# 北京邮电大学软件学院

# 2020-2021学年第1学期实验报告

**课程名称： XML & Web Service**

**实验名称： 实验1 基于Java语言的DOM & SAX实验**

**实验完成人：**

**指导教师：**\_\_\_\_\_\_\_\_\_\_吴国仕、傅湘玲\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**日 期： 2020 年 11 月 22 日**

1. **实验目的**
2. 理解DOM和SAX的运行原理。
3. 通过实验，基于Java编程语言，学习DOM和SAX对XML文件进行解析的语法和操作。
4. **实验内容**

根据实验提供的ipo.xml、ABC\_COMP.xml、IBM\_COMP.xml三个文件，在Java环境下对此文件进行如下操作：

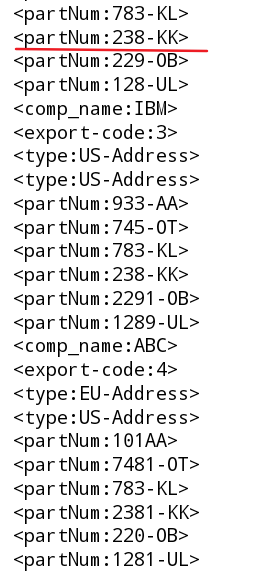
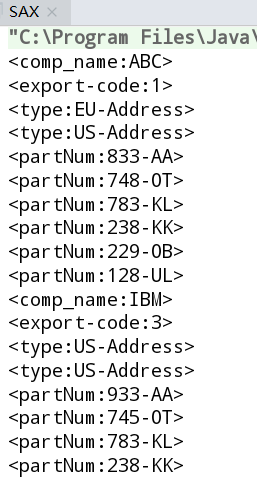
* 1. 使用**SAX模型**对ipo.xml中的所有属性信息进行提取，并以<key : value>的形式输出结果；
  2. 使用**DOM模型**，以ipo.xml文件中的 “**comp\_name**”属性的值作为区分，将ipo.xml中的对应内容分别保存到在ABC\_COMP.xml和IBM\_COMP.xml中；
  3. 根据你的实验过程，在实验报告中写出你对DOM和SAX模型的理解。

