# Overview

## StAX

Java StAX API是一种用于读取和写入XML流的流式API。因此，它类似于较旧的SAX API，除了SAX API只能用于读取XML，不能写入外。

SAX和StAX API的主要区别在于，当使用SAX时，您将向SAX解析器提供handler。该handler具有对与XML文档中发现的实体（元素，注释，文本等）对应的调用方法，。SAX解析器控制对XML文档遍历。而使用StAX解析器，您可以来控制对XML文档遍历。 当您觉得准备好处理下一个元素，文本节点等时，您可以调用next()方法（或相应的方法）。从next()方法获取可以告诉您XML中遇到的实体的对象。

通过这样的XML文档控制遍历可能是一个优点。 迭代可以保持在一个方法的范围内（可能调用子方法）。 这意味着在处理一段文本时，可以使用/访问相同的局部变量，在处理元素时也一样。而使用SAX，两个实体在handler对象中是由两种不同的方法处理。 因此，如果您需要从这些不同的方法访问共享变量，这些变量必须是handler对象中的成员变量。 尽管这不是很大的差异，但是在许多情况下，采用局部变量仍然是最好方式。

StAX接口是Java 6自带，但是没有实现。有一个标准的实现在stax.codehaus.org中.

## SAX Parser

SAX解析器是在XML文档遍历期间按一个实体接一个实体（元素，文本，注释等）进行处理的第一个API。 当您使用SAX解析器时，需要将一个handler对象传递给SAX解析器。 在遍历XML文档的过程中，handler对象对于你想处理的每个“事件”都有一个对应方法。

SAX解析器主要适用于处理可以单独处理每个元素的XML文档。一个文档你想要访问前面部分或者后面部分，用来处理一个给定的元素，这对handler并不是一个容易的事。那么，如果你需要访问前面的部分，那么可以在handler对象中的成员变量中存储那个较早的部分。但是如果你想处理后面的元素，这就有点困难了。要想处理后面遇到的元素，除非先处理前面的元素。但是，如果在处理XML文档时需要跳过XML文档，可能会更容易使用DOM解析器。

在大多数情况下使用StAX比SAX更好。

SAX接口在Java中自带并且有实现（从Java 5起）

## DOM Parser

DOM（文档对象模型）解析器将XML文档解析为对象图。整个文档被转换成一个大对象。一旦创建，您可以随意遍历对象图。您可以随时在图表中上下移动。这个对象图占用了大量的内存，所以这只能在没有其他选项合适的情况下使用。

DOM接口在Java中自带并且有实现（从Java 5起）

## XPath Evaluator

Java自带一个内置的XPath评估器。您设置了一个XPath表达式，并使评估器对DOM对象上的表达式进行评估。评估器然后返回与XPath表达式匹配的元素。XPath可以很方便地查找您需要处理的节点，而不是自己查找到它们。

## XSL Processor

Java自带一个内置的XSL处理器。根据XSL样式表，XSL处理器将输入XML文档转换为输出文档（不一定是XML）。一个样式表的典型应用将是将包含数据的XML文档转换为HTML文档，该文档以更人性化的格式呈现，例如在表格，列表等中。

## JAXB

JAXB是一个类似于DOM的API。 JAXB允许您从XSL Schema生成类，以匹配该模式中定义的XML文档。JAXB允许您将符合该架构的XML文档读入从生成的对象构建的对象结构中。您还可以将该对象结构序列化回磁盘或网络。

生成的JAXB类看起来更像是常规域对象。 它们具有名称与元素名称匹配的getter和setter。 DOM API的方法就像addElement（）等等，其中具体的元素名称是一个参数，您需要知道在DOM结构中的任何给定元素可以添加哪些元素作为子元素。 JAXB生成的clasess通过在类和方法名称中反映允许的结构来帮助您更多。

**Java SAX Parser**

SAX是“Simple API for XML“的缩写。一个Java SAX XML解析器是一种面向流的XML解析器。它在“监听器”遇到XML中的某些结构元素的时候调用某些方法来遍历XML文档。例如，它会为以下事件调用监听器对象。

