是属性名字不是值

String idValue2=id.getValue();//打印：值 ：1

String idValue=id.getStringValue();//也是打印：1

System.out.println(shu.element("作者").getStringValue());//只有StringValue这个属性，打印：金庸

Element shuming=shu.element("书名");

shuming.getText(); //获取到书名：倚天屠龙记

System.out.println();

封装方法：Dom4j是一次性加载到内存中的：可以重复调用

封装加载xml方法，使用成员变量接收document对象

public static void loadDocument(){ //封装

try{

SAXReader saxReader = new SAXReader();

doc = saxReader.read(new File("a.xml"));

}catch (Exception e) {

// TODO: handle exception

e.printStackTrace();

}

}

查询操作

public static void showPhoneInfo() {

//第一步 获取XML的根节点

Element root = doc.getRootElement();

//第二步 遍历所有的Brand标签

for (Iterator itBrand = root.elementIterator(); itBrand.hasNext();) {

Element brand = (Element) itBrand.next();

// 输出标签的name属性

System.out.println("品牌：" + brand.attributeValue("name"));

// 遍历Type标签

for (Iterator itType = brand.elementIterator(); itType.hasNext();) {

Element type = (Element) itType.next();

// 输出标签的name属性

System.out.println("\t型号：" + type.attributeValue("name"));

}

}

}

添加操作

public static void addNewPhoneInfo(){

// 获取XML的根节点

Element root = doc.getRootElement();

// 创建Brand标签

Element el = root.addElement("Brand");

// 给Brand标签设置属性

el.addAttribute("name", "三星");

// 创建Type标签

Element typeEl = el.addElement("Type");

// 给Type标签设置属性

typeEl.addAttribute("name", "Note4");

saveXML("src/收藏信息.xml");

}

修改操作

public static void updatePhoneInfo(){

// 获取XML的根节点

Element root = doc.getRootElement();

int id = 0;

for (Iterator itBrand = root.elementIterator(); itBrand.hasNext();) {

Element brand = (Element) itBrand.next();

id++;

brand.addAttribute("id", id + "");

}

saveXML("src/收藏信息.xml");

}

删除操作

public static void deleteItem(){

// 获取XML的根节点

Element root = doc.getRootElement();

int id = 0;

for (Iterator itBrand = root.elementIterator(); itBrand.hasNext();) {

Element brand = (Element) itBrand.next();

if (brand.attributeValue("name").equals("华为")) {

brand.getParent().remove(brand);

}

}

// saveXML("src/收藏信息.xml");

}

进行修改、增加、删除最后需要进行重新保存操作

public static void saveXML(String path){

try {

OutputFormat format = OutputFormat.createPrettyPrint();

format.setEncoding("GBK"); // 指定XML编码

XMLWriter writer;

writer = new XMLWriter(new FileWriter(path), format);

writer.write(doc);

writer.close();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

### SAX

SAX的优点：

解析速度快，占用内存少

SAX的缺点：

无法知道当前解析标签（节点）的上层标签，及其嵌套结构，仅仅知道当前解析的标签的名字和属性，要知道其他信息需要程序猿自己编码

只能读取XML，无法修改XML

无法随机访问某个标签（节点）

SAX解析适用场合

对于CPU资源宝贵的设备，如Android等移动设备

对于只需从xml读取信息而无需修改xml

解析步骤

1、得到xml文件对应的资源，可以是xml的输入流，文件和uri

2、得到SAX解析工厂（SAXParserFactory）

3、由解析工厂生产一个SAX解析器（SAXParser）

4、传入输入流和handler给解析器，调用parse()解析

# 协议

## HTTP协议

## 一、HTTP协议详解之URL篇

http（超文本传输协议）是一个基于请求与响应模式的、无状态的、应用层的协议，常基于TCP的连接方式，HTTP1.1版本中给出一种持续连接的机制，绝大多数的Web开发，都是构建在HTTP协议之上的Web应用。

URI与URL区别

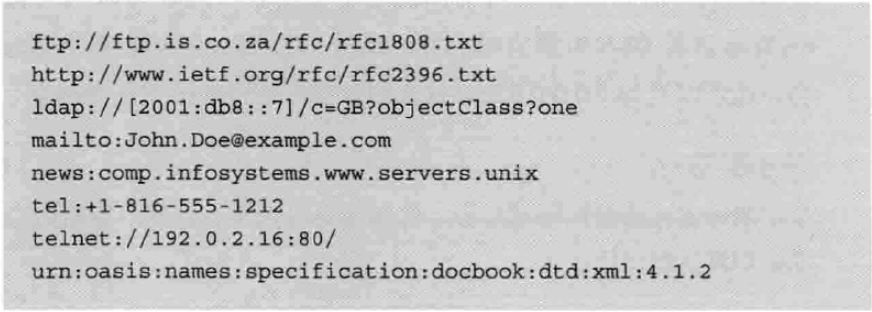
URL：(Uniform/Universal Resource Locator 的缩写，统一资源定位符)。

URI：(Uniform Resource Identifier 的缩写，统一资源标识符)（代表一种标准）。

关系：URI 属于 URL 更高层次的抽象，一种字符串文本标准。URI 属于父类，而 URL 属于 URI 的子类。

区别:URI 表示请求服务器的路径，定义这么一个资源。而 URL 同时说明要如何访问这个资源（http://）。

下图是URI示例：



http://host[":"port][ path]的解释：

http表示要通过HTTP协议来定位网络资源；

host表示合法的Internet主机域名或者IP地址；

port指定一个端口号，为空则使用缺省端口80；

path指定请求资源的URI；如果URL中没有path，请求URI时，是以“/”的形式给出(系统自动)

1、输入：www.guet.edu.cn浏览器自动转换成：http://www.guet.edu.cn/

2、http:192.168.0.116:8080/index.jsp

## 二、HTTP协议详解之请求篇

http请求由三部分组成，分别是：请求行、消息报头（请求头）、请求正文（请求体）

### 请求行

请求行以一个方法符号开头，以空格分开，后面跟着请求的URI和协议的版本，

格式如下：Method Request-URI HTTP-Version CRLF ： POST /WEB12/form.html HTTP/1.1

其中 Method：请求方法；Request-URI是一个统一资源标识符；HTTP-Version：HTTP协议版本；CRLF表示回车和换行（除了作为结尾的CRLF外，不允许出现单独的CR或LF字符）。

请求方法（所有方法全为大写）有多种，各个方法的解释如下：

GET 请求获取Request-URI所标识的资源 除了表单提交指定方法外都是GET请求

POST 在Request-URI所标识的资源后附加新的数据 常用于提交表单

HEAD 请求获取由Request-URI所标识的资源的响应响应头

PUT 请求服务器存储一个资源，并用Request-URI作为其标识

DELETE 请求服务器删除Request-URI所标识的资源

TRACE 请求服务器回送收到的请求信息，主要用于测试或诊断

CONNECT 保留将来使用

OPTIONS 请求查询服务器的性能，或者查询与资源相关的选项和需求

HEAD方法与GET方法几乎是一样的，对于HEAD请求的回应部分来说，它的HTTP头部中包含的信息与通过GET请求所得到的信息是相同的。利用这个方法，不必传输整个资源内容，就可以得到Request-URI所标识的资源的信息。该方法常用于测试超链接的有效性，是否可以访问，以及最近是否更新。

### 请求头

Accept: 客户端向服务器端表示，我能支持什么类型的数据。

Referer ： 真正请求的地址路径，全路径

Accept-Charset请求报头域用于指定客户端接受的字符集 Accept-Charset:iso-8859-1,gb2312

Accept-Language: 支持语言格式

User-Agent: 用户代理 向服务器表明，当前来访的客户端信息。

Content-Type： 提交的数据类型。经过urlencoding编码的form表单的数据

Accept-Encoding： gzip, deflate ： 压缩算法 。

Host ： 主机地址

Content-Length： 数据长度

Connection : Keep-Alive 保持连接

Cache-Control ： 对缓存的操作

请求报头举例：

GET /form.html HTTP/1.1 (CRLF)

Accept: image/jpeg,application/x-shockwave-flash,application/vnd.ms-powerpoint,application/msword,\*/\* (CRLF)

Accept-Language:zh-cn (CRLF)

Accept-Encoding:gzip,deflate (CRLF) 如果请求消息中没有设置这个域服务器假定客户端对各种内容编码都可以接受

If-Modified-Since:Wed,05 Jan 2007 11:21:25 GMT (CRLF)

If-None-Match:W/"80b1a4c018f3c41:8317" (CRLF)

User-Agent:Mozilla/4.0(compatible;MSIE6.0;Windows NT 5.0) (CRLF)

Host:www.guet.edu.cn (CRLF)

Connection:Keep-Alive (CRLF)

