Dom4j Learning Report

Version 1.0

刘卓轩 2019.11.6

声明

欢迎任课老师和助教以及其他来自互联网的读者阅读此报告。我于 2019 年秋季加入国科大“面向对象程序设计”课程，并选择 dom4j 作为源码学习环节的目标项目。此文即为我的源码学习报告。由于这是面向对象课程的作业，所以和算法相关的内容不是解析的重点。

本文分析的目标是 dom4j 开源项目，源码地址为 https://github.com/dom4j/dom4j ,版本为 2.1.1。文中会用到一些示例测试程序和 UML 图分析，我使用的工具是 JetBrains 公司的 IntelliJ IDEA.

欢迎各位老师、同学和来自互联网的读者参与课程讨论。如果你在文中发现了任何你确认是错误的内容，请与作者联系，我将对此不胜感激!

刘卓轩 2019 年 11 月 6 日

邮箱：liuzhuoxuan17@mails.ucas.ac.cn

第一章：功能分析与建模

本章将围绕 Dom4j 的提供的主要功能进行需求分析，并举出一些用例来印证笔者归纳的 Dom4j 的特性。

1.1 什么是 XML

XML(Extensible Markup Language)，全称可扩展标记语言。是由 W3C(万维网联盟)制定的一种标准数据格式。它在设计之初就强调了简洁、通用，因而同时满足了人和机器的易读性。W3C 严格规定了 XML 的语法结构，而程序员也已经开发了许多应用程序编程接口来帮助处理 XML 数据。

目前基于XML的应用有许多 ,主要分布于以下三个方面 :1.很多办公软件依赖基于XML的格式，包括 Microsoft Office，开源的 LibreOffice，苹果的 iWork 等等。2.基于 XML 制定了许多行业数据标准，使得不同的格式能被标准化为 XML 模式。3.进而广泛应用于面向服务的体系结构(SOA)，使得不同的系统通过交换 XML 信息相互通信。

1.2 什么是 dom4j

Dom4j 意为 DOM for java，是一款 java 开源库，主要用于解析 XML,此外还协同处理 XPATH 和 XSLT。

它相比于其他 XML 解析库如 DOM, SAX, JDOM 具有以下两个特点：高度灵活、内存使用高效。

DOM 接口通过一种分层对象模型来访问 XML 文档信息，即用一棵节点树的形式来描述 XML 的文档结构。但 DOM 由于需要一次性把整个文档转化成树结构，所以对内存的需求就比较高，遍历操作带来的开销较大。这就可能会导致内存溢出等极端情况发生。

1.3 XML 文档树结构

下面我们简要介绍一下 XML 的文档树结构，理解这种树的构成对理解 DOM4j 的处理流程有重要帮助。下面是一个典型的 XML 文档。

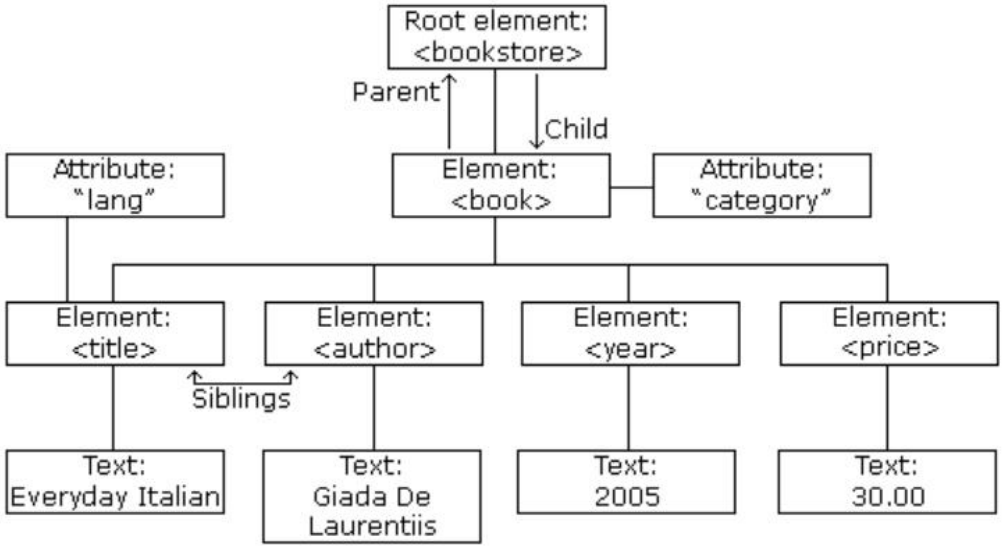
```xml
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>

  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>

  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>

</bookstore>
```