- startDocument

- startElement

- characters

- comments

- processing instructions

- endElement

- endDocument

这个事件列表可能不全，但是足够明白它是如果工作的下面我们继续看如果创建使用一个Java SAX 解析器。

SAXParserFactory factory = SAXParserFactory.newInstance();

try {

InputStream xmlInput = new FileInputStream("theFile.xml");

SAXParser saxParser = factory.newSAXParser();

DefaultHandler handler = new SaxHandler();

saxParser.parse(xmlInput, handler);

} catch (Throwable err) {

err.printStackTrace ();

}

当你调用SAXParser.parse()方法，SAX解析器开始处理XML文档。xmInput InputSTream作为参数被传递给parse()方法是读取的XML文档的输入流。

需要注意的是一个SaxHandler实例被创建并且作为参数传递给paser()方法。SaxHandler类是org.xml.sax.hepers.DefaultHandler的子类。而DfaultHandler是JDK自带的。

在处理XML时，SAXParser会调用DefaultHandler子类（这里是SaxHandler）实例中对应于解析器在XML文件中发现的实例中的方法。要对这些方法调用做出回应，您将重写DefaultHandler子类中的相应方法。这是一个例子：

public class SaxHandler extends DefaultHandler {

public void startDocument() throws SAXException {

}

public void endDocument() throws SAXException {

}

public void startElement(String uri, String localName,

String qName, Attributes attributes)

throws SAXException {

}

public void endElement(String uri, String localName, String qName)

throws SAXException {

}

public void characters(char ch[], int start, int length)

throws SAXException {

}

public void ignorableWhitespace(char ch[], int start, int length)

throws SAXException {

}

}

DefaultHandler子类的责任是通过这些方法从XML中提取任何必要的信息。如果您需要构建基于XML文件的对象图，则必须在DefaultHandler子类中构建该对象图。

# Java SAX DefaultHandler

org.xml.sax.helpers.DefaultHandler类是SAX 2.0中“侦听器”的基本类。

如本教程中SAX的第一个文本所示，您将创建一个DefaultHandler的子类，并在子类中重写某些继承的方法。在本文中，我将显示一个非常简单的DefaultHandler子类示例，该子类只打印有关XML文件的详细信息。这是代码：

public class SaxHandler extends DefaultHandler {

public void startDocument() throws SAXException {

System.out.println("start document : ");

}

public void endDocument() throws SAXException {

System.out.println("end document : ");

}

public void startElement(String uri, String localName,

String qName, Attributes attributes)

throws SAXException {

System.out.println("start element : " + qName);

}

public void endElement(String uri, String localName, String qName)

throws SAXException {

System.out.println("end element : " + qName);

}

public void characters(char ch[], int start, int length)

throws SAXException {

System.out.println("start characters : " +

new String(ch, start, length));

}

}

当您运行此代码时，使用此文件作为输入：

<root>

<child>

<grandchild>text 1</grandchild>

</child>

<child>

<grandchild>text 2</grandchild>

</child>

</root>

...您可以将以下输出打印到System.out中：

start document :

start element : root

characters :

start element : child

characters :

start element : grandchild

characters : text 1

end element : grandchild

characters :

end element : child

characters :

start element : child

characters :

start element : grandchild

characters : text 2

end element : grandchild

characters :

end element : child

characters :

end element : root

end document :

这是在处理上面显示的XML文件时，SAXParser调用SaxHandler实例中相应方法的顺序。

你可能已经注意到，有时characters()方法打印出一个换行符。这是因为，characters()方法由SAXParser调用，空格字符位于父元素开始标签的末尾和子标签开始标签之间。换句话说，这里使用点（...）标记的空白字符：

