今日内容

1. XML
    1. 概念
    2. 语法
    3. 解析

XML

概念

> *Extensible Markup Language 可扩展标记语言*

- 可扩展：标签都是自定义的。
- 功能
    - 存储数据
        1. 配置文件
        2. 在网络中传输
- xml与html的区别
    1. xml标签都是自定义的，html标签是预定义。
    2. xml的语法严格，html语法松散
    3. xml是存储数据的，html是展示数据
- w3c:万维网联盟

语法

基本语法

1. xml文档的后缀名 .xml
2. xml第一行必须定义为文档声明，而且注意文档声明之前不能有空格
3. xml文档中有且仅有一个根标签
4. 属性值必须使用引号(单双都可)引起来，比如id="1"
5. 标签必须正确关闭，可以使用自闭合，比如
6. xml标签名称区分大小写

```
<?xml version='1.0' ?>
<users>
    <user id='1'>
        <name>zhangsan</name>
        <age>23</age>
        <gender>male</gender>
        <br/>
```

```
    </user>

    <user id='2'>
        <name>lisi</name>
        <age>24</age>
        <gender>female</gender>
    </user>
</users>
```

组成部分

1. 文档声明
    1. 格式：`<?xml 属性列表 ?>`
    2. 属性列表：
        - version：版本号，必须的属性
        - encoding：编码方式。告知解析引擎当前文档使用的字符集，默认值：ISO-8859-1
        - standalone：是否独立
            - 取值：
                - yes：不依赖其他文件
                - no：依赖其他文件

指令(了解，不用学)：结合css的

- `<?xml-stylesheet type="text/css" href="css路径" ?>`

标签：标签名称自定义的

- 规则：
- 名称不能以数字或者标点符号开始
- 名称可以包含字母、数字以及其他的字符
    - 名称不能以字母 xml（或者 XML、Xml 等等）开始
    - 名称不能包含空格

属性：

- id属性值唯一（虽然这个起名是id，但是其实不是id）

文本：

- CDATA区：在该区域中的数据会被原样展示
    - 格式：`<![CDATA[ 数据 ]]>`

```
<?xml version='1.0' encoding='utf-8'?>
<users>
    <user id='1'>
```

```
        <name>张三</name>
        <age>23</age>
        <gender>male</gender>

        <code>
            <!-- CDATA区，这里面的回原样展示，如果不写在CDATA区，
            那么特殊符号就要用转义字符来进行使用 -->
            <![CDATA[
                if(a<b && a>c)
            ]]>
        </code>
    </user>
</users>
```

约束：规定xml文档的书写规则

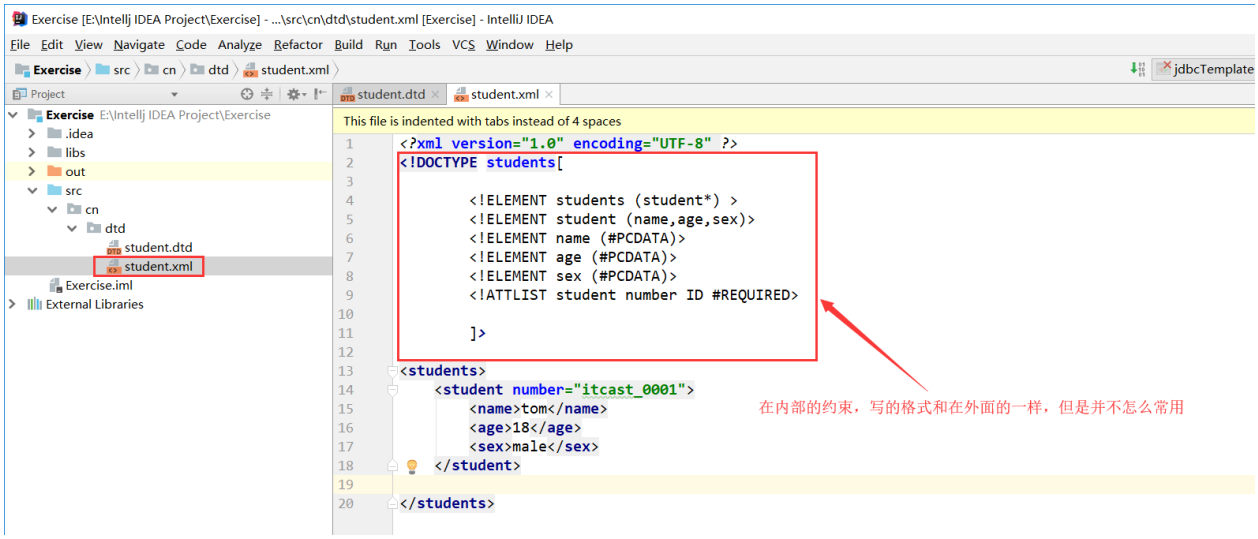
- 作为框架的使用者(程序员)：
    1. 能够在xml中引入约束文档
    2. 能够简单的读懂约束文档
- 分类：
    1. DTD:一种简单的约束技术
    2. Schema:一种复杂的约束技术