(CRLF)

### 请求体

发送的数据呈现的是key=value ,如果存在多个数据，那么使用 &

firstname=zhang&lastname=sansan

### GET和POST的区别

请求中：POST多了两个请求头：Content-Type和Content-Length

Content-Type：是普通内容还是要上传文件类型

Content-Type: application/x-www-form-urlencoded 这是普通内容

Content-Length：请求体中的参数内容长度

Content-Length: 25 最新浏览器都隐藏了一个&：firstname=zhang&lastname=sansan

## 三、HTTP协议详解之响应篇

HTTP响应也是由三个部分组成，分别是：状态行、消息报头、响应正文

### 响应行

响应行格式如下：

HTTP-Version Status-Code Reason-Phrase CRLF HTTP/1.1 200 OK （CRLF）

HTTP-Version：服务器HTTP协议的版本；Status-Code：服务器发回的响应状态代码；Reason-Phrase：状态代码的文本描述。

状态代码有三位数字组成，第一个数字定义了响应的类别，且有五种可能取值：

1xx：指示信息--表示请求已接收，继续处理

2xx：成功--表示请求已被成功接收、理解、接受

3xx：重定向--要完成请求必须进行更进一步的操作

4xx：客户端错误--请求有语法错误或请求无法实现

5xx：服务器端错误--服务器未能实现合法的请求

常见状态代码、状态描述、说明：

200 OK //客户端请求成功

400 Bad Request //客户端请求有语法错误，不能被服务器所理解

401 Unauthorized //请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用

403 Forbidden //服务器收到请求，但是拒绝提供服务

404 Not Found //请求资源不存在，eg：输入了错误的URL

500 Internal Server Error //服务器发生不可预期的错误

503 Server Unavailable //服务器当前不能处理客户端的请求，一段时间后可能恢复正常

### 响应头

Location响应报头域用于重定向接受者到一个新的位置。Location响应报头域常用在更换域名的时候。

Server: 服务器是哪一种类型。 Tomcat

Content-Type ： 服务器返回给客户端你的内容类型 Content-Type:text/html;charset=ISO-8859-1

Content-Length ： 返回的数据长度

Date ： 通讯的日期，响应的时间

Last-Modified实体报头域用于指示资源的最后修改日期和时间

### 响应体

如果是：Content-Type：text/html;charset=ISO-8859-1则返回整个HTMl页面内容

# 服务器

## Tomcat

### 安装和配置

1、tomcat5需jdk1.4以上，tomcat6需jdk1.5以上，tomcat7需jdk1.6以上。安装目录不能包含中文和空格

2、JAVA\_HOME环境变量: 指定tomcat启动时使用的jdk所在的位置

3、Catalina\_Home环境变量: startup.bat启动哪个tomcat由此环境变量指定，如果不配置则启动当前tomcat，推荐不要配置此环境变量

### 目录结构

bin ----存放tomcat启动关闭所用的批处理文件

conf --- tomcat的配置文件，最终要的是server.xml(更改tomcat运行所在的端口号，从8080改为80)

lib --- tomcat运行时需要的jar包所在的目录

logs ---运行时产生的日志文件

temp --- tomcat运行时产生的临时文件存放的目录,不需要我们管理

webapps --- 开发中最常用的目录,web应用放置到此目录下浏览器可以直接访问

work --- 工作目录,tomcat运行时产生的工作文件存放在这个目录中

### 配置访问路径

#### 虚拟主机

一个tomcat可以认为是一台真实主机.在一台真实主机中可以配置多个站点,这些站点在访问者看来访问他们就像在访问各自独立的主机一样,所以我们可以认为这些站点都运行在tomcat这台真实主机当中的各自的虚拟主机当中.一个网站就可以认为是一个虚拟主机

默认的虚拟主机名称为：localhost，虚拟主目录：webapps

<Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true"> </Host>

自定义配置虚拟主机

第一步在conf/server.xml中<Engin>标签下配置<Host>标签就可以为tomcat增加一台虚拟主机

<Host name="www.wwl.com" appBase="D:\zlf"></Host> 意思是：配置了名称为www.wwl.com，虚拟主目录：d盘下的zlf文件夹

name -- 指定虚拟主机的名称,浏览器通过这个名称访问虚拟主机

appBase -- 虚拟主机管理的目录,放置在这个目录下的web应用当前虚拟主机可以自动加载

第二步 在C:\Windows\System32\drivers\etc\hosts添加内容

127.0.0.1 www.wwl.com 把此www.wwl.com转换为ip:127.0.0.1

这两步就可以通过：www.wwl.com:8080/a.jsp访问

注意：localhost是不用配置的windows默认就有

由于浏览器访问地址www.wwl.com时,需要将地址翻译成对应的ip才能找到服务器,windows系统先去找hosts文件有没有，然后才去找网络上的dns服务器翻译.此时可以使用hosts文件模拟dns的功能

配置默认虚拟主机

通过http://127.0.0.1:8080/访问进入的是默认tomcat的猫主页，进行更改的步骤如下：

<Engine name="Catalina" defaultHost="localhost">把defaultHost的值改为自己配置的虚拟主机即可

#### web工程

一个虚拟主机中会有很多的web资源,但是web资源不能直接交给虚拟主机管理,需要按照一定方式组织成web应用虚拟主机才能使用.一般来说我们会按照功能将某一功能相关的所有的web资源组织成一个web应用后再交给虚拟主机.

目录结构

工程根路径：Html、css、js、jsp、WEB-INF(目录它必须有)

WEB-INF目录下：classes(java字节码)、lib(包)、web.xml

web.xml文件的作用：

某个web资源配置为web应用首页

将servlet程序映射到某个url地址上

为web应用配置监听器

为web应用配置过滤器

但凡涉及到对web资源进行配置，都需要通过web.xml文件

Tomcat配置WEB工程有三个方法

第一种：最简单的tomcat默认配置好的目录

直接将web工程放入虚拟主机管理的webapps目录下即可：如工程名为a文件夹

访问：http://localhost:8080/a/user

如果把工程名为a文件夹中的所有资源放入webapps/ROOT/下访问就不用工程名访问(缺省web工程名)

访问：http://localhost:8080/a.jsp

第二种 自定义目录和webapps目录一样的功能(需要重启tomcat服务 )

步骤：在conf/server.xml的<Host>标签下配置<Context path="虚拟路径" docBase="真实路径目录">

配置为<Context path="/c" docBase="D:\wwl"/> 注意：它是自闭标签格式要写对，docBase路径是指定到目录

访问http://localhost:8080/c/ a.jsp即可

配置为<Context path="" docBase="D:\wwl"/> path为空，即为缺省路径

http://localhost:8080/a.xml

第三种在tomcat/conf/引擎目录名/虚拟主机目录名/之下建立一个.xml文件

1、其中文件名用来指定虚拟路径，如果名字有’\_’的用#代替表示，文件名为ROOT.xml是缺省web工程名

2、文件内容：<Context docBase="真实目录">，

注意：Tomcat默认的路径：conf\Catalina\localhost;查看方式：server.xml中：<Engine name="Catalina" defaultHost="localhost">

### 配置代码自动更新

Conf目录下的context.xml内容：

<Context>

<WatchedResource>WEB-INF/web.xml</WatchedResource> //这就是每个web工程为什么要这样配置WEB-INF/web.xml

<Context reloadable="true"> 让tomcat能够自动重新加载，我们修改过的代码和配置

注意： reloadable设为true，目的是为了方便开发阶段， 它会影响tomcat性能；

</Context>

### 全局web.xml

里面有全局默认的Servlet配置程序进行解析：如：jsp，静态资源的解析(html，js)

# JavaWeb技术

1）servlet技术

2）filter技术---过滤器

3）listener技术---监听器

## Servlet

Servlet 运行在服务端的Java小程序，是sun公司提供一套规范（接口），用来处理客户端请求、响应给浏览器的动态资源。但servlet的实质就是java代码，通过java的API动态的向客户端输出内容

javaEE包括13门规范主要学习 servlet技术 和 jsp技术

其中 servlet规范包括三个技术点：servlet listener filter

web项目启动顺序：监听器-->过滤器-->Servlet.

项目启动后,容器会首先创建声明的各种监听器,为后继的各个事件监听做准备,然后创建过滤器,最后是Servlet.

