# 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2 基于Java编程语言的DOM & SAX实验**

**实验完成人：**

**姓名：**\_\_\_\_王衔飞\_\_\_\_**学号：**\_\_\_\_\_\_\_\_**成绩：**\_\_\_\_\_\_\_\_

**指导教师：**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**日 期： 2020 年 9 月 10 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**

本实验提供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**

Intellij idea 2020 , zulu-jdk15（说明本实验需要的环境）

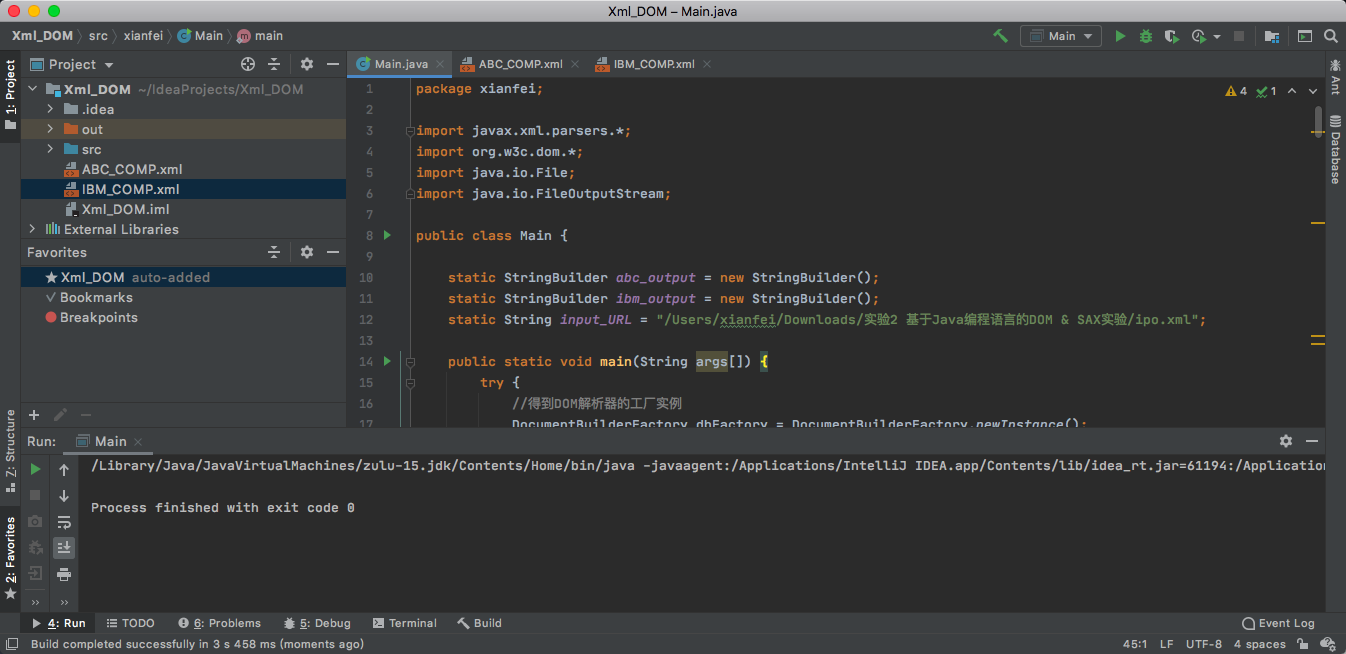
1. **实验结果**
2. SAX模型对ipo.xml中的所有属性信息进行提取