DTD：

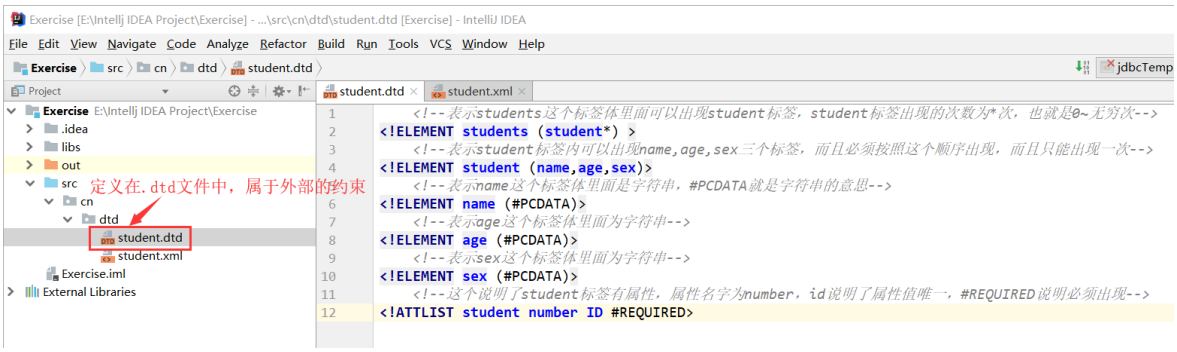
```
    <!--表示students这个标签体里面可以出现student标签，student标签出现的次数为*次，也就是0~无穷次-->
<!ELEMENT students (student*) >
    <!--表示student标签内可以出现name,age,sex三个标签，而且必须按照这个顺序出现，而且只能出现一次-->
    <!ELEMENT student (name,age,sex)>
        <!--表示name这个标签体里面是字符串，#PCDATA就是字符串的意思-->
    <!ELEMENT name (#PCDATA)>
        <!--表示age这个标签体里面为字符串-->
    <!ELEMENT age (#PCDATA)>
        <!--表示sex这个标签体里面为字符串-->
    <!ELEMENT sex (#PCDATA)>
        <!--这个说明了student标签有属性，属性名字为number，id说明了属性值唯一，#REQUIRED说明必须出现-->
    <!ATTLIST student number ID #REQUIRED>
```

