# 北京邮电大学软件学院

# 2019-2020学年第1学期实验报告

**课程名称： XML & Web Service**

**实验名称：**  实验一 Using DOM & SAX Parser to Split XML File

**实验完成人：**

**姓名：**\_平雅霓\_\_\_**学号：**\_2017211949\_**成绩：**\_\_\_\_\_\_\_\_

**指导教师：** 吴国仕

**日 期： 2019 年 11 月 25 日**

1. **实验目的**

掌握xml解析的两种方式DOM和SAX，并完成实验说明中的内容。

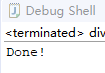
1. **实验内容**
2. 使用DOM模型，将IPO.XML文件分割成两个文件IBM\_COMP.XML 和ABC\_COMP. XML。
3. 使用SAX模型遍历 IPO.XML文件中所有属性信息并在控制台显示出来。
4. 根据自己的编程过程描述DOM和SAX的区别。
5. **实验环境**

Eclipse、Windows10操作系统

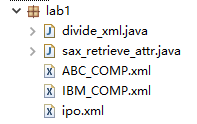
1. **实验结果**
2. **将IPO.xml文件分割成两个文件IBM\_COMP.xml 和ABC\_COMP. Xml**

**注意，运行后请刷新目录，因为输出文件在同级目录下**

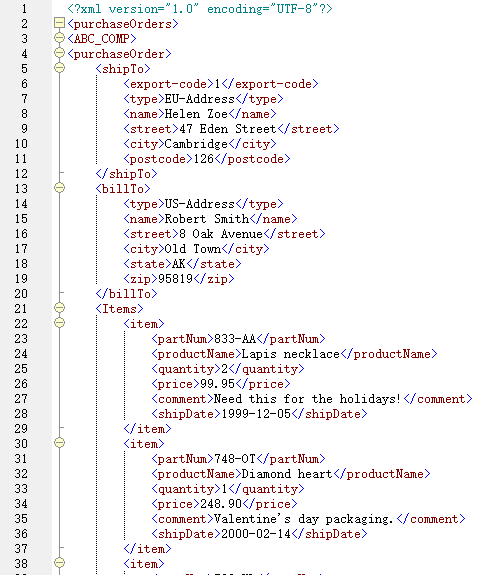
程序运行结束后，控制台显示“Done！”。



在同级目录下生成ABC\_COMP.xml和IBM\_COMP.xml文件。

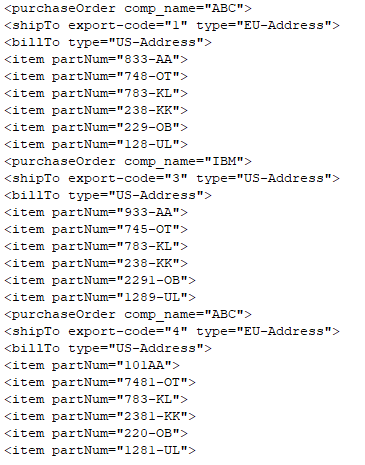


使用XMLSpy软件打开ABC\_COMP.xml和IBM\_COMP.xml文件查看结果如下：





1. **使用SAX模型遍历IPO.XML文件中所有属性信息并在控制台显示出来。**



在控制台输出了所有含有属性的元素以及属性信息。

1. **根据自己的编程过程描述DOM和SAX的区别。**
2. DOM模型会将所有文件读取到内存中形成DOM树，访问的时候相当于是数据结构中树的遍历，访问较慢，不适用于十分大的文件，因为会占很大的内存。

