XML技术

# 主要内容：

XML及其语法

XML约束之DTD

XML约束之Schema

XML解析技术

# 需要掌握知识点：

一、XML概述

二、XML的语法

三、XML约束(看懂。引入DTD或Schema，必须会)

四、XML解析(JAVA解析)

# 什么是XML文件

XML是指可扩展标记语言(eXtensible Markup Language)，它是一种标记语言，很类似HTML。它被设计的宗旨是传输数据，而非显示数据。

XML技术是W3C组织(World Wide Web Consortium万维网联盟)发布的，目前遵循的是W3C组织于2000年发布的XML1.0规范。

XML标签没有被预定义，需要用户自行定义标签。

XML被广泛认为是继Java之后在Internet上最激动人心的新技术。

# XML文件的作用：主要是用来存储数据

XML是一种通用的数据交换格式。

在XML语言中，它允许用户自定义标签。一个标签用于描述一段数据；一个标签可分为开始标签和结束标签，在起始标签之间，又可以使用其它标签描述其它数据，以此来实现数据关系的描述。

XML中的数据必须通过软件程序来解析执行或显示，如IE；这样的解析程序称之为Parser(解析器)。

# XML常见应用

在Java开发中，传统的配置文件是\*.properties属性文件(key=value)，而XML表示的数据更为丰富。

XML技术除用于描述有关系的数据外，还经常用作软件配置文件，以描述程序模块之间的关系。(如后面将要学习到的Spring和springMVC,Mybatis都是基于XML作为配置文件的)

在一个软件系统中，通过XML配置文件可以提高系统的灵活性。即程序的行为是通过XML文件来配置的，而不是硬编码。

# 1.3XML基础语法

一个XML文件一般由以下几部分组成：

文档声明

元素

元素的属性

注释

CDATA区

特殊字符

处理指令(PI:Processing Instruction)

## 文档声明

编写XML文档时，需要先使用文档声明来声明XML文档。且必须出现在文档的第一行。

最简单的语法:<?xml version=“1.0”?>

用encoding属性说明文档所使用的字符编码，默认为UTF-8。保存在磁盘上的文件编码要与声明的编码一致。

如：<?xml version=“1.0” encoding=“GB2312”?>

用standalone属性说明文档是否独立，即是否依赖其他文档。

如：<?xml version=“1.0” standalone=“yes”?>

## 元素

XML元素指XML文件中出现的标签。一个标签分为起始和结束标签(不能省略)。一个标签有如下几种书写形式：

包含标签主体：<mytag>some content</mytag>

不含标签主体：<mytag/>

一个标签中可以嵌套若干子标签，但所有标签必须合理的嵌套，不允许有交叉嵌套。

<mytag1><mytag2></mytag1></mytag2> WRONG

一个XML文档必须有且仅有一个根标签，其他标签都是这个根标签的子标签或孙标签。

注：由于在XML中，空格和换行都作为原始内容被处理，所以，在编写XML文件时，使用换行和缩进等方式来让原文件中的内容清晰可读的“良好”书写习惯可能要被迫改变

元素(标签)的名称可以包含字母、数字、减号、下划线和英文句点，但必须遵守下面的一些规范：

严格区分大小写；<P> <p>

只能以字母或下划线开头；abc \_abc

不能以xml(或XML、Xml等)开头----W3C保留日后使用；

名称字符之间不能有空格或制表符；ab

名称字符之间不能使用冒号； (有特殊用途)

