Chapte4-XML和DOM

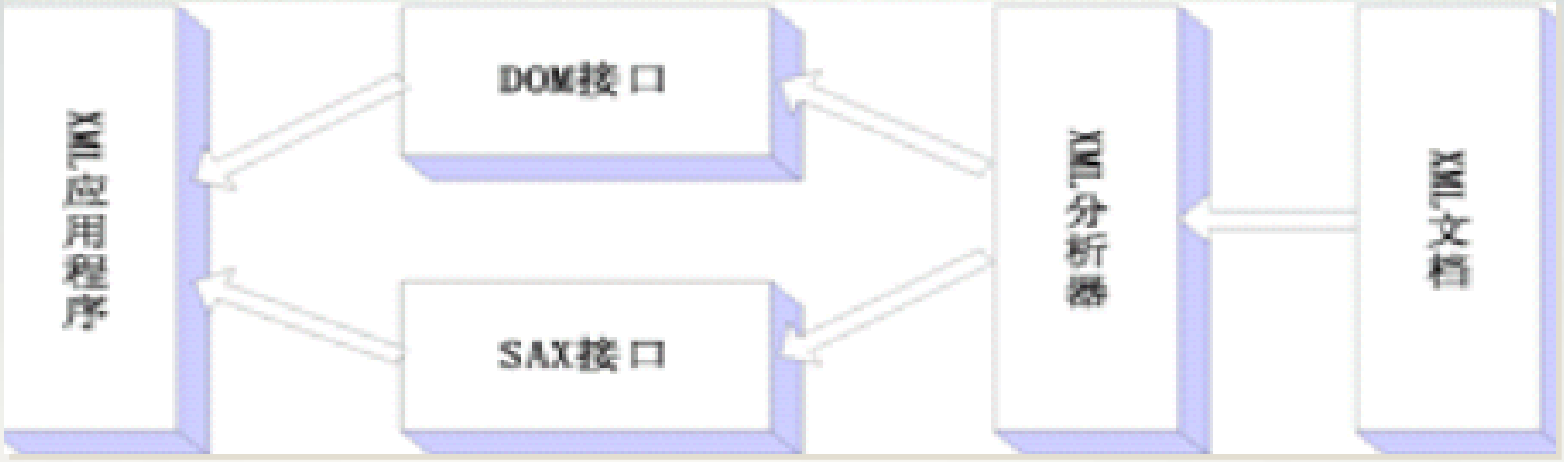
本章目标

- DOM简介
- 解析XML文档
- 操纵DOM的对象、接口、属性、方法

DOM和SAX

- W3C制定了一套书写XML分析器的标准接口规范--DOM。除此之外，XML_DEV邮件列表中的成员根据应用的需求也自发地定义了一套对XML文档进行操作的接口规范--SAX。这两种接口规范各有侧重，互有长短，应用都比较广泛

- 下面，我们给出DOM和SAX在应用程序开发过程中所处地位的示 意图。从图中可以看出，应用程序不是直接对XML文档进行操作的，而是首先由XML分析器对XML文档进行分析，然后，应用程序通过XML分析器所提供的DOM接口或SAX接口对分析结果进行操作，从而间接地实现了对XML文档的访问



DOM

DOM的全称是Document Object Model，也即文档对象模型。在应用程序中，基于DOM的XML分析器将一个XML文档转换成一个对象模型的集合（通常称DOM树），应用程序正是通过对这个对象模型的操作，来实现对XML文档数据的操作。通过DOM接口，应用程序可以在任何时候访问XML文档中的任何一部分数据，因此，这种利用DOM接口的机制也被称作随机访问机制。

DOM接口提供了一种通过分层对象模型来访问XML文档信息的方式，这些分层对象模型依据XML的文档结构形成了一棵节点树。无论XML文档中所描述的是什么类型的信息，即便是制表数据、项目列表或一个文档，利用DOM所生成的模型都是节点树的形式。也就是说，DOM强制使用树模型来访问XML文档中的信息。由于XML本质上就是一种分层结构，所以这种描述方法是相当有效的。