1. **实验环境**

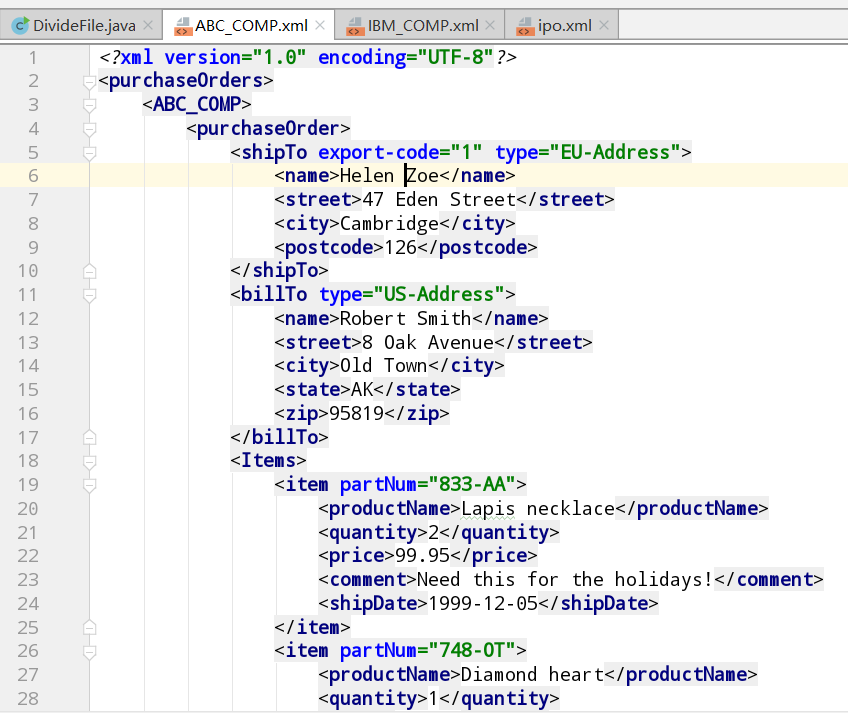
Windows10 Java语言 IDEA

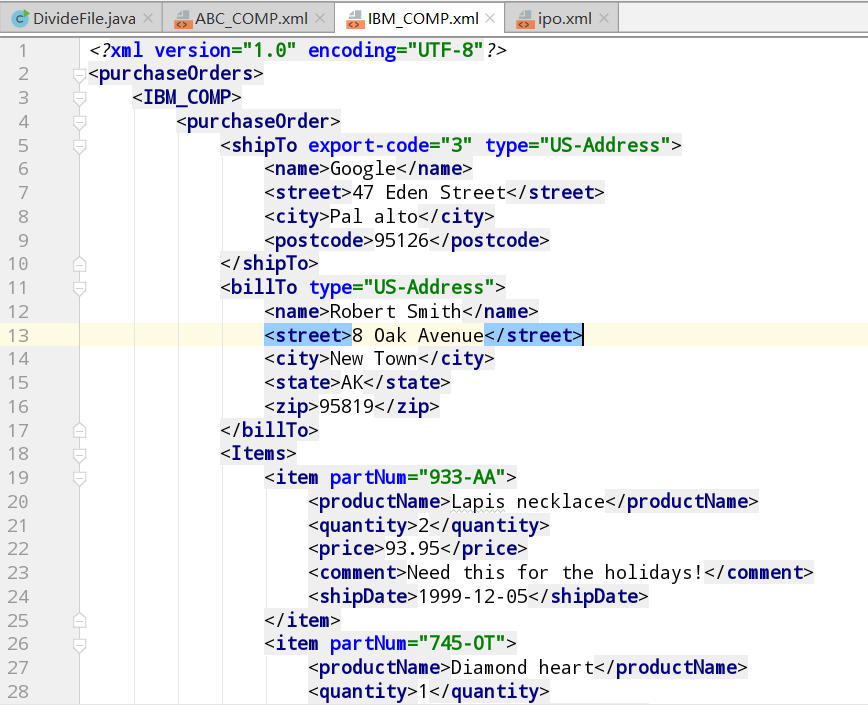
1. **实验结果**
2. 使用SAX模型对ipo.xml中的所有属性信息进行提取

* 在控制台以<key : value>的形式输出结果。



1. 使用DOM模型，将ipo.xml中的对应内容分别保存到在ABC\_COMP.xml和IBM\_COMP.xml中；

* ABC\_COMP.xml和IBM\_COMP.xml中的部分内容如下(格式化后的显示结果)



1. **对DOM和SAX的理解**

DOM模型会将XML文件读取到内存中形成DOM树，可以随意读取文件树中的任何部分，或者修改文件树中的内容然后进行保存等操作。但是不适用于十分大的文件，因为会占很大的内存。

SAX模型是将XML文档进行快速扫描，当遇到一个元素就发送一个事件进行处理。访问十分快，因此可以适用于较大的文件。但是只能对文件顺序解析一遍然后读取，不支持对文件的随意存取与修改。

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，目前大部分浏览器都使用DOM来对HTML进行解析，方便在形成的DOM树中插入CSS文件信息。

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，用户可以自定义输出结果

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。

1. **实现讲解**

* ***SAX提取 ipo.xml中的信息***

自定义MySAXhandler类，它继承了DefaultHandler类，之后复写startElement()

方法，在该方法中创建一个循环，按照自定义格式连续输出属性信息。

* ***DOM将 ipo.xml中的信息分别存储到对应文档中***

将ipo.xml文件中的信息读到程序中，将其名为purchaseOrder的节点存到purchaseOrderList中，之后创建ABC\_COMP与IBM\_COMP两个元素，循环遍历purchaseOrderList中的节点信息，如果comp\_name属性值为:”ABC”则将其存入到ABC\_COMP元素中，如果comp\_name属性值为:”IBM”则将其存入到IBM\_COMP元素中，在进行一些元素添加处理后，使用TransformerFactory将ABC\_COMP和IBM\_COMP中的信息存入到对应xml文件中，最后使用IDEA对xml进行格式化显示。