## 元素的属性

一个元素可以有多个属性，每个属性都有它自己的名称和取值，例如：<mytag name=“value” …/>

属性值一定要用引号(单引号或双引号)引起来。

属性名称的命名规范与元素的命名规范相同

元素中的属性是不允许重复的

在XML技术中，标签属性所代表的信息也可以被改成用子元素的形式来描述，例如：

<mytag>

<name>

<firstName/>

<lastName/>

</name>

</mytag>

## 注释

XML中的注释语法为：<!--这是注释-->

注意：

XML声明之前不能有注释

注释不能嵌套，例如：

<!--大段注释

…

<!--有一段注释-->

…

-->

## CDATA区

CDATA是Character Data的缩写

作用：把标签当做普通文本内容；

语法：<![CDATA[内容]]>

<![CDATA[

<ali>www.ali.cn</ali>

]]>

## 特殊字符

对于一些特殊字符，若要在元素主体内容中显示，必须进行转义



## 处理指令

处理指令，简称PI(Processing Instruction)。

作用：用来指挥软件如何解析XML文档。

语法：必须以“<?”作为开头，以“?>”作为结尾。

常用处理指令：

XML声明：<?xml version=“1.0” encoding=“GB2312”?>

xml-stylesheet指令：

<?xml-stylesheet type=“text/css” href=“some.css”?>

注：对中文命名的标签元素不起作用

## xml语法规范总结

1、所有xml元素必须结束

<abc></abc> ----- 在xml 文件中如果标签中没有内容 自结束 <abc />

2、标签不允许交叉嵌套

3、有且仅有一个根元素

4、xml文件中 空格和换行 会被解析器进行解析的

5、严格区分大小写，元素和属性名称不能以数字开始

属性定义语法 <元素名称 属性名称="属性值" 属性名称="属性值" ....>

一个元素定义多个属性的，每个属性值必须使用单引号和双引号

6、属性的值必须用单引号或者双引号 引起来

7、xml注释与html注释相同 <!-- --> 注释不能嵌套

CDATA区域内 保留不被解析器解析的内容 ----通常存放特殊代码，这些代码造成xml格式混乱

CDATA和转义字符在哪些情况使用？区别？

\* 优先使用CDATA XML技术存储数据，传输数据、解析数据

8、通过PI指令控制xml显示样式 （指令之前不能有注释）

当xml 使用pi指令 引用CSS --- 效果等同于HTML，进行页面数据展示

\* 通过xml进行显示数据 ，通过XSLT 修饰xml数据显示 （类似CSS

## XML与HTML对比总结

HTML中的标记是用来显示数据的，而XML中的标记用来描述数据的性质和结构。

HTML是不区分大小写的，而XML是严格区分大小写的。

HTML可以有多个根元素，而XML有且只能有一个根元素。

HTML中，属性值的引号是可用可不用的，而XML中，属性值必须放在引号中。

HTML中，空格是可用自动过滤的，而XML中，空格等部分则不会自动删除。

HTML中的标记是预定义的，而XML中的标记是可以随便定义的，并且可扩展。

# XML的约束

xml约束包括 DTD约束以及schema约束

# XML约束之DTD

为什么需要约束

XML都是用户自定义的标签，若出现小小的错误，软件程序将不能正确地获取文件中的内容而报错。(如：Tomcat)

XML技术中，可以编写一个文档来约束一个XML的书写规范，这个文档称之为约束。

两个概念：

格式良好的XML：遵循XML语法的XML

有效的XML：遵循约束文档的XML

总之：约束文档定义了在XML中允许出现的元素名称、属性及元素出现的顺序等等。

## 常用的约束技术

XML DTD

XML Schema

XDR

SOX

## DTD快速入门

DTD(Document Type Definition)：文档类型定义。

作用：约束XML的书写规范

## 验证XML的有效性

如何根据DTD中定义的内容来验证XML书写是否正确呢？

答：需要软件程序，即解析器

根据能否对XML文档进行约束模式校验，可以将解析器分为两类：

非校验解析器，如IE

校验解析器

## 编写DTD的方式

DTD约束文档可以在XML文档中直接定义，也可以作为单独的文档进行编写(单独的文档必须以UTF-8编码进行保存) 。

在XML文档中编写DTD示例

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<!DOCTYPE 商城 [

<!ELEMENT 商城 (订单+)>

<!ELEMENT 订单 (编号,姓名,地址)>

<!ELEMENT 编号 (#PCDATA)>

<!ELEMENT 姓名 (#PCDATA)>

<!ELEMENT 地址 (#PCDATA)>

]>

<商城>

<订单>

<编号>ali0110</编号>

<姓名>屈东</姓名>

<地址>杭州</地址>

</订单>

...

