1. Schema的约束 （schema是一个约束文件 xml是被约束文件）

XML Schema文件自身就是一个XML文件，但它的扩展名通常

为.xsd和XmL文件一样，一个XMLSchema文档也必须有一个根结点，

但这个根结点的名称为Schema

Dtd的语法<!Element 元素名称 约束>

Schema 符合xml的语法，xml语句

一个xml中可以有多个schema

多个schema如何区分？多个schema使用名称空间区分(类似于java中的包名)

Dtd开面的PCDATE,但是在schema里面可以支持更多的数据类型

比如 年龄只能是整数，在schenma里面可以直接定义一个整数

Sschema结构和语法比较复杂

1. schema的快速入门
2. 创建schema文件,后缀名是 .xsd

W3c 规定使用schema作为根节点

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

表示创建的是约束文档

targetNamespace=*"http://www.example.org/1"*

怎么引入schema 路径

elementFormDefault=*"qualified"*

表示质量良好

</schema>

步骤（1）看xml中有多少个元素

<element>

（2）看简单的元素和复杂的元素

如果是复杂元素

<element name=*"person"*>

<complexType >

<sequence>

子元素

</sequence>

</complexType>

</element>

简单元素

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

<element name=*"age"* type=*"int"*></element>子元素

1. 简单元素写在复杂元素中

<element name=*"person"*>

<complexType >

<sequence>

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

<element name=*"age"* type=*"int"*></element>子元素

</sequence>

</complexType>

</element>

<sequence> 指示器 sequence 表示元素的出现按照顺序来写，将xml中的元素调换位置会报错

<all>表示元素只能出现一次

<choice>元素只能出现其中的一个

maxOccurs

<element name=*"name"* type=*"string" maxOccurs=“unbounded”*></element>

表示改元素可以出现的次数 表示出现无限次数

<any><any> :表示任意元素

Attribute可以约束属性

1. 写在复杂的元素里面
2. 写在</complexType>之前

<attribute name=*"id1"* type=*"int"* use=*"required"*></attribute>

name：属性名称

type：属性类型

use:属性是否必须出现 required

1. 在xml中引入约束的文件

<person xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* //与xsd相同便是该xml文件是是一个被约束文件

xmlns=[*http://www.example.org/1*](http://www.example.org/1) *//约束文档的地址或者路径*

xsi:schemaLocation=*"http://www.example.org/1 1.xsd"*>

// targetNamespace 空格 约束文档的路径地址

<name>zhangsan </name>

<age>20</age>

person>

xsd文件

<schema xmlns=*"http://www.w3.org/2001/XMLSchema"* targetNamespace=*"http://www.example.org/1"* xmlns:tns=*"http://www.example.org/1"* elementFormDefault=*"qualified"*>

