# xml使用场景

主要用于:

1. 作为配置文件
2. 作为小型数据库

# xml语法

## 2.1标签

<student>学生</student>

**注意:**

1. 有空标签:<student/>
2. 标签是区分大小写的<student>学生</Student>是错的
3. 标签名不能以数字开头
4. 标签一定要配对
5. 有且仅有一个根标签

## 2.2属性

<student id=”1” name=”lw”></student>

注意:

1. 一个属性分为属性名和属性值,属性名和属性值用=分割
2. 属性值必须要用单引号或者双引号包围,建立用双引号
3. 一个标签内可以包含多个属性,但是不能包含同名属性

## 2.3注释

和html注释一样<!--hahah -->

## 2.4文档声明

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

version:xml的版本号

encoding:表示解析xml文件内容时,使用的码表

## 2.5转义字符

和html一样

## 2.6CDATA块

把CDATA块中的内容全部原样输出

|  |
| --- |
| <student>  <![CDATA[  <name>李四</name>  ]]>  </student |

## 2.7处理指令

01.xml内容:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <?xml-stylesheet type="text/css" href="01.css"?>  <!-- xml你好-->  <student>  <name>张三</name>  <age>20</age>  </student> |

01.css内容

|  |
| --- |
| student{  background-color:red;  font-size:20px;  color:blue;  } |

注意:这个功能不用了

# xml实战

通讯录程序:

联系人:编号 姓名 性别 手机 地址 QQ

要求:

1. 编写一个xml文件(contact.xml),该文件描述联系人信息
2. 该文件可以同时描述多个联系人

|  |
| --- |
| <contact-list>  <contact id="001" name="eric">  <name>张三</name>  <gender>男</gender>  <phone>234234234</phone>  <email>34234@qq.com</email>  <address>杭州西湖</address>  </contact>  <contact id="002">  <name>李四</name>  <gender>男</gender>  <phone>234234234</phone>  <email>34234@qq.com</email>  <address>杭州西湖</address>  </contact>  <student>  </student>  </contact-list> |

# xml解析

有两种方式:

1. DOM解析
2. SAX解析

## 4.1DOM解析

**DOM解析:**

xml解析引起会把一个xml文件一次性加载进内存中,在内存中创建一个document对象,

通过树上的对象来获取或者设置xml上的信息

编程方式:面向对象编程方式

**问题:如果文件大的话,使用dom解析效率比较低,甚至会导致内存溢出**

concat-list

根节点

concat

**. . .**

concat

phone

gender

name

**. . .**

xml及解析引擎会把一个xml文件在内存中创建一个树形结构,树上的分支叫节点(Node)

xml信息 dom模型

节点:

节点名称

节点类型(标签,属性,文本)

Node对象

标签:

标签名

包含属性

包含文本

Element对象

Attribute对象

属性:

属性名

属性值

文本:

文本内容

Text对象

Document对象,代表一个xml文件,从Document对象开始获取xml文档树上的每个节点(标签节点,属性节点,文本节点)

### 4.1.1DOM解析工具

Sun官方jaxp工具(不好用)

jdom工具(也不好用)

dom4j工具(三大框架都使用):重点(非官方)

寓意:dom for java

### 4.1.2继承关系图

Node

接口

ProcessingInstruction 接口

Entity

接口

DocumentType

接口

Branch

接口

CharacterData

接口

CDATA

接口

Document

接口

Element

接口

Text

接口

Attribute

接口

Comment

接口

### 4.1.3Node接口

public interface Node

extends java.lang.Cloneable

子接口Attribute, Branch, CDATA, Comment, Document, DocumentType, Element, Entity, CharacterData, ProcessingInstruction, Text:

常用的方法:

|  |  |
| --- | --- |
| java.lang.String getName() | 获取节点的名称 |

### 4.1.4Document接口

public interface Document

extends Branch

子类有: AbstractDocument, DefaultDocument, DOMDocument

常用方法:

|  |  |
| --- | --- |
| Element getRootElement() | 获取根标签 |

### 4.1.5Element接口

public interface Element

extends Branch

子类: AbstractElement, BaseElement, BeanElement, DatatypeElement, DefaultElement, DOMElement, IndexedElement, NonLazyElement, UserDataElement