</商城>

--orders.dtd

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!ELEMENT orders (order)>  <!ELEMENT order (id,name,address)>  <!ELEMENT id (#PCDATA)>  <!ELEMENT name (#PCDATA)>  <!ELEMENT address (#PCDATA)> |

--orders.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE orders SYSTEM "orders.dtd">  <orders>  <order>  <id>001</id>  <name>葫芦丝</name>  <address>杭州</address>  </order>  </orders> |

## 引入外部DTD文档

XML使用DOCTYPE声明语句来指明它所遵循的DTD文档，有两种形式：

当引用的DTD文档在本地时，采用如下方式：

<!DOCTYPE 根元素 SYSTEM “DTD文档路径”>

如：<!DOCTYPE 商城 SYSTEM “book.dtd”>

当引用的DTD文档在公共网络上时，采用如下方式：

<!DOCTYPE 根元素 PUBLIC “DTD名称” “DTD文档的URL”>

如:

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

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app\_2\_3.dtd">

## DTD语法详解

DTD文档的语法主要涉及以下内容的定义：

定义元素

定义属性

## DTD-定义元素

在DTD文档中使用ELEMENT关键字来声明一个XML元素。

语法：<!ELEMENT 元素名称 使用规则>

使用规则：

(#PCDATA):指示元素的主体内容只能是普通的文本.(Parsed Character Data)

EMPTY：用于指示元素的主体为空。比如<br/>

ANY:用于指示元素的主体内容为任意类型。

(子元素)：指示元素中包含的子元素

定义子元素及描述它们的关系:

如果子元素用逗号分开，说明必须按照声明顺序去编写XML文档。

如: <!ELEMENT FILE (TITLE,AUTHOR,EMAIL)

如果子元素用“|”分开，说明任选其一。

如:<!ELEMENT FILE (TITLE|AUTHOR|EMAIL)

用+、\*、？来表示元素出现的次数

如果元素后面没有+\*?:表示必须且只能出现一次

+:表示至少出现一次，一次或多次

\*：表示可有可无，零次、一次或多次

?:表示可以有也可以无，有的话只能有一次。零次或一次

如： <!ELEMENT MYFILE ((TITLE\*, AUTHOR?, EMAIL)\* | COMMENT)>

在DTD文档中使用ATTLIST关键字来为一个元素声明属性。

语法：

<!ATTLIST 元素名

属性名1 属性值类型 设置说明

属性名2 属性值类型 设置说明

…

>

例如：

<!ATTLIST 商品

类别 CDATA #REQUIRED

颜色 CDATA #IMPLIED

>

对应的XML为：<商品 类别=“服装” 颜色=“黄色”/>

## 属性值类型

CDATA：表示属性的取值为普通的文本字符串

ENUMERATED (DTD没有此关键字)：表示枚举，只能从枚举列表中任选其一，如(鸡肉|牛肉|猪肉|鱼肉)

ID:表示属性的取值不能重复

设置说明

#REQUIRED：表示该属性必须出现

#IMPLIED：表示该属性可有可无

#FIXED:表示属性的取值为一个固定值。语法：#FIXED "固定值"

直接值：表示属性的取值为该默认值

## DTD--定义属性示例

1、<!ATTLIST 页面作者

姓名 CDATA #IMPLIED

年龄 CDATA #IMPLIED

联系信息 CDATA #REQUIRED

网站职务 CDATA #FIXED "页面作者"

个人爱好 CDATA "上网"

