链滴

java通过DOM操作xml

作者：crick77
原文链接：https://ld246.com/article/1450771793427
来源网站：链滴

XML

XML (eXtensible Markup Language), 可扩展标记语言,发明之初是为了取代HTML，但在使用过程，开发者发现这种规范的语言格式，在数据传输方面有着明显的优势。

这里只将XML作为一种数据交换格式。

比如说java语言本身的javaBean数据，在程序内使用没有问题，但是如果涉及到与其他语言进行交互则会出现很多问题。所以通过XML进行数据传输通信，可以拥有更好的跨平台性和可移植性，并且让层数据预备了可读性。

本篇文章的重点在于介绍如果通过java DOM对XML文件进行解析与操作。

dom

DOM是W3C处理XML的标准API，多种语言都实现了该标准，java对dom的实现在org.w3c.dom包。很多工具类都是在此基础上进行了封装和扩充，如jdom、dom4j等，这里使用原生实现来完成对xm文档的基本操作。

DOM的实现原理是将XML作为树结构全部读入内存，再进行操作。好处是简单快捷，可以修改结构数据，而造成的隐患则是是在读取大型XML文件时，可能会造成过多的内存占用。

代码

读取解析xml文件

需要读取的xml文件如下，传递了商品订单信息，包括商品名、价格、购买数量。通过程序读取数据并计算出订单总价格。

因为本次重点在于xml的解析操作，所以价格直接用float类型处理。如果是生产环境，一定要使用Big ecimal操作，避免float的精度问题！！

```
<?xml version="1.0" encoding="UTF-8" ?>
<shopping>
   <goods>
      <name>品名1</name>
      <price>3</price>
      <number>4</number>
   </goods>
   <goods>
      <name>品名2</name>
      <price>1.2</price>
      <number>3</number>
   </goods>
</shopping>
```

java 读取XML代码

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

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class XmlParser {

    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

    public Document parseDoc(String filePath) {
        Document document = null;
        try {
            DocumentBuilder builder = factory.newDocumentBuilder();
            document = builder.parse(new File(filePath));
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        } catch (SAXException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return document;
    }

    public static void main(String[] args) {
        XmlParser parser = new XmlParser();
        Document document = parser.parseDoc("D://shopping.xml");
        Element rootElement = document.getDocumentElement();
        List<Goods> goodsList = new ArrayList<Goods>();
        NodeList goodsNodeList = rootElement.getElementsByTagName("goods");
        for (int i = 0; i < goodsNodeList.getLength(); i++) {
            Element child = (Element) goodsNodeList.item(i);
            Goods goods = new Goods(child);
            goodsList.add(goods);
        }

//        NodeList goodsNodeList = rootElement.getChildNodes();
//        for (int i = 0; i < goodsNodeList.getLength(); i++) {
//            Node node = goodsNodeList.item(i);
//            if (node.getNodeType() == Node.ELEMENT_NODE) {
//                Element child = (Element) node;
//                Goods goods = new Goods(child);
//                goodsList.add(goods);
//            }
//        }
        float total = 0;
        int sum = 0;
        for (Goods goods : goodsList) {
            total += goods.getTotal();
            sum += goods.getNumber();
```

```
        }
        System.out.println(total);
        System.out.println(sum);
    }

    static class Goods {
        private float price;
        private int number;
        public Goods(Element element) {
            this.price = Float.parseFloat(element.getElementsByTagName("price").item(0).getText
ontent());
            this.number = Integer.parseInt(element.getElementsByTagName("number").item(0).ge
TextContent());
        }
        public float getTotal(){
            return this.price * this.number;
        }
        public int getNumber(){
            return number;
        }
    }
}
```

node和element的关系

element一定是node但是node不一定是element，node可能是元素节点、属性节点、文本节点，而e ement表示包含开始标签和结束标签的完整元素。

所以上面的代码中 用

```
NodeList goodsNodeList = rootElement.getElementsByTagName("goods");
```

获取了NodeList，可以直接转型为Element： Element child = (Element) node;

如果获取的是node节点

```
NodeList goodsNodeList = rootElement.getChildNodes();
```

则必须在循环中增加判断if (node.getNodeType() == Node.ELEMENT_NODE) {} 判断当前节点是为Element元素。

生成xml文件

将统计后的订单信息以xml格式输出，生成文件格式如下