常用的方法:

|  |  |
| --- | --- |
| Element element(java.lang.String name) | 获取第一个指定名称的子元素 |
| java.util.List<Element> elements(java.lang.String name) | 获取指定名称的所有子元素 |
| java.util.List<Element> elements() | 获取所有的子元素 |

### 4.1.6例子

#### 1)获取元素

|  |
| --- |
| {  public static void main(String args[]) throws Exception{  //1.创建xml解析器对象  SAXReader reader = new SAXReader();  //2.读取xml文件  File f = new File("src/com/company/01.xml");  Document doc = reader.read(f);  //3.读取标签  //3.1读取根标签  Element rootElement = doc.getRootElement();  //3.2获取标签名  String name = rootElement.getName();  //3.3获取第一个子标签(根据标签名获取指定的标签)  Element conElem = rootElement.element("contact");  //3.4获取所有的子标签(根据指定的名称获取所有同名子标签)  List<Element> list = rootElement.elements("contact");  //遍历list  for(Element e:list){  System.out.println(e);  }  System.out.println("==========================");  //4.获取所有子标签(不指定名称)  List<Element> eList = rootElement.elements();  for(Element e: eList){  System.out.println(e);  }  System.out.println("==========================");  //返回孙子标签  Element nameElem = rootElement.element("contact").element("name");  System.out.print(nameElem);  }  } |

#### 2)属性

|  |
| --- |
| public class Test{  public static void main(String args[]) throws Exception{  //1.创建xml解析器对象  SAXReader reader = new SAXReader();  File f = new File("src/com/company/01.xml");  Document doc = reader.read(f);  //读取属性,必须先得到属性所在的标签  //拿到id=001的属性  Element conElem = doc.getRootElement().element("contact");  //根据属性的名称获取对象的属性值  String value = conElem.attributeValue("id");  System.out.println(value);  //拿到id=002的属性  Element conElem2 = doc.getRootElement().elements("contact").get(1);  Attribute idAttr = conElem2.attribute("id");  //属性名 属性值  System.out.println(idAttr.getName());  System.out.println(idAttr.getValue());  //获取标签的所有属性  List<Attribute> attList = conElem.attributes();  for(Attribute attr : attList){  System.out.println(attr.getName()+"="+attr.getValue());  }  }  } |

#### 3)文本

|  |
| --- |
| public class Test{  public static void main(String args[]) throws Exception{  //1.创建xml解析器对象  SAXReader reader = new SAXReader();  File f = new File("src/com/company/01.xml");  Document doc = reader.read(f);  //2.读取文本  //2.1拿到所在的标签上的文本内容  Element nameElem = doc.getRootElement().element("contact").element("name");  String content = nameElem.getText();  System.out.println(content);  //2.2通过父标签来获取指定子标签的文本内容  Element conElem = doc.getRootElement().element("contact");  content = conElem.elementText("name"); //子标签文本  System.out.println(content);  }  } |

4)将xml信息读取到对象中

a)创建一个Contact类

|  |
| --- |
| package com.company;  public class Contact{  private String id;  private String name;  private String gender;  private String phone;  private String email;  private String address;  public String getId() {  return id;  }  public void setId(String id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public String getGender() {  return gender;  }  public void setGender(String gender) {  this.gender = gender;  }  public String getPhone() {  return phone;  }  public void setPhone(String phone) {  this.phone = phone;  }  public String getEmail() {  return email;  }  public void setEmail(String email) {  this.email = email;  }  public String getAddress() {  return address;  }  public void setAddress(String address) {  this.address = address;  }  @Override  public String toString() {  return "Contact{" +  "id='" + id + '\'' +  ", name='" + name + '\'' +  ", gender='" + gender + '\'' +  ", phone='" + phone + '\'' +  ", email='" + email + '\'' +  ", address='" + address + '\'' +  '}';  }  } |