DOM树所提供的随机访问方式给应用程序的开发带来了很大的灵活性，它可以任意地控制整个XML文档中的内容。然而，由于DOM分析器把整个XML文档转化成DOM树放在了内存中，因此，当文档比较大或者结构比较复杂时，对内存的需求就比较高。而且，对于结构复杂的树的遍历也是一项耗时的操作。所以，DOM分析器对机器性能的要求比较高，实现效率不十分理想。不过，由于DOM分析器所采用的树结构的思想与XML文档的结构相吻合，同时鉴于随机访问所带来的方便，因此，DOM分析器还是有很广泛的使用价值的。

Dom的组成

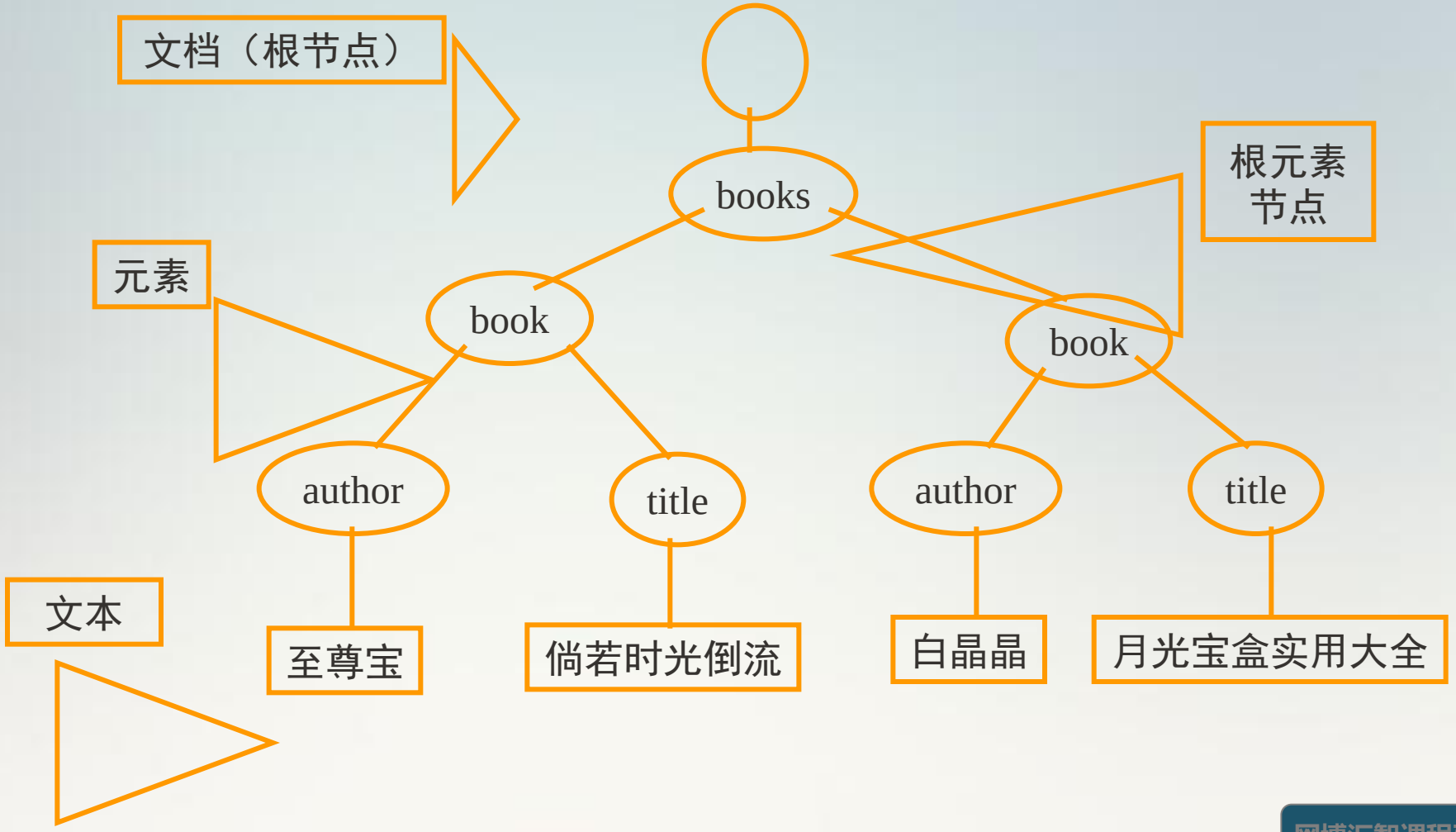
对于XML应用开发来说，DOM就是一个对象化的XML数据接口，一个与语言无关、与平台无关的标准接口规范。它定义了HTML文档和**XML文档的逻辑结构，给出了一种访问和处理HTML文档和XML文档的**方法。利用DOM，程序开发人员可以动态地创建文档，遍历文档结构，添加、修改、删除文档内容，改变文档的显示方式等等。可以这样说，文档代表的是数据，而DOM则代表了如何去处理这些数据。无论是在浏览器里还是在浏览器外，无论是在服务器上还是在客户端，只要有用到XML的地方，就会碰到对DOM的应用。