项目销毁：过滤器-->Servlet-->监听器

### Servlet对象创建

#### 实现Servlet接口

1）创建类实现Servlet接口、实现Servlet接口方法：有5个方法

public class MyServlet implements Servlet {

何时执行：init方法可以认为是在构造方法中的，创建对象只调用一次

参数ServletConfig：代表的是该servlet对象的配置信息

public void init(ServletConfig config) throws ServletException {

System.out.println("我只调用一次");

//1、获得servlet的name----<servlet-name>abc</servlet-name>

String servletName = config.getServletName();

System.out.println(servletName);//abc

//2、获得该servlet的初始化的参数--- <init-param> </init-param>

String initParameter = config.getInitParameter("url");

System.out.println(initParameter);

//3、获得Servletcontext对象

ServletContext servletContext = config.getServletContext();

}

何时执行：每次请求都会执行

参数ServletRequest ：代表请求 可以认为ServletRequest 内部封装的是:http请求的信息（如请求头的key和value）、请求体

参数ServletResponse ：代表响应 可以认为要封装的是响应的信息返回给客户端的：响应头和响应体

public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException {

System.out.println("我只要有请求都调用");

}

何时执行：servlet销毁的时候执行

public void destroy() {

System.out.println("服务器正常关闭调用，也就是Servlet对象销毁");

}

public ServletConfig getServletConfig() {

return null;

}

public String getServletInfo() {

return null;

}

}

2）在web.xml进行servlet的配置

<servlet> 第一 创建标签servlet：必须有子标签servlet-name和servlet-class

<servlet-name>b</servlet-name>

<servlet-class>wwl.MyServlet2</servlet-class> 实现Servlet或者继承Servlet子类的类

</servlet>

<servlet-mapping> 第二 创建标签servlet-mapping：

<servlet-name>b</servlet-name> servlet-name子标签和servlet标签中的servlet-name名字要一样；

<url-pattern>/ccc2</url-pattern> url-pattern：访问此servlet虚拟路径

</servlet-mapping>

#### 继承HttpSrvlet

在实际开发中，我们不会直接去实现Servlet接口，因为那样需要覆盖的方法太多，继承HttpServlet

### Servlet生命周期

以前搞晕了，以为每个工程就一个servlet，其实是可以创建很多servlet，下面单指一个servlet

1）每一个Servlet何时创建对象

默认第一次请求访问某servlet时创建该servlet的对象

如果希望某一个Servlet类要在Tomca启动就创建：在web.xml中进行设置如下：

<servlet>

<servlet-name>a</servlet-name>

<servlet-class>wwl.MyServlet</servlet-class>

<init-param>

<param-name>configContextLocation</param-name>

<param-value>classpath:ni.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup> //为Servlet-name名称为a的设置成启动tomcat时候就创建该对象

</servlet>

2）每一个Servlet何时销毁

服务器关闭servlet就销毁了

3）每次访问必然执行的方法

service(ServletRequest req, ServletResponse res)方法

### Servlet的xml配置

xml中servlet基本配置

<servlet>

<servlet-name>b</servlet-name>

<servlet-class>wwl.MyServlet2</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>b</servlet-name>

<url-pattern>/ccc2</url-pattern>

</servlet-mapping>

访问虚拟路径配置：其中url-pattern的配置方式：

1）完全匹配 访问的资源与配置的资源完全相同才能访问到

<url-pattern>/ccc2</url-pattern>

2）目录匹配格式：/虚拟的目录../\* \*代表任意

<url-pattern>/aaa/bbb/\*</url-pattern>

3）扩展名匹配格式：\*.扩展名

<url-pattern>\*.abcd </url-pattern>

注意：第二种与第三种不要混用 /aaa/bbb/\*.abcd（错误的）

4）缺省的servlet 就配置一个 /

当访问的同工程下的资源地址所有servlet都不匹配的时候找缺省的servlet

注意：这样会造成静态资源都找不到：访问HTML都会匹配到你的缺省servlet中。原理：原来tomcat全局的web.xml配置了默认的缺省servlet用于找静态资源的

<url-pattern>/</url-pattern>

扩展小知识：静态资源tomcat是如何执行的？还有jsp？懂得这个就能明白为啥会静态资源找不到

1、先去自己配置的<servlet-mapping>中匹配，

2、没有就会去找tomcat全局的web.xml配置的默认servlet，此servlet会将静态html通过getWriter()通过流把内容逐句打印给浏览器

在tomcat中目录conf/下有全局的web.xml配置了默认的servlet对象用于寻找静态资源：

<servlet>

<servlet-name>default</servlet-name>

<servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class> //现在明白这个类的作用了

<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>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>/</url-pattern> 它这里也配置了：/

</servlet-mapping>

还有jsp的servlet处理器类，具体自己看全局web.xml:

<servlet-name>jsp</servlet-name>

<servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class> 这是处理jsp的servlet类

因为java就近原则：你也配置了缺省的servlet导致会跑到你配置的servlet中去，从而找不到静态资源

服务器启动实例化Servlet配置

Servlet的何时创建：默认第一次访问时创建

配置让servlet启动tomcat就初始化servlet对象

当在servlet的配置时 加上一个配置 <load-on-startup> servlet对象在服务器启动时就创建

## ServletContext

### ServletContext对象

ServletContext代表是一个web应用的环境（上下文）对象，ServletContext对象 内部封装是该web应用的信息，ServletContext对象一个web应用只有一个

ServletContext对象的生命周期

创建：该web应用被加载（服务器启动或发布web应用）

销毁：web应用被卸载（服务器关闭，移除该web应用）

怎样获得ServletContext对象

1）ServletContext servletContext = config.getServletContext();

2）ServletContext servletContext = this.getServletContext();//其实都一样只是HttpServlet父类封装了1方法

### ServletContext的作用

(1)获得web应用全局的初始化参数

在web.xml配置全局参数：

<context-param>

<param-name>driver</param-name>

<param-value>com.mysql.jdbc.Driver</param-value>

</context-param>

代码中通过context获得参数

ServletContext context = getServletContext();

String initParameter = context.getInitParameter("driver");

System.out.println(initParameter);

(2)获得web应用中任何资源的绝对路径

方法：String path = context.getRealPath(相对于该web应用的相对地址);

(3)ServletContext是一个域对象

什么是域对象？什么是域？存储数据的区域就是域对象

ServletContext域对象的作用范围：整个web应（所有的web资源都可以随意向 servletcontext域中存取数据，数据可以共享）

域对象的通用的方法：

setAtrribute(String name,Object obj); 通过key存储数据

getAttribute(String name); 通过key获取数据

removeAttribute(String name); 通过key 删除数据

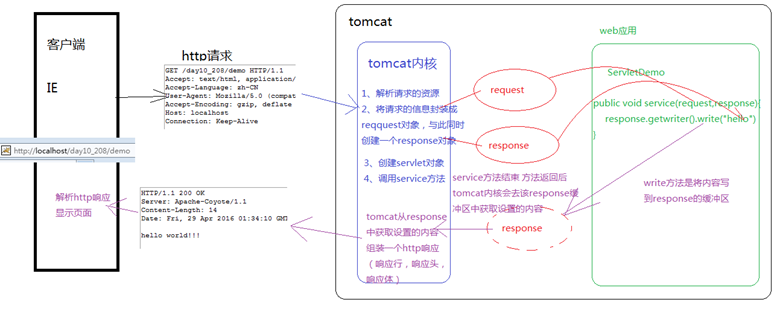
## response

HttpServletResponse概述

我们在创建Servlet时会覆盖service()方法，或doGet()/doPost(),这些方法都有两个参数，一个为代表请求的request和代表响应response。

service方法中的response的类型是ServletResponse，而doGet/doPost方法的response的类型是HttpServletResponse，HttpServletResponse是ServletResponse的子接口，功能和方法更加强大

### 运行流程



### 设置响应行

响应行 HTTP/1.1 200 OK

setStatus(int sc) 通过此方法就可以更改状态：（一般都不改，tomcat帮我们自动设置）

response.setStatus(404);//此时就把默认的访问成功的200设置成了404

### 设置响应头

响应头：

Content-Length 128

Content-Type text/html;charset=UTF-8

Date Wed, 26 Dec 2018 05:44:14 GMT

Server Apache-Coyote/1.1

响应头可以做很多功能：具体看[例子](#_例子)

设置响应头方法

1、addHeader(String name, String value) 同样的响应头会添加两个（可以自定义头名称）

response.addHeader("name", "abc");

response.addHeader("name", "def");

浏览器响应头就会添加两个：

name:abc

name:def

2、addDateHeader(String name, long date)

response.addDateHeader("datesds", new Date().getTime());

浏览器响应头就会添加响应头datesds：

datesds:Mon, 24 Dec 2018 09:34:40 GMT

3、setHeader(String name, String value) 同样的响应头会修改覆盖（可以自定义头名称）

response.setHeader("name", "abc");

response.setHeader("name", "def");

浏览器响应头就会添加两个：

name:def

4、setDateHeader(String name, long date) 修改响应头操作

其中，add表示添加，而set表示设置

### 设置响应体

响应体：response把整个内容通过getWriter()输出到浏览器显示

(1)响应体设置文本

**PrintWriter getWriter()** 获得字符流，通过字符流的write(String s)方法可以将字符串设置到response缓冲区中，随后Tomcat会将response缓冲区中的内容组装成Http响应返回给浏览器端。

