APIs Java para XML

Por Gustavo Link Federizzi

Índice

1. Introdução
- 1.1 Document Object Model
- 1.2 Simple API for XML

2. A API DOM
- 2.1 Os princípios do processamento DOM
- 2.2 Introdução a navegação no DOM
- 2.3 Exemplo de utilização DOM
  - 2.3.1 Uma classe biblioteca de DOMs
  - 2.3.2 Localizando Elementos pelo Tipo
  - 2.3.3 Navegando no Modelo DOM
  - 2.3.4 Acessando Atributos por nome
  - 2.3.5 Modificando um DOM

3. A API SAX
- 3.1 Entrada do Parser SAX
- 3.2 A Interface SAX DocumentHandler
- 3.3 Relatório SAX de Erro
- 3.4 Classe HandlerBase
- 3.5 Exemplo de SAX
  - 3.5.1 Criando HandlerBase
  - 3.5.2 Localizando Parse Erros

4. Links Relacionados

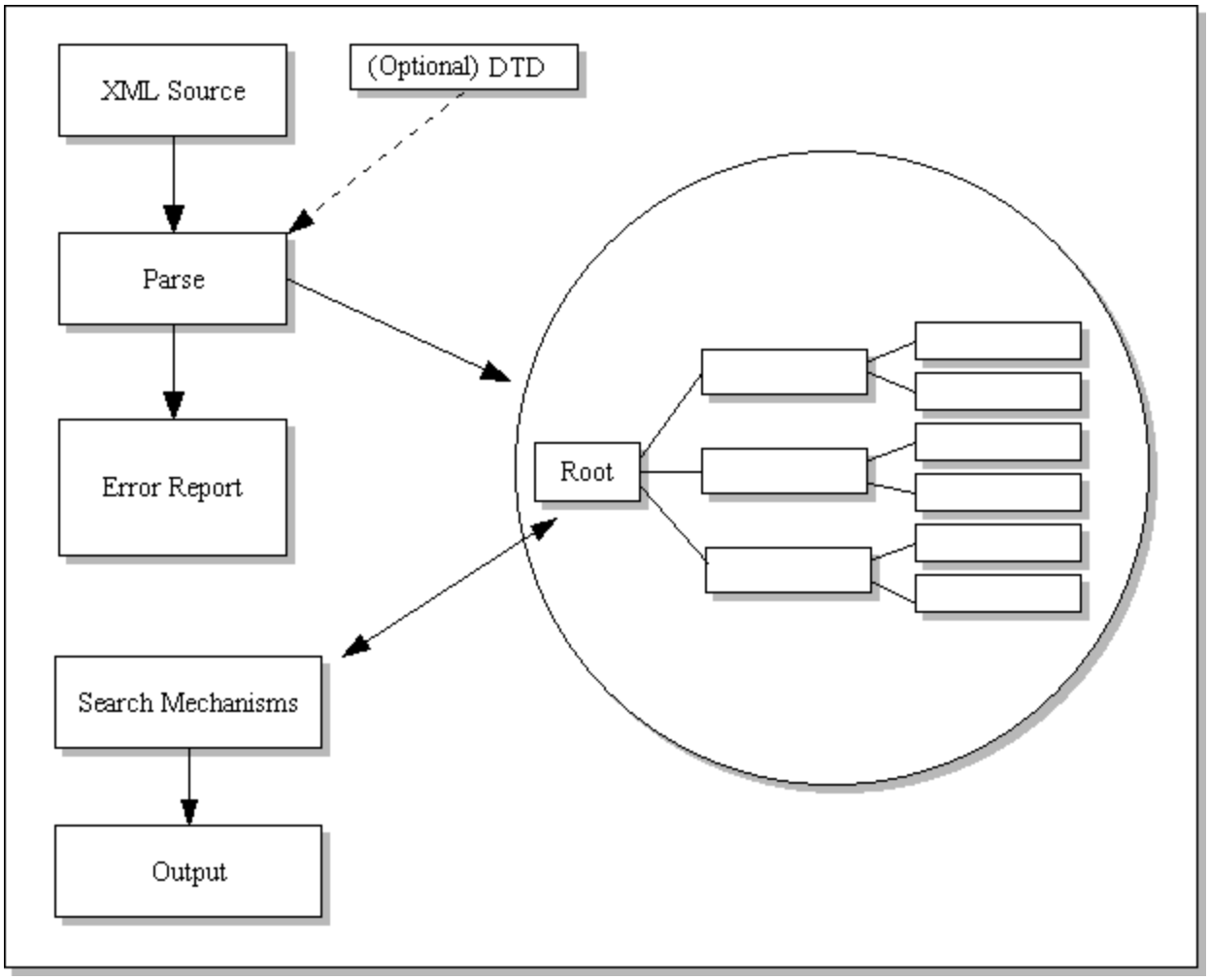
5. Bibliografia

1. Introdução

Este trabalho procura mostrar como diferentes filosofias de programação Java para XML são utilizados. Duas APIs são apresentadas: DOM e SAX.