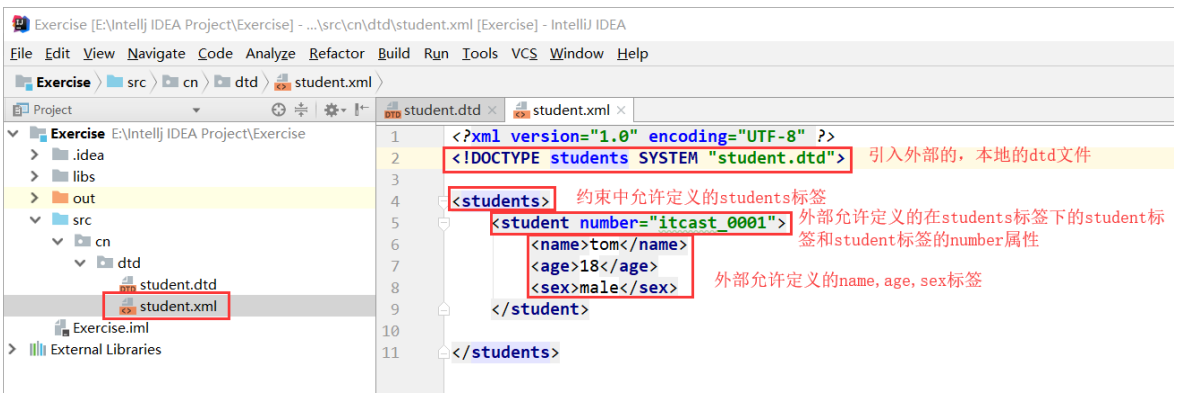
引入dtd文档到xml文档中

- 后缀名：.dtd
- 内部dtd：将约束规则定义在xml文档中



- 外部dtd：将约束的规则定义在外部的dtd文件中
  - 本地：<!DOCTYPE 根标签名 SYSTEM "dtd文件的位置">





  - 网络：<!DOCTYPE 根标签名 PUBLIC "dtd文件名字（随便起）" "dtd文件的位置URL">
  - dtd的缺陷：
    1. 虽然可以进行类型的约束，但是不能约束具体的内容，比如要选择性别，来一个字符串"呵呵"，针对这样的问题，Schema就进行了修改

Schema

- 后缀名：.xsd
- 其实Schema文档本身就是xml文档

> *Schema文档*

```xml
<?xml version="1.0"?>
<xsd:schema xmlns="http://www.itcast.cn/xml"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.itcast.cn/xml"
elementFormDefault="qualified">

    <!--定义了一个元素(element)，元素名字是students，类型(type)为自定义的
studentsType，但是自定义的元素类型需要声明-->
    <xsd:element name="students" type="studentsType"/>

    <!--这就是对studentsType类型进行声明，其中xsd:complexType为复合类型-->
    <xsd:complexType name="studentsType">
        <!--sequence(按顺序出现)-->
        <xsd:sequence>
            <!--student中元素(element)有student，类型为studentType(又是一个新的类
型)，最小出现次数(minOccurs)为0次，最大(maxOccurs)为unbounded(没有指定)-->
            <xsd:element name="student" type="studentType" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <!--对新的类型studentType进行声明-->
    <xsd:complexType name="studentType">

        <!--按顺序出现-->
        <xsd:sequence>
            <!--元素有name,age,sex-->
                <!--类型为xsd:string，这个意思是xsd规定好的类型：字符串-->
            <xsd:element name="name" type="xsd:string"/>
                <!--新的自定义类型ageType-->
            <xsd:element name="age" type="ageType" />
            <!--新的自定义类型sexType-->
            <xsd:element name="sex" type="sexType" />
        </xsd:sequence>

        <!--属性number，类型为自定义类型numberType，use(需求)为required(必须的)-->
        <xsd:attribute name="number" type="numberType" use="required"/>
    </xsd:complexType>

    <!--自定义类型为sexType，xsd:simpleType(简单类型)-->
    <xsd:simpleType name="sexType">

        <!--base(基础格式)为字符串-->
        <xsd:restriction base="xsd:string">
            <!--enumeration(枚举)值的选项之一为male-->
            <xsd:enumeration value="male"/>
            <!--enumeration(枚举),值的选项之一为male-->
            <xsd:enumeration value="female"/>
        </xsd:restriction>
    </xsd:simpleType>
```

```
        <!--自定义类型为ageType，xsd:simpleType(简单类型)-->
        <xsd:simpleType name="ageType">
            <!--base(基础格式)为数字-->
            <xsd:restriction base="xsd:integer">
                <!--最小值(minInclusive)为0-->
                <xsd:minInclusive value="0"/>
                <!--最大值(maxInclusive)为256-->
                <xsd:maxInclusive value="256"/>
            </xsd:restriction>
        </xsd:simpleType>

        <!--自定义类型为numberType，xsd:simpleType(简单类型)-->
        <xsd:simpleType name="numberType">
            <!--base(基础格式)为字符串-->
            <xsd:restriction base="xsd:string">
                <!--pattern(组成格式)必须为heima_四位数的数字，/d在正则中表示数字{4}表示4
位-->
                <xsd:pattern value="heima_\d{4}"/>
            </xsd:restriction>
        </xsd:simpleType>


    </xsd:schema>
```

引入

1.填写xml文档的根元素，如<students></stuents> 2.引入xsi前缀. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"，这是个固定的格式，代表w3c万维网2001的约束，其实这里有很多选择，这里选择2001的 3.引入xsd文件命名空间. xsi:schemaLocation="http://www.itcast.cn/xml student.xsd"，在这里引入文件命名空间，其中xsi:schemaLocation="xxx student.xsd"中，student.xsd表示文件的路径，xxx表示给这个路径起了一个名字，称为名称空间或者命名空间，以后要使用student.xsd里面的东西都要在标签前带一个xxx，用冒号连接

```
<?xml version="1.0" encoding="UTF-8" ?>
 <http://www.itcast.cn/xml:students
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.itcast.cn/xml  student.xsd"
             >

    <http://www.itcast.cn/xml:student number="heima_0001">
    </student>

 </students>
```