//第一步 告诉服务器要发送的内容编码为utf-8 可以省略，下面第二步底层是有调用此行代码的

Response.setCharacterEncoding(“utf-8”);

//第二步 告诉浏览器用utf-8接收

response.setHeader("Content-Type", "text/html;charset=utf-8");//第一种：古老的

Response.setContentType(“text/html;charset=utf-8”);//注意此底层其实是有调用Response.setCharacterEncoding(“utf-8”);的

//第三步 发送信息给浏览器

Response.getWrite().write(‘浏览器你好啊，我是服务器！’);

---------------------------------------------------------关于设置中文的乱码问题------------------------------------------------------------------------------

原因：response缓冲区的默认编码是iso8859-1，此码表中没有中文，可以通过response的setCharacterEncoding(String charset) 设置response的编码

但我们发现客户端还是不能正常显示文字:

原因：我们将response缓冲区的编码设置成UTF-8，但浏览器的默认编码是本地系统的编码，因为我们都是中文系统，所以客户端浏览器的默认编码是GBK，我们可以手动修改浏览器的编码是UTF-8。

我们还可以在代码中指定浏览器解析页面的编码方式，

通过response的setContentType(String type)方法指定页面解析时的编码是UTF-8

response.setContentType("text/html;charset=UTF-8");

上面的代码不仅可以指定浏览器解析页面时的编码，同时也内含setCharacterEncoding的功能，所以在实际开发中只要编写response.setContentType("text/html;charset=UTF-8");就可以解决页面输出中文乱码问题。

(2)响应体设置字节：如图片

**ServletOutputStream getOutputStream()**

获得字节流，通过该字节流的write(byte[] bytes)可以向response缓冲区中写入字 节，在由Tomcat服务器将字节内容组成Http响应返回给浏览器。

注意：getWriter()和getOutputStream()不能同时使用，不然报错

### 例子

#### 简单入门：向浏览器发送信息

//第一步 告诉服务器要发送的内容编码为utf-8 可以省略，下面第二步底层是有调用此行代码的

Response.setCharacterEncoding(“utf-8”);//注意request也有这个方法：用来得到的请求参数转为utf-8

//第二步 告诉浏览器用utf-8接收

response.setHeader("Content-Type", "text/html;charset=utf-8");//第一种：古老的

Response.setContentType(“text/html;charset=utf-8”);//注意此底层其实是有调用Response.setCharacterEncoding(“utf-8”);的

//第三步 发送信息给浏览器

Response.getWrite().write(‘浏览器你好啊，我是服务器！’);

#### 请求重定向

redirect是服务端根据逻辑,发送一个状态码,告诉浏览器重新去请求那个地址.所以地址栏显示的是新的URL.所以redirect等于客户端向服务器端发出两次不同request对象，同时也接受两次response

Response.setStatus(302);//设置状态302

Response.setHeader(“Location”,”/wwl/a.jsp”);

Response.sendRedirect（”/wwl/a.jsp”）;//底层帮你自动设置响应码，和响应头

#### 实现文件下载

希望浏览器不要解析显示：设置响应头 Content-Disposition

//第一步 设置响应头，使用url编码解决中文不显示问题，浏览器只认UTF-8，浏览器会自动解码成utf-8字符

response.setHeader("Content-Disposition", "attachment;filename="+URLEncoder.encode("美女.jpg", "utf-8"));

//第二步 设置输入流（指定服务器中需要下载的文件）和输出流

InputStream in = new FileInputStream(this.getServletContext().getRealPath("1.jpg"));//单1.jpg是在tomcat的bin目录找

OutputStream out = response.getOutputStream();

byte[] bs = new byte[1024];

int i = 0;

while((i=in.read(bs))!=-1){

out.write(bs,0,i);

}

in.close();//输入流需要关闭，输出流是通过响应response拿到的不需要关闭

#### 实现定时刷新页面

设置响应头refresh

Response.setHeader(“Refresh”,”1”);//每一秒刷新本页面

Response.setHeader(“refresh”,”3;url=”/wwl/.a.jsp”);//3 秒后跳转到阿 a.jsp

在HTML可以利用<meta http-equiv= "" content="">标签模拟响应头的功能

#### 控制浏览器是否缓存

浏览器默认是会话级别的缓存（关闭浏览器就清空）

//不缓存操作

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

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

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

保存一个月:必须是long型：现在算法已经超出了int的范围会变成负数，当前时间进行加负数就会变成以前时间

response.setDateHeader("Expires", System.currentTimeMillis()+100l\*60\*60\*24\*30);

#### 验证码

Java代码部分：缓存问题要注意，使用万国码找单个字

randNum方法返回int（int 开始,和结束）：第五步画随机直线和第六步的随机字体颜色、随机万国码的字需要用到

//随机生成一个x轴或y轴， 此方法用在第五步直线drawLine参数中

public static int randNum(int begin,int end) {

Random r =new Random();

//begin 到 end之间的值操作方式，Random类没有提供

return r.nextInt(end-begin)+begin;

}

第一步 在内存中构建一张图片

int height=30;

int width=90;

//参数：宽，高，（静态像素老师选择最差的）8位RGB颜色分量的整数像素的图像，

BufferedImage img= new BufferedImage(width, height,BufferedImage.TYPE\_INT\_RGB);

第二步 就像Windows画图工具一样：获取图像上的画布

Graphics2D g = (Graphics2D) img.getGraphics();

第三步 设置此画布的背景色 此步就会在页面形成长方形了

g.setColor(Color.GRAY);//设置颜色

//此方法是填充整个画布的方法：x坐标、y坐标、宽度、高度

g.fillRect(0, 0, width, height);

第四步 （可以省略）设置一个边框

g.setColor(Color.BLUE);

//此方法不填充只是画四周边缘

g.drawRect(0, 0, width-1, height-1);

第五步（可以省略）画干扰线让用户较难辨认，x轴和y轴构建一个点，两点构建一条直线

g.setColor(Color.RED);

//画直线 x轴和y轴决定一个点，两点决定一条直线 (int x1,int y1,int x2,int y2)

g.drawLine(0, 0, width, height);//x坐标：0，y坐标：0固定一个点在画布的起始位置

使用到RandNum方法操作

//g.drawLine(randNum(0, width), randNum(0, height), width, height);

第六步 写字 字符串是万国码unicode格式万国码进行操作

String base ="\u7684\u4e86\u662f\u6211\u4e00";//unicode码,万国码一部分

//for循环获得4个字：因为给你的string是一个字的，

for (int i = 0; i < 4; i++) {

//让颜色随机：Color构造方法参数：红、绿、蓝

g.setColor(new Color(randNum(0, 255), randNum(0, 255), randNum(0, 255)));

g.setFont(new Font("黑体",Font.BOLD,10));//设置字体 构造方法参数：名称、样式、大小

//参数:值、宽度、高度

g.drawString(base.charAt(randNum(0, base.length()-1))+"", 5+(20\*i), 22);

//字体旋转这里就不写了

}

第七步 让浏览器不要进行缓存，不然刷新不了永远都是那一张

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

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

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

第八步 将图片输出到浏览器、或者指定路径

ImageIO.write(img, "jpg",response.getOutputStream());//输出到浏览器上

ImageIO.write(img, "jpg", new FileOutputStream("D:\\afg.jpg"));//指定位置

}

JSP页面 使用IMG标签 javascript代码中内容不一致了才会有效果

Javascript点击更换验证码操作

<script type="text/javascript">

function changeImg(img){ //需要加上时间确保内容不一样，和ajax效果一样

img.src="/Day04/servlet/ValiImg?time="+new Date().getTime();

}

</script>

</head>

<body>

<form action="#" method="POST">

用户名:<input type="text" name="username"/>

密码:<input type="password" name="password"/>

验证码:<input type="text" name="valistr"/>

<img src="/Day04/servlet/ValiImg" style="cursor: pointer;" onclick="changeImg(this)"/>

<input type="submit" value="注册" />

</form>

</body>

## request

request对象封装了http协议(请求行，请求头，请求体)内容和客户机信息(IP)

### 获得请求行和客户机信息

请求行： GET /wwl/a.jsp HTTP/1.1

1获取客户机的请求方式

String method=Request.getMethod();//打印 GET

2获取客户端URI（资源部分名称）

Request.getRequestURI(); 打印：/wwl/a.jsp

客户机信息

1获取客户端请求的完整URL

Request.getRequestURL();打印：http://localhost/wwl/a.jsp

2获取请求行中参数部分

String qstr=Request.getQueryString();打印：name=wenwenliang&age=199

3获取返回发出请求的客户机的IP地址

String ip=Reuqest.getRemoteAddr();//打印：127.0.0.1 192.168.1.220

4获取当前WEB应用的名称

Request.getContextPath();// 打印 /wwl

### 获得请求头

请求头：可翻看[HTTP协议](#_请求头)