文本

描述已自动生成

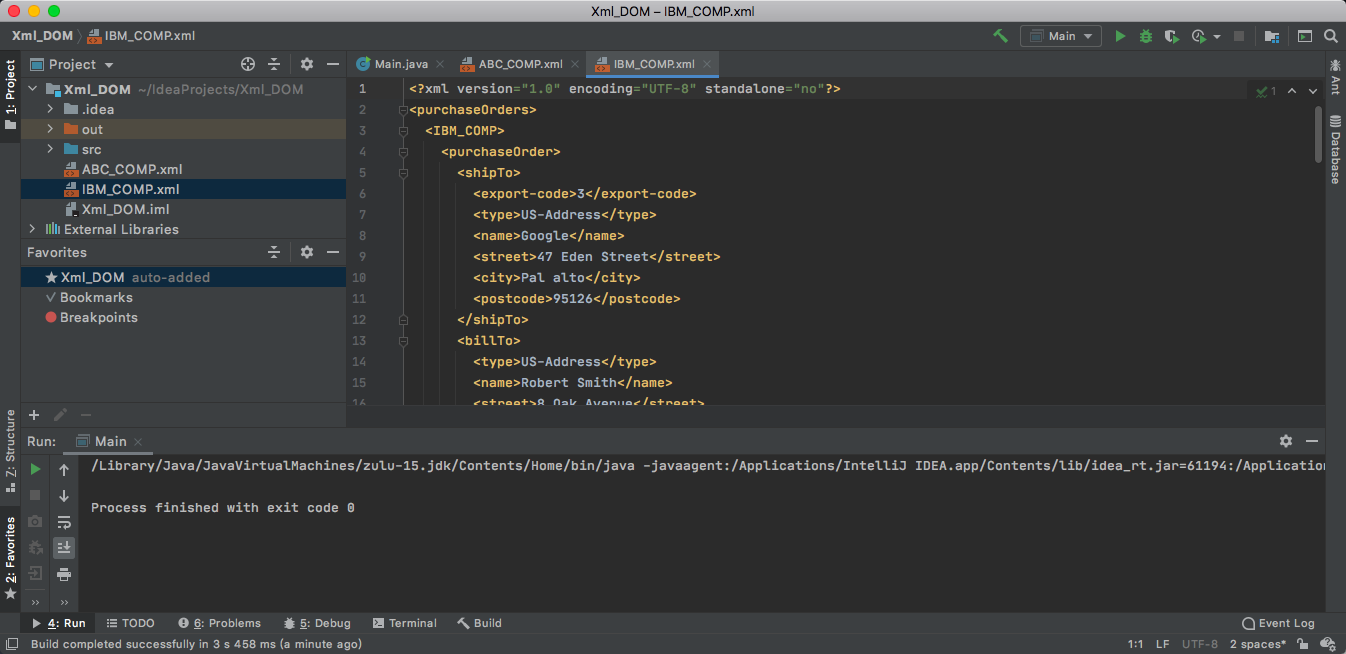
完成实验要求，输出结果和代码见附录。

1. 使用**DOM模型**，以ipo.xml文件中的 “**comp\_name**”属性的值作为区分



文本

描述已自动生成



完成实验要求，输出与给出的示例文件一致

心得：通过本次实验，学会了如何使用SAX和DOM模型在Java语言中对XML文件进行分析，并执行响应的操作。

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

代码：

1. **package** xianfei;
3. **import** java.io.\*;
4. **import** org.xml.sax.\*;
5. **import** javax.xml.parsers.\*;
6. **import** org.xml.sax.helpers.DefaultHandler;
8. **public** **class** Main **extends** DefaultHandler {
9. **static** String inputURL = "/Users/xianfei/Downloads/实验2 基于Java编程语言的DOM & SAX实验/ipo.xml";
11. **public** **static** **void** main(String args[]) {
12. SAXParserFactory saxFactory = SAXParserFactory.newInstance();
13. **try** {
14. SAXParser saxParser = saxFactory.newSAXParser();
15. saxParser.parse(**new** File(inputURL), **new** Main());
16. } **catch** (Throwable ignored) {
17. }
18. }
20. **public** **void** startElement(String uri, String localName,
21. String qualifiedName, Attributes attributes) {
22. **if** (attributes != **null**) {
23. **int** numberAttributes = attributes.getLength();
24. **for** (**int** loopIndex = 0; loopIndex < numberAttributes; loopIndex++) {
25. System.out.print("<");
26. System.out.print(attributes.getQName(loopIndex));
27. System.out.print(":");
28. System.out.print(attributes.getValue(loopIndex));
29. System.out.println(">");
30. }
31. }
32. }
33. }