4.假如以后要为每一个xsd约束前都加上名称空间太麻烦，所以为每一个xsd约束声明一个前缀,作为标识，xmlns:a="http://www.itcast.cn/xml"，这样代表的是加入了一个把名称空间替换成了a，以后就要在这之前加上一个a前缀

```xml
<?xml version="1.0" encoding="UTF-8" ?>
 <a:http://www.itcast.cn/xml:students
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xmlns:a="http://www.itcast.cn/xml"
            xsi:schemaLocation="http://www.itcast.cn/xml  student.xsd"
             >

    <a:student number="heima_0001">
        <a:name>tom</name>
        <a:age>18</age>
        <a:sex>male</sex>
    </student>

 </students>
```

如果不写就代表空前缀，这也是默认前缀，xmlns="http://www.itcast.cn/xml"

```xml
<?xml version="1.0" encoding="UTF-8" ?>
 <students   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xmlns="http://www.itcast.cn/xml"
            xsi:schemaLocation="http://www.itcast.cn/xml  student.xsd"
             >
    <student number="heima_0001">
        <name>tom</name>
        <age>18</age>
        <sex>male</sex>
    </student>

 </students>
```

前缀的作用就是为了区分xml文档，当以后有了第二个，第三个文档的时候就需要前缀来进行

解析

> *操作xml文档，将文档中的数据读取到内存中*

操作xml文档

1. 解析(读取)：将文档中的数据读取到内存中
2. 写入：将内存中的数据保存到xml文档中。持久化的存储

解析xml的方式

1. DOM：将标记语言文档一次性加载进内存，在内存中形成一颗dom树。一般用于服务端，如pc
    - 优点：操作方便，可以对文档进行CRUD的所有操作
    - 缺点：占内存
2. SAX：逐行读取，基于事件驱动的。一般用于小型的设备，如手机安卓等
    - 优点：不占内存。
    - 缺点：只能读取，不能增删改

xml常见的解析器

1. JAXP：sun公司提供的解析器，支持dom和sax两种思想，性能特低，基本没人用
2. DOM4J：一款非常优秀的解析器
3. Jsoup：jsoup 是一款Java 的HTML解析器，可直接解析某个URL地址、HTML文本内容。它提供了一套非常省力的API，可通过DOM，CSS以及类似于jQuery的操作方法来取出和操作数据。
4. PULL：Android操作系统内置的解析器，sax方式的。

Jsoup

> *jsoup 是一款Java 的HTML解析器，可直接解析某个URL地址、HTML文本内容。它提供了一套非常省力的API，可通过DOM，CSS以及类似于jQuery的操作方法来取出和操作数据。*

- 快速入门：
  - 步骤：
    1. 导入jar包，jsoup-1.11.2.jar
    2. 获取Document对象
    3. 获取对应的标签Element对象
    4. 获取数据

```
package XML;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

import java.io.File;
import java.io.IOException;

public class JsoupDemo1 {

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

//        导入jar包

//         获取Document对象

        /*根据类加载器获取文件路径*/
        String path =
JsoupDemo1.class.getClassLoader().getResource("student.xml").getPath();

        /*解析xml文档，加载文档进内存，获取dom树-->Document*/
        Document document = Jsoup.parse(new File(path), "utf-8");

        /*
        * 获取xml文档中name标签
        * 返回Elements
        * Elements extends ArrayList<Elements>
        * */
```

```
        Elements elements = document.getElementsByTag("name");

//          获取数据

        /*获取第一个name的Element对象*/
        Element element = elements.get(0);

        /*获取数据*/
        String name = element.text();

        System.out.println(name);
    }
}
```

> *几个注意事项：*
>
> 1. *xml文件和xsd文件尽量放在根目录src下面，不要在src目录下的目录里面，因为暂时不会如何去找*
> 2. *项目文件夹的名字不要有空格，中文等，否则使用.getPath()方法会乱码*