SAX文件是顺序读入所需要的文件内容，不会一次性全部读取，访问十分快，因此可以适用于较大的文件，当遇到一个元素的时候发送一个事件。

1. DOM树在内存中形成，使用”随机访问“协议，可以随意存放或读取文件树，没有次数限制。

SAX模型中只能对文件按顺序从头到尾读取XML文件内容，是事件驱动的解析。

1. DOM模型可以任意修改文件树。

SAX模型但不能修改文件。

1. DOM模型易于理解，易于开发，可以较简单的按树的规则对节点进行访问。

SAX模型需要用户自定义事件处理器，继承了DefaultHandler类，对其中一些方法进行重写。

1. DOM模型中系统为使用者自动建立DOM树，XML对象模型由系统提供。

SAX模型对开发人员更加灵活，可以用SAX建立自己的XML对象模型

1. SAX使用ad-hoc标准，DOM使用W3C标准，SAX因为适用于大文件和访问快的特性，更受欢迎。
2. **附录**

**问题分析与设计方案：**

1. **将IPO.xml文件分割成两个文件IBM\_COMP.xml 和ABC\_COMP. Xml**

在本题目中，要求将IPO.xml文件分割，所以我采用了两个File和FileOutputStream类的对象实现了文件的创建和写入操作，先将文件的前两行分别输入到IBM\_COMP.xml和ABC\_COMP.xml文件中，然后调用childLoop(Node node, String indentation,String outputtype)函数来实现遍历，传入的参数为document。在遍历时第一次为case Node.*DOCUMENT\_NODE*的分支***,***在这个情况分支中实现了从document中筛选出<purchaseOrders>根元素，然后遍历其子节点，它的子节点有3个，即含有3个<purchaseOrder>子元素，然后我们根据<purchaseOrder>的属性comp\_name来判断向哪个文件中输入。

需要注意的是，childLoop(Node node, String indentation,String outputtype)函数传入的参数中有个String outputtype，这个是用来传递写入的文件名的，若传入的为“ABC”则向ABC\_COMP.xml文件中写入信息，若为“IBM”则向IBM\_COMP.xml文件中写入文件。与之一起使用的是outputstream(String type,String data)函数，这个函数用来向文件中写入数据，type用来传递文件类型，可以传递的类型有“ABC”和”IBM”，参数data为需要写入的数据，在childLoop（）函数中调用此函数来向文件中写入内容。

1. **使用SAX模型遍历 IPO.XML文件中所有属性信息并在控制台显示出来。**

在使用SAX解析器时，需要一个处理器来为各种解析器事件定义事件动作，因此我在此java文件中定义了一个类来处理xml文件，它继承了DefaultHandler类，并在该类中重写了若干个在解析文档时解析器会调用的方法。下面是最重要的几个方法：

startElement()和endElement()在每当遇到起始或终止标签时调用。

characters()在每当遇到字符数据时调用。

startDocument()和endDocument()分别在文档开始和结束时各调用一次。

processingInstruction()在处理注释时调用。

warning()，error()，fatalError()用来处理警告和错误。

因为本题目只要求显示属性，而属性包含在element中，所以只需实现startElement()方法即可，其他方法也保留在了代码中，但函数体为空。

在startElement()方法中传入的参数有该元素的Attributes（即属性），所以若要打印所有的属性信息，需要使用Attributes来筛选出含有属性的元素，通过调用attributes. getLength()方法来判断该元素是否有属性，若长度为0则表示该元素无属性，若不为零则表示存在属性，将这个元素和属性信息输出在控制台。

**程序**

1. **分割IPO.xml文件的程序**

**package** lab1;

**import** java.io.\*;

**import** org.xml.sax.\*;

**import** javax.xml.parsers.\*;

**import** org.xml.sax.helpers.DefaultHandler;

**public** **class** sax\_retrieve\_attr **extends** DefaultHandler {

**static** **int** *numberLines* = 0;

**static** String *indentation* = "";

**static** String *displayText*[] = **new** String[1000];

**static** **boolean** *displayBoolean*;

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

sax\_retrieve\_attr obj = **new** sax\_retrieve\_attr();

obj.childLoop("src/lab1/ipo.xml");

**for** (**int** index = 0; index < *numberLines*; index++) {

System.***out***.println(*displayText*[index]);

}

}

**public** **void** childLoop(String uri) {

DefaultHandler saxHandler = **this**;

SAXParserFactory saxFactory = SAXParserFactory.*newInstance*();

**try** {

SAXParser saxParser = saxFactory.newSAXParser();

saxParser.parse(**new** File(uri), saxHandler);