>

2、<?xml version = "1.0" encoding="GB2312" standalone="yes"?>

<!DOCTYPE 购物篮 [

<!ELEMENT 肉 EMPTY>

<!ATTLIST 肉 品种 ( 鸡肉 | 牛肉 | 猪肉 | 鱼肉 ) "鸡肉">

]>

<购物篮>

<肉 品种="鱼肉"/>

<肉 品种="牛肉"/>

<肉/>

</购物篮>

# XML约束之Schema

## XML Schema

XML Schema也被称为XML Schema定义（XML Schema Definition，XSD）。和DTD一样，Schema也是XML的约束，同样用于定义合法的XML文档构建模块。与DTD不同的是，XML Schema是用一套预先定义好的XML元素和属性创建的，这些元素和属性规定了XML文档的结构和内容模式，且XML Schema规定XML文档实例的结构和每一个元素或属性的数据类型。另外，Schema相对于DTD有一个明显的好处就是，Schema是基于XML编写的，自己本身也是一个XML文档（文件后缀名为.xsd），而不是像DTD有自成一套的语法，这也是Schema能比DTD更被广泛应用的原因

## Schema约束快速入门

XML Schema 文件自身就是一个XML文件，但它的扩展名通常为.xsd。

一个XML Schema文档通常称之为模式文档(约束文档)，遵循这个文档书写的xml文件称之为实例文档。

和XML文件一样，一个XML Schema文档也必须有一个根结点，但这个根结点的名称为schema。

编写了一个XML Schema约束文档后，通常需要把这个文件中声明的元素绑定到一个ＵＲＩ地址上，在XML Schema技术中有一个专业术语来描述这个过程，即把XML Schema文档声明的元素绑定到一个名称空间上，以后XML文件就可以通过这个URI（即名称空间）来告诉解析引擎，xml文档中编写的元素来自哪里，被谁约束。

## Schema入门案例



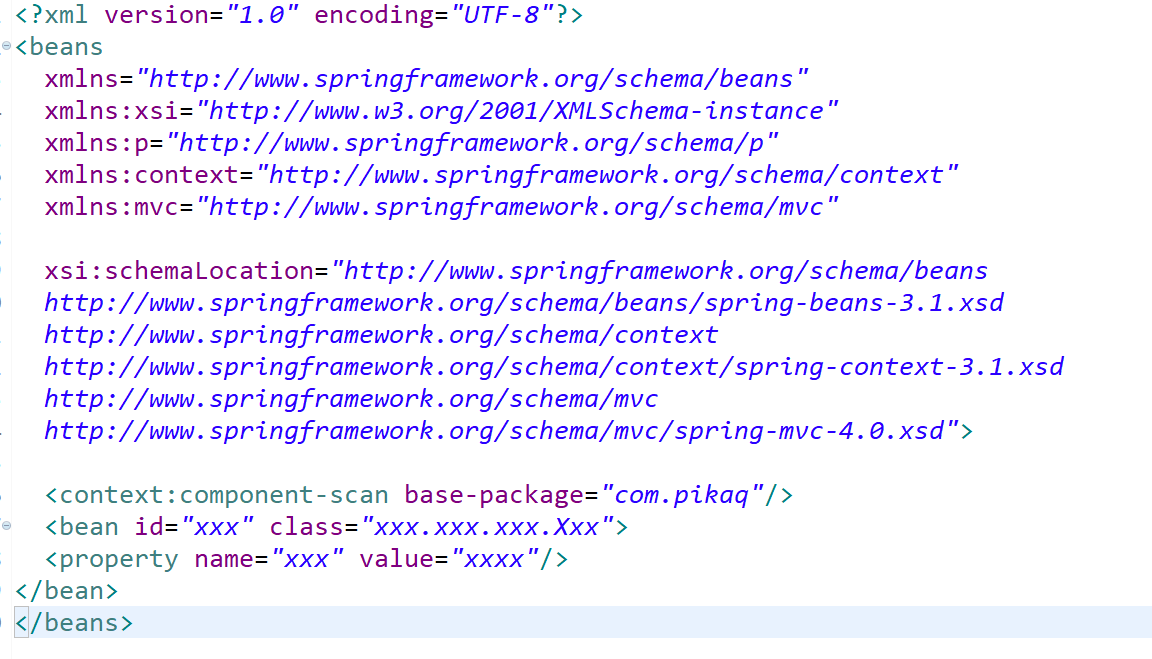
## 名称空间的概念

### 为什么需要xmlns

因为xml文件有成千上万，谁也不能保证你的标签是独一无二的，总是会冲突的，这时就需要xmlns了！

### 怎么使用xmlns 呢

使用语法： xmlns:namespace-prefix="namespaceURI"。其中namespace-prefix为自定义前缀，只要在这个XML文档中保证前缀不重复即可；namespaceURI是这个前缀对应的XML Namespace的定义。例如：