对象的使用

1. Jsoup：工具类，可以解析html或xml文档，返回Document
    - parse：解析html或xml文档，返回Document
        - parse(File in, String charsetName)：解析xml或html文件的。
            - in：File对象
            - charsetName：设置字符集编码，如"utf-8"
        - parse(String html)：解析xml或html字符串

          html：就是html或者xml的内容就是<html></html>等标记语言
        - parse(URL url, int timeoutMillis)：通过网络路径获取指定的html或xml的文档对象
            - url：统一资源定位符，URL url = new URL("xxxx");
            - timeoutMillis：超时时间，毫秒值
2. Document：文档对象。代表内存中的dom树
    - 获取Element对象
        - getElementById(String id)：根据id属性值获取唯一的element对象
        - getElementsByTag(String tagName)：根据标签名称获取元素对象集合
        - getElementsByAttribute(String key)：根据属性名称获取元素对象集合
        - getElementsByAttributeValue(String key, String value)：根据对应的属性名和属性值获取元素对象集合

```
<?xml version="1.0" encoding="UTF-8" ?>
 <students>
    <student number="heima_0001">
        <name id="itcast">tom</name>
        <age>18</age>
```

```
        <sex>male</sex>
    </student>

    <student number="heima_0002">
        <name>jack</name>
        <age>19</age>
        <sex>female</sex>
    </student>

 </students>
```

```
package XML;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

import java.io.File;
import java.io.IOException;

public class JsoupDemo1 {

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


        String path =
JsoupDemo1.class.getClassLoader().getResource("student.xml").getPath();

        Document document = Jsoup.parse(new File(path), "utf-8");


        /*通过id的值来获取*/
        Element itcast = document.getElementById("itcast");
        System.out.println(itcast);

        System.out.println("-----------------");

        /*通过标签名来获取*/
        Elements student = document.getElementsByTag("age");
        System.out.println(student);

        System.out.println("-----------------");

        /*通过属性名称来获取*/
        Elements number = document.getElementsByAttribute("number");
        System.out.println(number);

        System.out.println("-----------------");
```

```
        /*通过对应的属性名和属性值来获取*/
        Elements id_itcast = document.getElementsByAttributeValue("id",
 "itcast");
        System.out.println(id_itcast);

        System.out.println("-----------------");
    }
}
```

3. Elements：元素Element对象的集合。可以当做 ArrayList来使用
4. Element：元素对象
5. 获取子元素对象

- getElementById(String id)：根据id属性值获取唯一的element对象
- getElementsByTag(String tagName)：根据标签名称获取元素对象集合
- getElementsByAttribute(String key)：根据属性名称获取元素对象集合
- getElementsByAttributeValue(String key, String value)：根据对应的属性名和属性值获取元素对象集合

> *注意：*
>
> - *Element是获取子标签对象，确切地说并不是获取子元素对象，而应该说是通过Element元素获取对象，因为Element是一个单位，所以获取到的也是Element里面的单位*

2. 获取属性值

- String attr(String key)：根据属性名称获取属性值，属性名不区分大小写

3. 获取文本内容

- String text():获取文本内容
- String html():获取标签体的所有内容(包括子标签的字符串内容)

```
<?xml version="1.0" encoding="UTF-8" ?>
 <students>
  <student number="heima_0001">
      <name id="itcast">tom</name>
      <age>18</age>
      <sex>male</sex>
  </student>

  <student number="heima_0002">
      <name>jack</name>
      <age>19</age>
      <sex>female</sex>
  </student>
```

```
    </students>
```

```
package XML;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

import java.io.File;
import java.io.IOException;

public class JsoupDemo1 {

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


        String path =
JsoupDemo1.class.getClassLoader().getResource("student.xml").getPath();

        Document document = Jsoup.parse(new File(path), "utf-8");

        /*获取到了element对象*/
        Element student = document.getElementsByTag("student").get(0);

            /*通过element对象来获取对象*/
        Elements name = student.getElementsByTag("name");

        System.out.println(name);
        /*只获取到了一个对象，那么就是说获取的的确是子元素对象
          这样的话其实就是根据Document的子元素Element来获取Elements元素
            <name id="itcast">
                tom
            </name>
        */
        System.out.println("-----------------");


        /*通过Element对象来获取student属性值*/
        String number = student.attr("number");
        System.out.println(number);
        /*
            heima_0001
        */
        System.out.println("--------------------");

        /*获取文本内容*/
        String text = student.text();
        String html = student.html();
        System.out.println(text);
```

```
            System.out.println("------");
        System.out.println(html);    //tom 18 male
            System.out.println("-------");
            /*
            <name id="itcast">
             tom
            </name>
            <age>
             18
            </age>
            <sex>
             male
            </sex>
            */
    }
}
```