b)将xml信息读入

|  |
| --- |
| public class Test{  public static void main(String args[]) throws Exception{  //目标:将contact.xml的信息  //1.读取xml文件  SAXReader reader = new SAXReader();  Document doc = reader.read(new File("src/com/company/contact.xml"));  //2.创建List对象  List<Contact> list = new ArrayList<>();  //3.把xml信息->list对象  //3.1读取到所有contact标签  List<Element> conList = doc.getRootElement().elements("contact");  for(Element elem : conList){  //3.2创建Contact对象  Contact con = new Contact();  //3.3把contact标签数据放入contact对象中  con.setId(elem.attributeValue("id"));  con.setName(elem.elementText("name")); //子标签文本  con.setGender(elem.elementText("gender"));  con.setPhone(elem.elementText("phone"));  con.setEmail(elem.elementText("email"));  con.setAddress(elem.elementText("address"));  //3.4把contact对象放入到list对象  list.add(con);  }  //4输出  for(Contact con: list){  System.out.println(con);  }  }  } |

#### 4)拷贝xml文件

|  |
| --- |
| public class Demo{  public static void main(String args[]) throws Exception{  Document doc = new SAXReader().read(new File("./src/com/company/contact.xml"));  //1.1创建一个输出流  OutputStream out = new FileOutputStream("F:/contact.xml");  //1.2创建输出格式  // OutputFormat format = OutputFormat.createCompactFormat();//紧密的格式(空格和换行没了),上线时使用  OutputFormat format = OutputFormat.createPrettyPrint();//漂亮的格式(有空格和换行),测试时使用  //1.3设置编码格式  //1)影响xml文档声明的encoding  //2)影响了xml内容保存的编码  format.setEncoding("utf-8");  //1.4创建输出对象  XMLWriter writer = new XMLWriter(out,format);  //1.2写出数据  writer.write(doc);  writer.close();  }  } |

#### 5)添加标签

|  |
| --- |
| public class Demo{  public static void main(String args[]) throws Exception{  /\*\*  \* 添加  \*/  //1.1创建一个文档  Document doc = DocumentHelper.createDocument();  //1.2添加标签,返回添加的标签  Element conLisstElem = doc.addElement("contact-list");  //doc.addElement("contact-list");不能添加两个根标签  Element conElem = conLisstElem.addElement("contact");  conElem.addElement("name");  //1.3添加属性  conElem.addAttribute("id","001");  conElem.addAttribute("name","eric");  //1.2把文档写出到xml文件中  OutputStream out = new FileOutputStream("F:/contact.xml");  OutputFormat format = OutputFormat.createPrettyPrint();  format.setEncoding("utf-8");  XMLWriter writer = new XMLWriter(out,format);  writer.write(doc);  }  } |

#### 6）修改

|  |
| --- |
| public class Demo{  public static void main(String args[]) throws Exception{  Document doc = new SAXReader().read(new File("./src/com/company/contact.xml"));  /\*\*  \* 修改  \*/  //方法1:  Element conElem = doc.getRootElement().element("contact");  Attribute idAttr = conElem.attribute("id");  idAttr.setValue("003");  //方法2:在一个标签中添加一个同名的属性,来覆盖属性值  conElem.addAttribute("id","004");  //标签的文本  Element nameElem = doc.getRootElement().element("contact").element("name");  nameElem.setText("dsaf王五");  //1.2把文档写出到xml文件中  OutputStream out = new FileOutputStream("F:/contact.xml");  OutputFormat format = OutputFormat.createPrettyPrint();  format.setEncoding("utf-8");  XMLWriter writer = new XMLWriter(out,format);  writer.write(doc);  }  } |

#### 7.删除

|  |
| --- |
| public class Demo{  public static void main(String args[]) throws Exception{  Document doc = new SAXReader().read(new File("./src/com/company/contact.xml"));  /\*\*  \* 删除  \*/  //1.1删除标签  // Element conElem = doc.getRootElement().element("contact");  // conElem.detach(); //自杀  // conElem.getParent().remove(conElem);//他杀  //1.2删除属性  Attribute idAttr = doc.getRootElement().element("contact").attribute("id");  idAttr.detach();  //1.2把文档写出到xml文件中  OutputStream out = new FileOutputStream("F:/contact.xml");  OutputFormat format = OutputFormat.createPrettyPrint();  format.setEncoding("utf-8");  XMLWriter writer = new XMLWriter(out,format);  writer.write(doc);  }  } |

