DOM4J详解

# DOM4J api简介

DOM4J是 dom4j.org 出品的一个开源 XML 解析包。DOM4J应用于 Java 平台，采用了 Java 集合框架并完全支持 DOM，SAX 和 JAXP。DOM4J 使用起来非常简单。只要你了解基本的 XML-DOM 模型，就能使用。

Dom：把整个文档作为一个对象。DOM4J 最大的特色是使用大量的接口。它的主要接口都在org.dom4j里面定义：

* Attribute：定义了 XML 的属性。
* Branch：指能够包含子节点的节点。如XML元素(Element)和文档(Docuemnts)定义了一个公共的行为
* CDATA：定义了 XML CDATA 区域
* CharacterData：是一个标识接口，标识基于字符的节点。如CDATA，Comment, Text.
* Comment：定义了 XML 注释的行为
* Document：定义了XML 文档
* DocumentType：定义 XML DOCTYPE 声明
* Element：定义XML 元素
* ElementHandler：定义了Element 对象的处理器
* ElementPath：被 ElementHandler 使用，用于取得当前正在处理的路径层次信息
* Entity：定义 XML entity
* Node：为dom4j中所有的XML节点定义了多态行为
* NodeFilter：定义了在dom4j 节点中产生的一个滤镜或谓词的行为（predicate）
* ProcessingInstruction：定义 XML 处理指令
* Text：定义 XML 文本节点
* Visitor：用于实现 Visitor模式
* XPath：在分析一个字符串后会提供一个 XPath 表达式

接口之间的继承关系如下：

|  |
| --- |
| interface java.lang.Cloneable  interface org.dom4j.Node  interface org.dom4j.Attribute  interface org.dom4j.Branch  interface org.dom4j.Document  interface org.dom4j.Element  interface org.dom4j.CharacterData  interface org.dom4j.CDATA  interface org.dom4j.Comment  interface org.dom4j.Text  interface org.dom4j.DocumentType  interface org.dom4j.Entity  interface org.dom4j.ProcessingInstruction |

## 类：SAXReader

|  |
| --- |
| public void addHandler(String path,ElementHandler handler)：当解析到path指定的路径时，将调用参数handler指定的处理器。针对不同的节点可以添加多个handler实例。或者调用默认的Handler setDefaultHandler(ElementHandler handler); |

## 接口ElementHandler

|  |
| --- |
| public void onStart(ElementPath path)：该方法在解析到元素开始标签时被调用。  public void onEnd(ElementPath path)：该方法在解析到元素的结束标签时被调用 |

## 接口：ElementPath （假设有参数：ElementPath path）

|  |
| --- |
| public void addHandler(String path,ElementHandler)： 该方法与SAXReader类中的addHandler()方法的作用相同。路径path可以是绝对路径（路径以/开头），也可以是相对路径（假设是当前路径的子节点路径）。  public void removeHandler(String path)： 移除指定路径上的ElementHandler实例。路径可以是相对路径，也可以是绝对路径。  public String getPath()：该方法得到当前节点的路径。该方法返回的是完整的绝对路径  public Element getCurrent()： 该方法得到当前节点。 |

## Element类

|  |
| --- |
| getQName()：元素的QName对象  getNamespace()：元素所属的Namespace对象  getNamespacePrefix()：元素所属的Namespace对象的prefix  getNamespaceURI()：元素所属的Namespace对象的URI  getName()：元素的local name  getQualifiedName()：元素的qualified name  getText()：元素所含有的text内容，如果内容为空则返回一个空字符串而不是null  getTextTrim()：元素所含有的text内容，其中连续的空格被转化为单个空格，该方法不会返回null  attributeIterator()：元素属性的iterator，其中每个元素都是Attribute对象  attributeValue()：元素的某个指定属性所含的值  elementIterator()：元素的子元素的iterator，其中每个元素都是Element对象  element()：元素的某个指定（qualified name或者local name）的子元素  elementText()：元素的某个指定（qualified name或者local name）的子元素中的text信息  getParent：元素的父元素  getPath()：元素的XPath表达式，其中父元素的qualified name和子元素的  qualified name之间使用"/"分隔  isTextOnly()：是否该元素只含有text或是空元素  isRootElement()：是否该元素是XML树的根节点 |

# DOM4j使用

## Document对象相关

|  |
| --- |
| #1 读取XML文件,获得document对象.  SAXReader reader = new SAXReader();  Document document = reader.read(new File("input.xml"));    #2解析XML形式的文本,得到document对象.  String text = "<members></members>";  Document document = DocumentHelper.parseText(text);  #3主动创建document对象.  Document document = DocumentHelper.createDocument();  Element root = document.addElement("members");// 创建根节点 |