作为W3C的标准接口规范，目前，DOM由三部分组成，包括：核心（core）、HTML和XML。核心部分是结构化文档比较底层对象的集合，这一部分所定义的对象已经完全可以表达出任何HTML和XML文档中的数据了。HTML接口和XML接口两部分则是专为操作具体的HTML文档和**XML文档所提供的高级接口，使对这两类文件的操作更加方便。**

DOM模型结构


```
<?xml version="1.0" encoding="utf-8" ?>
<books>
        <book>
                <author>至尊宝</author>
                <title>倘若时光倒流</title >
        </book>
        <book>
                <author>白晶晶</author>
                <title>月光宝盒实用大全</title>
        </book>
</books>
```

DOM模型结构（续）



节点类型

- **常见的节点类型：**

  元素：元素是XML的基本构件。典型地，元素可以有其它元素、文本节点或两者兼有来作为其子节点。**元素节点还是可以有属性的唯一类型的节点**。

  属性：属性节点包含关于元素节点的信息，但实际上，不认为它是元素的子节点

  文本：确切来讲，文本节点是：文本。它可以包含许多信息或仅仅是空白。

  文档（根节点）：文档节点是整个文档中所有其它节点的父节点。(**根节点不等于根元素节点！**)

- **较不常见的节点类型：CDATA、注释、处理指令等**

DOM的四个基本接口

文档对象模型利用对象来把文档模型化，这些模型不仅描述了文档的结构，还定义了模型中对象的行为。换句话说，在上面给出的例子里，图中的节点不是数据结构，而是对象，对象中包含方法和属性。在DOM中，对象模型要实现：

- 用来表示、操作文档的接口
- 接口的行为和属性
- 接口之间的关系以及互操作

在DOM接口规范中，有四个基本的接口：Document，Node，NodeList以及NamedNodeMap。在这四个基本接口中，Document接口是对文档进行操作的入口，它是从Node接口继承过来的。Node接口是其他大多数接口的父类，象Document，Element，Attribute，Text，Comment等接口都是从Node接口继承过来的。NodeList接口是一个节点的集合，它包含了某个节点中的所有子节点。NamedNodeMap接口也是一个节点的集合，通过该接口，可以建立节点名和节点之间的一一映射关系，从而利用节点名可以直接访问特定的节点。