## 4.2XPath

导入jaxen-1.1-beta-6.jar 它是dom4j的插件包

使用XPath就两个方法:

List<Node> list = selectNodes(“xpath表达式”)查询多个节点

Node node = selectStringNode(“xpath表达式”)查询一个节点

### 4.2.1xpath表达式

/ 绝对路径 斜杠在最前面,代表xml文件的根,斜杠在中间,表示子元素

// 相对路径 选择后代元素(不分层次结构)

\* 通配 选择所有元素

[] 条件 选择什么条件下的元素,例如/AAA/BBB[1]选择第一个子元素

@ 属性 选取属性

= 内容(值)

and 逻辑与

text() 选取文本内容

范例:xpath表达式

|  |
| --- |
| public class Demo{  public static void main(String args[]) throws Exception{  //1.读取xml文件  Document doc = new SAXReader().read(new File("./src/com/company/contact.xml"));  //2.利用xpath方法查询xml文件  String xpath = "";  //2.1 /  xpath = "/contact-list"; //选取根标签contact-list  xpath = "/contact-list/contact"; //contact-list根标签下的子标签contact  //2.2 //  xpath = "//contact"; //选取所有的contact标签,不分层次  xpath = "//contact/name"; //选取所有父标签为contact的name标签  //2.3 \*  xpath = "/contact-list/\*"; //选取根标签contact-list下的所有子标签  xpath = "/contact-list//\*";//选取根标签contact-list下的所有后代标签,不分层次结构  //2.4 [] 注意[]表示的是第一个层次的  xpath = "//contact[1]"; //的第一个contact标签  xpath = "//contact[last()]";//最后一个contact标签  //2.5 @  xpath = "//@id"; //选择所有id属性,返回的类型是Attribute  xpath = "//contact[@id]"; //选择所有包含id属性的标签  //2.6 =  xpath = "//contact[@id='001']"; //选择id属性值为001f的contact标签  //2.7 and 逻辑与 在[]可以添加多个条件  //选取id属性为002,且name属性为eric的contact标签  xpath = "//contact[@id='002' and @name='eric']";  //2.8 text() 选取文本  xpath = "//name[1]/text()"; //选择同一个层次的第一个name标签的text文本  xpath = "//name[text()='李四']"; //文本内容为李四的name标签  List<Node> list = doc.selectNodes(xpath);  for(Node node:list){  System.out.println(node);  }  }  } |

范例:模拟用户登陆

|  |
| --- |
| public class Demo{  public static void main(String args[]) throws Exception{  //1)获取用户输入的用户名和密码  BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));  System.out.println("请输入用户名:");  String name = reader.readLine();  System.out.println("请输入密码:");  String password = reader.readLine();  //2)在user.xml中查询name标签的文本为'rose',password标签文本为"123456'的user标签  Document doc = new SAXReader().read(new File("./src/com/company/user.xml"));  //查询 name文本为'xxx'的name标签  Element nameElem = (Element)doc.selectSingleNode("//user/name[text()='"+ name +"']");  //判断name标签是否存在  if(nameElem !=null){  //存在  //判断密码是否正确  Element userElem = nameElem.getParent();  //判断password子标签的文本内容  String dbpwd = userElem.elementText("password");  System.out.println(dbpwd);  if(password.equals(dbpwd)){  System.out.println("登陆成功");  }else{  System.out.println("密码错误,请重新输入");  }  }else{  //不存在  System.out.println("该用户名不存在");  }  }  } |