## Element节点相关

|  |
| --- |
| #1获取文档的根节点.  Element rootElm = document.getRootElement();  #2取得某节点的单个子节点.  Element memberElm=root.element("member");// "member"是节点名    #3取得节点的内容  String text=memberElm.getText();  String text=root.elementText("name");取得根节点下的name子节点的文字.  #4.取得某节点下指定名称的所有节点并进行遍历.  List nodes = rootElm.elements("member");  for (Iterator it = nodes.iterator(); it.hasNext();) {  Element elm = (Element) it.next();  // do something  }    #5对某节点下的所有子节点进行遍历.  for(Iterator it=root.elementIterator();it.hasNext();){  Element element = (Element) it.next();  // do something  }    #6在某节点下添加子节点.  Element ageElm = newMemberElm.addElement("age");    #7设置文本节点.  ageElm.setText("29");    #8删除某节点.  parentElm.remove(childElm); // parentElm是其父节点  #9添加一个CDATA节点.  Element contentElm = infoElm.addElement("content");  contentElm.addCDATA(diary.getContent()); |

## Attribute 属性相关.

|  |
| --- |
| #1.取得节点的指定的属性  Element root=document.getRootElement();  Attribute attribute=root.attribute("size"); // 属性名name    #2.取得属性的文本  String text=attribute.getText();  String text2=root.element("name").attributeValue("firstname");  //这个是取得根节点下name字节点的firstname属性的值.    #3.遍历某节点的所有属性  Element root=document.getRootElement();  for(Iterator it=root.attributeIterator();it.hasNext();){  Attribute attribute = (Attribute) it.next();  String text=attribute.getText();  System.out.println(text);  }  #4.设置某节点的属性和文字.  newMemberElm.addAttribute("name", "sitinspring");  #5.设置属性的文字  Attribute attribute=root.attribute("name");  attribute.setText("sitinspring");    #6.删除某属性  Attribute attribute=root.attribute("size");// 属性名name  root.remove(attribute); |

## 将文档写入XML文件.

|  |
| --- |
| #1.文档中全为英文,不设置编码,直接写入.  XMLWriter writer = new XMLWriter(new FileWriter("output.xml"));  writer.write(document);  writer.close();  #2.文档中含有中文,设置编码格式再写入.  OutputFormat format = OutputFormat.createPrettyPrint();  format.setEncoding("GBK"); // 指定XML编码  XMLWriter writer = new XMLWriter(new FileWriter("output.xml"),format);  writer.write(document);  writer.close(); |

## 字符串与XML的转换

|  |
| --- |
| #1.将字符串转化为XML  String text = "<members> <member>sitinspring</member> </members>";  Document document = DocumentHelper.parseText(text);    2.将文档或节点的XML转化为字符串.  SAXReader reader = new SAXReader();  Document document = reader.read(new File("input.xml"));  Element root=document.getRootElement();  String docXmlText=document.asXML();  String rootXmlText=root.asXML();  Element memberElm=root.element("member");  String memberXmlText=memberElm.asXML(); |

# XML文档操作详解

## 创建xml文件

|  |
| --- |
| @Test  public void createXML() {  List<Book> books = new ArrayList<Book>();  Book book1 = new Book("1", "java性能优化", "zxx", "2016-10-08", "20", "zh-cn");  Book book2 = new Book("2", "python性能优化", "zqq", "2016-11-30", "20.1", "zh-cn");  books.add(book1);  books.add(book2);  Document doc = DocumentHelper.createDocument();  Element bookstore = doc.addElement("bookstore");  for (Book book : books) {  Element bookEle = bookstore.addElement("book");  bookEle.addAttribute("id", book.getId());  Element nameEle = bookEle.addElement("name");  nameEle.setText(book.getName());  Element authorEle = bookEle.addElement("author");  authorEle.setText(book.getAuthor());  Element yearEle = bookEle.addElement("year");  yearEle.setText(book.getYear());  Element priceEle = bookEle.addElement("price");  priceEle.setText(book.getPrice());  Element languageEle = bookEle.addElement("language");  languageEle.setText(book.getLanguage());  }  OutputFormat format = OutputFormat.createPrettyPrint();  format.setEncoding("UTF-8");  File file = new File("books.xml");  XMLWriter writer;  try {  writer = new XMLWriter(new FileOutputStream(file), format);  writer.setEscapeText(false);  writer.write(doc);  writer.close();  } catch (IOException e) {  e.printStackTrace();  }  } |

## 读取解析XML文件

|  |
| --- |
| @Test  public void parserXML() {  String file = XMLTest.class.getClassLoader().getResource("books.xml").getPath();  System.out.println(file);    List<Book> books = new ArrayList<Book>();  SAXReader reader = new SAXReader();  try {  Document doc = reader.read(new File(file));  Element bookStoreElement = doc.getRootElement();  Iterator<Element> it = bookStoreElement.elementIterator();  while (it.hasNext()) {  Book book = new Book();  Element bookEle = it.next();  List<Attribute> attributes = bookEle.attributes();  for (Attribute attr : attributes) {  if (attr.getName().equals("id")) {  book.setId(attr.getValue());  }  }  List<Element> childBookEles = bookEle.elements();  for (Element childBookEle : childBookEles) {  String name = childBookEle.getName();  String value = childBookEle.getStringValue();  if (name.equals("name")) {  book.setName(value);  }  else if (name.equals("author")) {  book.setAuthor(value);  }  else if (name.equals("year")) {  book.setYear(value);  }  else if (name.equals("price")) {  book.setPrice(value);  }  else if (name.equals("language")) {  book.setLanguage(value);  }  }  books.add(book);  }  System.out.println(books);  } catch (DocumentException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  } |