<root>**...**

**....**<child>

</child>

还有一些位于元素结束标记结束后的空格字符，直到下一个同级标签的开头，或父元素的结束标签的开头。

## 处理指令

DefaultHandler类还有一个方法，用于在XML文件中找到XML处理指令。以下是该方法的声明：

public void processingInstruction(String target, String data)

throws SAXException {

}

你不会经常使用处理说明，所以我不会在这里详细介绍。现在你知道这是在这里，你可以自己尝试。

## 异常

DefaultHandler类有三种可以重写的方法，用来处理在XML解析过程中遇到的异常。

public void warning(SAXParseException e) throws SAXException {

}

public void error(SAXParseException e) throws SAXException {

}

public void fatalError(SAXParseException e) throws SAXException {

}

假如解析器遇到非法的XML实体（像＆notLegal;）。然后，SAXParser会在断开解析之前调用fatalError()方法。

如果发生的错误不严重，SAXParser可能会调用error()或warning()方法。这样，您可以收集列表中的所有错误，并将它们全部返回，而不是一个接一个地返回。

# Java SAX Schema Validation

可以在使用SAX解析器解析的时候打开XML Schema验证。

Schema schema = null;

try {

String language = XMLConstants.W3C\_XML\_SCHEMA\_NS\_URI;

SchemaFactory factory = SchemaFactory.newInstance(language);

schema = factory.newSchema(new File(name));

} catch (Exception e) {

e.printStackStrace();

}

SAXParserFactory spf = SAXParserFactory.newInstance();

spf.setSchema(schema);

SAXParser parser = spf.newSAXParser();

parser.parse(...);

首先，一个Schema对象是从一些XML schema文件创建的。

其次，创建了一个SAXParserFactory，并且该工厂已经设置了Schema实例。此工厂创建的所有SAXParser实例将使用此XML模式来验证XML文件。

最后，XML文件被解析和验证。

# Java SAX vs. StAX

SAX和StAX都是基于流/事件的XML解析器，但它们的工作方式有微妙的差异。SAX使用“push”模式，StAX使用“pull”模式。理解这可能会令人困惑。因此，我将尝试在本文中更详细地解决这些模型的差异。

## The SAX Push Model

SAX push模型意味着是SAX解析器调用你的handler，而不是handler调用SAX解析器。因此是SAX解析器将时间“pushes”进你的handler。这里总结：

SAX Parser --> Handler

在push模型下，你无法控制解析器如何遍历文件。一旦您启动解析器，它会一直循环到最后，为输入XML文档中的每个XML事件调用handler。

## The StAX Pull Model

StAX pull模型疑问这你的“handler”类调用解析器，而不是其他方式。因此你的hadler类控制分析器何时移动到输入中的下一个事件。换句话说，您的handler将事件“pull”出解析器。此外，您可以随时停止解析。pull模型总结如下：

Handler --> StAX Parser

## Summary of Advantages and Disadvantages

StAX pull模型比SAX push模型有一些优势（少数情况下“控制反转”不是一个优点）。我在下表中总结了SAX和StAX的优缺点：

|  |  |  |  |
| --- | --- | --- | --- |
| SAX + | SAX - | StAX + | StAX - |
| + Schema Validation |  | + Subparsing / Delegation possible + Support for XML Writing | - No Schema Validati |

## StAX Allows Subparsing / Delegation

StAX相比SAX的一大优点在于，pull模式允许通过方法和组件对XML输入进行解析。那是什么意思？

首先，这里是一个XML示例：

<transportInfo>

<driver>...</driver>

<driver>...</driver>

<vehicle>...</vehicle>

<vehicle>...</vehicle>

</transportInfo>

第二，看看这个StAX StreamReader的例子：

public void parse(){

XMLStreamReader streamReader = factory.createXMLStreamReader(

new FileReader("data\\test.xml"));

while(streamReader.hasNext()){

streamReader.next();

if(streamReader.getEventType() == XMLStreamReader.START\_ELEMENT){

String elementName = streamReader.getLocalName();

if("driver".equals(elementName)){

**parseDriverAndAllChildren(streamReader);**

} else if("vehicle".equals(elementName)) {

**parseVehicleAndAllChildren(streamReader);**

}

}

}

}

public void parseDriverAndAllChildren(XMLStreamReader streamReader) {

while(streamReader.hasNext()){

streamReader.next();

if(streamReader.getEventType() == XMLStreamReader.END\_ELEMENT){

String elementName = streamReader.getLocalName();

if("driver".equals(elementName)){

return;

}

} else if(streamReader.getEventType() == XMLStreamReader.START\_ELEMENT){

//do something to child elements...