功能：可以设置请求头做一些功能：如防盗链、文件上传等，请翻转[例子](#_例子_1)

1、String getHeader(String name) 获取指定key请求头的值

2、Enumeration getHeaderNames() 获取所有的key返回枚举类型

3、Enumeration getHeaders(String name) 获取指定key的所有值(请求头有可能有两个)

4、long getDateHeader(String name) 获取时间的响应头一般不用

5、int getIntHeader(String name)

### 获得请求体

请求体中的内容就是提交post提交的参数如：

username=zhangsan&password=123&hobby=football&hobby=basketball

以上面参数为例，通过以下方法获得请求GRT和POST参数：

String getParameter(String name) 获得指定单个key

String[] getParameterValues(String name) 获得key中所有的值(有时候不止一个值)

Enumeration getParameterNames() 获得所有key

Map<String,String[]> getParameterMap() 获得所有的key和val组成map

解决post提交方式的乱码：request.setCharacterEncoding("UTF-8");

解决get提交的方式的乱码：parameter = new String(parameter.getbytes("iso8859-1"),"utf-8");

### 域和转发

(1)request是一个域对象

request对象也是一个存储数据的区域对象，所以也具有如下方法：

setAttribute(String name, Object o)

getAttribute(String name)

removeAttribute(String name)

注意：request域的作用范围：一次请求中

(2)request完成请求转发

获得请求转发器----path是转发的地址

RequestDispatcher getRequestDispatcher(String path)

通过转发器对象转发

requestDispathcer.forward(ServletRequest request, ServletResponse response)

注意：转发与重定向的区别？

1）重定向两次请求，转发一次请求

2）重定向地址栏的地址变化，转发地址不变

3）重新定向可以访问外部网站 转发只能访问内部资源

4）转发的性能要优于重定向

注意：客户端地址与服务器端地址的写法？

客户端地址：是客户端去访问服务器的地址，服务器外部的地址，特点：需写上web应用名称

服务器端地址：服务器内部资源的跳转的地址，特点：不需要写web应用的名称。

### 生命周期

注意：ServletContext域与Request域的生命周期比较？

ServletContext：

创建：服务器启动

销毁：服务器关闭

域的作用范围：整个web应用

request：

创建：访问时创建request

销毁：响应结束request销毁

域的作用范围：一次请求中

### 例子

#### 防盗链

Referer请求头：http://localhost:8080/wwl/b.jsp:如果有这个头证明：本网页是从：b.jsp页面跳转过来的

注意：点击a标签超链接跳转：会有referer请求头信息。其它像服务器跳转不会有此referer头的

public void doGet(HttpServletRequest request, HttpServletResponse responsethrows ServletException, IOException {

response.setContentType("text/html;charset=utf-8");

String ref = request.getHeader("Referer");//第一步 获取到此头内容

if(ref==null || "".equals(ref) || !ref.startsWith("http://localhost")){ //第二步条件判断

response.sendRedirect(request.getContextPath()+"/index.html");

return; //第三步结束此方法

}

response.getWriter().write("凤姐独家回忆..我在黑马的日子..收货了很多很多......");

}

#### 文件上传

Servlet3.0技术

Servlet3.0 与 Servlet2.5：

\* Servlet3.0需要运行在tomcat7以上的服务器中.

\* Servlet3.0以后web.xml就不是必须的.

Servlet3.0支持注解开发.

支持文件上传.

文件上传的抓包分析：

已经修改了enctype属性（部分）:

User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko

Content-Type: multipart/form-data; boundary=---------------------------7e02e526160b66

Accept-Encoding: gzip, deflate

Host: localhost:8080

Content-Length: 224

Connection: Keep-Alive

Cache-Control: no-cache

Cookie: JSESSIONID=99CD51DA9A47D29200168968AD983E9E

-----------------------------7e02e526160b66

Content-Disposition: form-data; name="upload"; filename="C:\Users\apple\Desktop\aaa.txt"

Content-Type: text/plain

Hello shouyi

-----------------------------7e02e526160b66—

【文件上传的技术】

JspSmartUpload: jspSmartUpload组件是应用JSP进行[B/S](http://baike.baidu.com/subview/8039/8039.htm)程序开发过程中经常使用的上传下载组件，它使用简单，方便。现在我又为其加上了下载中文名字的文件的支持，真个是如虎添翼，必将赢得更多开发者的青睐。-Model1年代的文件上传的工具.

FileUpload 是 Apache commons下面的一个子项目，用来实现[Java](http://baike.baidu.com/subview/29/12654100.htm)环境下面的文件上传功能，与常见的SmartUpload齐名.应用在Model2年代了.

Servlet3.0 :

Struts2 :

开发步骤 ①制作JSP页面②开发servlet类

JSP页面：

1.form表单的提交的方式必须是POST、 enctype属性的值multipart/form-data

2.文件上传项:<input type=”file”>,而且文件上传项必须有name属性和值.

\* <input type=”file” name=”upload”/>

代码的操作：

@WebServlet("/UploadServlet")

第一步：@MultipartConfig

**public** **class** UploadServlet **extends** HttpServlet {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

// 接收普通项:

request.setCharacterEncoding("UTF-8");

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

System.***out***.println("文件描述:"+desc);

第二步：通过请求获得Part对象：上传文件的所有内容

Part part = request.getPart("upload");

**long** size = part.getSize();// 获得上传的文件的大小

String type = part.getContentType();

System.***out***.println("文件类型"+type); // text/plain image/jpeg

第三步：获取文件名：Content-Disposition: form-data; name="file"; filename="er.jpg"

String header = part.getHeader("Content-Disposition");

int index=fileName.lastIndexOf("=");

String name=fileName.substring(index+2,fileName.length()-1)

第四步：指定文件要存储的路径

String diskpath=this.getServletContext().getRealPath("/WEB-INF/upLoad");

//UUID配合hashCode创建目录

String rName=UUID.randomUUID().toString()+"\_"+fileName;

String hasName=Integer.toHexString(rName.hashCode());

for(char c :hasName.toCharArray()) {

diskpath+="/"+c;

}

new File(diskpath).mkdirs();

第五步:得到上传文件数据输入流，通过输出流输出到指定位置

InputStream input=part.getInputStream();

OutputStream out= new FileOutputStream(new File(diskpath, rName));

byte[] b= new byte[1024];

int i=0;

while((i=input.read(b)) != -1) {

out.write(b, 0, i);

}

out.close();

input.close(); }

总结:

文件名重名的问题： UUID随机产生一个文件名.

文件上传的目录分离:

按用户分 ：一个用户创建一个或多个路径.

按时间分 ：按月，星期，天进行划分.

按个数分 ：一个路径中存3000个文件.

按分离算法分 ：按照一定的算法进行划分.

## 会话

会话技术简介

1．存储客户端的状态

由一个问题引出今天的内容，例如网站的购物系统，用户将购买的商品信息存储到哪 里？因为Http协议是无状态的，也就是说每个客户访问服务器端资源时，服务器并 不知道该客户端是谁，所以需要会话技术识别客户端的状态。会话技术是帮助服务器 记住客户端状态（区分客户端）

2．会话技术

从打开一个浏览器访问某个站点，到关闭这个浏览器的整个过程，成为一次会话。会 话技术就是记录这次会话中客户端的状态与数据的。

会话技术分为Cookie和Session：

Cookie：数据存储在客户端本地，减少服务器端的存储的压力，安全性不好，客户端 可以清除cookie

Session：将数据存储到服务器端，安全性相对好，增加服务器的压力

### Cookie

什么是Cookie？

数据存储在客户端本地，减少服务器端的存储的压力，安全性不好，客户端可以清除cookie

1请求头的Cookie (不显示时间和路径的)

Cookie:name=zhangsan

2响应头的Cookie

Set-Cookie name=zhangsan; Expires=Wed, 26-Dec-2018 07:21:37 GMT; Path=/ServletDemo

name：cookie的名称。自定义

Expires：( 到期、有效期):持久化的事件

Path：需要携带此cookie的url路径

3注意不管有没有设置(response.setHeader("Set-Cookie"))，访问jsp页面：

响应头：访问jsp页面会有Set-Cookie名称 JSESSOONID响应浏览器：jsp默认发送session，可以改为false

请求头：请求收到信息：第一次不会有cookie信息

#### 1.服务器端向客户端发送Cookie

##### 1）创建Cookie

Cookie cookie = new Cookie(String cookieName,String cookieValue);