## 4.3SAX解析

sax解析:加载一点,就处理一点,占用内存比较小

编程方式:基于事件的编程方式

### 4.3.1SAX解析工具

sun公司官方的sax解析工具,在org.xml.sax包中

### 4.3.2sax解析开发步骤

范例:监听器MyDefaultHandler1.java 监听器里重写了很多方法

|  |
| --- |
| public class MyDefaultHandler1 extends DefaultHandler{  /\*\*  \* 遇到xml文档的开始位置触发此方法  \* @throws SAXException  \*/  @Override  public void startDocument() throws SAXException {  System.out.println("MyDefaultHandler1.startDocument()");  }  /\*\*  \* 遇到开始标签触发此方法  \* @param uri  \* @param localName  \* @param qName 表示当前读到的开始标签名称  \* @param attributes 属性列表  \* @throws SAXException  \*/  @Override  public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {  System.out.println("MyDefaultHandler1.startElement()->" + qName);  }  /\*\*  \* 遇到每个结束标签触发此方法  \* @param uri  \* @param localName  \* @param qName 当前读到的结束标签名字  \* @throws SAXException  \*/  @Override  public void endElement(String uri, String localName, String qName) throws SAXException {  System.out.println("MyDefaultHandler1.endElement()->" + qName);  }  /\*\*  \* 遇到文本就会触发此方法  \* @param ch 表示到目前为止读到的文本内容  \* @param start 当前内容的起始位置  \* @param length 表示当前内容的长度  \* @throws SAXException  \*/  @Override  public void characters(char[] ch, int start, int length) throws SAXException {  String content = new String(ch,start,length);  System.out.println("MyDefaultHandler1.characters()->" + content);  }  /\*\*  \* 遇到xml文档的结尾  \* @throws SAXException  \*/  @Override  public void endDocument() throws SAXException {  System.out.println("MyDefaultHandler1.endDocument()");  }  } |

范例:主程序

|  |
| --- |
| public class Demo{  public static void main(String args[]) throws Exception{  //1.创建一个SAXParser解析对象  SAXParser parser = SAXParserFactory.newInstance().newSAXParser();  //2.解析xml  /\*\*  \* 参数1:需要解析的xml文件  \* 参数2:指定的DefaultHandler  \*/  /\*\*  \* 事件编程模式三要素:  \* 事件源:xml文件  \* 事件:解析到开始标签(包含属性), 解析到结束标签 ,解析到文本内容  \* 监听器:DefaultHandler  \*/  //类似注册监听器(将事件源和监听器绑定)  parser.parse(new File("./src/com/company/contact.xml"),new MyDefaultHandler1());  }  } |

### 4.3.3例子

#### 4.3.3.1将xml文件打印出来

MyDefaultHandler2.java

|  |
| --- |
| public class MyDefaultHandler2 extends DefaultHandler{  //存储contact.xml信息  //当contact.xml读取完毕之后，这个变量就有了所有xml信息了  private StringBuffer sb = new StringBuffer();  public String getContent(){  return this.sb.toString();  }  //开始标签  @Override  public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {  sb.append("<" + qName);  //属性列表  if(attributes!=null){  for(int i=0;i<attributes.getLength();i++){  String name = attributes.getQName(i);//根据下标返回属性名称  String value = attributes.getValue(i);//根据下标返回属性值  this.sb.append(" " + name + "=\"" + value + "\"");  }  }  sb.append(">");  }  //文本标签  @Override  public void characters(char[] ch, int start, int length) throws SAXException {  String content = new String(ch,start,length);  this.sb.append(content);  }  //结束标签  @Override  public void endElement(String uri, String localName, String qName) throws SAXException {  sb.append("</" + qName + ">");  }  } |

主程序

|  |
| --- |
| public class Demo{  public static void main(String args[]) throws Exception{  //1.创建一个SAXParser解析对象  SAXParser parser = SAXParserFactory.newInstance().newSAXParser();  //创建时间处理程序  MyDefaultHandler2 handler2 = new MyDefaultHandler2();  //2.读取xml文件  parser.parse(new File("./src/com/company/contact.xml"),handler2);  System.out.println(handler2.getContent());  }  } |