1.1. Document Object Model - DOM

Neste modelo, o documento XML inteiro é armazenado na memória num formato de árvore de nodos, todos descendendo de uma raiz. O programador pode então aplicar vários métodos para localizar e manipular os nodos. Este é seu modelo conceitual:

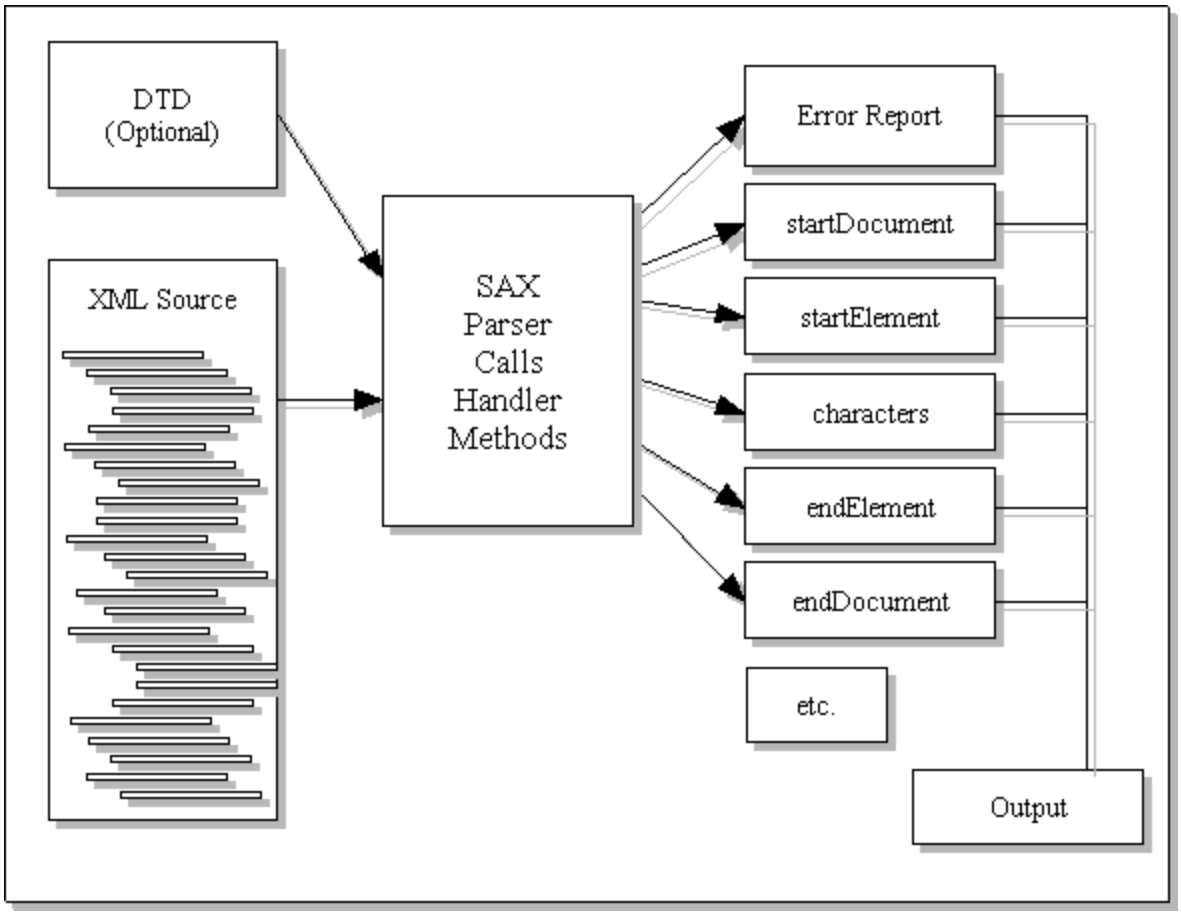


Essencialmente, o programador configura um parser com um XML fonte, e espera terminar. Se existir algum erro, o programador recebe um relatório do erro, caso contrário é retornado um objeto DOM que pode ser manipulado.

De uma maneira geral, para cada procura ou algum tipo de manipulação, é preciso começar pelo elemento raiz e ir subindo na hierarquia. Como todas as informações estão disponíveis na memória, é possível correlacionar e combinar informações como desejar.

1.2. Simple API for XML - SAX

Utilizando-se SAX , o documento XML é lido por um parser, que identifica cada elemento no momento em que é encontrado, fazendo uma chamada para um método especificado pelo programador enquanto o documento é lido. Este é o seu modelo conceitual:



Resumidamente, o programador configura um parser com uma fonte de entrada e o conecta com um conjunto de handler methods. Quando o parser é executado, ele dispara eventos que são capturados pelos handler methods. Cada vez que o parser detecta uma parte importante do documento XML, ele dispara o handler apropriado. Um erro pode acontecer em qualquer momento durante o parsing, então o programador receberá informações sobre os elementos do documento até o acontecimento do erro.