}

}

}

public void parseVehicleAndAllChildren(XMLStreamReader streamReader) {

while(streamReader.hasNext()){

streamReader.next();

if(streamReader.getEventType() == XMLStreamReader.END\_ELEMENT){

String elementName = streamReader.getLocalName();

if("vehicle".equals(elementName)){

return;

}

} else if(streamReader.getEventType() == XMLStreamReader.START\_ELEMENT){

//do something to child elements...

}

}

}

注意每个方法parseDriverAndAllChildren()和parseVehicleAndAllChildren()能够继续解析循环（while（streamReader.hasNext（）{...}并处理与它们各自关注的“dirver”/“vehicle”元素相关的所有元素。

## StAX 支持 XML 写入

SAX不支持写入XML。如果您不需要编写XML，这不是问题。如果您确实需要编写XML，这可能会令人烦恼。你将不得不编写自己的XML写作机制。而在StAX已经内建了一个写入机制。

## StAX不支持schema验证

据我所知，在撰写本文时，StAX不支持XML Schema验证，SAX具有这一点。

# Java StAX

用来处理XML的StAx Java API旨在解析XML流，就和SAX API一样。StAX和SAX API的主要区别是：

* StAX是一个“pull”API。 SAX是一个“push”API。
* StAX可以同时进行XML读取和写入。 SAX只能做XML读取。

“读+写”API与“读”功能的API之间的区别是非常明显的。但是，“pull”和“push”式API之间的区别并不那么明显，所以我稍稍谈谈一下。有关SAX和StAX的更多功能特征类型比较，请参阅文本SAX与StAX。

## "Pull" vs. "Push" Style API

SAX是一种push式API。这意味着SAX解析器遍历XML并调用由您提供的handler对象的方法。例如，当SAX解析器遇到XML元素的开始开始标签时，它会调用handler对象的startElement。它将XML中的信息“push”到您的对象中。因此名称为“push”式API。这也被称为“事件驱动”API。当在XML文档中找到有趣的东西（“有趣”=元素，文本，注释等）时，您的handler对象将被通知事件调用。

StAX是一个pull式API。这意味着您必须自己将XML文件中的StAX解析器从一个条目移动到下一个条目，就像使用标准Iterator或JDBC ResultSet一样。然后，您可以通过StAX解析器为XML文件（“条目”=元素，文本，注释等）中遇到的每个此类“条目”访问XML信息。

实际上，StAX有两种不同的读取API。一个看起来很像使用迭代器，一个很像使用ResultSet的。这些被称为“迭代”和“游标”读取。

那么这两种读取有什么区别呢？

迭代读取器从nextExvent()方法调用中返回一个XML事件对象。从这个事件对象你可以看到遇到的事件类型（元素，文本，注释等等）。这个事件元素是不可变，并且可以在你的应用程序的其他部分进行解析。您也可以在迭代到下一个事件时挂起到较早的事件对象。正如你所看到的，这样做非常像在迭代集合时如何使用普通迭代器。在这里，您只是迭代XML事件。这是：

XMLEventReader reader = ...;

while(reader.hasNext()){

XMLEvent event = reader.nextEvent();

if(event.getEventType() == XMLEvent.START\_ELEMENT){

StartElement startElement = event.asStartElement();

System.out.println(startElement.getName().getLocalPart());

}

//... more event types handled here...