5. Node：节点对象

- 是Document和Element的父类

快捷查询方式

1. selector:选择器

- 使用的方法：Elements select(String cssQuery)
  - 语法：参考Selector类中定义的语法
  - cssQuery：css选择器

```
<?xml version="1.0" encoding="UTF-8" ?>
 <students>
    <student number="heima_0001">
        <name id="itcast">tom</name>
        <age>18</age>
        <sex>male</sex>
    </student>

    <student number="heima_0002">
        <name>jack</name>
        <age>19</age>
        <sex>female</sex>
    </student>

 </students>
```

```
package XML;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
```

```java
import org.jsoup.select.Elements;

import java.io.File;
import java.io.IOException;

public class JsoupDemo1 {

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

        String path =
JsoupDemo1.class.getClassLoader().getResource("student.xml").getPath();

        Document document = Jsoup.parse(new File(path), "utf-8");

        /*查询age标签*/
        Elements age = document.select("age");
        System.out.println(age);

        System.out.println("------------");

        /*查询id值为itcast的元素*/
        Elements select = document.select("#itcast");
        System.out.println(select);

        System.out.println("------------");

        /*获取student标签并且number属性值为heima_0001的age子标签*/

            /*student标签并且number属性值为heima_0001*/
        Elements select1 = document.select("student[number='heima_0001']");
        System.out.println(select1);

            System.out.println("--------");

            /*student标签下的number属性值为heima_0001的age*/
        Elements select2 = document.select("student[number='heima_0001']>age");
        System.out.println(select2);
    }
}
/*
<age>
 18
</age>
<age>
 19
</age>
------------
<name id="itcast">
 tom
</name>
------------
```

```
<student number="heima_0001">
 <name id="itcast">
  tom
 </name>
 <age>
  18
 </age>
 <sex>
  male
 </sex>
</student>
--------
<age>
 18
</age>
*/
```

2. XPath：XPath即为XML路径语言，它是一种用来确定XML（标准通用标记语言的子集）文档中某部分位置的语言

- 使用Jsoup的Xpath需要额外导入jar包。
- 查询w3cshool参考手册，使用xpath的语法完成查询

```
<?xml version="1.0" encoding="UTF-8" ?>
 <students>
    <student number="heima_0001">
        <name id="itcast">tom</name>
        <age>18</age>
        <sex>male</sex>
    </student>

    <student number="heima_0002">
        <name>jack</name>
        <age>19</age>
        <sex>female</sex>
    </student>

 </students>
```

```
package XML;

import cn.wanghaomiao.xpath.exception.XpathSyntaxErrorException;
import cn.wanghaomiao.xpath.model.JXDocument;
import cn.wanghaomiao.xpath.model.JXNode;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

import java.io.File;
```

```
import java.io.IOException;
import java.util.List;

public class JsoupDemo1 {

    public static void main(String[] args) throws IOException,
XpathSyntaxErrorException {


        String path =
JsoupDemo1.class.getClassLoader().getResource("student.xml").getPath();

        Document document = Jsoup.parse(new File(path), "utf-8");


//          根据document对象来创建Xpath中的JXDocument对象

        /*创建JXDocument对象*/
        JXDocument jxDocument = new JXDocument(document);

        /*结合xpath语法进行查询*/
            /*查询所有的student标签*/
        List<JXNode> jxNodes = jxDocument.selN("//student");

        for (JXNode jxNode : jxNodes) {
            System.out.println(jxNode);//生成的jxNode可以点出Element：Element
element = jxNode.getElement();
        }
        System.out.println("-------------");

            /*查询所有的student下面的name标签*/
        List<JXNode> jxNodes1 = jxDocument.selN("//student/name");
        for (JXNode jxNode : jxNodes1) {
            System.out.println(jxNode);
        }

            /*查询student标签下带有id属性的name标签*/
        List<JXNode> jxNodes2 = jxDocument.selN("//student/name[@id]");

            /*查询student标签下带有id属性的name标签并且id属性值为itcast*/
        List<JXNode> jxNodes3 = jxDocument.selN("//student/name[@id='itcast']");
    }
}
```