例 1

XML 使用简单的具有自我描述性的语法。第一行是 XML 声明，它定义了 XML 的版本号(1.0)和所使用的编码方式(UTF-8)。

第二行便是 XML 的根元素，每个 XML 都需要一个根元素。它类似于描述了本文档的主题:bookstore。接下来的若干行可以从形如<>和</>的打开与关闭标签中分别看出包含关系与并列关系，我们称包括打开标签、关闭标签和它们之间的内容成为元素，元素之间的关系可以抽象成树结构中的 parent, children, sibling。

我们还可以发现打开标签中包含着一个等式，等式左右分别规定了属性与属性值，用于为元素提供额外信息。包含在元素内且没有打开关闭标签的称为文本，文本主要负责存储元素信息。我们将例 1 中的每个上述的元素、属性、文本写成如下的树结构，每个信息都成了一个节点。



总结 XML 文档树的结构如下

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

1.4 XML 命名空间

在 XML 中，如果两个不同的文档使用相同的元素名，在合并时就会发生冲突。为了避免冲突，可以通过使用名称前缀避免。在 XML 使用前缀时，一个在元素的开始标签的 xmlns 属性中定义的命名空间(Namespace)必须被定义。命名空间的声明按照如下语法：

xmlns:前缀= "URI"

我们举一个简单的例子：

```
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
<h:tr>
<h:td>Apples</h:td>
<h:td>Bananas</h:td>
</h:tr>
</h:table>
```

<f:table xmlns:f="http://www.w3cschool.cc/furniture">
<f:name>African Coffee Table</f:name>
<f:width>80</f:width>
<f:length>120</f:length>
</f:table>

</root>

在上面的实例中，<table>标签的 xmlns 属性定义了 h:和 f:前缀的合格命名空间。所有带有相同前缀的子元素都会和同一个命名空间相关联。另外 W3C 规定了 Qname 是带前缀或不带前缀的名字

1.5 需求分析

1. 解析 XML 文档

用例名称：解析（parse）

场景：文件或流、用户

用例描述：用户导入与 XML 相关的包，创建 SAXReader，从文件或流创建文档，通过调用 document.selectNodes()获取所需的节点。提取 XML 文档的根元素，然后通过遍历器寻找想要的元素，通过 attributeValue(属性)获取属性值，通过 elementText 获取元素的文本。

用例价值：用户得到 XML 文档的数据信息

2. 创建或修改 XML 文档

用例名称：创建（create）

场景：文件或流、用户

用例描述：用户先寻求 dom4j 的 DocumentHelper 在 Dom4j 内由文档工厂创建一个文档，然后通过文档的 addElement 方法在文档中添加节点，即为根节点。如果要在对某个

元素内部添加新的元素，调用新元素的 addElement 即可。如果有附属的属性和文本需要添加，Element 的返回值同样为 Element 的 addAttribute 和 addText 接口可以调用。OutputFormat 类会构造自身的一个对象，然后根据特定的格式需要来添加空格或空行。最后根据该格式构建一个 XML Writer 对象，由它将规整后的文档打印到文件或输出流中。

用例价值：用户可以方便地创建新的 XML 文档。

约束和限制：使用相同的元素名时应注意命名空间不同。

第二章：核心流程设计分析

我们在第一部分关注了 Dom4j 的功能，了解了 Dom4j 能做什么，接下来将进一步分析 Dom4j 内部在完成这些功能时的核心流程。Dom4j 的一个优势在于高度灵活，它的高度灵活可以从外部与内部两方面探究。外部模式体现在接口的封装与隔离上，内部模式体现在内部对其他包的类继承上。

