DOM4J 分析报告

咸宁 2017K8009929042

2020.1.1

一、引言

Dom4j 是一个易用的、开源的库，用于 XML，XPath 和 XSLT。它应用于 Java 平台，采用了 Java 集合框架并完全支持 DOM，SAX 和 JAXP[1]。DOM4J 可用于读取、写入、遍历、创建和修改 XML 文档。本文主要基于 DOM4J 创建 XML 文档的过程展开分析。

二、DOM4J 的 API

DOM4J 由 14 个包组成。org.dom4j 包中包含了 XML 各个组成部分的接口和创建文档及处理异常时所需要的帮助类。这些接口之间的层次关系如下。

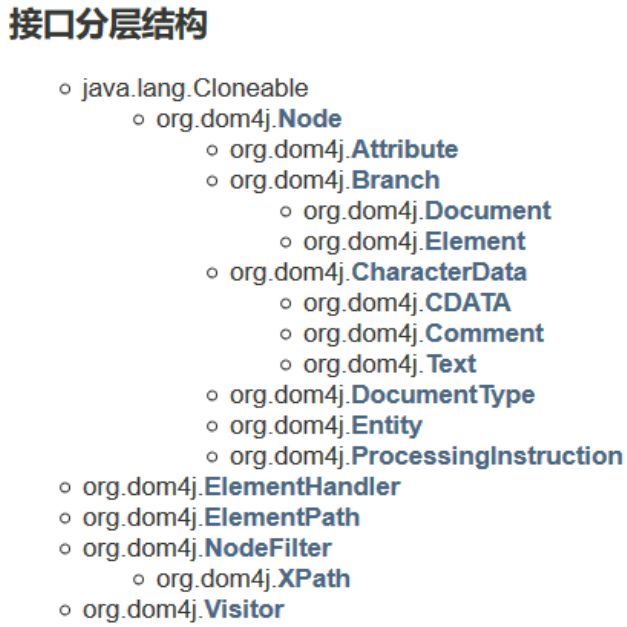

图 1　DOM4J 的接口分层结构

表示 XML 文档组成部分的接口都从 Node 直接或间接派生而来，Node 定义了所有 XML 节点的多态行为。

三、XML 文档创建过程

下面是创建一个描述下列信息的 XML 文档的过程。

Name	Author	Press	Category
Thinking in JAVA	Bruce Eckel	China Machine Press	Computer Science
Artifical Intelligence: A Morden Approach	Stuart J. Russell Perter Norvig	Tsinghua University Press	

表 1　两本书的信息

1.导入 dom4j 包

```
import org.dom4j.*;
```

2.创建 Document 对象

Dom4j 中的 DocumentHelper 类是一个帮助类，这个类可以帮助我们创建各种类和对象。我们可以调用其中的静态方法来创建一个 document 对象：

```
Document document = DocumentHelper.createDocument();
```

3.创建文档的根节点

所有实现了 Branch 接口的节点都包含 addElement 方法，用于添加子节点。因此可以如下生成根元素：

```
Element rootElement = document.addElement("books");
```

4.添加元素节点并设置文本内容

我们可以继续使用 addElement 方法在根节点下添加子节点，该方法返回新生成的节点，因此可以进行链式调用。Element 接口中定义了在节点下添加属性、文本等内容的方法：

```
Element book1 = rootElement.addElement("book")
                .addAttribute("category","Computer Science");
                book1.addElement("name")
                .addText("Thinking in JAVA");
                book1.addElement("author")
                .addText("Bruce Eckel");
                book1.addElement("press")
                .addText("China Machine Press");
```

5.将生成的 document 写入 XML 文档

Node 中定义了 write 方法，可以将节点序列化后写入文件：

```
public static void writeFile(Document doc) throws IOException {
    FileWriter out = new FileWriter("E:\\books.xml");
    doc.write(out);
    out.close();
}
```

6.生成结果

如图。

```
<?xml version="1.0" encoding="UTF-8"?>
- <books>
  - <book category="Computer Science">
      <name>Thinking in JAVA</name>
      <author>Bruce Eckel</author>
      <press>China Machine Press</press>
    </book>
  - <book category="Computer Science">
      <name>Artifical Intelligence: A Morden Approach</name>
      <author>Stuart J. Russell</author>
      <author>Perter Norvig</author>
      <press>Tsinghua University Press</press>
    </book>
</books>
```