## 使用名称空间引入Schema

为了在一个XML文档中声明它所遵循的Schema文件的具体位置，通常需要在Xml文档中的根结点中使用schemaLocation属性来指定，例如：

<aliyc:书架 xmlns:aliyc="http://www.aliyc.cn"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation=“http://www.aliyc.cn book.xsd">

此属性有两个值。第一个值是需要使用的命名空间。第二个值是供命名空间使用的 XML schema 的位置，两者之间用空格分隔。

注意，在使用schemaLocation属性时，也需要指定该属性来自哪里。

## 使用默认名称空间

基本格式：

xmlns="URI"

举例：

<商城 xmlns="http://www.aliyc.org/xmlbook/schema"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation=“http://www.aliyc.org/xmlbook/schema book.xsd">

<订单>

<编号>0001</编号>

<姓名>屈东</姓名>

<地址>108.00元</售价>

</订单>

<商城>

--orders.xml

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

<orders xmlns="aliyc"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="aliyc orders.xsd"

>

<order>

<name>Core Java</name>

<price>100</price>

</order>

</orders>

--orders.xsd

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

<schema xmlns="http://www.w3.org/2001/XMLSchema"

targetNamespace="aliyc"

elementFormDefault="qualified"

>

<!-- 每一个元素 对应 element -->

<element name="orders">

<!-- 包含子元素和属性的元素 复杂元素 -->

<complexType>

<!-- 强调子元素出现顺序 -->

<sequence>

<!-- 让book元素出现多次 \* -->

<element name="order" minOccurs="0" maxOccurs="unbounded">

<complexType>

<sequence>

<!-- 对于简单元素，编写type属性 -->

<element name="name" type="string"></element>

<element name="price" type="double"></element>

</sequence>

</complexType>

</element>

</sequence>

</complexType>

</element>

</schema>

# XML编程(用Java编写解析器)

## XML解析思想

XML解析方式分为两种：DOM方式和SAX方式

DOM：Document Object Model，文档对象模型。这种方式是W3C推荐的处理XML的一种方式。

SAX：Simple API for XML。这种方式不是官方标准，属于开源社区，几乎所有的XML解析器都支持它。

STAX是JDK6.0中除了DOM和SAX之外的又一种处理XML文档的API

## 什么是DOM和SAX/STAX

DOM Document Object Model ----- 文档对象模型

DOM思想： 将整个xml 加载内存中，形成文档对象，所有对xml操作都对内存中文档对象进行

DOM 是官方xml解析标准所有开发语言都支持的

SAX 思想：一边解析 ，一边处理，一边释放内存资源 ---- 不允许在内存中保留大规模xml 数据

解析xml 思想是仅仅是一种单纯的编程思想，没有具体代码，解析开发包是解析xml思想

的具体代码实现

## DOM和SAX区别

DOM 支持回写

会将整个XML载入内存，以树形结构方式存储

XML比较复杂的时候，或者当你需要随机处理文档中数据的时候不建议使用

SAX / STAX

相比DOM是一种更为轻量级的方案

采用串行方法读取 --- 文件输入流(字节、字符)读取

编程较为复杂

无法在读取过程中修改XML数据

## SAX与STAX的区别

StAX和SAX的区别——拉式解析器和推式解析器的区别

虽然StAX与SAX一样基于XML事件解析，相比于DOM将整个XML加载进内存来说效率高。不同的是，StAX在在处理XML事件的方式上使得应用程序更接近底层，所以在效率上比SAX更优秀。