<element name=*"person"*>

<complexType >

<sequence>

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

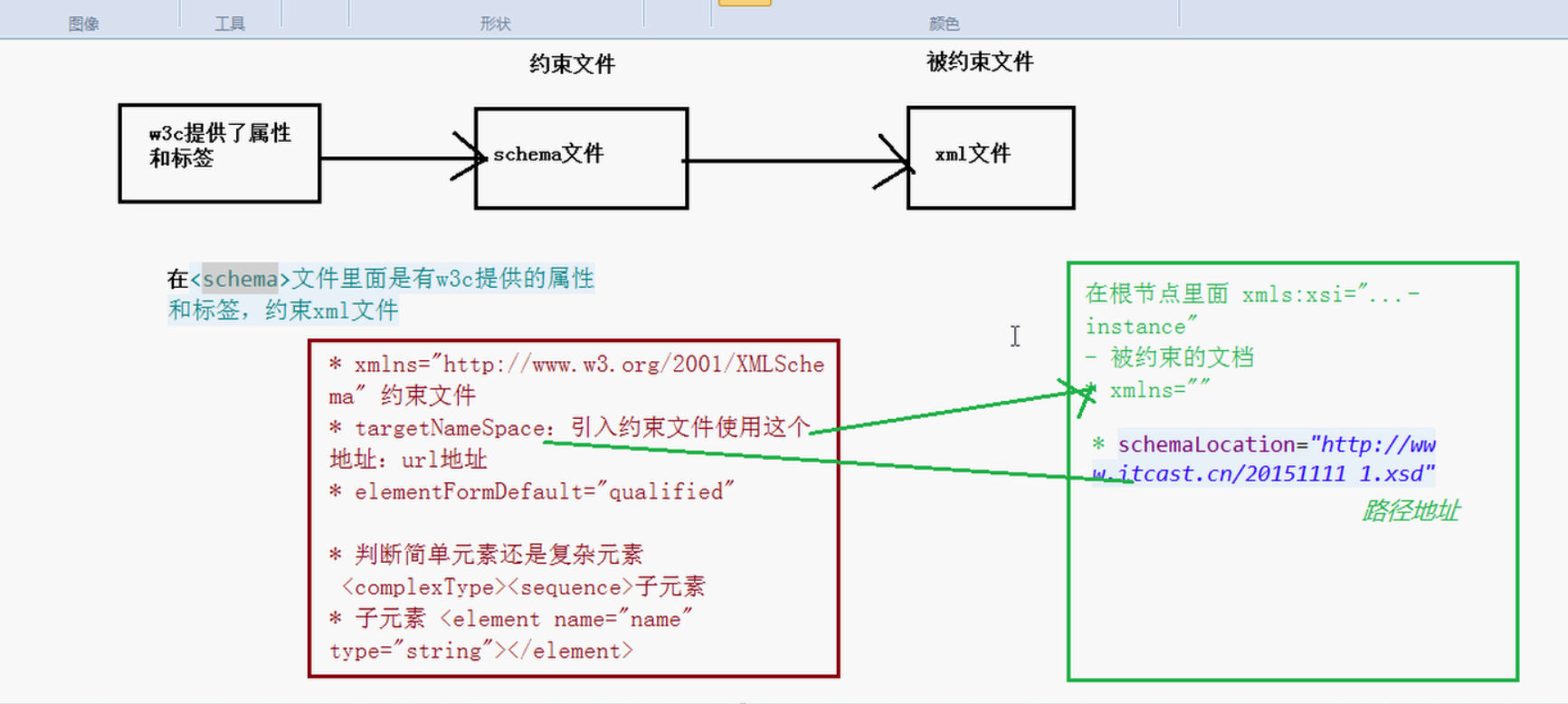
<element name=*"age"* type=*"int"*></element>