} **catch** (Throwable t) {

}

}

**public** **void** startDocument() {

}

**public** **void** processingInstruction(String target, String data) {

}

**public** **void** startElement(String uri, String localName,

String qualifiedName, Attributes attributes) {

*displayBoolean* = **true**;

**if** (*displayBoolean*) {

**if** (attributes.getLength() != 0) {

*displayText*[*numberLines*] = "";

*displayText*[*numberLines*] += '<';

*displayText*[*numberLines*] += qualifiedName;

**int** numberAttributes = attributes.getLength();

**for** (**int** loopIndex = 0; loopIndex < numberAttributes; loopIndex++) {

*displayText*[*numberLines*] += ' ';

*displayText*[*numberLines*] += attributes.getQName(loopIndex);

*displayText*[*numberLines*] += "=\"";

*displayText*[*numberLines*] += attributes.getValue(loopIndex);

*displayText*[*numberLines*] += '"';

}

*displayText*[*numberLines*] += '>';

*numberLines*++;

}

}

}

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

}

**public** **void** endElement(String uri, String localName, String qualifiedName) {

}

**public** **void** warning(SAXParseException exception) {

System.***err***.println("Warning: " + exception.getMessage());

}

**public** **void** error(SAXParseException exception) {

System.***err***.println("Error: " + exception.getMessage());

}

**public** **void** fatalError(SAXParseException exception) {

System.***err***.println("Fatal error: " + exception.getMessage());

}

}

1. **输出所有属性信息的程序**

**package** lab1;

**import** java.io.\*;

**import** org.xml.sax.\*;

**import** javax.xml.parsers.\*;

**import** org.xml.sax.helpers.DefaultHandler;

**public** **class** sax\_retrieve\_attr **extends** DefaultHandler {

**static** **int** *numberLines* = 0;

**static** String *indentation* = "";

**static** String *displayText*[] = **new** String[1000];

**static** **boolean** *displayBoolean*;

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

sax\_retrieve\_attr obj = **new** sax\_retrieve\_attr();

obj.childLoop("src/lab1/ipo.xml");

**for** (**int** index = 0; index < *numberLines*; index++) {

System.***out***.println(*displayText*[index]);

}

}

**public** **void** childLoop(String uri) {

DefaultHandler saxHandler = **this**;

SAXParserFactory saxFactory = SAXParserFactory.*newInstance*();

**try** {

SAXParser saxParser = saxFactory.newSAXParser();

saxParser.parse(**new** File(uri), saxHandler);

} **catch** (Throwable t) {

}

}

**public** **void** startDocument() {

}

**public** **void** processingInstruction(String target, String data) {

}

**public** **void** startElement(String uri, String localName,

String qualifiedName, Attributes attributes) {

*displayBoolean* = **true**;

**if** (*displayBoolean*) {

**if** (attributes.getLength() != 0) {

*displayText*[*numberLines*] = "";

*displayText*[*numberLines*] += '<';

*displayText*[*numberLines*] += qualifiedName;

**int** numberAttributes = attributes.getLength();

**for** (**int** loopIndex = 0; loopIndex < numberAttributes; loopIndex++) {

*displayText*[*numberLines*] += ' ';

*displayText*[*numberLines*] += attributes.getQName(loopIndex);

*displayText*[*numberLines*] += "=\"";

*displayText*[*numberLines*] += attributes.getValue(loopIndex);

*displayText*[*numberLines*] += '"';

}

*displayText*[*numberLines*] += '>';

*numberLines*++;

}

}

}

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

}

**public** **void** endElement(String uri, String localName, String qualifiedName) {

}

**public** **void** warning(SAXParseException exception) {

System.***err***.println("Warning: " + exception.getMessage());

}

**public** **void** error(SAXParseException exception) {

System.***err***.println("Error: " + exception.getMessage());

}

**public** **void** fatalError(SAXParseException exception) {

System.***err***.println("Fatal error: " + exception.getMessage());

}

}

**调试心得**

通过本次实验，我对xml的解析有了更深的了解，对xml解析的两种方式DOM和SAX有了更深的理解。在使用DOM模型的时候，对树的遍历和递归有了更深的感悟。在使用SAX模型的时候，明白了SAX的顺序访问是如何进行的。