2. A API DOM

A API para trabalhar com DOM é fornecido pela recomendação World Wide Web Consortium (W3C) . Mais detalhes é encontrado de www.w3c.org/DOM/.

A unidade básica considerada no DOM é o Nodo. A interface Node é implementada por todas as diferentes subcategorias de Node listadas na tabela abaixo. Esta forma de representação é muito interessante para as linguagens orientadas a objetos como Java. Todos os tipos de Node possuem uma interface correspondente em Java.

Tabela 2.1 : Subtipos de Node em DOM

Tipo de Nodo	Descrição
Document	Representa o elemento raiz.
Element	Tipo de nodo que representa a maioria dos objetos, exceto texto.
Attr	Um atributo do elemento.
Text	Coleção de caracteres de Attr ou Element.
CDATASection	Um bloco de texto que pode conter elementos de marcação. Se existir, a marcação não é sofre parsing.
EntityReference	Representa uma referência para uma entidade ainda não expandida.
Entity	Representa uma entidade XML.
ProcessingInstruction	Uma instrução que define o comportamento do parser.
Comment	Contém o texto de um comentário contido no XML.
DocumentType	Uma representação do DTD do documento.
DocumentFragment	Um subconjunto de um Document, baseado em um Node, contendo todos os filhos deste.
Notation	Representa uma notação em XML.

Além de vários subtipos de Node , DOM define interfaces de coleções de Nodes como o NodeList e o NamedNodeMap. DOM também especifica uma interface DOMException que pode ser utilizada para comunicar erros de exceção.

Essas interfaces são encontradas no pacote org.w3c.dom.

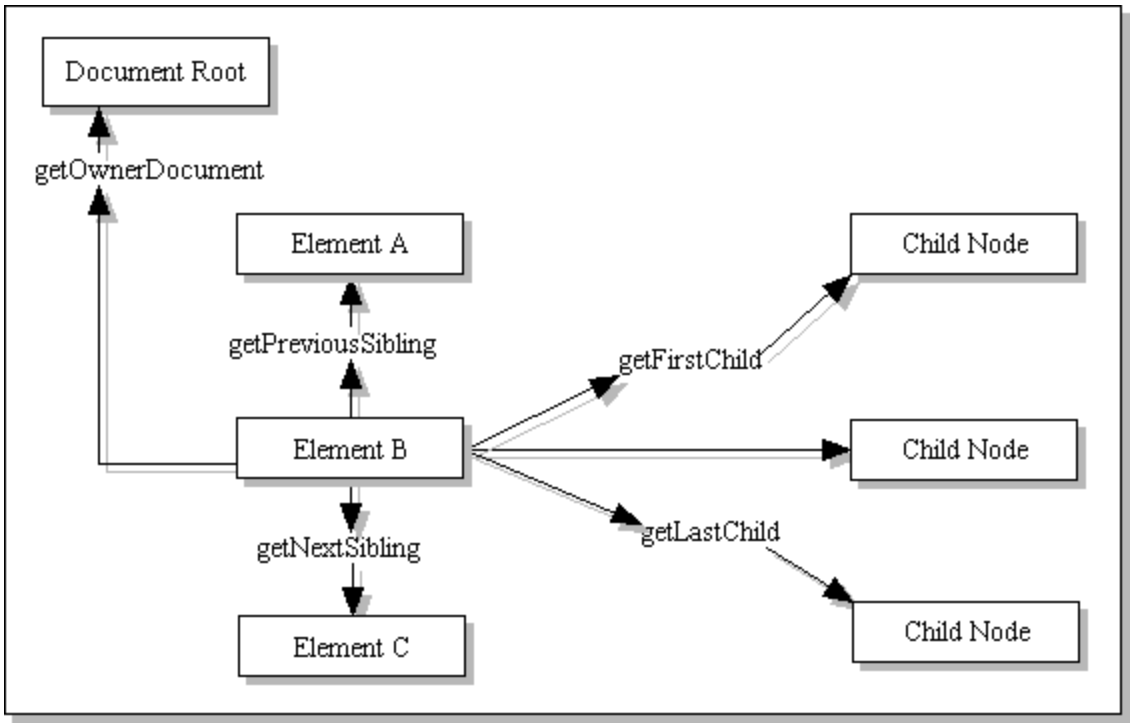
2.1. Os princípios do processamento DOM

A idéia básica é que um documento XML e transformado em um DOM constituído de objetos que implementam interfaces. Cada parte do documento é transformado em um objeto, e as conexões entre os objetos refletem a hierarquia do documento.