</sequence>

</complexType>

</element>

</schema>



复杂的约束



通过 别名：属性 对属性进行限制

别名的名字随便，但是别名的名字不能相同

Xml使用多个schema

如果schema部门名称的元素为任何类型 引入多个schema，通过别名来引入

Xml中的代码，实现对xml中代码的约束 不同的约束引入不同的别名来区分

e <employee age="30">

<!--部门名称-->I

<dept：name>100</dept:name>引入dept中的限制

<！--员工名称-->

<name>王晓晓</name>

</employee>

</company>

3、sax的解析原理

解析xml的两种技术 dom和sax

Dom解析根据xml的层级结构在内存中分配已个树形结构

把xml中的标签属性，文本封装成对象

Sax方式:事件驱动，边读变解析

（api文档）在java.xmlparsers包里面 有两个类

抽象类SAXParser

得到实例 从SAXParseFactory.newSAXParser（）方法获得

Parse（File F,DefaultHandler dh）

两个参数

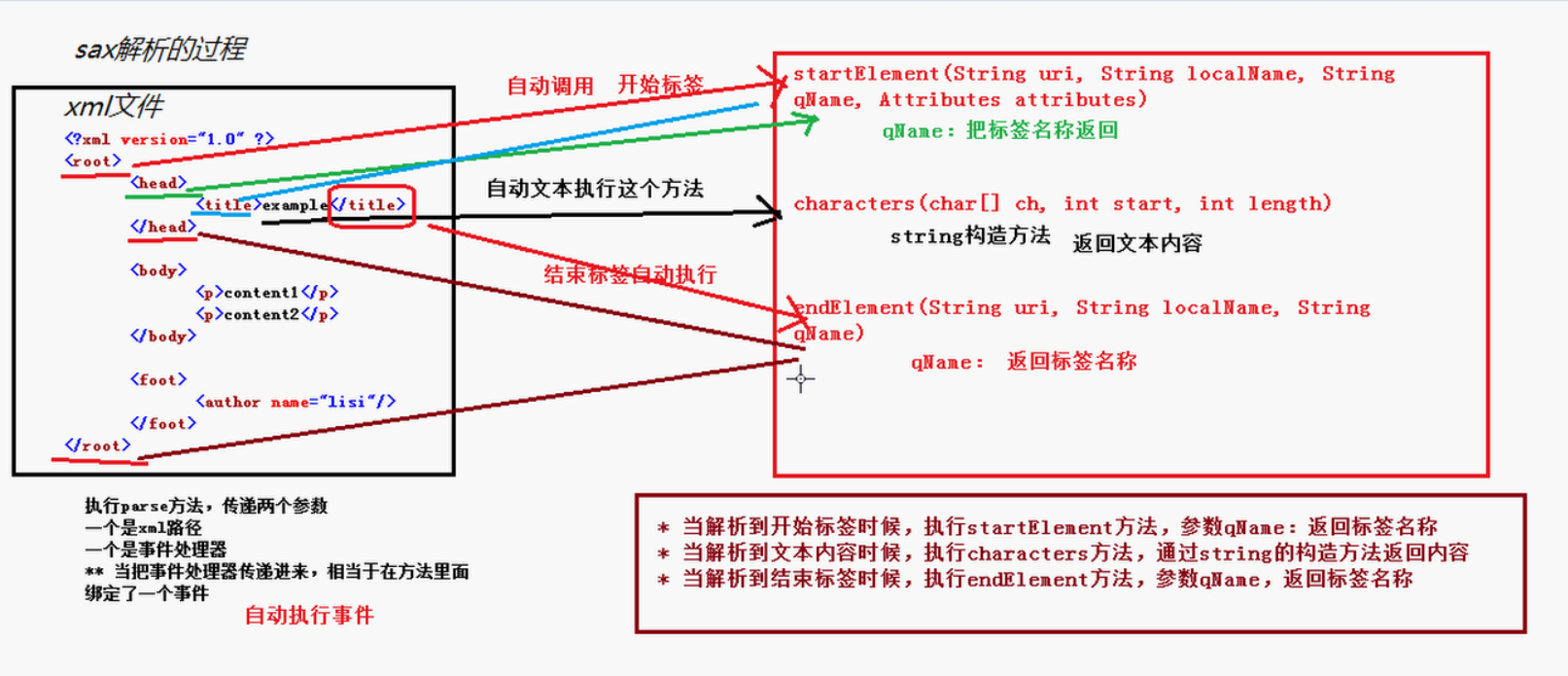
第一个参数 xml的路径

事件处理器

SAXParserFactory

得到实例 newInstance（）方法得到

Sax的执行过程



当解析到开始标签是自动执行startElement方法

当解析到文本的时候，自动执行characters方法

当解析到结束标签时自动执行endElement 方法

4、使用jasp的dax方式解析xml

Sax方式不能实现增删改操作，只能执行查询操作

打印出整个文档

执行parse方法第一个参数xml路径，第二个参数是时间处理器，创建一个雷，继承事件处理器的类，重写里面的三个方法

**public** **static** **void** main(String[] args) **throws** Exception {

SAXParserFactory saxParserFactory = SAXParserFactory.*newInstance*();

SAXParser saxParser = saxParserFactory.newSAXParser();

saxParser.parse("src/1.xml",**new** MyDefault());

}

}

**class** MyDefault **extends** DefaultHandler {

@Override

**public** **void** startElement(String uri, String localName, String qName, Attributes attributes) **throws** SAXException {

// **TODO** 自动生成的方法存根

**super**.startElement(uri, localName, qName, attributes);

System.***out***.print(qName);

}

@Override

**public** **void** endElement(String uri, String localName, String qName) **throws** SAXException {

// **TODO** 自动生成的方法存根

**super**.endElement(uri, localName, qName);

System.***out***.print(qName);

}

@Override

**public** **void** characters(**char**[] ch, **int** start, **int** length) **throws** SAXException {

// **TODO** 自动生成的方法存根

**super**.characters(ch, start, length);

System.***out***.print(**new** String(ch, start, length));

}

获取到所有name元素的值

**class** MyDefault1 **extends** DefaultHandler{

**boolean** flag = **false**;

@Override

**public** **void** startElement(String uri, String localName, String qName, Attributes attributes) **throws** SAXException {

// **TODO** 自动生成的方法存根

**super**.startElement(uri, localName, qName, attributes);

**if**("name" ==qName) {

flag = **true**;