图 2　XML 生成结果

四、创建文档时涉及的对象及其交互流程

1.Branch、Element 和 Document 接口

Branch 接口定义了所有可以包含子节点的节点的行为。继承自它的 element 和 document 接口分别定义了 XML 树中的元素和文档节点。

在上文的例子中，Branch 提供了 3 种 addElement 的抽象方法，Element 和 Document 继承自 Branch，因此也拥有 addElement 方法。此外，它们还有自己特有的抽象方法，如 Element 中的 addAttribute 和 Document 中的 getRootElement。

addElement 方法具体在 Org.dom4j.tree 包中由相应的抽象类实现。

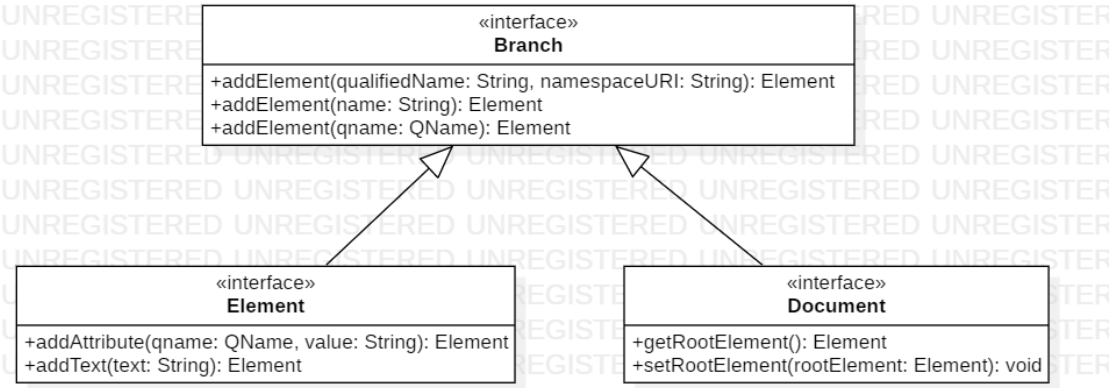

图 3 Branch 下接口继承关系图

2.DocumentHelper 类

生成新的 Document 对象时，我们使用了 DocumentHelper 类中的静态方法，这个类是 DOM4J 中帮助方法的一个集合。之前使用的 create 方法如下。

```
public static Document createDocument() {
    return getDocumentFactory().createDocument();
}
```

其中调用的 getDocumentFactory()方法如下。

```
private static DocumentFactory getDocumentFactory() {
    return DocumentFactory.getInstance();
}
```

在这个例子里，为了实现 create 方法，Helper 类访问了 Factory 类的单例实现，创建了一个 DocumentFactory 类，并调用了其中的 createDocument 方法来创建新对象。

3.DocumentFactory 类

DocumentFactory 类是 DOM4J 中工厂方法的一个集合，它的 create 方法返回 Org.dom4j.tree 包中的类的实例，比如 DefaultElement 和 DefaultAttribute。

DOM4J 中对象的创建由工厂类完成，需要创建对象时，先单例实现 DocumentFactory，

然后调用其中不同的 create 方法。在上文的例子中，它给出了 createDocument 方法的实现，返回了一个 DefaultDocument 对象，如下。

```
public Document createDocument() {
    DefaultDocument answer = new DefaultDocument();
    answer.setDocumentFactory(this);
    return answer;
}
```

图 4 DocumentFactory [2]

DocumentFactory 有许多子类，可满足用户不同的需求。比如可以产生 JavaBean 对象的 BeanDocumentFactory；生成对象继承了 org.w3c.dom.Document 接口和 org.dom4j.Document 接口的 DOMDocumentFactory 等。

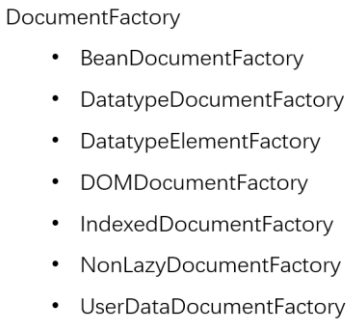

图 5 DocumentFactory 的子类

4.Org.dom4j.tree 中的抽象类

Org.dom4j.tree 包给出了 dom4j 文档对象的默认实现，包括 `DefaultDocument`，`DefaultElement`，`DefaultAttribute` 等。同时它还给出了许多抽象类（基类），如果用户需要实现自己的文档对象模型，可以使用这些基类。