Document接口

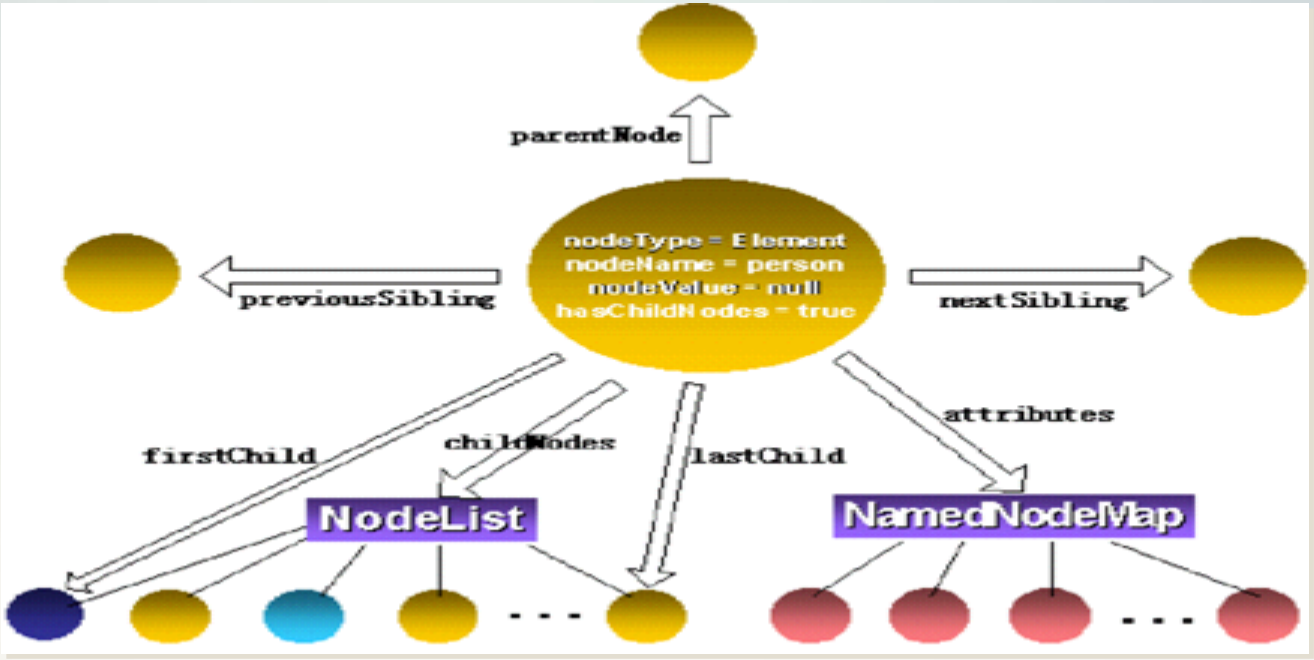
Document接口代表了整个XML/HTML文档，因此，它是整棵文档树的根，提供了对文档中的数据进行访问和操作的入口。

由于元素、文本节点、注释、处理指令等都不能脱离文档的上下文关系而独立存在，所以在Document接口提供了创建其他节点对象的方法，通过该方法创建的节点对象都有一个ownerDocument属性，用来表明当前节点是由谁所创建的以及节点同Document之间的联系。

在DOM树中，Document接口同其他接口之间的关系如下图所示：

Node接口

Node接口在整个DOM树中具有举足轻重的地位，DOM接口中有很大一部分接口是从Node接口继承过来的，例如，Element、Attr、CDATASection等接口，都是从Node继承过来的。在DOM树中，Node接口代表了树中的一个节点。一个典型的Node接口如下图所示：



NodeList接口

NodeList接口提供了对节点集合的抽象定义，它并不包含如何实现这个节点集的定义。NodeList用于表示有顺序关系的一组节点，比如某个节点的子节点序列。另外，它还出现在一些方法的返回值中，例如getElementsByTagName。

在DOM中，NodeList的对象是"live"的，换句话说，对文档的改变，会直接反映到相关的NodeList对象中。例如，如果通过DOM获得一个**NodeList对象，该对象中包含了某个Element节点的所有子节点的集合，**那么，当再通过DOM对Element节点进行操作（添加、删除、改动节点中的子节点）时，这些改变将会自动地反映到NodeList对象中，而不需DOM应用程序再做其他额外的操作。

NodeList中的每个item都可以通过一个索引来访问，该索引值从0开始。

NamedNodeMap接口

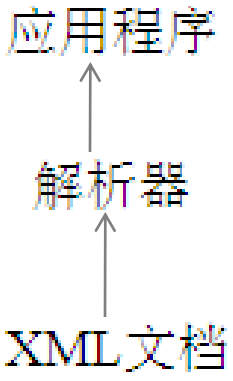
实现了NamedNodeMap接口的对象中包含了可以通过名字来访问的一组节点的集合。不过注意，NamedNodeMap并不是从NodeList继承过来的，它所包含的节点集中的节点是无序的。尽管这些节点也可以通过索引来进行访问，但这只是提供了枚举NamedNodeMap中所包含节点的一种简单方法，并不表明在DOM规范中为NamedNodeMap中的节点规定了一种排列顺序。