1. **源代码**
2. ***SAX\_* *experiment：***
3. **import** javax.xml.parsers.\*;
4. **import** org.xml.sax.\*;
5. **import** org.xml.sax.helpers.\*;
7. **public** **class** SAX {
8. **public** **static** **void** main(String args[])
9. **throws** Exception
10. {
11. // 创建解析器工厂
12. SAXParserFactory factory = SAXParserFactory.newInstance();
13. factory.setNamespaceAware(**true**);
14. // 创建解析器
15. SAXParser saxParser = factory.newSAXParser();
16. XMLReader parser = saxParser.getXMLReader();
17. // 创建处理器
18. MySAXhandler mySAXhandler = **new** MySAXhandler();
19. parser.setContentHandler(mySAXhandler);
20. // 读取解析ipo.xml文件
21. parser.parse("ipo.xml");
22. }
24. }
26. // 自定义处理器
27. **class** MySAXhandler **extends** DefaultHandler{
28. **public** **void** startElement(String namespaceURI, String locakName, String qualifiedName, Attributes attributes) **throws** SAXException {
29. // 循环输出属性名和属性值<key:value>
30. **for** (**int** i = 0; i < attributes.getLength(); i++) {
31. System.out.println("<" + attributes.getLocalName(i) + ":" + attributes.getValue(i) + ">");
32. }
33. }
34. }
35. ***DOM\_ experiment：***
36. **import** javax.xml.parsers.\*;
37. **import** org.w3c.dom.\*;
38. **import** javax.xml.transform.\*;
39. **import** javax.xml.transform.dom.DOMSource;
40. **import** javax.xml.transform.stream.StreamResult;
41. **import** java.io.File;

44. **public** **class** DOM\_experiment {
45. **public** **static** **void** main(String[] args) {
46. **try**{
47. // 创建文档构建器工厂
48. DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
49. DocumentBuilder builder = factory.newDocumentBuilder();
51. // 为ipo.xml构建文件解析器
52. Document ipo\_doc = builder.parse("ipo.xml");
53. NodeList purchaseOrderList = ipo\_doc.getElementsByTagName("purchaseOrder");
54. // 创建元素
55. Element ABC\_COMP = ipo\_doc.createElement("ABC\_COMP");
56. Element IBM\_COMP = ipo\_doc.createElement("IBM\_COMP");
58. **for** (**int** i = 0; i < purchaseOrderList.getLength(); i++) {
59. Node purchaseOrderNode = purchaseOrderList.item(i);
61. // 如果<purchaseOrder>第一个属性的值为"ABC"
62. **if** (purchaseOrderNode.getAttributes().item(0 ).getNodeValue().equals("ABC")){
63. childLoop(purchaseOrderNode, ipo\_doc, ABC\_COMP);
64. }
65. // 如果<purchaseOrder>第一个属性的值为"IBM"
66. **else** **if** (purchaseOrderNode.getAttributes().item(0 ).getNodeValue().equals("IBM")){
67. childLoop(purchaseOrderNode, ipo\_doc, IBM\_COMP);
68. }
69. }
71. Element ABC\_purchaseOrders = ipo\_doc.createElement("purchaseOrders");
72. ABC\_purchaseOrders.appendChild(ABC\_COMP);
74. Element IBM\_purchaseOrders = ipo\_doc.createElement("purchaseOrders");
75. IBM\_purchaseOrders.appendChild(IBM\_COMP);
77. // 将获取的信息存放到对应的文件中
78. TransformerFactory transformerFactory = TransformerFactory.newInstance();
79. Transformer transformer = transformerFactory.newTransformer();
80. transformer.transform(**new** DOMSource(ABC\_purchaseOrders), **new** StreamResult(**new** File("ABC\_COMP.xml")));
81. transformer.transform(**new** DOMSource(IBM\_purchaseOrders), **new** StreamResult(**new** File("IBM\_COMP.xml")));
83. // 输出后的文件在格式化后更容易观察, 在IDEA中按下ctrl+alt+l 使.xml文件格式化显示
85. }**catch**(Exception e){
86. System.out.println(e);
87. }
88. }
90. **public** **static** **void** childLoop(Node purchaseOrderNode, Document ipo\_doc, Element comp\_name){
91. // 创建一个ABC\_purchaseOrder 用于存放purchaseOrder所有的子节点
92. Element ABC\_purchaseOrder = ipo\_doc.createElement("purchaseOrder");
93. Element text = ipo\_doc.createElement("test");
95. // 创建一个purchaseOrder子节点的指针
96. Node nextElement = purchaseOrderNode.getFirstChild().getNextSibling();
97. Node nextText = purchaseOrderNode.getFirstChild();
99. **while**(nextElement != **null**){
100. // 将purchaseOrder子节点 放入到ABC\_purchaseOrder中
101. ABC\_purchaseOrder.appendChild(nextElement);
102. text.appendChild(nextText);
104. // 取下一个purchaseOrder的子节点
105. nextElement = purchaseOrderNode.getFirstChild().getNextSibling();
106. nextText = purchaseOrderNode.getFirstChild();
107. }
108. // 将ABC\_purchaseOrder放入
109. comp\_name.appendChild(ABC\_purchaseOrder);
110. }
111. }