2.1 主要外部接口

Everything is a node，这是 Dom4j 的接口继承关系图中表现出来的第一大特点。包括 Document, Element 在内的众多接口，都是我们在解析 XML 时极常用到的。节点这一概念，在第一章中已经交代过了，这里不再赘述。但 Node 这一底层接口是用户完全没必要接触的，也就是说，就算你刚入门 XML，也不需要了解内部的树接口，就可以对 XML 文档进行操作了。从下面的 UML 图中，我们还可以发现两个中间的接口，即 Branch 和 CharacterData,它们分别对可存在孩子的节点和不存在孩子的叶节点做了相应的特殊接口。Dom4j 主要外部接口的设计体现了面向对象设计原则中的接口隔离，即尽可能地细化接口，而不是创建一个庞大而臃肿的 Node 接口提供所有依赖于它的类进行调用。

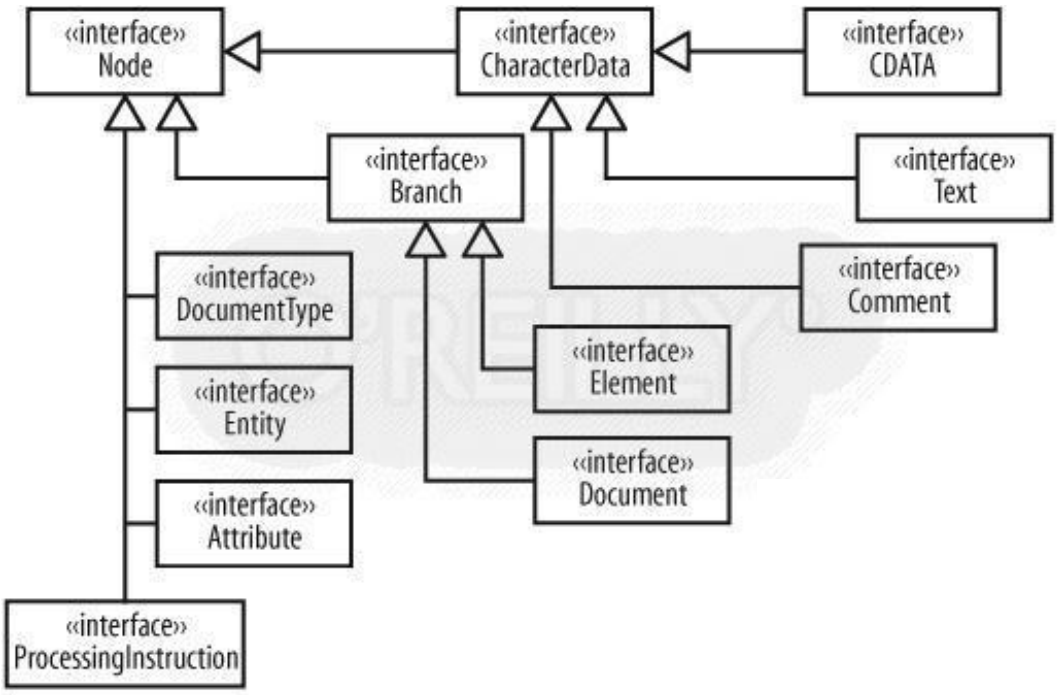


图 1

2.2 内部抽象类的结构

当深入跟踪 XML 解析过程的调用栈时，我们都会一次次进入 tree 包。从 AbstractNode 的类型名数组中就可以看出 Tree 包通过抽象类实现了图 1 中所有的外部接口。

```
public abstract class AbstractNode implements Node, Cloneable, Serializable {
    protected static final String[] NODE_TYPE_NAMES = {"Node", "Element",
            "Attribute", "Text", "CDATA", "Entity", "Entity",
            "ProcessingInstruction", "Comment", "Document", "DocumentType",
            "DocumentFragment", "Notation", "Namespace", "Unknown" };
```