Ao se considerar em adotar um determinado parser DOM, é preciso prestar atenção em alguns aspectos. Um deles é o fato do parser validar ou não validar o documento fonte. A validação requer mais memória e processamento, mas é preciso garantir que o documento é bem formado e válido. Outro aspecto diz respeito a compatibilidade com os padrões estabelecidos pelos consórcios.

2.2. Introdução a Navegação no DOM

Vários métodos na interface Node do DOM define relacionamentos entre nodos que provem formas de movimentação pelas conexões entre Nodes. O diagrama da figura abaixo ilustra isso.



2.3. Exemplo de utilização DOM

Neste exemplo, nos utilizaremos o parser XML da Sun: o JAXP API parser toolkit, que substitui diversos toolkits diferentes. A Sun buscou neste novo API flexibilidade, utilizando-se apenas dois arquivos JAR – jaxp.jar e parser.jar - que devem ser colocados no diretório de extensões padrão. Estes pacotes devem ser incluídos

```
import java.io.*;
import java.util.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.w3c.dom.*;
```

Pelo fato do parser fazer todo o trabalho,tudo que se deve fazer é usar um DocumentBuilderFactory para criar um DocumentBuilder e especificar o arquivo de entrada. Por exemplo:

```
File xmlFile = new File( src ); //onde src é uma string
DocumentBuilderFactory dbf = new DocumentBuilderFactory.newInstance();
DocumentBuider db = dbf.newDocumentBuilder();
Document doc = db.parse(xmlFile);
```

O Document é uma referência para um objeto que implementa a interface Document definida no pacote org.w3c.dom. A classe específica é fornecida pelo DocumentBuilder; mas não é preciso se preocupar com os detalhes internos: a interface Document fornece todos os métodos necessários.

O código de parsing também precisa considerar a captura de erros de exceção para reportar

os vários erros que podem acontecer, como:

- **ParserConfigurationException – acionado quando o DocumentBuilderFactory não consegue criar o parser.**
- **SaxParseException – ocorrendo quando o parser encontra um problema na formatação do arquivo XML. O objeto Exception carrega informações sobre a localização do erro no arquivo.**
- **SaxException – O erro mais genérico de um parser.**
- **IOException – Acionado quando um erro de arquivo ocorre.**

2.3.1. Uma classe biblioteca de DOM

Para ilustrar o uso de DOM, nos apresentamos uma classe util Java para manter objetos DOM residentes na memória. Essa classe pode ser útil em aplicações de servidores Web, mantendo na memória os arquivos que são freqüentemente usados, ao invés de fazer o parsing a cada pedido.

A primeira listagem de código nos mostra o método estático usado para obter uma referência a um único objeto DOMLibrary e as variáveis da instância.

```
import java.io.* ;
import java.util.* ;
import javax.xml.parsers.* ;
import org.xml.sax.* ;
import org.w3c.dom.* ;

public class DOMlibrary
{
 private static DOMlibrary theLib ;

 public synchronized static DOMlibrary getLibrary(){
  if( theLib == null ) theLib = new DOMlibrary();
  return theLib ;
 }

 // instance variables below this
 private Hashtable domHash ;

 private String lastErr = "none" ;
 // private constructor to ensure singleton
 private DOMlibrary(){
  domHash = new Hashtable();
 }
```

Um método completo para a criação de um DOM é mostrado na listagem abaixo, Este método é chamado com uma string, dando a localização de um arquivo XML e uma variável booleana com o valor true se a validação do documento é desejada. O método retorna uma referência para um Document caso execute com sucesso ou uma String contendo uma mensagem de erro

caso contrário. Note que se o erro for causado por uma SAXParseException, a String conterá detalhes sobre a localização do erro.

```
// retorna ou um Document ou uma String se acontecer algum erro
  private Object loadXML( String src, boolean validate ) {
    File xmlFile = new File( src ) ;
    String err = null ;
    try {
     DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
      dbf.setValidating( validate );
      DocumentBuilder db = dbf.newDocumentBuilder();
      Document doc = db.parse( xmlFile );
      return doc ;
    }catch(ParserConfigurationException pce){
       err = pce.toString();
    }catch(SAXParseException spe ){
       StringBuffer sb = new StringBuffer( spe.toString() );
       sb.append("\n  Line number: " + spe.getLineNumber());
       sb.append("\nColumn number: " + spe.getColumnNumber() );
       sb.append("\n Public ID: " + spe.getPublicId() );
       sb.append("\n System ID: " + spe.getSystemId() + "\n");
       err = sb.toString();
    }catch( SAXException se ){
      err = se.toString();
      if( se.getException() != null ){
        err += " caused by: " + se.getException().toString() ;
      }
    }catch( IOException ie ){
      err = ie.toString();
    }
    return err ;
  } // end loadXML
```