}

游标读取器next()方法调用不返回事件对象。而是该调用将游标移动到XML中的下一个“事件”。 然后，您可以直接在游标上调用方法来获取有关当前事件的更多信息。这与如何迭代JDBC ResultSet的记录非常相似，并调用getString()或getLong()方法来获取ResultSet指向的当前记录中的值。

XMLStreamReader streamReader = ...;

while(streamReader.hasNext()){

int eventType = streamReader.next();

if(eventType == XMLStreamReader.START\_ELEMENT){

System.out.println(streamReader.getLocalName());

}

//... more event types handled here...

}

所以，主要的区别之一是，您可以在使用迭代器样式API时挂起早期的XML事件对象。使用游标式API时，则不能执行此操作。将游标移动到XML流中的下一个事件之后，您就没有关于上一个事件的信息。这样的话迭代器样式的API更加有优势。

但是，游标样式API被认为比迭代器样式API更具有内存效率。因此，如果您的应用程序需要绝对的顶级性能，请使用游标样式API。

这两个StAX API都将在后面的文章中有更详细的介绍。请参阅本页右侧的目录。

## Java StAX 实现

在编写时（Java 6），只有StAX接口与JDK捆绑在一起。Java内置没有StAX实现。但是，有一个标准的实现可以在这里找到： <http://stax.codehaus.org/>

# Java StAX: XMLInputFactory

javax.xml.stream.XMLInputFactory类是Java StAX API的根组件。从这个类可以创建一个XMLStreamReader和一个XMLEventReader。这里有两个例子：

XMLInputFactory factory = XMLInputFactory.newInstance();

XMLEventReader eventReader =

factory.createXMLEventReader(

new FileReader("data\\test.xml"));

XMLStreamReader streamReader =

factory.createXMLStreamReader(

new FileReader("data\\test.xml"));

## XMLInputFactory Properties

您可以使用setProperty()方法在XMLInputFactory实例上设置各种属性。这是一个例子：

factory.setProperty(XMLInputFactory.IS\_NAMESPACE\_AWARE, true);

有关属性及其含义的完整列表，请参阅StAX API的官方JavaDoc（在Java 6中）。

# Java StAX: XMLOutputFactory

javax.xml.stream.XMLOutputFactory类是Java StAX API的根组件。从这个类可以创建一个XMLStreamWriter和一个XMLEventWriter。这里有两个例子：

XMLOutputFactory factory = XMLOutputFactory.newInstance();

XMLEventWriter eventWriter =

factory.createXMLEventWriter(

new FileWriter("data\\test.xml"));

XMLStreamWriter streamWriter =

factory.createXMLStreamWriter(

new FileWriter("data\\test.xml"));

## XMLOutputFactory Properties

您可以使用setProperty()方法在XMLOutputFactory实例上设置一个属性。这是一个例子：

factory.setProperty(XMLOutputFactory.IS\_REPAIRING\_NAMESPACES, true);

有关属性及其含义的完整列表，请参阅StAX API的官方JavaDoc（在Java 6中）。

# Java StAX: XMLEventReader - The Iterator API

Java StAX中的XMLEventReader类提供了一种用于解析XML的迭代器样式API。换句话说，它允许您从XML中的事件移动到事件，让您控制什么时候移动到下一个事件。在这种情况下，一个“事件”就是一个元素的开头，一个元素的结尾，一组文本等等。换句话说，你可以从SAX解析器获得几乎相同的事件。

您可以通过javax.xml.stream.XMLInputFactory类创建一个XMLEventReader。下面是示例代码：

XMLInputFactory factory = XMLInputFactory.newInstance();

//get Reader connected to XML input from somewhere..

Reader reader = getXmlReader();

try {

**XMLEventReader eventReader =**

**factory.createXMLEventReader(reader);**

} catch (XMLStreamException e) {

e.printStackTrace();

}

一旦创建后，你可以通过Reader的底层来遍历迭代XML输入。下面是使用范例。

while(eventReader.hasNext()){

XMLEvent event = eventReader.nextEvent();

if(event.getEventType() == XMLStreamConstants.START\_ELEMENT){

StartElement startElement = event.asStartElement();

System.out.println(startElement.getName().getLocalPart());

}

//handle more event types here...