#### 4.3.3.2将xml文件信息封装为List对象

MyDefaultHandler3.java

|  |
| --- |
| public class MyDefaultHandler3 extends DefaultHandler{  //存储所有Contact对象信息  private List<Contact> conList = new ArrayList<>();  //用于存储一个contact标签中的信息  private Contact contact = null;  //用于记录当前标签  private String curTag;  public List<Contact> getConList() {  return this.conList;  }  //开始标签  @Override  public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {  this.curTag = qName;  if(qName.equals("contact")){  //创建contact对象  this.contact = new Contact();  String id = attributes.getValue("id");  this.contact.setId(id);  }  }  //文本内容  @Override  public void characters(char[] ch, int start, int length) throws SAXException {  String content = new String(ch,start,length);  if("name".equals(this.curTag)){  this.contact.setName(content);  }  if("phone".equals(this.curTag)){  this.contact.setPhone(content);  }  if("email".equals(this.curTag)){  this.contact.setEmail(content);  }  if("address".equals(this.curTag)){  this.contact.setAddress(content);  }  if("gender".equals(this.curTag)){  this.contact.setGender(content);  }  }  //结束标签  @Override  public void endElement(String uri, String localName, String qName) throws SAXException {  this.curTag = null;  if(qName.equals("contact")){  this.conList.add(contact);  }  }  } |

Contact.java

|  |
| --- |
| public class Contact{  private String id;  private String name;  private String gender;  private String phone;  private String email;  private String address;  public String getId() {  return id;  }  public void setId(String id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public String getGender() {  return gender;  }  public void setGender(String gender) {  this.gender = gender;  }  public String getPhone() {  return phone;  }  public void setPhone(String phone) {  this.phone = phone;  }  public String getEmail() {  return email;  }  public void setEmail(String email) {  this.email = email;  }  public String getAddress() {  return address;  }  public void setAddress(String address) {  this.address = address;  }  @Override  public String toString() {  return "Contact{" +  "id='" + id + '\'' +  ", name='" + name + '\'' +  ", gender='" + gender + '\'' +  ", phone='" + phone + '\'' +  ", email='" + email + '\'' +  ", address='" + address + '\'' +  '}';  }  } |

contact.xml

|  |
| --- |
| <contact-list>  <contact id="001">  <name>张三</name>  <gender>男</gender>  <phone>234234234</phone>  <email>34234@qq.com</email>  <address>杭州西湖</address>  </contact>  <contact id="002" name="eric">  <name>李四</name>  <gender>男</gender>  <phone>234234234</phone>  <email>34234@qq.com</email>  <address>杭州西湖</address>  </contact>  <student>  </student>  </contact-list> |

主程序

|  |
| --- |
| public class Demo3{  public static void main(String args[]) throws Exception{  //1.读取contact.xml文件  SAXParser parser = SAXParserFactory.newInstance().newSAXParser();  MyDefaultHandler3 handler3 = new MyDefaultHandler3();  parser.parse(new File("./src/com/company/contact.xml"),handler3);  //获取封装好的List对象  List<Contact> conList = handler3.getConList();  for(Contact contact:conList){  System.out.println(contact);  }  }  } |

## 4.4Dom解析和SAX对比

|  |  |  |
| --- | --- | --- |
|  | Dom解析 | SAX解析 |
| 原理 | 一次性加载进内存,内存占用大 | 加载一点就读取一点,内存占用较小 |
| 读取顺序 | 可以读取xml文件的任意位置的信息,甚至是往回读 | 只能是从上往下依次读取,不能往回读 |
| 操作方式 | 可以进行查询,也可以进行修改 | 只能查询 |
| 编程方式 | 面向对象编程方式 | 基于事件的编程方式 |

# xml约束

约束:开发者指定的对xml文件内容的规范(开发者根据业务指定的)

约束的分类:

1. DTD约束:相对简单,数据类型简单的

场景:hibernate structs2

1. Schema约束:相对复杂,功能很强大,数据类型非常丰富,Schema的出现就是为了替代DTD的

## 5.1DTD约束

### 5.1.1内部DtD

|  |
| --- |
| <?xml version="1.0"?>  <!DOCTYPE note [  <!ELEMENT note (to,from,heading,body)>  <!ELEMENT to (#PCDATA)>  <!ELEMENT from (#PCDATA)>  <!ELEMENT heading (#PCDATA)>  <!ELEMENT body (#PCDATA)>  ]>  <note>  <to>Tove</to>  <from>Joni</from>  <heading>Reminder</heading>  <body>Don't forget the meeting!</body>  </note> |