}

}

@Override

**public** **void** characters(**char**[] ch, **int** start, **int** length) **throws** SAXException {

// **TODO** 自动生成的方法存根

**super**.characters(ch, start, length);

**if**(flag==**true**) {

System.***out***.println(**new** String(ch, start, length));

}

}

@Override

**public** **void** endElement(String uri, String localName, String qName) **throws** SAXException {

// **TODO** 自动生成的方法存根

**super**.endElement(uri, localName, qName);

**if**("name" ==qName) {

flag = **false**;

}

}

}

只打印name值

**class** MyDefault1 **extends** DefaultHandler{

**boolean** flag = **false**;

**int** index = 1;

@Override

**public** **void** startElement(String uri, String localName, String qName, Attributes attributes) **throws** SAXException {

// **TODO** 自动生成的方法存根

**super**.startElement(uri, localName, qName, attributes);

**if**("name" ==qName) {

flag = **true**;

}

}

@Override

**public** **void** characters(**char**[] ch, **int** start, **int** length) **throws** SAXException {

// **TODO** 自动生成的方法存根

**super**.characters(ch, start, length);

**if**(flag==**true** && index == 2) {

System.***out***.println(**new** String(ch, start, length));

}

}

@Override

**public** **void** endElement(String uri, String localName, String qName) **throws** SAXException {

// **TODO** 自动生成的方法存根

**super**.endElement(uri, localName, qName);

**if**("name" ==qName) {

flag = **false**;

index++;

}

}

}

\*\*\*执行parse方法，第一个参数xm1路径，第二个参数是事件处理器

\*\*\*创建一个类，继承事件处理器的类，

\*\*\*重写里面的三个方法

\*获取到所有的name元素的值

\*\*定义一个成员变量flag=false

\*\*判断开始方法是否是name元素，如果是name元素，把flag值设置成true

\*\*如果flag值是true，在characters方法里面打印内容

\*\*当执行到结束方法时候，把flag值设置成false

\*获取第一个name元素的值I

\*\*定义一个成员变量idx=1

\*\*在结束方法时候，idx+1 idx+H

\*\*想要打印出第一个name元素的值，

-在characters方法里面判断，

--判断flag=true并且idx==1，在打印内容

5、使用dom4j解析xml

DOM在解析的时候把整个XML文件映射到Document的树型结构中，XML中的元素、属性、文本都能在Document中看清，但是它消耗内存、查询速度慢。SAX是基于事件的解析，解析器在读取XML时根据读取的数据产生相应的事件，由应用程序实现相应的事件处理，所以它的解析速度快，内存占用少。但是它需要应用程序自身处理解析器的状态，实现起来比较麻烦，而且它只支持对XML文件的读取，不支持写入。

Dom4j 是一个组织，针对xml得劲解析，提供解析器dom4j

Dom4j不是javase的一部分，要想使用第一步需要怎么做

导入dom4j提供的jar 包

导入方法 创建一个lib文件夹，复制文件到lib文件夹下

右键点击jar包 build path add to buildpath

看到jar包 变成奶瓶的样子 表示导入成功】

得到document

SAXReader reader = new SAXReader();

Document document = reader.read(url);

document的父接口是Node 在document中找不到方法在Node中去找

node 中的方法

document getRootElement（）获取根节点 返回的是Element

Element 也是一个接口，父接口是Node

Element和Node中的方法

getParent（）获取父节点

addElement（）添加标签

实现方法 element(qname)表示获取标签下面的第一个字标签

Qname：标签 的名称

elements(qname)

获取下级字标签

Qname:标签名称

elements():获取下面所有级别的字标签

6、使用dom4j查询dom

<p1>

<name>zhangsan</name>

<age>20</age>

</p1>

<p1>

<name>lisi</name>

<age>30</age>

</p1>

需求：查询name元素里面的值

1. 创建解析器
2. 得到document
3. 得到根节点
4. 得到所有的p1的标签

Element（“p1” ）返回list集合

遍历list的每一个p1

1. 得到name
2. 得到name中的值
3. **public** **static** **void** selectName() **throws** Exception{
4. SAXReader saxReader = **new** SAXReader();
5. Document document = saxReader.read("src/1.xml");
6. Element root = document.getRootElement();
7. List<Element> list=root.elements("p1");
8. **for** (Element element : list) {
9. Element name1=element.element("name");
10. String s = name1.getText();
11. System.***out***.println(s);
13. }

得到第一个name的值

**public** **static** **void** selectName1() **throws** Exception{

SAXReader saxReader = **new** SAXReader();