}

您可以通过调用其nextEvent()方法从XMLStreamReader获取一个XMLEvent对象。通过调用其getEventType()方法，可以从事件对象中检查事件的类型。根据您遇到的事件类型，您将执行不同的操作。

## XML Stream Events

以下是可以在XML流中遇到的事件的列表。 每个事件在javax.xml.stream.XMLStreamConstants接口中都有一个对应的常量。

* ATTRIBUTE
* CDATA
* CHARACTERS
* COMMENT
* DTD
* END\_DOCUMENT
* END\_ELEMENT
* ENTITY\_DECLARATION
* ENTITY\_REFERENCE
* NAMESPACE
* NOTATION\_DECLARATION
* PROCESSING\_INSTRUCTION
* SPACE
* START\_DOCUMENT
* START\_ELEMENT

## XMLEvent Processing

从XMLEvent对象可以访问相应的XML数据。你还可以获得在XML流中遇到事件的位置（行号+列号）信息。

您可以通过调用以下3种方法之一将事件对象转换为更具体的事件类型对象：

1. asStartElement()
2. asEndElement()
3. asCharacters()

究竟如何处理事件，如START\_DOCUMENT，NAMESPACE或PROCESSING\_INSTRUCTION，我还不知道。我会更新这个文本。幸运的是，我们通常只需要知道START\_ELEMENT，END\_ELEMENT和CHARACTERS事件，所以缺乏这些并不重要

### XMLEvent.asStartElement()

asStartElement()方法返回一个java.xml.stream.StartElement对象。从该对象可以获取元素的名称，获取元素的命名空间以及元素的属性。有关更多详细信息，请参阅Java 6 JavaDoc。

### XMLEvent.asEndElement()

asEndElement()方法返回一个java.xml.stream.EndElement对象。从这个对象可以得到元素名和命名空间。

### XMLEvent.asCharacters()

asCharacters()方法返回一个java.xml.stream.Characters对象。从这个对象可以自己获取字符，以及查看字符是CDATA，空格还是可忽略的空格。

# Java StAX: XMLEventWriter - The Iterator Writer API

Java StAX API中的XMLEventWriter类允许您将StAX XMLEvent写入Writer，OutputStream或Result（特殊JAXP对象）中。

XMLOutputFactory factory = XMLOutputFactory.newInstance();

XMLEventFactory eventFactory = XMLEventFactory.newInstance();

try {

XMLEventWriter writer =

factory.createXMLEventWriter(

new FileWriter("data\\output.xml"));

XMLEvent event = eventFactory.createStartDocument();

writer.add(event);

event = eventFactory.createStartElement(

"jenkov", "http://jenkov.com", "document");

writer.add(event);

event = eventFactory.createNamespace(

"jenkov", "http://jenkov.com");

writer.add(event);

event = eventFactory.createAttribute

("attribute", "value");

writer.add(event);

event = eventFactory.createEndElement(

"jenkov", "http://jenkov.com", "document");

writer.add(event);

writer.flush();

writer.close();

} catch (XMLStreamException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

执行此代码的结果是以下XML文件（插入可换行符）：

<?xml version='1.0' encoding='UTF-8'?>

<jenkov:document xmlns:jenkov="http://jenkov.com" attribute="value">

</jenkov:document>

您可以看到，可以使用XMLEvent和XMLEventWriter生成XML。但是，如果您想要快速输出一些XML，那么使用XMLStreamWriter可能会更好。

## Chaining XMLEventReader and XMLEventWriter

可以将XMLEventReader中的XMLEvent可以直接添加到XMLEventWriter。换而言之，你只是简单将XML事件直接从reader中添加到writer里。您可以使用XMLEventWriter.add(XMLEventReader)方法。

# Java StAX: XMLStreamReader - The Cursor API

ava StAX中的XMLStreamReader类提供了一种用于解析XML的游标样式API。像迭代API一样，它允许您从XML中的事件移动到事件，从而控制何时移动到下一个事件。在这种情况下，一个“事件”就是一个元素的开头，一个元素的结尾，一组文本等等。换句话说，你可以从SAX解析器获得几乎相同的事件。

您可以通过javax.xml.stream.XMLInputFactory类创建一个XMLStreamReader。这是一个Java XML StreamStreamReader示例：

XMLInputFactory factory = XMLInputFactory.newInstance();

//get Reader connected to XML input from somewhere..