示例：

Cookie cookie = new Cookie("username"，"zhangsan");注意：Cookie中不能存储中文

那么该cookie会以响应头的形式发送给客户端：Cookie:”name=zhangsan”

##### 2）设置Cookie在客户端的持久化时间

cookie.setMaxAge(int seconds); ---时间秒

注意：如果不设置持久化时间，cookie会存储在浏览器的内存中，浏览器关闭cookie信息销毁（会话级别的cookie），如果设置持久化时间，cookie信息会被持久化到浏览器的磁盘文件里

示例：

cookie.setMaxAge(10\*60);

设置cookie信息在浏览器的磁盘文件中存储的时间是10分钟，过期浏览器 自动删除该cookie信息

##### 3）设置Cookie的携带路径

cookie.setPath(String path);

注意 ：没有设置路径都是在发送 cookie的所在的路径或子路径Url才会带着Cookie过来

示例：

如果不设置路径:如你的servlet在：/WEB16/demo/cookieServlet:在demo下的子路径或孙路径都会携带cookie信息

cookie.setPath("/WEB16");代表访问WEB16应用中的任何资源都携带cookie

cookie.setPath("/WEB16/cookieServlet");代表访问WEB16中的cookieServlet才携带cookie信息

##### 4）向客户端发送cookie

response.addCookie(Cookie cookie);

##### 5）删除客户端的cookie

删除cookie：需要名字、path、Domain(一般都不用了)、3样都要一致

Cookie c = new Cookie("wenwenliang","wwl");

c.setMaxAge(0); 设置为0；覆盖掉之前的cookie

c.setPath(request.getContextPath());//设置Url根路经，整个web应用都会带着cookie信息过来

response.addCookie(c);

##### 使用响应头操作

设置响应头：Set-Cookie 请求头 Cookie

一 浏览器第一次访问到此代码response.setHeader("Set-Cookie", "wenwenliang=eee");

二 打开浏览器的请求信息和响应信息：响应信息会有Set-Cookie：wenwenliang= eee，请求头无cookie信息

三 如果再次访问带有设置Cookie的servlet会进行覆盖原来相同名字的cookie

#### 2.服务器端接收客户端的Cookie

第一种 获取请求头，然后可以通过字符串截取第一个”=”

System.out.println(request.getHeader("Cookie")); //打印wenwenliang=%E6%88%91%E6%98%AF%E8%B0%81;

第二种 sun公司封装好的request获取Cookie数组

Cookie[] cookie=request.getCookies();

for(Cookie c : cookie) {

if("wenwenliang".equals(c.getName())) {

System.out.println(URLDecoder.decode(c.getValue(), "utf-8"));//URL解码 ：打印：我是谁

}

#### 例子

##### 中文操作

第一步 创建对象，参数：第一个是Cookie名字，第二个是值：用中文使用URL编码

Cookie c = new Cookie("wenwenliang",URLEncoder.encode("我是谁", "utf-8"));

第二步 使用response发送Cookie信息 参数是cookie对象

response.addCookie(c);

##### 显示上次用户访问时间

第一步 把cookie值使用当前系统时间的毫秒值字符串进行存储

Cookie c = new Cookie("wenwenliang",new Date().getTime()+"");

response.addCookie(c);

第二步 在另一个servlet程序中获取这个cookie信息，进行判断获取此毫秒值进行转换为时间

Cookie[] cookie=request.getCookies();

Cookie cookie2=null; //用来接收wenwenliang的cookie信息

if(cookie !=null) {

for(Cookie c : cookie) {

if("wenwenliang".equals(c.getName())) {

cookie2 =c;

}

}

}

response.setContentType("text/html;charset=utf-8");

if(cookie2 ==null) { //判断是否为null

response.getWriter().write("你是第一次访问本网站");

}else {

response.getWriter().write("你上次的访问时间是："+

new Date(Long.parseLong(cookie2.getValue())).toLocaleString());

}

第一：保存Cookie信息一个月；第二:浏览器访问服务器带着cookie信息的条件

Cookie c = new Cookie("wenwenliang","wwl");

c.setMaxAge(3600\*24\*30);//设置保存一个月

response.addCookie(c); //浏览器响应头：Set-Cookie:wenwenliang=wwl;Expires=Thu,26-Jul-2018 03:08:06 GMT ；请求头无变化

注意 ：没有设置路径都是在发送 cookie的所在的路径或子路径Url才会带着Cookie过来

如 http://localhost:8080/Test/wwl/Test此网页是发送cookie信息的路径：wwl/以后的才会带着cookie

第一：http://localhost:8080/Test/wwl这个路径是在wwl下，会带着请求cookie信息过来

而http://localhost:8080/Test/t这个路径不在wwl下，不会带着请求cookie信息过来

设置为web服务器url根路径：就可以全部都会带cookie信息

Cookie c = new Cookie("wenwenliang","wwl");

c.setMaxAge(3600\*24\*30); //设置保存一个月

c.setPath(request.getContextPath());//设置Url根路经，整个web应用都会带着cookie信息过来

response.addCookie(c);//浏览器响应头：Set-Cookie:wenwenliang=wwl;Expires=Thu,26-Jul-2018 03:08:06 GMT ；Path=/Test

设置Domain现代浏览器不会接收：参数用来放域名：www.baidu.com

### Session

什么是session？

基于cookie技术的，借助cookie存储JSESSIONID将数据存储到服务器端，安全性相对好，增加服务器的压力

1、请求头的session (不显示时间和路径的)

Cookie:JSESSIONID=ED7A92C20096A2BC1EBB460A08B2D35D

2、响应头的session

Set-Cookie JSESSIONID=SDFSGFD345; Expires=Wed, 26-Dec-2018 07:21:37 GMT; Path=/ServletDemo

JSESSIONID: cookie中专属名称。

Expires( 到期、有效期):持久化的事件

Path: 需要携带此cookie的url路径

3注意不管有没有设置(response.setHeader("Set-Cookie"))，访问jsp页面：

响应头：访问jsp页面会有Set-Cookie名称 JSESSOONID响应浏览器：jsp默认发送session，可以改为false

请求头：请求收到信息：第一次不会有cookie信息

#### 1．获得Session对象

HttpSession session = request.getSession(); 值永远不会为null:

该方法会智能的判断客户端有无带：session的cookie，有直接将Session返回，无自动创建一个

此方法会获得专属于当前会话的Session对象，如果服务器端没有该会话的Session对象会创建一个新的Session返回，如果已经有了属于该会话的Session直接将已有的Session返回（实质就是根据JSESSIONID判断该客户端是否在服务器上已经存在session了）

#### 2．怎样向session中存取数据（session也是一个域对象）

Session也是存储数据的区域对象，所以session对象也具有如下三个方法：

session.setAttribute(String name,Object obj);

session.getAttribute(String name);

session.removeAttribute(String name);

#### 3．Session对象的生命周期（面试题/笔试题）

创建：第一次执行request.getSession()时创建

销毁：

1）服务器（非正常）关闭时：正常关闭:会有钝化和活化

2）session过期/失效（默认30分钟）：可以在web.xml配置，tomgcat全局web.xml配置就是30分钟

web.xml中进行配置

<session-config>

<session-timeout>30</session-timeout>

</session-config>

3）手动销毁session

session.invalidate();

问题：时间的起算点 从何时开始计算30分钟？

从不操作服务器端的资源开始计时

作用范围：

默认在一次会话中，也就是说在，一次会话中任何资源公用一个session对象

面试题：浏览器关闭，session就销毁了？ 不对

因为：浏览器关闭只是把自己的session清空，session是在服务器端开辟了一个专属的空间：再打开等同于是新的客户端：进行访问带有request.getSerssion()的servlet获取新的session。不是之前的session，

这时候销毁：默认是30分钟进行自动销毁

#### 4. 钝化和活化

session的钝化和活化 条件：①需要tamcat正常关闭②有未过时session对象

需要①正常关闭服务器②session还没过时：称为钝化

D:\apache-tomcat-7.0.78\work\Catalina\localhost\Test目录下有文件SESSIONS.ser

服务器开启此工程如有此文件SESSIONS.ser会把它加载到内存中 同时此文件消失

注意 对象的类（实体类）如果要被钝化必须系列化：实体类需实现Serializable

#### 5、其它（了解）

简单入门 浏览器不会主动创建session

第一步 当浏览器第一次访问到有Request.getSeeion();代码//浏览器响应头会有Set-cookie：JSESSIONID和独一无二的值

Set –Cookie：JSESSIONID=FF95C76911632721BD4BF886B233F0C1;Path=/Test/;HttpOnly

第二步 当第二次访问到有Request.getSeeion();代码，浏览器响应头不会再有Set-Cookie

因为服务器是先判断请求头有无JSESSIONID名称的Cookie，有：不再此创建session，无：进行创建

Session\_URL重写：被设置成浏览器拒绝接受Cookie（了解即可）

对Url路径进行改造：

Request.getSession(); 先有此代码 此方法支持URL重写带session

String url1 = request.getContextPath()+”/wwl/t?name=电视”;