### 5.1.2外部DTD

note.xml

|  |
| --- |
| <!DOCTYPE note SYSTEM "note.dtd">  <note>  <to>Tove</to>  <from>Joni</from>  <heading>Reminder</heading>  <body>Don't forget the meeting!</body>  </note> |

引入外部的方式为:<!DOCTYPE 根标签 SYSTEM/PUBLIC “dtd路径”>

SYSTEM:表示dtd文件在本地

PUBLIC:表示dtd文件咋网路上,而不在本地

### 5.1.3dtd语法

1)约束标签:

<!ELEMENT 元素名称 类别> 或 <!ELEMENT 元素名称(元素内容)>

**类别:**

1. EMPTY:空标签
2. PCDATA:普通字符串(不能包含子标签)
3. ANY:任意内容

例如: <!ELEMENT to (#PCDATA)>

**顺序问题**:

(子元素1,子元素2,....):一定要一次出现1,2,...

例如: <!ELEMENT note (to,from,heading,body)>

**数量问题:**

子元素:有且仅有1个

+: 1个或者多个

\* 0个或者多个

？ 0个或者1个

例如:

|  |
| --- |
| <!DOCTYPE note SYSTEM "note.dtd">  <note>  <to>Tove</to>  <from>Joni</from>  <heading>Reminder</heading>  <body>Don't forget the meeting!</body>  <body>Don't forget the meeting!</body>  </note> |

|  |
| --- |
| <!DOCTYPE note [  <!ELEMENT note (to,from,heading,body+)>  <!ELEMENT to (#PCDATA)>  <!ELEMENT from (#PCDATA)>  <!ELEMENT heading (#PCDATA)>  <!ELEMENT body (#PCDATA)>  ]> |

2)约束属性

<!ATTLIST 元素名称 属性名称 属性类型 默认值>

默认值有三个:

1)#REQUIRED:属性是必须的

2)#IMPLIED: 属性不是必须的

3)#FIXED value:属性值是固定的,可以没有属性,如果有属性,则一定是value值

属性类型:

1)CDATA:普通字符串

2)(en1|en2|...):枚举中的一个值

3)ID:唯一的值

4)

例如:

|  |
| --- |
| <!DOCTYPE note SYSTEM "note.dtd">  <note>  <to>Tove</to>  <from>Joni</from>  <heading>Reminder</heading>  <body id="001">Don't forget the meeting!</body>  </note> |

|  |
| --- |
| <!DOCTYPE note [  <!ELEMENT note (to,from,heading,body)>  <!ELEMENT to (#PCDATA)>  <!ELEMENT from (#PCDATA)>  <!ELEMENT heading (#PCDATA)>  <!ELEMENT body (#PCDATA)>  <!ATTLIST body id CDATA #REQUIRED>  ]> |

### 5.1.4例子

根据dtd文件写出xml文件

|  |
| --- |
| <!DOCTYPE NEWSPAPER[  <!ELEMENT NEWSPAPER (ARTICLE+)>  <!ELEMENT ARTICLE (HEADLINE,BYLINE,LEAD,BODY,NOTES)>  <!ELEMENT HEADLINE (#PCDATA)>  <!ELEMENT BYLINE (#PCDATA)>  <!ELEMENT LEAD (#PCDATA)>  <!ELEMENT BODY (#PCDATA)>  <!ELEMENT NOTES (#PCDATA)>  <!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>  <!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>  <!ATTLIST ARTICLE DATE CDATA #IMPLIED>  <!ATTLIST ARTICLE EDITION CDATA #IMPLIED>  ]>  <NEWSPAPER>  <ARTICLE AUTHOR="xiaoming" EDITOR="fsaf" EDITION="fdsf">  <HEADLINE>abc</HEADLINE>  <BYLINE>1212</BYLINE>  <LEAD>34</LEAD>  <BODY>455</BODY>  <NOTES>lff</NOTES>  </ARTICLE>  <ARTICLE AUTHOR="id" EDITOR="fsaf">  <HEADLINE>234j</HEADLINE>  <BYLINE>你好</BYLINE>  <LEAD>什么</LEAD>  <BODY>使得</BODY>  <NOTES>haha</NOTES>  </ARTICLE>  </NEWSPAPER> |

## 5.2Schema约束