Outras classes acessam documentos na biblioteca DOM chamando o método getDOM mostrado no código abaixo. Este método é chamado passando uma String com a localização do arquivo XML. Se este arquivo já sofreu parsing, a referência para o objeto Document é retornada. Caso contrário, o arquivo sofrerá o parsing através da chamada do método loadXML.

```
// ou retorna um Document ou null se tiver problemas
public synchronized Document getDOM( String src, boolean validate ){
    Object doc = domHash.get( src );
    File f = null ;
    if( doc == null ){
       System.out.println("DOMlibrary.getDOM new " + src );
       doc = loadXML( src, validate );
       domHash.put( src, doc );
       if( doc instanceof String ){
          lastErr = (String) doc ;
       }
```

```
    }
    // se não for um documento, então deve ser uma String de erro
    if( doc instanceof Document ) {
      return (Document) doc ;
    }
    return null ;
  }
```

O final da biblioteca DOMLibrary é mostrada abaixo com alguns métodos úteis.

```
  // utilize isto para forçar a remoção de um DOM. Retorna
  // a última cópia do DOM ou null caso não exista
  public synchronized Document removeDOM( String src ){
    Document dom = (Document)domHash.get( src );
    if( dom != null ){
      domHash.remove( src );
      // System.out.println("Removed " + src );
    }
    return dom ;
  }

  // utilize isto para forçar uma atualização de um DOM
public synchronized Document reloadDOM( String src, boolean validate ){
    if( domHash.get( src ) != null ){
      domHash.remove( src );
    }
    return getDOM( src, validate );
  }

  public String getLastErr(){ return lastErr ; }

}
```

2.3.2. Localizando Elementos pelo Tipo

Depois que o Document é criado, todas as operações de programação o usarão como ponto de partida. O arquivo XML é fechado e o parser é descartado. Vamos dar uma olhada no que se pode fazer com o documento.

Na nomenclatura para DOMs descrita no W3C, todas as partes do documento é representada por um objeto do tipo Node. Os subtipos de Node definem vários tipos de comportamentos requeridos. O tipo de Node mais comum é o do tipo Element. O DOM fornece um tipo especial de interface de coleção chamada NodeList usada para referenciar uma lista de referências do tipo Node.

Suponha que temos uma lista de livros em XML com o elemento raiz Publications,onde cada tag Book contem uma série de outros elementos descritos no código XML da próxima seção. Obtemos uma NodeList de elementos Book da seguinte maneira:

```
Element dE = doc.getDocumentElement();
NodeList booklist = dE.getElementsByTagName("Book");
```

Sabemos o número de Elements desta maneira:

```
int bookCt = booklist.getLenght();
```

e podemos recuperar um determinado Elemento desta maneira:

```
Element bookE = (Element) booklist.item( n );
```

Onde n é o índice entre 0 e bookCt menos 1. Note que NodeList preserva a ordem dos Elementos encontrados no documento original. Além disso, a NodeList não é uma estrutura estática, mas sim dinamicamente reflete as mudanças no DOM feitos pelo programa. Também note que é preciso fazer um cast ao recuperar um elemento da NodeList, porque o método retorna uma referência a um objeto do tipo Node.

2.3.3. Navegando no Modelo DOM

Para trabalharmos sobre alguns exemplos práticos de código Java para navegação no modelo DOM, vamos utilizar o documento XML descrito abaixo, criado para armazenar informações pertinentes a livros e publicações:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Publications SYSTEM "publications.dtd" >
<Publications date="May 5, 2000">
<Book isbn="1576102912" >
 <Title>Java 2 Exam Cram</Title>
 <Author>Bill Brogden</Author>
 <Edition edition="1" />
 <DatePublished year="1999" />
 <Publisher>The Coriolis Group</Publisher>
 <Press>Certification Insider Press</Press>
 <Series>Exam Cram</Series>
 <Size pp="388"/>
 <Cover img="images/j2ec.gif" />
 <Topic>Java</Topic>
 <Topic>Certification</Topic>
 <Topic>Exam 310-025</Topic>
 <Topic>Certified Programmer for the Java 2 Platform</Topic>
 <Topic>Study Guide</Topic>
 <Errata code="ecj2" />
 <BriefDescription>This compact study guide concentrates on the topics covered
 in Sun's Java 2 programmer certification exam. Numerous questions
 similar to those on the real exam are presented and discussed.
 </BriefDescription>
</Book>
<Book isbn="076450360X">
 <Title>HTML 4 For Dummies</Title>
```

```
 <Edition edition="5"/>
 <Publisher>IDG Books Worldwide</Publisher>
 <Series>Dummies</Series>
 <DatePublished month="7" year="1999" />
 <Size pp="400"/>
 <Author>Natanya Pitts</Author>
 <Author>Ed Tittel</Author>
 <Topic>HTML</Topic>
 <Topic>WWW</Topic>
 <BriefDescription> The fifth edition of this introductory book about the
  Hypertext Markup Language, the markup used to build documents for use
  on the Web. This book covers elements of page design, comprehensive
  markup definitions and examples, and includes a CD with examples
  taken from the text, along with a set of Web pages, templates,
  and online resources built specifically for the book's readers.
 </BriefDescription>
</Book>
</Publications>
```

Dado um Element representando um Book, é possível obter uma lista dos filhos este elemento. Por exemplo, o seguinte fragmento de código recupera uma NodeList de Author Elements pertencentes a um determinado livro. Caso nenhum elemento deste tipo é encontrado, a NodeList terá comprimento zero:

```
NodeList authors = bookE.getElementsByTagName("Author");
```

O texto associado com o Author Element é tratado como um Node filho; logo, para imprimir o nome dos autores neste exemplo, nos utilizamos o seguinte código:

```
for( int i = 0; i < authors.getLength(); i++) {
    Element aE = (Element) authors.item( i );
    String txt = aE.getFirstChild().getNodeValue();
    System.out.println("Author" + txt );
}
```

Localizando Nodos Filhos

O texto de um Element pode ser armazenado no DOM em mais de um Node filho. No exemplo XML de um questionário a seguir, uma seção CDATA é utilizada para que marcação HTML seja incorporada:

```
<Qtext>
<![CDATA[Please fill in <b>all</b> Fields
]]>
</Qtext>
```

O modelo DOM resultante tem três filhos Nodes. O final de linha depois de <Qtext> é um Node, a seção CDATA é um Node, e a linha após o]]> é um Node. Para recuperar todo o texto de um elemento, utilize o método a seguir:

```
String getChildrenText (Element e) {
    StringBuffer sb = new StringBuffer();
    NodeList nl = e.getChildNodes();
    for( int i = 0 ; i < nl.getLength(); i++){
        sb.append( nl.item(i).getNodeValue(); );
        }
return sb.toString();
}
```

Navegando entre irmãos

Um objeto Element também conhece seus relacionamentos próximos no DOM. É possível localizar elementos anteriores e próximos no mesmo nível de hierarquia com o seguintes métodos:

```
Element prev = aE.getPreviousSibling();
Element next = aE.getNextSibling();
```

Estes métodos retornam null quando não existem irmãos naquele ponto.

2.3.4 Acessando atributos por nome

Atributos pertencentes a um Element podem ser acessados por nome para recuperar o seu valor, como em:

```
String isbnStr = bookE.getAttribute("isbn");
```

A String retornada será vazia caso não exista atributo com aquele nome. Este método funciona bem em XML quando os valores de atributos são apenas texto. Caso seu XML tenha referências de entidades nos atributos, como a seguir, onde Attr é uma extensão da interface Node:

```
Attr isbnStr = bookE.getAttribute("isbn");
```

É possível recuperar todos os atributos relacionados com um elemento com um NamedNodeMap, como a seguir:

```
NamedNodeMap map = bookE.getAttributes();
```

O conteúdo de um determinado atributo pode ser acessado pelo nome:

```
Node nd = map.getNamedItem("isbn");
```

Note que será retornado um Node do tipo Attr, ou null caso o atributo não exista. Para realmente recuperar o conteúdo do atributo faça o seguinte:

```
String value = nd.getNodeValue();
```

realmente recuperar o conteúdo do atributo faça o seguinte:

2.3.5 Modificando um DOM

Quando temos um Document na memória, podemos modifica-lo através de diversas maneira. Por exemplo, caso quiséssemos adicionar um atributo printing no tag Edition de um determinado livro, podemos fazer da seguinte maneira:

```
NodeList editNL = bookE.getElementsByTagName("Edition");
Element edition = (Element) editNL.item( 0 );
Edition.setAttribute("printing", "3");
```

A interface Document especifica métodos para criar objetos Node de todos os tipos. Por exemplo, se desejamos adicionar um novo livro ao Document exemplo:

```
Element addBook = doc.createElement("Book");
doc.appendChild( addBook );
```

Title, Author e outros elementos podem ser criados e adicionados como filhos do objeto addBook. Existe também um método de inserir um Node filho antes de um determinado Node.

Para extrair porções de um Document ou reagrupar partes de um documento, DOM fornece a interface DocumentFragment.

3. A API SAX

O padrão SAX cresceu do fato do método DOM era muito complexo e inadequado para várias aplicações. Além disso, até então, cada parser XML para Java tinha seu próprio padrão de interface. Programadores determinados a trazer ordem para este caos produziram os primeiros rascunhos do SAX – num curto espaço de tempo.

O padrão SAX é hoje aceito largamente e forma a base dos parsers do modelo DOM bem como parsers que simplesmente provem uma interface SAX. O pacote que contém todas essas interfaces é o org.xml.sax.

Tabela 3.1 : Interfaces SAX

Interface SAX	Descrição
Parser	Providencia os métodos básicos para criação de um parser
DocumentHandler	Define métodos disparados que são disparados pelos principais eventos durante o parsing.

Locator	**Define um método utilizado para localizar o local do evento no parsing.**
AttributeList	**Define a maneira que o programa acessa uma lista de atributos associado com um elemento.**
DTDHandler	**Define métodos utilizados pelo parser para relatar notações e entidades definidas no DTD.**
ErrorHandler	**Define métodos de comunicação de erro.**
EntityResolver	**Define métodos onde uma aplicação pode prover resolução personalizada de entidades externas.**

Basicamente o programador cria um parser SAX para ler uma determinada fonte e registra objetos implementando as várias interfaces com o parser. Quando o parser é executado, os objetos registrados são notificados quando eventos disparados pelo parser ocorrem.

3.1. Entrada do Parser SAX

A generalização das possíveis fontes de um parser é provido por uma classe InputSource. Uma InputSource pode ser criada por um fluxo de caracteres unicode ou um fluxo de bytes. Como as classes de IO do Java são na sua maioria orientadas a ler ou escrever em fluxo de bytes ou caracteres, existem muitas maneiras de criar um objeto InputSource.

A outra forma de fonte do parser é especificada em termos de sistema de nomeação Uniform Resource Identifier (URI). A fonte pode ser um nome de arquivo ou mesmo uma Uniform Resource Location (URL) para o uso de um arquivo remoto.

3.2. A Interface SAX DocumentHandler

Tipicamente a maioria do trabalho em um programa que usa o método SAX acontece nos métodos definido na interface DocumentHandler. Os métodos são listados abaixo:

Tabela 3.2 : Eventos disparados pelo parser SAX

Parsing Event Handler	Descrição
StartDocument	**Parsing começou.**
EndDocument	**O fim do documento foi alcançado. Este é o último evento disparado.**
StartElement	**Um tag de elemento sofreu parsing. Uma lista de atributos é fornecida.**
EndElement	**O tag de final de um elemento sofreu parsing.**
Characters	**O parser localizou um bloco de caracteres. Pode ser chamada várias vezes para um elemento.**

IgnorableWhitespace	**O parser localizou um bloco de espaços em brancos geralmente ignorado no padrão XML.**
processingInstruction	**O parser localizou uma processing instruction que é retornada como uma String.**
SetDocumentLocator	**Esse evento retorna um objeto que implementa a interface Locator que aponta o local no documento que disparou o evento.**

3.3. Relatório SAX de Erro

A classe SAXException é uma classe básica para relatar erros. Pelo fato de que muita coisa errada pode acontecer durante um parsing, a classe SAXException é usada como um capturador genérico de problemas como erro de rede e arquivos. Maiores detalhes são fornecidos por uma SAXParseException no caso da descoberta de um erro de estruturação do documento pelo parser.

A interface ErrorHandler dispara três métodos que recebem uma SAXParseException: warning, error e fatalError. Quando ocorre um fatalError, a sua chamada é o ultimo evento disparado pelo parser antes de terminar. Métodos da SAXParseException permitem localizar exatamente o ponto onde ocorreu o erro no documento, em termos de linha e posição do caractere que causou o erro.

3.4. Classe HandlerBase

O pacote org.xml.sax prove uma classe chamada HandlerBase. Essa classe implementa as interfaces DocumentHandler, DTDHandler, ErrorHandler e EntityResolver com métodos que não fazem nada. Um programador tem que implementar apenas aqueles métodos relacionados com os eventos que lhe interessam.

3.5. Exemplo de SAX

Programar em SAX requer uma grande mudança de vista em relação a programação DOM. O parser SAX passa apenas uma vez pelo documento fonte, e o programador deve fazer tudo que ele precisa nesta única passagem. Uma das vantagens do SAX é que ele requer pouca memória, e a quantidade não depende do tamanho do documento XML.

Vamos utilizar na construção do nosso exemplo o seguinte documento XML de notícias:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE moreovernews SYSTEM "moreovernews.dtd">
```

```
<moreovernews>
<article id="_8510757">
<url>http://c.moreover.com/click/here.pl?x8510756</url>
<headline_text>Cyclone Commerce Poised to Fulfill Promise
of E-Signature Legislation</headline_text>
<source>Java Industry Connection</source>
<media_type>text</media_type>
<cluster>Java news</cluster>
<tagline> </tagline>
<document_url>http://industry.java.sun.com/javanews/more/hotnews/
</document_url>
<harvest_time>Jul 25 2000  8:34AM</harvest_time>
<access_registration> </access_registration>
<access_status> </access_status>
</article>
<article id="_8514989">
<url>http://c.moreover.com/click/here.pl?x8510853</url>
<headline_text>Schlumberger Showcases Integrated Smart Card-Based
Solutions for Campus Market at Nacubo 2000</headline_text>
<source>Java Industry Connection</source>
<media_type>text</media_type>
<cluster>Java news</cluster>
<tagline> </tagline>
<document_url>http://industry.java.sun.com/javanews/more/hotnews/
</document_url>
<harvest_time>Jul 25 2000  8:34AM</harvest_time>
<access_registration> </access_registration>
<access_status> </access_status>
</article>
```

