XML 主流的解析方式有DOM方式、Stream方式和转换成JavaBean的方式

1. DOM 方式：

① 将xml转换成Document对象，保存到内存中。因此解析大量的xml时，该方式并不合适。

org.dom4j.Document doc = null;  
try {  
 doc = DocumentHelper.*parseText*(xml);  
} catch (DocumentException e) {  
 e.printStackTrace();  
 failCause.append("parse xml failed.");  
 return false;  
}  
if (doc == null) {  
 failCause.append("doc is null.");  
 return false;  
}

② 可以使用XPath获取Node或Element对象

Node node = doc.selectSingleNode(path);

Element element = (Element) node;

List nodes = doc.selectNodes(path);

List<Element> nodes = doc.selectNodes(path);

③ 可以修改Element的属性和值

((Element) node).addAttribute("red", redValue);

2. Stream 方式：

这种方式是通过流解析的，只能一次性解析，提取出相应的xml结构

public class SAXParser {  
  
 private static SAXParser *saxParser* = new SAXParser();  
 private static XMLReader *parser*;  
 private static BookHandler *bookHandler*;  
  
 private SAXParser() {  
 try {  
 *parser* = XMLReaderFactory.*createXMLReader*();  
 *bookHandler* = new BookHandler();  
 *parser*.setContentHandler(*bookHandler*);  
 } catch (SAXException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public static SAXParser getInstance() {  
 return *saxParser*;  
 }  
  
 private static class BookHandler extends DefaultHandler {  
 private Policy policy;  
 private List<Rule> rules;  
 private Charset charset;  
 private Rule rule;  
 private XmlTag tagName;  
  
 // 在xml解析开始的时候调用  
 @Override  
 public void startDocument() throws SAXException {  
 System.*out*.println("Start parsing document");  
 rules = new ArrayList<>();  
 policy = new Policy();  
 }  
  
 // 在xml解析结束的时候调用  
 @Override  
 public void endDocument() throws SAXException {  
 System.*out*.println("End");  
 policy.setRules(rules);  
 }  
  
 */\*\*  
 \* 开始解析一个元素  
 \*/* @Override  
 public void startElement(String uri, String localName, String qName, Attributes atts) throws SAXException {  
 // 使用 qualified name, 在此没有使用xmlns prefixes  
 System.*out*.println(qName);  
 if (qName != null && qName.contains("word")) {  
 tagName = XmlTag.*WORD*;  
 } else {  
 tagName = XmlTag.*value*(qName);  
 }  
  
 switch (tagName) {  
 case *POLICY*:  
 policy.initAttr(atts); break;  
 case *RULE*:  
 rule = new Rule();  
 rule.initAtts(atts);  
 rule.setVersion(policy.getNewVersion());  
 rules.add(rule);  
 break;  
 case *WORD*:  
 charset = Charset.*forName*(atts.getValue("encoding"));  
 }  
 }  
  
 @Override  
 public void endElement(String namespaceURI, String localName, String qName) throws SAXException {  
 tagName = XmlTag.*NullResult*;  
 }  
  
 @Override  
 public void characters(char[] ch, int start, int length) {  
 // Processing character data inside an element  
 if (tagName == XmlTag.*WORD*) {  
 rule.addWord(new String(ch, start, length));  
 }  
 }  
 }

}

3. JavaBean方式

使用注解的方式将xml转换成JavaBean。

import javax.xml.bind.JAXBContext;  
import javax.xml.bind.JAXBException;  
import javax.xml.bind.Unmarshaller;

import java.io.StringReader;

JAXBContext jc = JAXBContext.*newInstance*(DeriveData.class);  
Unmarshaller unmar = jc.createUnmarshaller();  
DeriveData deriveData = (DeriveData) unmar.unmarshal(new StringReader(result));

@XmlRootElement(name = "result")  
public class DeriveData {  
  
 private List<HintElement> hints;  
  
 private List<KmapElement> elements;  
  
 @XmlElementWrapper(name = "kmap")  
 @XmlElement(name = "element")  
 public List<KmapElement> getElements() {  
 return elements;  
 }  
  
 public void setElements(List<KmapElement> elements) {  
 this.elements = elements;  
 }  
  
 @XmlElementWrapper(name = "hint")  
 @XmlElement(name = "title")  
 public List<HintElement> getHints() {  
 return hints;  
 }  
  
 public void setHints(List<HintElement> hints) {  
 this.hints = hints;  
 }  
}