NamedNodeMap表示的是一组节点和其唯一名字的一一对应关系，这个接口主要用在属性节点的表示上。

与NodeList相同，在DOM中，NamedNodeMap对象也是"live"的。

解析器基础

- XML解析器实际上就是一段代码，它读入一个XML文档并分析其结构。

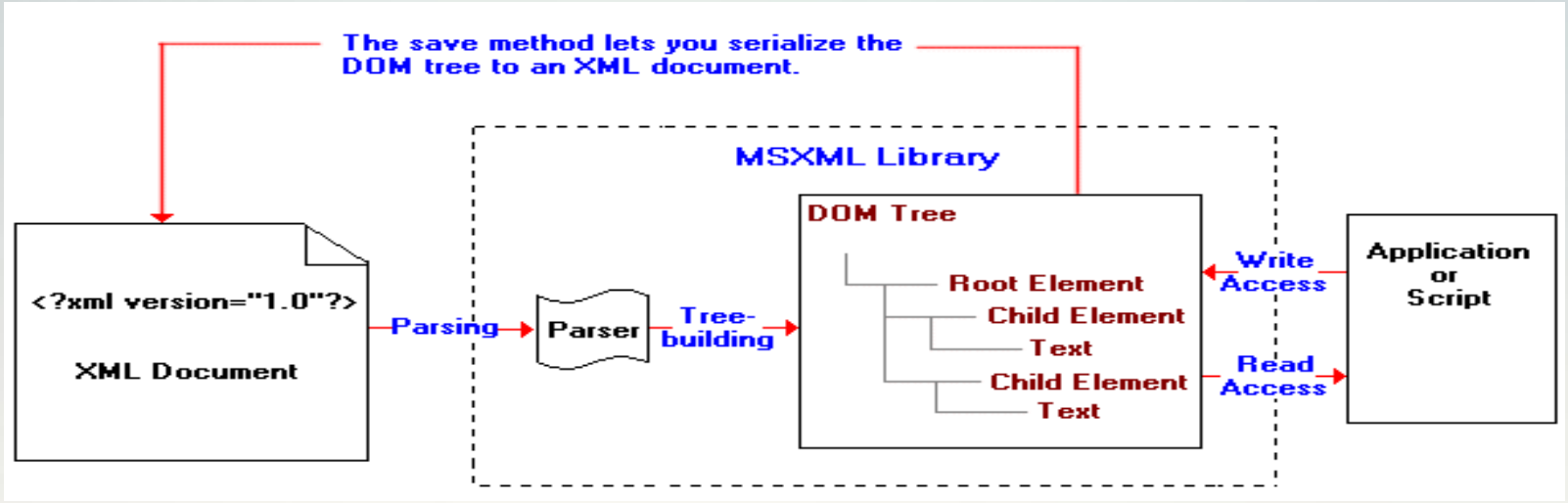


- 分类：
  - 带校验的解析器
  - 不校验的解析器（效率高）

  - 支持DOM的解析器（W3C的官方标准 ）
  - 支持SAX的解析器（事实上的工业标准 ）

Dom基础

- 文档对象模型
- 通过解析XML文档，为XML文档在逻辑上建立一个树模型，树的节点是一个个对象，通过存取这些对象就能够存取XML文档的内容。

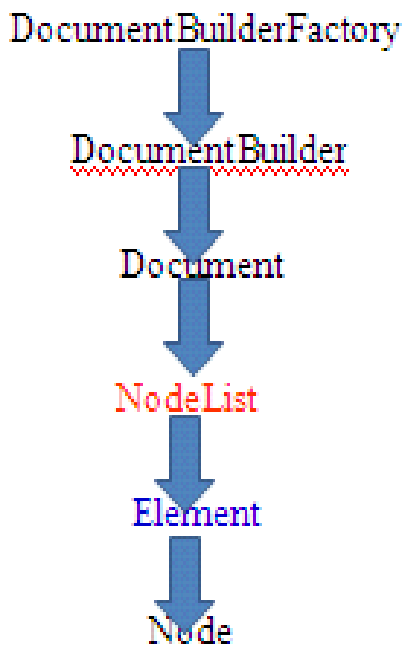


DOM基本对象

- 一切都是节点（对象）

- Node对象：DOM结构中最为基本的对象
- Document对象：代表整个XML的文档
- NodeList对象：包含一个或者多个Node的列表
- Element对象：代表XML文档中的标签元素

文档处理流程




```
<persons>
    <person id = "p01">
        <name>张三</name>
        <age>15</age>
        <address>南京市</address>
    </person>
    <person id = "p02">
        <name>李小龙</name>
        <age>20</age>
        <address>美国</address>
    </person>
</persons>
```

DOM方式解析


```
public static void main(String args[]) {
    try {
        DocumentBuilderFactory factory = DocumentBuilderFactory
                .newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document doc = builder.parse("src/test.xml");
        NodeList nl = doc.getElementsByTagName("person");
        for (int i = 0; i < nl.getLength(); i++) {
            Element node = (Element) nl.item(i);
            System.out.print("name: ");
            System.out.println(node.getElementsByTagName("name").item(0)
                    .getFirstChild().getNodeValue());