使用SAX时，我们知道XML事件是由解析器调用开发人员编写的回调方法来处理的，也就是说应用程序是被动于解析器的。应用程序只能被动的等待解析器将XML事件推送给自己处理，对于每一种事件都需要在解析开始之前就做好准备。这种方式被称为“推（push）”。而StAX正好相反，StAX采用一种“拉（pull）”的方式，由应用程序主动从解析器中获取当前XML事件然后根据需求处理（保存或者忽略）。StAX使得应用程序掌握了主动权，可以简化调用代码来准确地处理它预期的内容，或者发生意外时停止解析。此外，由于该方法不基于处理程序回调，应用程序不需要像使用 SAX 那样模拟解析器的状态。

## DOM和SAX/STAX解析方式的选择

选择 DOM 还是 SAX/STAX，这取决于几个因素

应用程序的目的：如果必须对数据进行更改，并且作为 XML 将它输出，则在大多数情况下，使用 DOM解析方式

数据的数量：对于大文件，SAX/STAX 是更好的选择

将如何使用数据：如果实际上只使用一小部分数据，则使用 SAX /STAX将数据抽取到应用程序中，这种方法更好些

需要速度：通常，SAX/STAX 实现比 DOM 实现快

## XML解析技术

DOM、SAX、STAX 只是XML解析方式，解析思想，没有API(仅仅是指导思想)

JAXP是 Sun 提供的一套XML解析API

JAXP很好的支持DOM和SAX解析思想

JDK6.0 开始支持 STAX 解析方式

JAXP 开发包是JavaSE的一部分，它由javax.xml、org.w3c.dom 、org.xml.sax 包及其子包组成

在 javax.xml.parsers 包中，定义了几个工厂类，程序员调用这些工厂类，可以得到对xml文档进行解析的 DOM 或 SAX 的解析器对象

Jaxp(sun)、xml pull 、dom4j

JAXP：是SUN公司推出的解析标准实现。

Dom4J：是开源组织推出的解析开发包。(牛，大家都在用，包括SUN公司的一些技术的实现都在用)

## XML解析开发包

JAXP 是sun官方推出实现技术 同时支持 DOM SAX STAX

DOM4j 是开源社区开源框架 支持 DOM 解析方式

XML PULL Android 移动设备内置xml 解析技术 支持 STAX 解析方式

## DOM4J解析XML文档

Dom4j是一个简单、灵活的开放源代码的库。Dom4j是由早期开发JDOM的人分离出来而后独立开发的。与JDOM不同的是，dom4j使用接口和抽象基类，虽然Dom4j的API相对要复杂一些，但它提供了比JDOM更好的灵活性。

Dom4j是一个非常优秀的Java XML API，具有性能优异、功能强大和极易使用的特点。现在很多软件采用的Dom4j，例如Hibernate，包括sun公司自己的JAXP也用了Dom4j。

使用Dom4j开发，需下载dom4j相应的jar文件。

## DOM4J的dom方式与sax方式案例（XPath方式选择xml中的节点）

|  |
| --- |
| public static void Dom4j\_dom() throws UnsupportedEncodingException, DocumentException{  //创建dom4j的解析器  SAXReader saxReader = new SAXReader();  //创建文件对象  File xmlfile = new File("config/01Student.xml");  //利用解析器读取xml文件。获取domcument对象  Document doc = saxReader.read(xmlfile);  //XPath方式选取  List<Element> studentNodes = doc.selectNodes("/students/student"); //返回student节点集合，很耗时    for(Element e : studentNodes){  String id = e.attributeValue("id");  System.out.println(id);  }  } |

## DOM4J编程练习

Xml File

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <students>  <student>  <name>屈东</name>  <college>河南大学</college>  <telephone>qq:1766181826</telephone>  <notes>程序员</notes>  </student>    </students> |

Code:

|  |
| --- |
| public static void main(String[] args) {  try {  //创建解析器  SAXReader reader = new SAXReader();  //通过解析器的read方法将配置文件读取到内存中，生成一个Document[org.dom4j]对象树  Document document = reader.read("conf/students.xml");  //获取根节点  Element root = document.getRootElement();  //开始遍历根节点  for(Iterator<Element> rootIter = root.elementIterator();rootIter.hasNext();){  Element studentElt = rootIter.next();  for(Iterator<Element> innerIter = studentElt.elementIterator();innerIter.hasNext();){  Element innerElt = innerIter.next();  String innerValue = innerElt.getStringValue();  System.out.println(innerValue);  }  System.out.println("-------------------------------");  }  } catch (Exception e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  } |