Document document = saxReader.read("src/1.xml");

Element root = document.getRootElement();

Element p1 = root.element("p1");

Element name1 = p1.element("name");

String s = name1.getText();

System.***out***.println(s);

}

得到第二个name

\*1、创建解析器

\*2、得到document

\*3、得到根节点

\*4、得到所有的p1

\*\*返回list集合

\*5、遍历得到第二个p1I

\*\*使用list下标得到get方法，集合的下标从0开始，想要得到第二个值，下标写1

\*6、得到第二个p1下面的name

\*7、得到name的值

**public** **static** **void** selectName2() **throws** Exception{

SAXReader saxReader = **new** SAXReader();

Document document = saxReader.read("src/1.xml");

Element root = document.getRootElement();

List<Element> list = root.elements("p1");

Element p2=list.get(1);

Element name2=p2.element("name");

String s = name2.getText();

System.***out***.println(s);

}

7、使用dom4j实现添加操作

<p1>

<name>zhangsan</name>

<age>20</age>

</p1>

<p1>

<name>lisi</name>

<age>30</age>

</p1>

1. 操作在第一个p1标签的末尾添加一个标签<sex></sex>
2. \*1、创建解析器
3. \*2、得到document
4. \*3、得到根节点
5. \*4、获取到第一个p1
6. \*使用element方法
7. \*5、在p1下面添加元素
8. \*在p1上面直接使用addElement（\"标签名称\"）方法返回一个Element
9. \*6、在添加完成之后的元素下面添加文本
10. \*在sex上直接使用setText（\"文本内容\"）方法
11. \*7、回写xml
12. \*格式化outputFormat，使用createprettyPrint方法，表示一个漂亮的格式
13. \*使用类xMLWriter直接new这个类，传递两个参数
14. I\*\*\*第一个参数是xm1文件路径 new Fileoutputstream(\"路径\"）
15. \*\*\*第二个参数是格式化类的值
16. \*\*/

**public** **static** **void** addSex() **throws** Exception {

SAXReader saxReader = **new** SAXReader();

Document document = saxReader.read("src/1.mxl");

Element root = document.getRootElement();

Element p1 = root.element("p1");

Element sex1 = p1.addElement("sex");

sex1.setText("nv");

OutputFormat format = OutputFormat.*createCompactFormat*();

XMLWriter xmlWriter = **new** XMLWriter(**new** FileOutputStream("src/1.xml"),format);

xmlWriter.write(document);

xmlWriter.close();

1. }

8、在特定的位置添加一盒元素

实现 在第一个p1下面的age标签之前调价<school>edit</school>

\*1、创建解析器

\*2、得到document

\*3、得到根节点

\*4、获取到第一个p1

\*5、获取p1下面的所有的元素

\*\*elements（）方法返回list集合

\*\*使用list里面的方法，在特定位置添加元素

\*\*首先创建元素在元素下面创建文本

-使用DocumentHelper类方法createElement创建标签

-把文本添加到标签下面使用setText（\"文本内容”）方法

\*\*\*list集合里面的 add(int index,E element)

-第一个参数是位置下标，从0开始

-第二个参数是要添加的元素

\*6、回写xm1

**public** **static** **void** addAgeBrfore() **throws** Exception{

SAXReader saxReader = **new** SAXReader();

Document document = saxReader.read("src/1.mxl");

Element root = document.getRootElement();

Element p1 = root.element("p1");

List list = p1.elements();

Object element;

Element school =DocumentHelper.*createElement*("school");

school.setText("ecit");

list.add(1, school);

OutputFormat format = OutputFormat.*createPrettyPrint*();

XMLWriter xmlWriter = **new** XMLWriter(**new** FileOutputStream("src/1.xml"),format);

xmlWriter.write(document);

xmlWriter.close();

}

封装

**public** **static** Document getDocument(String path) {

SAXReader reader = **new** SAXReader();

**try** {

Document document =reader.read(path);

**return** document;

} **catch** (DocumentException e) {

// **TODO** 自动生成的 catch 块

e.printStackTrace();

}

**return** **null**;

}

**public** **static** **void** xmlWriters(String path,Document document) {

**try** {

OutputFormat format = OutputFormat.*createPrettyPrint*();

XMLWriter xmlWriter = **new** XMLWriter(**new** FileOutputStream(path),format);

xmlWriter.write(document);

xmlWriter.close();