- org.dom4j.tree.**AbstractNode** (implements java.lang.Cloneable, org.dom4j.Node, java.io.Serializable)
  - org.dom4j.tree.**AbstractAttribute** (implements org.dom4j.Attribute)
    - org.dom4j.tree.**FlyweightAttribute**
      - org.dom4j.tree.**DefaultAttribute**
  - org.dom4j.tree.**AbstractBranch** (implements org.dom4j.Branch)
    - org.dom4j.tree.**AbstractDocument** (implements org.dom4j.Document)
      - org.dom4j.tree.**DefaultDocument**
    - org.dom4j.tree.**AbstractElement** (implements org.dom4j.Element)
      - org.dom4j.tree.**BaseElement**
      - org.dom4j.tree.**DefaultElement**
  - org.dom4j.tree.**AbstractCharacterData** (implements org.dom4j.CharacterData)
    - org.dom4j.tree.**AbstractCDATA** (implements org.dom4j.CDATA)
      - org.dom4j.tree.**FlyweightCDATA** (implements org.dom4j.CDATA)
        - org.dom4j.tree.**DefaultCDATA**
    - org.dom4j.tree.**AbstractComment** (implements org.dom4j.Comment)
      - org.dom4j.tree.**FlyweightComment** (implements org.dom4j.Comment)
        - org.dom4j.tree.**DefaultComment**
    - org.dom4j.tree.**AbstractText** (implements org.dom4j.Text)
      - org.dom4j.tree.**FlyweightText** (implements org.dom4j.Text)
        - org.dom4j.tree.**DefaultText**
  - org.dom4j.tree.**AbstractDocumentType** (implements org.dom4j.DocumentType)
    - org.dom4j.tree.**DefaultDocumentType**
  - org.dom4j.tree.**AbstractEntity** (implements org.dom4j.Entity)
    - org.dom4j.tree.**FlyweightEntity**
      - org.dom4j.tree.**DefaultEntity**
  - org.dom4j.tree.**AbstractProcessingInstruction** (implements org.dom4j.ProcessingInstruction)
    - org.dom4j.tree.**FlyweightProcessingInstruction**
      - org.dom4j.tree.**DefaultProcessingInstruction**
  - org.dom4j.**Namespace**
    - org.dom4j.tree.**DefaultNamespace**

图 6 Org.dom4j.tree 包树状关系图

对于上文中提到的 `addElement` 方法，`AbstractBranch`、`AbstractDocument`、`AbstractElement` 分别给出了不同的实例化方法，即子类继承父类的同时又重写了父类的方法。下面是 `AbstractDocument` 中给出的 `addElement` 方法。

```
public Element addElement(String name) {
    Element element = getDocumentFactory().createElement(name);
    add(element);
    return element;
}
```

5.不同对象之间的交互流程

可以用下面这张 UML 图来概括上述对象之间的关系。接口中定义了抽象方法，抽象类中给出了方法的具体实现，实现过程调用了工厂类的 `create` 方法，默认类继承了抽象类的方法。