Reader reader = getXmlReader();

try {

**XMLStreamReader streamReader =**

**factory.createXMLStreamReader(reader);**

} catch (XMLStreamException e) {

e.printStackTrace();

}

一旦创建后，你可以通过Reader的底层来遍历迭代XML输入。下面是使用范例。

XMLStreamReader streamReader = factory.createXMLStreamReader(

new FileReader("data\\test.xml"));

while(streamReader.hasNext()){

streamReader.next();

if(streamReader.getEventType() == XMLStreamReader.START\_ELEMENT){

System.out.println(streamReader.getLocalName());

}

}

通过调用XMLStreamReader.getEventType（）方法获取事件类型。当您知道事件类型时，您可以根据需要处理给定的事件。

## XML Stream Events

以下是可以在XML流中遇到的事件的列表。 每个事件在javax.xml.stream.XMLStreamConstants接口中都有一个对应的常量。

* ATTRIBUTE
* CDATA
* CHARACTERS
* COMMENT
* DTD
* END\_DOCUMENT
* END\_ELEMENT
* ENTITY\_DECLARATION
* ENTITY\_REFERENCE
* NAMESPACE
* NOTATION\_DECLARATION
* PROCESSING\_INSTRUCTION
* SPACE
* START\_DOCUMENT
* START\_ELEMENT

## XML Event Processing

从XMLStreamReader可以访问相应的XML数据。你还可以获得在XML流中遇到事件的位置（行号+列号）信息。

# Java StAX: XMLStreamWriter - The Cursor Writer API

Java StAX API中的XMLStreamWriter类允许您将XML事件（元素，属性等）写入Writer，OutputStream或Result（特殊JAXP对象）

以下是使用FileWriter将一系列事件写入磁盘的简单示例：

XMLOutputFactory factory = XMLOutputFactory.newInstance();

try {

XMLStreamWriter writer = factory.createXMLStreamWriter(

new FileWriter("data\\output2.xml"));

writer.writeStartDocument();

writer.writeStartElement("document");

writer.writeStartElement("data");

writer.writeAttribute("name", "value");

writer.writeEndElement();

writer.writeEndElement();

writer.writeEndDocument();

writer.flush();

writer.close();

} catch (XMLStreamException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

执行此代码的结果是以下XML文件（插入可换行符）：

<?xml version='1.0' encoding='utf-8'?>

<document><data name="value"></data></document>

# Java DOM

用于XML解析的Java DOM API旨在使用XML作为内存中的对象图（“文档对象模型（DOM）”）。解析器遍历XML文件并创建相应的DOM对象。这些DOM对象以树结构链接在一起。解析器完成后，您就可以从返回的解析结果获得此DOM对象结构。然后，您可以根据需要遍历DOM结构。

以下是一个示例XML文件，以及一个DOM树，说明将XML转换为DOM的原理：

<book>

<title>Fun Software</title>

<author>Jakob Jenkov</author>

<ISBN>0123456789</ISBN>

</book>

## 创建Java DOM XML解析器

使用javax.xml.parsers.DocumentBuilderFactory类创建Java DOM XML解析器。这是一个例子：

DocumentBuilderFactory builderFactory =

DocumentBuilderFactory.newInstance();

DocumentBuilder builder = null;

try {

**builder = builderFactory.newDocumentBuilder();**

} catch (ParserConfigurationException e) {

e.printStackTrace();

