# 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

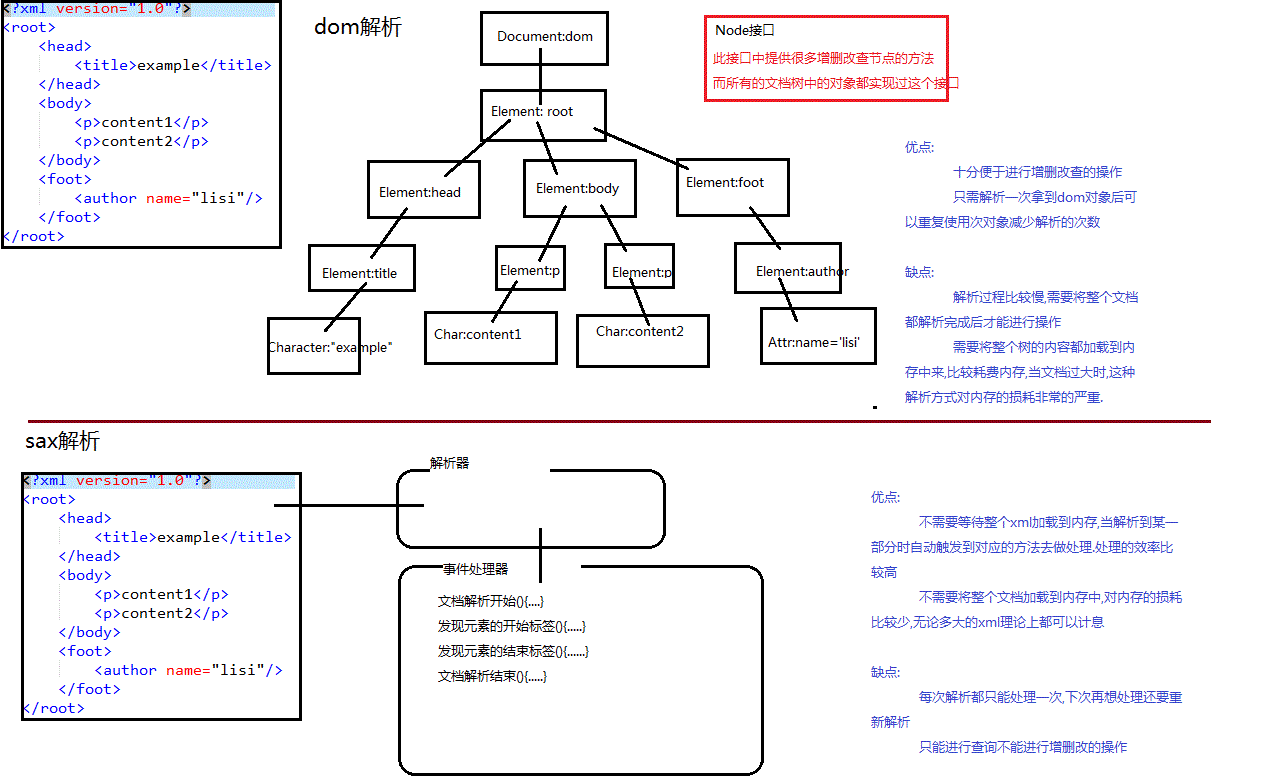
(1)CDATA区可以成段的进行转义，而转义字符一次只能转义一个字符

(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

# 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">

### Eclipse发布工程

第一步 打开Eclipse窗口选择Tomcat进行绑定

windows 🡪preferences🡪server🡪runtime🡪add🡪选择Apache文件夹🡪tomcat7.0🡪browse指定到目录点击完成

第二步 把系统默认的tomcat改为自定tomcat

windows 🡪show view🡪other🡪server🡪servers调试面板打开一个servers面板双击打开一个窗口

1、找到server Locations选择：Use Tomcat installation(takes control of….)

2、把Deploy path:改名为webapps

第三步 发布工程：右击面板中的Tomcat选择add and reomve进行添加工程

### 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>

# 协议

## HTTP协议