Response.encodeURL(url1); //把路径放入此方法中即可，了解即可，很麻烦，要把全部URL进行改造

<a href=”/Test/Test/t;jsessionid=B81EB72CF092436462K?name=电视”>电视机</a> 看源码会发现拼接了session信息

#### 例子

##### 保存Session信息一个月

因为session就是基于cookie技术的：它的名字是JSESSIONID，需要同名、同path才能进行设置

HttpSession session=request.getSession(); 第一步 创建session对象

session.setAttribute("A", "B");

Cookie c = new Cookie("JSESSIONID",session.getId()); 第二步 name要和session名字一样，它的值通过getId()获得

c.setMaxAge(60\*60\*30\*24); 第三步 设置时间为30天

c.setPath(request.getContextPath()); 第四步 设置路径上下文路径，因为session就是上下文路径

response.addCookie(c); 第五步 发送cookie信息

响应头变成带时间：Set-Cookie: JSESSIONID=FF95C76911632721BD4BF886B233F0C1;26-Jul-2018 03:08:06 GMT ;Pth=/Test

## listener

javaEE包括13门规范主要学习 servlet技术 和 jsp技术

其中 servlet规范包括三个技术点：servlet listener filter

web项目启动顺序：监听器-->过滤器-->Servlet.

项目启动后,容器会首先创建声明的各种监听器,为后继的各个事件监听做准备,然后创建过滤器,最后是Servlet.

项目销毁：过滤器-->Servlet-->监听器

什么是监听器？

监听器就是监听某个对象的的状态变化的组件

有8大监听器

### 简单入门

**第一步** 创建类实现监听器接口类；**第二步** ①web.xml进行注册或 ②注解形式@WebListener

### 监听三大对象

监听三大作用域的创建和销毁监听器：请求对象、会话对象、全局对象

1、ServletContextListener 监听WEB工程的启动和关闭销毁

@WebListener //注解形式操作

public class TestListen implements ServletContextListener{

public void contextDestroyed(ServletContextEvent arg0) { //

System.out.println("销毁了"+arg0.getServletContext());

}

public void contextInitialized(ServletContextEvent arg0) { //web工程启动的期间：调用初始化方法

System.out.println("创建了"+arg0.getServletContext());

}

当启动web工程加载完成创建了当前SetvletContext对象后立即调用了初始化方法

六月 28, 2018 10:16:26 上午 org.apache.catalina.startup.TaglibUriRule body

信息: TLD skipped. URI: http://java.sun.com/jsp/jstl/xml is already defined

创建了org.apache.catalina.core.ApplicationContextFacade@604da796

六月 28, 2018 10:16:26 上午 org.apache.catalina.core.StandardContext reload

信息: Reloading Context with name [/Test] is completed

2、HttpSessionListener 三种死法①默认30分钟未使用②调用invalidate()③未正常关闭工程

第一步 第一次触发了request.getSession();第二次是不会再触发下面方法，因为已经有session对象了，不会再创建

public class TestListen implements ServletContextListenerrequest{

public void sessionCreated(HttpSessionEvent arg0) {

System.out.println("创建了session对象");

}

第二步 ①默认30分钟未使用②调用invalidate()③未正常关闭工程 都会触发下面的方法

public void sessionDestroyed(HttpSessionEvent arg0) {

System.out.println("销毁了session对象");

}

3、ServletRequestListener 监听ServletRequest创建和销毁

注意 请求的创建和销毁之间是很快的：一打开servlet类就直接两个都打印出到控制台了

public void requestDestroyed(ServletRequestEvent arg0) {

System.out.println("销毁了request对象");

}

public void requestInitialized(ServletRequestEvent arg0) {

System.out.println("创建了request对象");

} 控制台同时打印了上面两句话

### 监听三大对象域

监听三大对象域：和上面的几乎一致 3个监听器接口

**1、**ServletContextAttributeListener**2、**HttpSessionAttributeListener、**3、**ServletRequestAttributeListener

它们3个都有三个方法：**加入、更改、移除**，这三大监听器就不写了

### 监听实体类会话域状态

Javabean(实体类)感知自己在Session中状态变化的最后两个监听器

触发条件 ①被绑定到session②被移除session③随着session被钝化④随着session被活化

这两个监听器不用在web.xml注册，只需实体类实现这两个接口即可

绑定和移除session接口监听器：HttpSessionBindingListener

活化和钝化session接口接口监听器：HttpSessionActivationListener

public class T implements HttpSessionBindingListener,HttpSessionActivationListener,Serializable{

private Integer id;

public Integer getId() {return id; }

public void setId(Integer id) {this.id = id; }

public void valueBound(HttpSessionBindingEvent arg0) {

System.out.println("我被绑定到session了");

}

public void valueUnbound(HttpSessionBindingEvent arg0) {

System.out.println("我被移除了session");

}

public void sessionDidActivate(HttpSessionEvent arg0) {

System.out.println("我活化了");

}

public void sessionWillPassivate(HttpSessionEvent arg0) {

System.out.println("我钝化了");

}

用户登录列表踢人案例 注意 ①hashMap存储机制②使用了两个监听器对象

第一步 自定义类实现ServletContextListener 监听器进行域属性添加：msp<String, HttpSession>集合

….省略了一些代码；

public void contextInitialized(ServletContextEvent sce) {

ServletContext context= sce.getServletContext(); //获得ServletContext对象

context.setAttribute("usermap", new LinkedHashMap<Integer,HttpSession>());//把空Map存进application属性域中

第二步 登录操作：通过数据库查询出来的对象存进session后再存储到第一步map中，实体类有绑定监听器看第三步

//这里需要注意如果相同用户登录：需要顶掉登录相同用户

ServletContext context=this.getServletContext(); **//获得servletContext对象**

Map<String,HttpSession> map=(Map<String, HttpSession>) context.getAttribute("usermap"); **//获取存到域中的空map**

HttpSession session=map.get(u.getName());  **//这步是查询map中有没有这个值，如有：进行移除操作**

if(session != null) {

session.invalidate(); **//一旦移除会触发另一个监听器 HttpSessionBindingListener**

}

request.getSession().setAttribute("user", u); //一旦添加实体类触发：实体类绑定监听器中的方法看第三步

response.sendRedirect(request.getContextPath()+"/b.jsp");

第三步 实体类中实现了HttpSessionBindingListener监听器：一旦移除、绑定session触发事件

用户绑定到属性session中：request.getSession().setAttribute("user", u)触发此方法

public void valueBound(HttpSessionBindingEvent event) {

HttpSession session=event.getSession(); //第一步 获取session对象

ServletContext servlet=session.getServletContext(); //第二步 通过session获取ServletContext对象

Map<String,HttpSession> map=(Map<String, HttpSession>) servlet.getAttribute("usermap"); //第三步 获取第一步中map对象

map.put(this.getName(), session); //第四步 存入session对象：当前用户直接this.名字

}

用户从session中移除：session.invalidate()此代码会触发此方法、超时下线

public void valueUnbound(HttpSessionBindingEvent event) {

HttpSession session=event.getSession();

ServletContext servlet=session.getServletContext();

Map<String,HttpSession> map=(Map<String, HttpSession>) servlet.getAttribute("usermap");

map.remove(this.getName()); //移除当前对象的名字

}

第四步 踢人操作 页面获取到name值，获取此name的HttpSession值，再进行杀死session，触发销毁监听方法

String s=request.getParameter("name");

byte[] b1=s.getBytes("iso8859-1");

String name =new String(b1, "utf-8");

Map<Integer,HttpSession> map=(Map<Integer, HttpSession>) request.getServletContext().getAttribute("usermap");

HttpSession session=map.get(name);

if(session != null) {

session.invalidate();

}

response.sendRedirect("userList.jsp");

## filter

javaEE包括13门规范主要学习 servlet技术 和 jsp技术

其中 servlet规范包括三个技术点：servlet listener filter

web项目启动顺序：监听器-->过滤器-->Servlet.

项目启动后,容器会首先创建声明的各种监听器,为后继的各个事件监听做准备,然后创建过滤器,最后是Servlet.