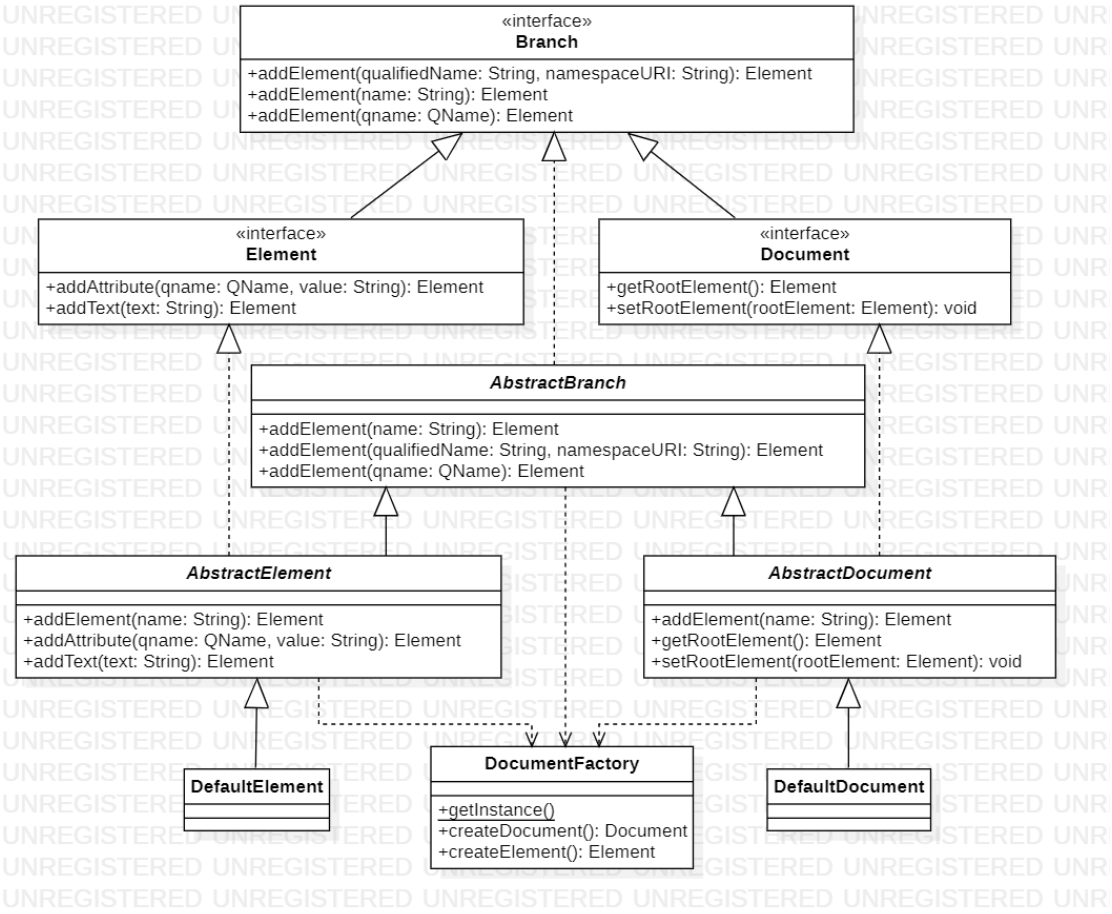

图 7 接口、抽象类与工厂类

在本文的例子中，所有的方法都由接口定义，并将其实现放在抽象类中，这样既有利于各个模块之间的交互，也有利于后期的升级维护，体现了“面向接口编程”的思想。

五、创建文档时涉及到的设计模式

1.单例模式

单例模式的设计意图是保证一个类仅有一个实例，并提供一个访问它的全局访问点。当一个全局使用的类被频繁创建和销毁时，可以使用单例模式来控制实例数目，节省系统资源。

创建 XML 文档时使用的 `DocumentFactory` 这个工厂类就应用了单例模式，当需要调用工厂类中的方法时，就可以使用 `DocumentFactory.getInstance()`来获取一个工厂类。

```
public static synchronized DocumentFactory getInstance() {
    if (singleton == null) {
        singleton = createSingleton();
    }
    return singleton.instance();
}
```

单例模式有多种实现方法，DOM4J 中默认使用的是懒汉模式，可以保证线程之间的安全性，确保即使多线程情况下也不会出现错误。调用 `getInstance` 方法时，如果 *singleton*

不存在就创建一个新的出来，否则返回已有的 *singleton*。

同时，DOM4J 也给出了非懒汉模式的工厂类单例实现方式。NoLazyDocumentFactory 是 DocumentFactory 的一个子类，他与父类唯一的区别在于 *singleton* 是一开始就产生的而不需要在调用时生成。这样虽然使用了更多的内存，但却可以使得同一个文档的实例在修改之前可以在不同线程之间共享。

```
protected static transient NonLazyDocumentFactory singleton
        = new NonLazyDocumentFactory();
public static DocumentFactory getInstance() {
    return singleton;
}
```

2.享元模式

享元模式主要用于减少创建对象的数量，以减少内存占用和提高性能。当有大量对象时，为避免内存溢出，可以将其中共同的部分抽象出来，如果有相同的业务请求，直接返回内存中已有的对象，避免重新创建。

创建 XML 文档时，会用到大量的细粒度对象如 Attribute、Entity、Comment、Text 等，DOM4J 为节省内存空间，对这些对象采用享元模式。默认类 DefaultAttribute 并不是像 Element 和 Document 那样直接继承自抽象类 AbstractAttribute，而是继承自抽象类的子类 FlyweightAttribute。

每个 Element 对象中都有一个 AttributeList，其中保存所有使用过的 Attribute 对象。AbstractElement 中实例化了几种 attribute 方法，可以根据 QName，Name 等信息查找 AttributeList 中的 Attribute 对象，其中调用的 getQname 方法在 FlyweightAttribute 中实例化。

```
public Attribute attribute(QName qName) {
    for (Attribute attribute : attributeList()) {
        if (qName.equals(attribute.getQName())) {
            return attribute;
        }
    }
    return null;
}
```

当调用 Element 中的 addAttribute 方法时，首先根据 QName 检查 attribute 方法返回值是否为空，为空则说明这个 Attribute 没有使用过，需要新建一个，否则可以直接得到之前的 Attribute 对象。最后再根据原本 Attribute 的属性决定是否复用该对象。

```
public Element addAttribute(QName qName, String value) {
```

```
        // adding a null value is equivalent to removing the attribute
        Attribute attribute = attribute(qName);
        if (value != null) {
            if (attribute == null) {
                add(getDocumentFactory().createAttribute(this, qName, value));
            } else if (attribute.isReadOnly()) {
                remove(attribute);
                add(getDocumentFactory().createAttribute(this, qName, value));
            } else {
                attribute.setValue(value);
            }
        } else if (attribute != null) {
            remove(attribute);
        }
        return this;
    }
```

六、结语

DOM4J 是一个优秀的开源库，仅从创建 XML 文档的功能来看，它简单易用又十分灵活。其中包含的大量面向对象编程思想值得我们学习。

参考文献

[1] Elliotte Rusty Harold.Java 语言与 XML 处理教程[M].北京:电子工业出版社,2003-11-01.

[2] Brett D.McLaughlin & Justin Edelson.Java and XML, 3rd Edition[M].O'Reilly,2006-12-01.