直观上去看这个包的文件列表和依赖关系如下，几乎每一个接口都拥有抽象类，享元类和默认创建类。

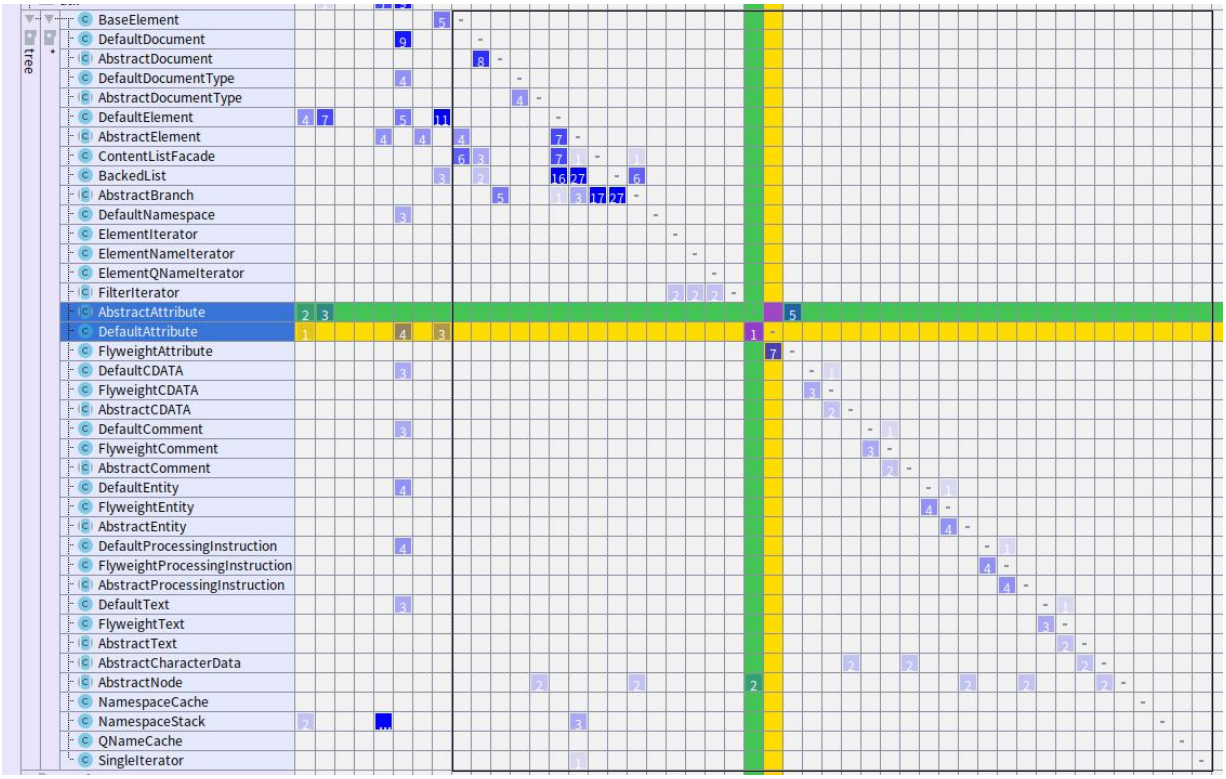


图 2

Tree 包内部类的耦合度极低，反映在具体的类图上如下，以 Attribute 为例。抽象类 AbstractAttribute 负责对上层的接口进行小幅度的封装，形成一些较为复杂的功能，而不局限于单独的 Attribute 节点。Flyweight 享元类用于减少存储开销，这在第三章中还会详细介绍。而 defaultAttribute 类则在工厂实例化时可以用到。

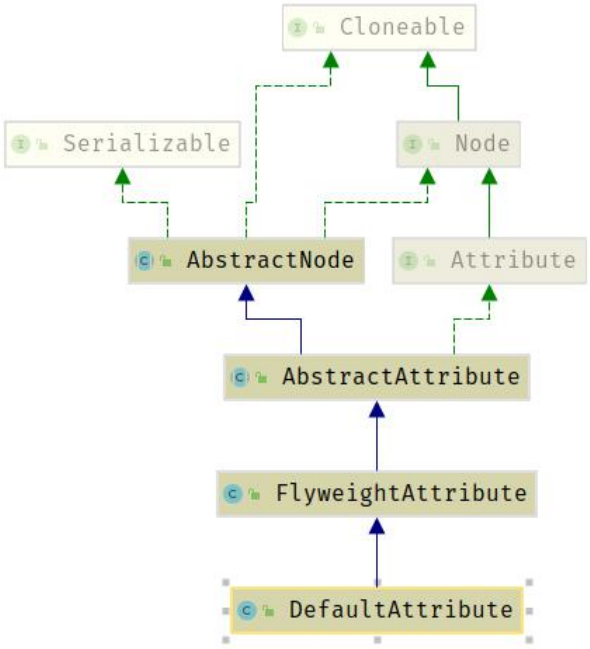


图 2

第三章：高级设计意图分析

在第二章中，我们看到大量的 DOM 节点都采用了 flyweight 类继承 abstract 抽象类，去添加一些 default 类没有的关键信息。接下来，我们会重点以享元模式为线来深入理解 Dom4j 为内存优化而改进的设计意图，同时还会涉及工厂模式和适配器模式。