项目销毁：过滤器-->Servlet-->监听器

### 简单入门

第一步 创建类实现接口Filter类

@WebFilter("/\*") //注解直接在这里写/\*拦截URL工程下得所有路径

public class TestFilter implements Filter{

public void destroy() {}

public void doFilter(ServletRequest arg0, ServletResponse arg1, FilterChain arg2)throws IOException, ServletException {

System.out.println("被我拦截了");

arg2.doFilter(arg0, arg1);//经测试没有此句话进来都是空白的网页，不显示内容也不报错，控制台打印”被我拦截了”

}

public void init(FilterConfig arg0) throws ServletException {

System.out.println("我是过滤器");}

第二步 web.xml配置 一个Filter可以多个filter-mapping、多个url-partten

<filter-mapping>

<filter-name>Demo1</filter-name> //跟Fileter标签相同的名字

**第一种** 一个Filtermapping中可以配置多个url-partten,这个url-partten的写法和servlet-mapping中的写法相同

<url-pattern>/servlet/A</url-pattern> //指定单独 A类

<url-pattern>/servlet/\*</url-pattern> //指定servlet/路径下得所有类

<url-pattern>/\*</url-pattern> //工程下得所有

<url-pattern>\*.do</url-pattern> //后缀是do的类

**第二种**  配置多个servlet-name,其中填入servlet的名字明确的通知要拦截哪个名字的Servlet

<servlet-name>Demo3Servlet</servlet-name>

**第三种** 指定拦截REQUEST、FORWARD、INCLUDE、ERROR四个值,可以配置多个dispatcher,如果不配默认是REQUEST

<dispatcher>REQUEST</dispatcher>

<dispatcher>FORWARD</dispatcher>

</filter-mapping>

测试 结果：过滤器只认URL路径 只要URL符合要求，不管路径是否正确都会拦截，

第一种 /\* 代表工程名下URL路径全部拦截，只认URL路径，不管此路径对不对；如工程Test：

http://localhost:8080/Test/会拦截，不管有没有资源都会执行doFilter方法

http://localhost:8080/login.jsp 不会拦截，不是Test路径下的文件

第二种 测试默认配置：转发不会拦截，因为默认只是request请求的拦截（重定向也会拦截）

request.getRequestDispatcher("/login.jsp").forward(request, response);//打印一次：我被调用了，（注意请求servlet需要一次）

response.sendRedirect("/Test/login.jsp");//打印两次：我被调用了

第三种 arg2.doFilter(arg0, arg1);方法等于就是进行调用另一个方法，另一个方法执行完，再来执行后续的代码

测试：两个拦截器执行arg2.doFilter(arg0, arg1)前后打印A、B和C、D；最后访问的资源：打印”资源”

public void doFilter(ServletRequest arg0, ServletResponse arg1, FilterChain arg2) throws IOException, ServletException {

System.out.println("A");

arg2.doFilter(arg0, arg1);

System.out.println("B");

}//两个Filter一样的操作另一个是：C、D进行打印

执行方法后打印顺序是：A、C、资源、D、B // 等同于方法调用了另一个方法，另一个方法之行完了回来再执行下面代码

### 过滤条件设置

1、访问资源有4种：请求、重定向、转发、错误

dispatcher：访问的方式(了解)

REQUEST：默认值，代表直接访问某个资源时执行filter

FORWARD：转发时才执行filter

INCLUDE: 包含资源时执行filter

ERROR：发生错误时 进行跳转是执行filter

2、URL路径的配置

1）完全匹配 /sertvle1

2）目录匹配 /aaa/bbb/\* ----最多的

/user/\*：访问前台的资源进入此过滤器

/admin/\*：访问后台的资源时执行此过滤器

3）扩展名匹配 \*.abc \*.jsp

4)）指定servlet的名称配置

<filter>

<filter-name>abc</filter-name>

<filter-class>wwl.filter.MyFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>abc</filter-name>

<servlet-name>a</servlet-name> //指定要拦截的servlet

<url-pattern>/\*</url-pattern> //拦截所有web资源

<dispatcher>INCLUDE</dispatcher>//指定访问途径是什么操作？请求还是重定向等

</filter-mapping>

### 生命周期

public class MyFilter implements Filter { 3个方法的调用

//1、web工程启动的期间创建

public void init(FilterConfig filterConfig) throws ServletException {

//获取<filter>标签中配置的<init-param>标签中的Value

filterConfig.getInitParameter("CCC");

//获取ServletContext对象

filterConfig.getServletContext();

}

//2、默认(可以设置条件：请求、包含、错误、重定向)每次请求的时候调用此方法

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)

throws IOException, ServletException {

HttpServletRequest req=(HttpServletRequest) request;

//此代码如何知道下一个资源路径？：在web中<filter-mapping>标签从上至下顺序执行，如果没有了就访问最后的servlet资源

chain.doFilter(request, response);//放行

}

//3、web工程销毁的时候调用

public void destroy() {

}

}

### 作用

1、公共代码的提取

2、可以对request和response中的方法进行增强(装饰者模式/动态代理)

3、进行权限控制

### 例子

#### 解决乱码

第一步 使用装饰模式：创建类继承HttpServletRequestWrapper，（测试了动态代理也可以操作）

注意getParameterMap第一次获取就缓存在内存中，后续就从缓存中取解决过乱码的内容

public class A extends HttpServletRequestWrapper{

boolean flag=true; // 设置此成员布尔值是为了只进行一次解决乱码 的代码执行

public A(HttpServletRequest request) { super(request); } //继承的(父类没有无参构造)

public Map<String, String[]> getParameterMap() {

HttpServletRequest request=(HttpServletRequest)getRequest();

if(request.getMethod().equals("POST")) {

request.setCharacterEncoding("utf-8");

return super.getParameterMap();

}else if(request.getMethod().equals("GET")) {

Map<String, String[]> v=request.getParameterMap();

//只执行一次解决乱码,因为此方法有缓存机制,后续导致获取相同参数会把正确解码内容给解码成乱码

if(flag) { Collection<String[]> l=v.values();

for(String[] e:l) {

//增强for循环是不能修改的，只能进行赋值后进行重新保存；

for(int i=0;i<e.length;i++) {

e[i]= new String(e[i].getBytes("iso8859-1"),"utf-8");}}

flag=false;//设置为false，不再进行第二次

return v; }}

return super.getParameterMap();}

public String getParameter(String name) {return getParameterValues(name)[0]; }//调Values的方法

public String[] getParameterValues(String name) {return getParameterMap().get(name); }//调解决乱码的Map方法

第二步 Filter类中的调用 测试过直接动态代理proxy，可以操作

public void doFilter(ServletRequest arg0, ServletResponse arg1, FilterChain arg2) throws IOException, ServletException {

response.setContentType("text/html;charset=utf-8"); //--解决响应乱码

arg2.doFilter(new A((HttpServletRequest)arg0), arg1); //使用A类的request对象，

}

//此初始方法可以获取web中的init配置文件：utf-8可以不用写死，可以通过此方法获取：A类弄个两参数的构造方法传值

public void init(FilterConfig arg0) throws ServletException {}

## JSP

jsp它其实就是一个servlet

# 路径专题

1获得web工程中任何资源的绝对路径

需要ServletContext(上下文)对象进行获取：context.getRealPath(相对于该web工程的相对地址);

此路径是相对于web工程目录开始寻找的路径:

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

String path = this.getServletContext().getRealPath("");

System.out.println("Eclipse会将工程发布到Tomcat中路径下"+path); 打印：I:\tomcat \webapps\wwl

}

2通过类加载器获取绝地路径 类.class.getClassLoader().getResource("").getPath();

相对于类加载器加载字节码的地方开始寻找：

普通工程类加载地方：是Eclipse发布在本工程目录bin下开始寻找

web工程类加载地方：tomcat下web工程—>WEB-INF —>classes目录

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

String str= MyServlet3.class.getClassLoader().getResource("").getPath();

System.out.println("Eclipse会将工程发布到Tomcat下路径是"+str);打印： I:/tomcat/webapps/wwl/WEB-INF/classes/

}

3通过本身字节码获取绝对路径 类.class.getResource("")

以相对于本身字节码路径来开始寻找

上面2的方法看出是相对于类加载字节码地方开始寻找，不用类加载器获取的路径是相对于本身位置来开始寻找

String path2 = MyServlet3.class.getResource("").getPath();

System.out.println(path2); I:/tomcat/ webapps/ServletDemo/WEB-INF/classes/wwl/

2和3还有一个getFile()区别在于：getFile还会加上getQuery()的值。getQuery()返回的什么暂时不明白

4 了解：System.getProperty("user.dir")运行找到的是eclipse的路径

System.getProperty("user.dir")： user.dir是固定的,key：System.getPropertys()可以获得很多key系统信息

应用：可以后面拼接字符串

//它居然打印的是eclipse 开发工具的目录，明明是已经发布到了tomcat中了

System.out.println(System.getProperty("user.dir")); I:\eclipse\eclipse-jee-2018-09-win32-x86\_64\eclipse

FileReader b=new FileReader(System.getProperty("user.dir")+"\\a.txt"));

System.out.println(b.readLine());