改进版：

|  |
| --- |
| public static void main(String[] args) {  try {  //创建解析器  SAXReader reader = new SAXReader();  //通过解析器的read方法将配置文件读取到内存中，生成一个Document[org.dom4j]对象树  Document document = reader.read("config/01Student.xml");  //获取根节点  Element root = document.getRootElement();    Iterator<Element> rootIter = root.elementIterator();  while (rootIter.hasNext()) {  Element studentElt = rootIter.next();  Iterator<Element> innerIter2 = studentElt.elementIterator();  while (innerIter2.hasNext()) {  Element innerElt = innerIter2.next();  String innerValue = innerElt.getStringValue();  System.out.println(innerValue);  }    }  } catch (Exception e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  } |

## JAXP解析技术

JAXP（Java API for [XML](https://baike.baidu.com/item/XML" \t "https://baike.baidu.com/item/JAXP/_blank)Processing，意为XML处理的Java API）是Java XML程序设计的[应用程序接口](https://baike.baidu.com/item/%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F%E6%8E%A5%E5%8F%A3/10418844" \t "https://baike.baidu.com/item/JAXP/_blank)之一，它提供解析和验证XML文档的能力。JAXP是在Java社区进程下开发的，包括JSR 5 （JAXP 1.0）和 JSR 63 （JAXP 1.1和1.2）两个规范。

JAXP解析XML的三种基本接口为：

[文档对象模型](https://baike.baidu.com/item/%E6%96%87%E6%A1%A3%E5%AF%B9%E8%B1%A1%E6%A8%A1%E5%9E%8B/1033822" \t "https://baike.baidu.com/item/JAXP/_blank)解析接口或DOM接口

XML简单API解析接口或SAX接口

XML流API或STAX接口（是JDK 6的一部分，为JDK 5提供单独的包）

除了解析接口，JAXP还提供了[XSLT](https://baike.baidu.com/item/XSLT" \t "https://baike.baidu.com/item/JAXP/_blank)接口用来对XML文档进行数据和结构的转换。

百度百科：https://baike.baidu.com/item/JAXP/3669704?fr=aladdin#1

## JAXP的dom方式SAX方式解析XML

在使用 DOM 解析 XML 文档时，需要读取整个 XML 文档，在内存中构架代表整个 DOM 树的Doucment对象，从而再对XML文档进行操作。此种情况下，如果 XML 文档特别大，就会消耗计算机的大量内存，并且容易导致内存溢出。

SAX解析允许在读取文档的时候，即对文档进行处理，而不必等到整个文档装载完才对文档进行操作。

SAX采用事件处理的方式解析XML文件，利用 SAX 解析 XML 文档，涉及两个部分：解析器和事件处理器：

解析器可以使用JAXP的API创建，创建出SAX解析器后，就可以指定解析器去解析某个XML文档。

解析器采用SAX方式在解析某个XML文档时，它只要解析到XML文档的一个组成部分，都会去调用事件处理器的一个方法，解析器在调用事件处理器的方法时，会把当前解析到的xml文件内容作为方法的参数传递给事件处理器。

事件处理器由程序员编写，程序员通过事件处理器中方法的参数，就可以很轻松地得到sax解析器解析到的数据，从而可以决定如何对数据进行处理

|  |
| --- |
| Jaxp的dom方式与sax方式解析详见资料 |

## jAXP的SAX编程练习

|  |
| --- |
| public class MySAXParser {  /\*\*  \* @param args  \*/  public static void main(String[] args) {  try {  //创建解析器工厂  SAXParserFactory saxParserFactory = SAXParserFactory.newInstance();  //创建解析器  SAXParser saxParser = saxParserFactory.newSAXParser();  //通过解析器的parser方法  saxParser.parse("conf/persons.xml", new MyDefaultHandler());  } catch (Exception e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  }  }  class MyDefaultHandler extends DefaultHandler{  @Override  public void startElement(String uri, String localName, String qName,  Attributes attributes) throws SAXException {  System.out.print("<" + qName + ">");  }  @Override  public void characters(char[] ch, int start, int length)  throws SAXException {  System.out.print(new String(ch,start,length));  }  @Override  public void endElement(String uri, String localName, String qName)  throws SAXException {  System.out.print("</" + qName + ">");  }  } |