```
<?xml version="1.0" encoding="utf-8"?>
<order>
    <total>15.6</total>
    <sums>7</sums>
</order>
```

生成xml的操作和读取的顺序类似，先创建rootElement，然后添加childElement，再将rootElemen

放到document中，最后通过io输出xml文件到指定路径。

代码如下：

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class XmlParser {

    DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance(
;
    TransformerFactory transformerFactory = TransformerFactory.newInstance();

    public Document parseDoc(String filePath) {
        Document document = null;
        try {
            DocumentBuilder builder = documentBuilderFactory.newDocumentBuilder();
            document = builder.parse(new File(filePath));
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        } catch (SAXException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return document;
    }

    public void generateXml(String filePath, Document document){
        DOMSource source = new DOMSource(document);
        Transformer transformer = createtransformer();
        PrintWriter pw = null;
        try {
            pw = new PrintWriter(new FileOutputStream(filePath));
            StreamResult result = new StreamResult(pw);
```

```
            transformer.transform(source, result);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (TransformerConfigurationException e1) {
            e1.printStackTrace();
        } catch (TransformerException e) {
            e.printStackTrace();
        } finally {
            pw.close();
        }
    }

    public Document createDoc() {
        Document document = null;
        try {
            DocumentBuilder builder = documentBuilderFactory.newDocumentBuilder();
            document = builder.newDocument();
            document.setXmlStandalone(true);
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        }
        return document;
    }

    public Transformer createtransformer(){
        Transformer transformer = null;
        try {
            transformer = transformerFactory.newTransformer();
            //default former
//            transformer.setOutputProperty(OutputKeys.STANDALONE, "yes");
            transformer.setOutputProperty(OutputKeys.ENCODING, "utf-8");
            transformer.setOutputProperty(OutputKeys.INDENT, "yes");
        } catch (TransformerConfigurationException e) {
            e.printStackTrace();
        }
        return transformer;
    }

    public static void main(String[] args) {
        XmlParser parser = new XmlParser();
        Document document = parser.parseDoc("D://shopping.xml");
        Element rootElement = document.getDocumentElement();
        List<Goods> goodsList = new ArrayList<Goods>();
        NodeList goodsNodeList = rootElement.getElementsByTagName("goods");
        for (int i = 0; i < goodsNodeList.getLength(); i++) {
            Element child = (Element) goodsNodeList.item(i);
            Goods goods = new Goods(child);
            goodsList.add(goods);
        }
        float total = 0;
        int sum = 0;
        for (Goods goods : goodsList) {
            total += goods.getTotal();
            sum += goods.getNumber();
```

```
        }
        Document orderDocument = parser.createDoc();
        Order order = new Order(total, sum);
        Element orderElement = order.getElement(orderDocument);
        orderDocument.appendChild(orderElement);

        parser.generateXml("D://order.xml", orderDocument);

    }

    static class Order {
        private float total = 0;
        private int   sum   = 0;

        public Order(float total, int sum) {
            this.total = total;
            this.sum = sum;
        }

        public Element getElement(Document document) {
            Element rootElement = document.createElement("order");

            Element totalElement = document.createElement("total");
            totalElement.setTextContent(String.valueOf(this.total));
            rootElement.appendChild(totalElement);

            Element sumElement = document.createElement("sum");
            sumElement.setTextContent(String.valueOf(this.sum));
            rootElement.appendChild(sumElement);

            return rootElement;
        }

    }

    static class Goods {
        private float price;
        private int   number;

        public Goods(Element element) {
            this.price = Float.parseFloat(element.getElementsByTagName("price").item(0).getText
ontent());
            this.number = Integer.parseInt(element.getElementsByTagName("number").item(0).ge
TextContent());
        }

        public float getTotal() {
            return this.price * this.number;
        }

        public int getNumber() {
            return number;
        }
    }
```

原文链接：java通过DOM操作xml

```
}
```

standalone

这里有一个小坑 就是生成的xml中有一个属性为standalone="no"

standalone表示是否为独立文件，也就是说是否依赖于其他外部文件，如果为yes，则表示为不依赖他外部文件的独立文件，默认为yes。

但是生成之后的standalone="no"，不符合预期，并且

```
transformer.setOutputProperty(OutputKeys.STANDALONE, "yes");
```

设置格式之后standalone="no"仍然为no。这时需要设置document中的setXmlStandalone属性，

```
document.setXmlStandalone(true);
```

再次输出，可以去掉standalone属性。

结语

理解xml的结构之后，和dom的树形结构后，无论是使用原生支持还是通过第三方类库去操作xml文，都可以很容易的上手。

and **我喜欢用json**