}**catch**(Exception e) {

e.printStackTrace();

}

9、使用dom4j实现修改节点的操作

\*1、得到document

\*2、得到根节点，然后再得到第一个p1元素

\*3、得到第一个p1下面的age

element(\"\")方法

\*4、修改值是30

\*\*使用setText（\"文本内容\")方法

\*5、回写xml

\*

\*\*/I

**public** **static** **void** modifyAge() **throws** Exception {

Document document = Dom4jUtils.*getDocument*("src/a.xml");

Element root =document.getRootElement();

Element p1 = root.element("p1");

Element age = p1.element("age");

age.setText("300");

Dom4jUtils. *xmlWriters*("src/a.xml", document);

}

快速导包 ctrl shift o

10、使用dom4j实现删除节点的操左

删除第一个p1下面的<school>ecit</school> 的元素

<person>

<p1>

<name>zhangsan</name>

<school>>ecit</school>

<age>20</age>

</p1>

<p1>

<name>lisi</name>

<age>30</age>

</p1>

</person>

\*1、得到document

\*2、得到根节点

\*3、得到第一个p1标签

\*4、得到第一个p1下面的school元素

\*5、删除（使用p1删除school)

\*\*得到school的父节点

-第一种直接得到p1

-使用方法getparent方法得到

\*删除操作

-在p1上面执行remove方法删除节点

\*6、回写xml

**public** **static** **void** delSch() **throws** Exception{

Document document = Dom4jUtils.*getDocument*("src/a.xml");

Element root =document.getRootElement();

Element p1 = root.element("p1");

Element sch = p1.element("school");

p1.remove(sch);

Dom4jUtils. *xmlWriters*("src/a.xml", document);

}

10、使用dom4j获取属性的操作

获取第一标签id1的值

\*1、得到document

\*2、得到根节点

\*3、得到第一个p1元素

\*4、得到p1里面的属性值

-pl.attributeValue(\"id1\");

-在p1上面执行这个方法，里面的参数是属性名称

**public** **static** **void** getValue() **throws** Exception {

Document document = Dom4jUtils.*getDocument*("src/a.xml");

Element root =document.getRootElement();

Element p1 = root.element("p1");

String s = p1.attributeValue("id1");

System.***out***.println(s);

}

12使用dom4j指出xpath的操作（xpath）

可以直接获取某个元素

第一种形式 /AAA/BBB/CCC 表示一层一层的AAA下面DDD下面BBB

第二种形式 //BBB 表示这个名称相同的 表示只要是名称是BBB都得到

//@id id属性相统统都可以得到

第三种形式 /\*表示元素

第四种形式 /AAA/BBB[1] AAA中的第一个BBB元素

/AAA/BBB[last()] AAA中最后一个BBB元素

第五种形式 //BBB[@id]表示只要BBB有id属性都可以得到

第六种形式 //BBB[@id=“aaa”] 表示只要BBB有id属性并且id的属性为“aaa”都可以得到（三个条件）

13、使用dom4j支持xpath具体操作

默认情况下 dom4j里面有xpath

第一步需要 引入指出xpath的jar包，使用jaxen-1.1-beat-6.jar

需要吧jar包导入到项目中

在dom4j里面提供了两个方法，用来支持xpath

selectNodes（“xpath表达式”）

获取多个节点

selectSingleNode（““xpath表达式”）

获取单一的节点

使用xpath查询xml中所有name元素的值

所有name元素 xpath表示//name

**public** **static** **void** test1() **throws** Exception{

Document document = Dom4jUtils.*getDocument*("src/a.xml");

List<Node> list = document.selectNodes("//name");

**for** ( Node node : list) {

String s = node.getText();

System.***out***.println(s);

}

}

使用xpath实现：获取第一个p1下面的值

//p1[@id1]=’aaa’]/name

使用到selectSingleNode（“”//p1[@id1=’aaa’]/name“）单引号

\*1、得到documentI

\*2、直接使用selectsingleNode方法实现

\*-xpath：//p1[@idl='aaaa']/name

”）

**public** **static** **void** test2() **throws** Exception{

Document document = Dom4jUtils.*getDocument*("src/a.xml");

Node name1 = document.selectSingleNode("//p1[@id1='aaaa']/name");

String s1 = name1.getText();

System.***out***.println(s1);

}

14、实现简单的学生管理系统