            System.out.println();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }

}
```

DOM详解

- DOM的基本对象有5个：Document，Node，NodeList，Element和Attr
- Document对象代表了整个XML的文档，所有其它的Node，都以一定的顺序包含在Document对象之内，排列成一个树形的结构，程序员可以通过遍历这颗树来得到XML文档的所有的内容，这也是对XML文档操作的起点。我们总是先通过解析XML源文件而得到一个Document对象，然后再来执行后续的操作。此外，Document还包含了创建其它节点的方法，createAttribute()用来创建一个Attr对象。它所包含的主要的方法有

DOM详解

- **createAttribute(String)：用给定的属性名创建一个Attr对象，并可在其后使用setAttributeNode方法来放置在某一个Element对象上面。**
- **createElement(String)：用给定的标签名创建一个Element对象，代表XML文档中的一个标签，然后就可以在这个Element对象上添加属性或进行其它的操作。**
- **createTextNode(String)：用给定的字符串创建一个Text对象，Text对象代表了标签或者属性中所包含的纯文本字符串。如果在一个标签内没有其它的标签，那么标签内的文本所代表的Text对象是这个Element对象的唯一子对象。**

DOM详解

- **getElementsByTagName(String)：返回一个NodeList对象，它包含了所有给定标签名字的标签。**
- **getDocumentElement()：返回一个代表这个DOM树的根元素节点的Element对象，也就是代表XML文档根元素的那个对象。**

DOM详解

- Node对象是DOM结构中最为基本的对象，代表了文档树中的一个抽象的节点。在实际使用的时候，很少会真正的用到Node这个对象，而是用到诸如Element、Attr、Text等Node对象的子对象来操作文档Node对象为这些对象提供了一个抽象的、公共的根。虽然在Node对象中定义了对其子节点进行存取的方法，但是有一些Node子对象，比如Text对象，它并不存在子节点，这一点是要注意的。

DOM详解

- Node对象所包含的主要的方法有
- appendChild(org.w3c.dom.Node)：为这个节点添加一个子节点，并放在所有子节点的最后，如果这个子节点已经存在，则先把它删掉再添加进去。
- getFirstChild()：如果节点存在子节点，则返回第一个子节点，对等的，还有getLastChild()方法返回最后一个子节点。
- getNextSibling()：返回在DOM树中这个节点的下一个兄弟节点，对等的，还有getPreviousSibling()方法返回其前一个兄弟节点。
- getNodeName()：根据节点的类型返回节点的名称。
- getNodeType()：返回节点的类型。