Persons.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <person>  <p1>  <name>zhangsan</name>  <age>20</age>  </p1>  <p1>  <name>wangwu</name>  <age>29</age>  </p1>  </person> |

# XPath

## XPath概念

XPath即为[XML](https://baike.baidu.com/item/XML" \t "https://baike.baidu.com/item/XPath/_blank)路径语言（XML Path Language），它是一种用来确定XML文档中某部分位置的语言。XPath就是用来选取和确定XML文件中具体元素节点的技术

XPath基于XML的树状结构，提供在数据结构树中找寻节点的能力。起初XPath的提出的初衷是将其作为一个通用的、介于[XPointer](https://baike.baidu.com/item/XPointer" \t "https://baike.baidu.com/item/XPath/_blank)与[XSL](https://baike.baidu.com/item/XSL" \t "https://baike.baidu.com/item/XPath/_blank)间的语法模型。但是XPath很快的被开发者采用来当作小型查询语言

## XPath 语法选取节点





## XPATH案例1

## Server.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <server>  <service>  <connector port="8080"></connector>  </service>  </server> |

XPATH code

|  |
| --- |
| public static void main(String[] args) {  try {  //创建解析器  SAXReader saxReader = new SAXReader();  //通过解析器的read方法将配置文件读取到内存中，生成一个Document对象树  Document document = saxReader.read("conf/server.xml");  //获取connector节点元素对象的路径：server -> service -> connector  //获取connector节点元素对象的xpath路径：/server/service/connector  //获取connector节点元素对象的xpath路径：server//connector  //获取connector节点元素对象的xpath路径：//connector  Element connectorElt = (Element) document.selectSingleNode("//connector");    //获取connectorElt节点元素对象的port属性对象  Attribute portAttr = connectorElt.attribute("port");  //获取portAttr属性对象的值  String port = portAttr.getStringValue();    String portValue = connectorElt.attributeValue("port");    System.out.println(portValue);    } catch (Exception e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  } |

## XPATH案例2

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <config>  <database-info>  <driver-name>com.mysql.jdbc.Driver</driver-name>  <url>jdbc:mysql://localhost:3366/aliyc</url>  <user>root</user>  <password>123</password>  </database-info>  </config> |

|  |
| --- |
| public static void main(String[] args) {  try {  //创建解析器  SAXReader reader = new SAXReader();  //通过解析器的read方法将配置文件读取到内存中，生成一个Dcoumente【org.dom4j】对象树  Document document = reader.read("conf/sys-config.xml");  //driver-name节点元素的路径：config -> database-info -> driver-name  // driver-name节点元素的xpath路径：/config/database-info/driver-name  Element driverNameElt = (Element) document.selectSingleNode("/config/database-info/driver-name");  //获取driverNameElt节点元素对象的文本内容  String driverName = driverNameElt.getStringValue();  System.out.println(driverName);    //url节点元素的路径： config -> database-info -> url  //url节点元素的xpath路径： /config/database-info/url  //url节点元素的xpath路径： config//url  //url节点元素的xpath路径： //url  Element urlElt = (Element) document.selectSingleNode("config//url");  String url = urlElt.getStringValue();  System.out.println(url);    //user节点元素的路径:config -> database-info -> user  //user节点元素的xpath路径:/config/database-info/user  //user节点元素的xpath路径:config//user  //user节点元素的xpath路径://user  Element userElt = (Element) document.selectObject("//user");  String user = userElt.getText();  System.out.println(user);    Element passwordElt = (Element) document.selectSingleNode("//password");  String password = passwordElt.getTextTrim();  System.out.println(password);  } catch (Exception e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  } |