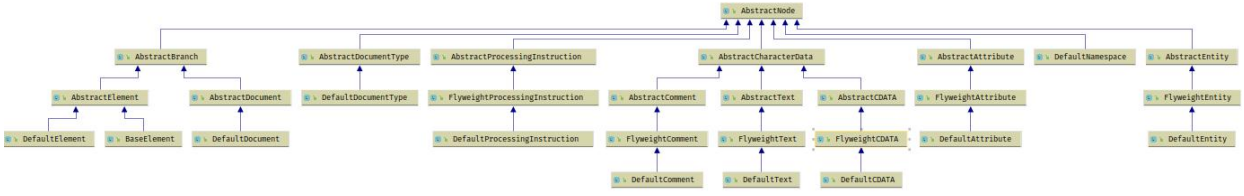


图 3

3.1 意图

第一章提到过，在过去 DOM 的实现中，内存的大小局限着 DOM 可处理 XML 文档数据的

大小，DOM 对 XML 树的节点化模型虽然可以单独处理每个元数据和它们之间的数据关系，但同时因为存在着大量对象，创建开销极大。同时，这些对象还存在着大量的重复状态，比如许多属性都关联着同一个命名空间。这时候不需要将这个共享的数据放在每个类中不断构造，而是可以采用享元模式。

3.2 解决方案

解决方案

GOF(Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides)在《设计模式：可复用面向对象软件的基础》中对 Flyweight 模式的定义是：

利用共享方法更高效地支持大数量的细粒度对象。

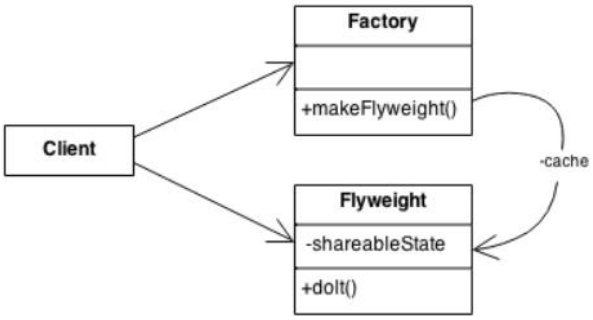


图 4

Flyweight 的主要参与者是 Flyweight Factory 和 Client，享元工厂往往负责管理、创建和存储享元对象，而 client 是对享元对象的引用。那 Dom4j 中如何界定这种 client 和 factory 的任务分工呢？我原本以为从下面的 Dom4j 的类图中可以看出 Default Attribute 维持着对享元的引用，但后来才发现自己理解错了。

类图：继承关系

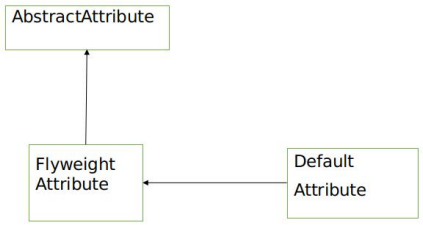


图 5

仔细解构 dom4j 享元层与 client 可以发现, default client 往往存储着所挂载的父元素节点，而剩下的例如文本的信息，属性的 Qname 和属性值被存储在了享元层中

Client 示例：

```
public class DefaultAttribute extends FlyweightAttribute {
    /** The parent of this node */
    private Element parent;

public class DefaultText extends FlyweightText {
    /** The parent of this node */
    private Element parent;
```

图 6

Flyweight 示例:

```
public class FlyweightAttribute extends AbstractAttribute {
    /** The <code>QName</code> for this element */
    private QName qname;

    /** The value of the <code>Attribute</code> */
    protected String value;
```

图 7

在元素添加属性和属性值时，在 Document Factory 中构造属性时，所有的属性都只需要两个值 qname 和 value，也可以看出 Dom4j 设计者精巧的 Adapter 适配器模式思想，将一个类的接口转化为客户希望的另一个接口，这种接口的改动封装了内部的算法设计，体现了对客户的信息隐藏。

```
public Attribute createAttribute(Element owner, QName qname, String value) {
    return new DefaultAttribute(qname, value);
}

public Attribute createAttribute(Element owner, String name, String value) {
    return createAttribute(owner, createQName(name), value);
}
```