}

这个DocumentBuilder实例就是DOM解析器。使用此DOM解析器，您可以将XML文件解析为DOM对象，如下一节所示。

## 使用Java DOM解析器解析XML

如下代码使用DocumentBuilder将XML文件解析为DOM树：

try {

**Document document = builder.parse(**

**new FileInputStream("data\\text.xml"));**

} catch (SAXException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

您现在可以遍历从DocumentBuilder接收到的Document实例。如何遍历Document对象将在下一个文本中讨论（见下面的链接）。

# Java DOM: The Document Object

DOM Document对象表示XML文档。当您使用Java DOM解析器解析XML文件时，将返回一个Document对象。在这个文本中，我会从头开始遍历一个DOM图的。

DOM的两个最常用的功能是：

* + 1. 访问元素的子元素
    2. 访问元素的属性

这是本文涵盖的两个主要功能。

Document接口和所有相关接口位于Java包org.w3c.dom中，因为它们是由万维网联盟 World Wide Web Consortium（W3C）设计的。在JavaDoc中查找DOM接口时需要知道这一点。

## DOM Document元素

DOM对象包含许多以树状结构连接的不同节点。顶级是Document对象。 Document对象只有一个根元素，通过调用getDocumentElement()返回返回给根元素，如下所示：

Element rootElement = document.getDocumentElement();

## DOM Elements, Child Elements, and the Node Interface

根元素的子节点，可以是元素，注释，处理指令，字符等。你可以如下所示得到一个元素的子节点：

NodeList nodes = element.getChildNodes();

for(int i=0; i<nodes.getLength(); i++){

Node node = nodes.item(i);

if(node instanceof Element){

//a child element to process

Element child = (Element) node;

String attribute = child.getAttribute("width");

}

}

getChildNodes()方法返回一个NodeList对象，它是由Node元素组成的List。 Node接口是DOM中几乎所有不同节点类型的父接口。这意味着Document接口继承Node，Element接口继承了Node，Attr（attribute）接口继承了Node等。

Node是DOM中所有节点接口的父接口的事实意味着您有时必须在Node接口中查找所需的方法，如方法getChildNodes()。当尝试迭代Document图时，这是需要注意的。

## DOM Element Attributes

您已经看到，您可以通过Element接口访问元素的属性。有两种方法可以这样做：

String attrValue = element.getAttribute("attrName");

Attr attribute = element.getAttributeNode("attrName");

大多数时候使用getAttribute()方法就行。

Attr接口继承了Node接口。它允许您通过getOwnerElement()方法访问其所属的元素。如果您需要将属性传递给一个或多个方法，这些方法需要访问有关该属性的更多信息才能处理该属性时，则通过此接口访问属性通常很方便。

您可以使用Document对象和相关节点执行更多操作，但您90％的时间内使用的操作只是访问元素的子元素和属性。其余的可以通过查看JavaDoc来查阅。

# Java DOM Schema Validation

在Java中，可以根据XML Schema来验证DOM图。我将在这里展示的技术是在创建DOM图之后的验证，而不是在创建过程中，否则是属于DTD验证的规范。

首先，您必须将XML Schema 加载到一个Schema对象中。一旦加载到Schema对象中，您可以使用相同的Schema对象来验证多个DOM图。这是很明智的，因为那你只需要解析一次XML Schema。如以前的DTD验证机制一样，相比于为每个XML文档同时加载并解析XML文档和XML Schema/DTD，这样在性能上会显著提高。

以下是将XML Schema 加载到Schema实例中的方法：

Schema schema = null;

try {

String language = XMLConstants.W3C\_XML\_SCHEMA\_NS\_URI;

SchemaFactory factory = SchemaFactory.newInstance(language);

schema = factory.newSchema(new File(name));

} catch (Exception e) {

e.printStackStrace();

}

还有一些额外的newSchema()方法，例如一个可以从Java URL对象加载架构的方法。

加载完毕后，您将如下验证DOM文档：

Validator validator = schema.newValidator();

validator.validate(new DOMSource(document));

Schema类可以在java包javax.xml.validation中找到。 DOMSource类可以在java包javax.xml.transform中找到。

如果验证失败，就会抛出异常。您还可以在Validator对象上设置错误处理程序。因此，您可以在同一文档中收集多个schema 错误。