输出结果：

<comp\_name:ABC>

<export-code:1>

<type:EU-Address>

<type:US-Address>

<partNum:833-AA>

<partNum:748-OT>

<partNum:783-KL>

<partNum:238-KK>

<partNum:229-OB>

<partNum:128-UL>

<comp\_name:IBM>

<export-code:3>

<type:US-Address>

<type:US-Address>

<partNum:933-AA>

<partNum:745-OT>

<partNum:783-KL>

<partNum:238-KK>

<partNum:2291-OB>

<partNum:1289-UL>

<comp\_name:ABC>

<export-code:4>

<type:EU-Address>

<type:US-Address>

<partNum:101AA>

<partNum:7481-OT>

<partNum:783-KL>

<partNum:2381-KK>

<partNum:220-OB>

<partNum:1281-UL>

Process finished with exit code 0

1. DOM模型 代码

输出文件见项目文件夹。

1. **package** xianfei;
3. **import** javax.xml.parsers.\*;
4. **import** org.w3c.dom.\*;
5. **import** java.io.File;
6. **import** java.io.FileOutputStream;
8. **public** **class** Main {
10. **static** StringBuilder abc\_output = **new** StringBuilder();
11. **static** StringBuilder ibm\_output = **new** StringBuilder();
12. **static** String input\_URL = "/Users/xianfei/Downloads/实验2 基于Java编程语言的DOM & SAX实验/ipo.xml";
14. **public** **static** **void** main(String args[]) {
15. **try** {
16. //得到DOM解析器的工厂实例
17. DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
18. //从DOM工厂中获得DOM解析器
19. DocumentBuilder dbBuilder = dbFactory.newDocumentBuilder();
20. //把要解析的xml文档读入DOM解析器
21. Document document = dbBuilder.parse(**new** File(input\_URL));
22. abc\_output.append("<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?>\n" + "<purchaseOrders>\n" +
23. "  <ABC\_COMP>\n" );
24. ibm\_output.append("<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?>\n"+"<purchaseOrders>\n" +
25. "  <IBM\_COMP>\n");
26. // 处理根节点
27. basedLoop(document);
28. abc\_output.append("  </ABC\_COMP>\n" +
29. "</purchaseOrders>\n");
30. ibm\_output.append("  </IBM\_COMP>\n" +
31. "</purchaseOrders>\n");
32. } **catch** (Exception e) {
33. e.printStackTrace();
34. }
35. **try** {
36. FileOutputStream abc\_fos = **new** FileOutputStream("ABC\_COMP.xml");
37. abc\_fos.write(abc\_output.toString().getBytes());
38. abc\_fos.close();
39. FileOutputStream ibm\_fos = **new** FileOutputStream("IBM\_COMP.xml");
40. ibm\_fos.write(ibm\_output.toString().getBytes());
41. ibm\_fos.close();
42. } **catch** (Exception e) {
43. e.printStackTrace();
44. }
45. }
47. **private** **static** **void** basedLoop(Node node) { // 处理根节点
48. **if** (node == **null**) {
49. **return**;
50. }
51. **int** type = node.getNodeType();
52. **switch** (type) {
53. **case** Node.DOCUMENT\_NODE ->
54. basedLoop(((Document) node).getDocumentElement());
55. **case** Node.ELEMENT\_NODE -> {
56. **boolean** isAbc = **false**, isIBM = **false**;
57. **int** length = (node.getAttributes() != **null**) ? node.getAttributes().getLength() : 0;
58. Attr attributes[] = **new** Attr[length];
59. **for** (**int** loopIndex = 0; loopIndex < length; loopIndex++)
60. attributes[loopIndex] = (Attr) node.getAttributes().item(loopIndex);
61. // 检测是abc还是ibm
62. **for** (Attr attribute : attributes) {
63. **if** (attribute.getNodeName().equals("comp\_name") && attribute.getNodeValue().equals("ABC"))
64. isAbc = **true**;
65. **if** (attribute.getNodeName().equals("comp\_name") && attribute.getNodeValue().equals("IBM"))
66. isIBM = **true**;
67. }
68. **if** (isAbc) {
69. processChild(node, abc\_output);
70. } **else** **if** (isIBM) {
71. processChild(node, ibm\_output);
72. } **else** {
73. NodeList childNodes = node.getChildNodes();
74. **if** (childNodes != **null**) {
75. length = childNodes.getLength();
76. **for** (**int** loopIndex = 0; loopIndex < length; loopIndex++) {
77. basedLoop(childNodes.item(loopIndex));
78. }
79. }
80. }
81. }
82. }
83. }
85. **private** **static** **void** processChild(Node node, StringBuilder abc\_output) {
86. **int** length;
87. abc\_output.append("    <purchaseOrder>");
88. NodeList childNodes = node.getChildNodes();
89. **if** (childNodes != **null**) {
90. length = childNodes.getLength();
91. **for** (**int** loopIndex = 0; loopIndex < length; loopIndex++) {
92. childLoop(abc\_output, childNodes.item(loopIndex), "      ");
93. }
94. abc\_output.append("\n    </purchaseOrder>\n");
95. }
96. }
98. **static** **boolean** isNewLine = **true**;
100. **private** **static** **void** childLoop(StringBuilder target, Node node, String indentation) {
101. **if** (node == **null**) **return**;
102. **int** type = node.getNodeType();
103. **switch** (type) {
104. **case** Node.ATTRIBUTE\_NODE, Node.ELEMENT\_NODE -> { // 对于属性和元素节点
105. target.append("\n").append(indentation); // 换行与缩进
106. target.append("<");
107. target.append(node.getNodeName());
108. target.append(">");
110. **int** length = (node.getAttributes() != **null**) ?
111. node.getAttributes().getLength() : 0;
112. Attr attributes[] = **new** Attr[length];
113. **for** (**int** loopIndex = 0; loopIndex < length; loopIndex++)
114. attributes[loopIndex] = (Attr) node.getAttributes().item(loopIndex);
116. **for** (**int** loopIndex = 0; loopIndex < attributes.length;
117. loopIndex++) {
118. target.append("\n").append(indentation).append("  "); // 换行与缩进
119. Attr attribute = attributes[loopIndex];
120. target.append("<");
121. target.append(attribute.getNodeName());
122. target.append(">");
123. target.append(attribute.getNodeValue());
124. target.append("</");
125. target.append(attribute.getNodeName());
126. target.append(">");
127. }
128. NodeList childNodes = node.getChildNodes();
129. **if** (childNodes != **null**) {
130. length = childNodes.getLength();
131. **for** (**int** loopIndex = 0; loopIndex < length; loopIndex++)
132. childLoop(target, childNodes.item(loopIndex), indentation + "  ");
133. }
134. **if** (isNewLine) target.append("\n").append(indentation); // 换行与缩进
135. isNewLine = **true**;
136. target.append("</");
137. target.append(node.getNodeName());
138. target.append(">");
139. }
140. **case** Node.TEXT\_NODE -> { // 对于文字
141. String trimmedText = node.getNodeValue().trim();
142. **if** (trimmedText.length() > 0) {
143. target.append(trimmedText);
144. isNewLine = **false**;
145. }
146. }
147. }
148. }
149. }