图 8

我们的工厂会将父元素节点和属性名的信息过滤掉，以 Qname 的方式进行构造。默认属性构造方法会将 Qname 和 value 交给享元属性类进行存储。

在执行下面的代码时

```
.addAttribute( name: "company", value: "Ferrai");
```

我们清楚地看到

```
public FlyweightAttribute(QName qname, String value) {  qname: "org.dom4j.QName@38a73c7d [name: company namespace: "org.dom4j.Nam
    this.qname = qname;  qname: "org.dom4j.QName@38a73c7d [name: company namespace: "org.dom4j.Namespace@babe [Namespace: prefix
    this.value = value;  value: "Ferrai"
```

图 9

享元层把一个 XML 元素属性的属性名和名词空间打包递交给了 qname，把 value 交给成员变量 value。此时一个对象被构造完毕，也是真正进入了内存。按照前面所述，如果 FlyweightAttribute 是所谓的享元，那费解的地方在于，qname 和 value 似乎包含了一个节点的所有信息，DefaultAttribute 中的 parent 成员变量也只不过在 Element add attribute 时指针调动一下，享元在哪里呢？

需要理解的是，java 里面对象的复制只是一种软拷贝，维护的变量 qname 和 value 也只不过是一种赋值关系，对众多节点而言是必要的，而且并不会耗费大量内存。真正考虑内存时应该关心的也就是硬数据的存储位置。

我们把视线转向图 8 中 createAttribute 紧接调用的 createQname

```
public class DocumentFactory implements Serializable {
    private static SingletonStrategy<DocumentFactory> singleton = null;

    protected transient QNameCache cache;
```

图 10

QnameCache 是真正的享元工厂，完成了对 Qname 的存储与创建。QnameCache 的 get 具有享元模式的明显特点:去寻找享元池（cache）中具有该外蕴状态(name)的对象，将其返回，如果不存在，则将这个 name 放入享元池中。

```
public QName get(String name, Namespace namespace) {
    Map<String, QName> cache = getNamespaceCache(namespace);
    QName answer = null;

    if (name != null) {
        answer = cache.get(name);
    } else {
        name = "";
    }

    if (answer == null) {
        answer = createQName(name, namespace);
        answer.setDocumentFactory(documentFactory);
        cache.put(name, answer);
    }

    return answer;
}
```

图 11

QnameCache 的内部实现是 Hash 表,这在 Qname 的成员变量中就可见端倪。

```
public class QName implements Serializable {
    /** The Singleton instance */
    private static SingletonStrategy<QNameCache> singleton = null;

    private String name;

    /** The qualified name of the element or attribute */
    private String qualifiedName;

    /** The Namespace of this element or attribute */
    private transient Namespace namespace;

    /** A cached version of the hashcode for efficiency */
    private int hashCode;
```

图 12

另外值得一提的 QnameCache 的成员变量并不是 NamespaceCache,而是一种

synchronizedMap,有别于 NamespaceCache 所使用的 ConcurrentHashMap。这两种 map 都支持多线程同步访问，我也暂未发现这两种数据结构对两种 Cache 的应用有何区分。

结语

Dom4j 作为一款成功的软件包，其性能的优秀和它的内部结构紧密相关。它集成了 DOM、SAX 精华，DOM 的树结构，SAX 的事件驱动都能被它通过适配器的方式化为己用。同样 Dom4j 的 memory-efficient 也在我们对享元模式的分析中展露出来。此外还有像 detach 方法等用于调控内存消耗，在这里也不一一赘述。可以说对这块高效解析库的分析让我看到一款软件为同时具备多种强大优势所作出的设计细化。当然，由于时间关系，我无暇去关注解析 XML 之外的其他功能，因而对部分设计的原因也只能猜测，包括可序列化的 DocumentFactory 和不可被序列化的 QnameCache 等等。

这门课无疑给我带来非常大的帮助， 由于同时在本学期选修了数据库，大作业也用到了面向对象中的里氏替换等原则，感受颇深。最后，感谢一学期以来老师和助教的悉心指导！