DOM详解

- getNodeValue()：返回节点的值。
- hasChildNodes()：判断是不是存在有子节点。
- hasAttributes()：判断这个节点是否存在有属性。
- getOwnerDocument()：返回节点所处的
- Document对象。
- insertBefore(org.w3c.dom.Node new, org.w3c.dom.Node ref)：在给定的一个子对象前再插入一个子对象。
- removeChild(org.w3c.dom.Node)：删除给 定的子节点对象

- replaceChild(org.w3c.dom.Node new, org.w3c.dom.Node old)：用一个新的Node对象代替给定的子节点对象。
- NodeList对象，顾名思义，就是代表了一个包含了一个或者多个Node的列表。可以简单的把它看成一个Node的数组，我们可以通过方法来获得列表中的元素：
- getLength()：返回列表的长度。
- item(int)：返回指定位置的Node对象

DOM详解

- Element对象代表的是XML文档中的标签元素，继承于Node，亦是Node的最主要的子对象。在标签中可以包含有属性，因而Element对象中有存取其属性的方法，而任何Node中定义的方法，也可以用在Element对象上面。
- getElementsByTagName(String)：返回一个NodeList对象，它包含了在这个标签中其下的子孙节点中具有给定标签名字的标签。
- getTagName()：返回一个代表这个标签名字的字符串。

DOM详解

- **getAttribute(String)：返回标签中给定属性名称的属性的值。在这儿需要注意的是，因为XML文档中允许有实体属性出现，而这个方法对这些实体属性并不适用。这时候需要用到getAttributeNode()方法来得到一个Attr对象来进行进一步的操作**
- **getAttributeNode(String)：返回一个代表给定属性名称的Attr对象。**

Attr对象代表了某个标签中的属性。Attr继承于Node，但是因为Attr实际上是包含在Element中
的，它并不能被看作是Element的子对象，因而在DOM中Attr并不是DOM树的一部分，所以Node中的getparentNode(), getpreviousSibling()和getnextSibling()返回
的都将是null。也就是说，Attr其实是被看作包含它的Element对象的一部分，它并不作为DOM树中单独的一个节点出现。这一点在使用的时候要同其它的Node子对象相区别

创建xml的几个类

- TransformerFactory：
  - 主要用来取得Transformer类的实例
- Transformer：
  - 完成输出
- DomSource：
  - 主要用来接收document对象
- StreamResult
  - 指定要使用的输出流对象

```
<student>
  <name>张三</name>
  <age>15</age>
  <address>南京市</address>
</student>
```

表 3-6 StreamResult 类的构造方法

No.	方　法	类　型	描　述
1	public StreamResult(File f)	构造	指定输出的文件
2	public StreamResult(OutputStream outputStream)	构造	指定输出的输出流

创建xml的几个类


```
// 建立DocumentBuilderFactory对象
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = null;
Document doc = null;
db = dbf.newDocumentBuilder();
doc = db.newDocument();
// 建立节点
Element student = doc.createElement("student");
// 创建属性节点
Node sex = (Node) doc.createAttribute("sex");
// 设置属性节点的值
sex.setNodeValue("男");
// 设置属性节点到student节点上
student.setAttributeNode((Attr) sex);
// 建立id节点
Element id = doc.createElement("id");
// 建立名字节点
Element name = doc.createElement("name");
// 建立年龄节点
Element age = doc.createElement("age");
// 为id节点添加内容
id.appendChild(doc.createTextNode("id01"));
// 为name节点添加内容
name.appendChild(doc.createTextNode("张三"));
// 为年龄节点添加内容
age.appendChild(doc.createTextNode("15"));
// 将三个子节点加入到父节点中
student.appendChild(id);
student.appendChild(name);
student.appendChild(age);
// 将根元素加入到根节点上
doc.appendChild(student);
// 输出文档到文件
TransformerFactory tf = TransformerFactory.newInstance();
Transformer t = null;
t = tf.newTransformer();
// 设置编码
t.setOutputProperty(OutputKeys.ENCODING, "utf-8");
// 创建源
DOMSource ds = new DOMSource(doc);
// 创建目标
StreamResult result = new StreamResult(new File("src/output.xml"));
// 输出
t.transform(ds, result);
```