A implementação do parser SAX da Sun no pacote javax.xml.parsers é bastante flexível. Um parser é obtido via SAXParserFactory da seguinte maneira:

```
SAXParserFactory fac = SAXParserFactory.newInstance();
SAXParser parser = fac.newSAXParser();
```

O parser é iniciado chamando um dos métodos parse fornecido. Estes diferem na maneira em que foi especificado o documento de entrada do parser. É possível usar um objeto InputStream, um objeto File, um URI, ou um objeto org.xml.sax.InputSource. O método parse também requer uma referência para um objeto que estende a classe HandlerInput.

3.5.1 Criando HandlerBase

HandlerBase é uma classe de utilidade que provê métodos nulos para cada método que o parser irá chamar quando ele detecta os eventos de parsing. A classe criada pelo programador deve estender a HandlerBase e prover métodos que sobrescrevem os eventos nos quais está interessado. Numa aplicação típica, este métodos são:

```
public void startDocument() {}
public void endDocument() {}
```

```
public void startElement ( String name, AttributeList attrib) {}
public void endElement( String name ) {}
public void characters ( char[] buf, int start, int length) {}
```

O método characters retorna o texto contido em um elemento, como o headline_text do nosso exemplo. Entretanto, uma chamada deste método não garante que todos os caracteres contidos no elemento serão retornados. Vejamos um exemplo:

```
<article id="_8510757">
<url>http://c.moreover.com/click/here.pl?x8510756</url>
<headline_text>Aqui vai uma manchete explicando a notícia</headline_text>
```

Os métodos chamados neste exemplo são os seguintes em ordem:

1. **startElement chamado com name="article" e attrib contendo um único valor para id**
2. **startElement chamado com name="url"**
3. **characters chamado uma ou mais vezes com o conteúdo do elemento url**
4. **endElement chamado com name="url"**
5. **startElement chamado com name="headline_text"**
6. **characters chamado uma ou mais vezes com o conteúdo do elemento headline_text**
7. **endElement chamado com name="headline_text"**

O problema da programação em SAX é que o programador tem apenas uma passada para fazer o que precisa. Caso se esteja procurando headlines que contem o valor "virus", por exemplo, quando o programador detectou a palavra, o elemento url já havia passado, se ele não guardou o elemento url em algum lugar, já era.

A seguir o código de um exemplo simples de programação SAX. Ele usa o arquivo XML descrito acima e salva apenas os elementos url e headline_text em um arquivo contendo links chamado System.out. Caso uma palavra seja especificada como argumento de execução, apenas headlines com aquela palavra serão gravados.

```
import java.io.* ;
import java.util.* ;
import org.xml.sax.* ;
import javax.xml.parsers.* ;

public class SaxTest extends org.xml.sax.HandlerBase
{
 public static void main(String[] args){
  if( args.length < 1 ){
   System.out.println("Expects xml file name on command line");
   System.exit(1);
  }
  try {
   SaxTest st = new SaxTest( args );
   st.parse();
  }catch(Exception ex){
   ex.printStackTrace( System.out );
  }
```

```
}

// instance variables
File sourceFile ;
String keyword ;
StringBuffer urlSB, headlineSB ;
boolean inUrl, inHeadline ;

SaxTest(String[] args ) {
 sourceFile = new File( args[0] ) ;
 if( args.length > 1 ){
  keyword = args[1].toUpperCase();
 }
}

void parse() throws Exception {
 SAXParserFactory fac = SAXParserFactory.newInstance();
 fac.setValidating( true );
 SAXParser parser = fac.newSAXParser();
 parser.parse( sourceFile, this );
}

public void startDocument(){
 System.out.println("Start parsing " + sourceFile.getAbsolutePath() );
}
public void endDocument(){
 System.out.println("End parsing " );
}

public void startElement( String name, AttributeList attrib ){
 if( inUrl = name.equals("url")){
  urlSB = new StringBuffer( 50 );
 }
 else if( inHeadline = name.equals("headline_text")){
  headlineSB = new StringBuffer(200);
 }
}

public void endElement( String name ){
 if( name.equals("headline_text") ){
  String tmp = headlineSB.toString();
  if( keyword != null ){
   if( tmp.toUpperCase().indexOf( keyword ) < 0 ){
      return ;
   }
  }
  String url = urlSB.toString();
  System.out.println( "
```

Note que para capturar o texto dos dois elementos de interesse, é preciso concatenar s arquivos recebidos de characters em um objeto StringBuffer. Variáveis booleanas são setadas e resetadas pelo método startElement. Essa abordagem é típica do processamento SAX em

qualquer linguagem.

3.5.2 Localizando Parse Errors

Problemas sérios de parsing causarão a chamada do SAXParseException. A listagem abaixo mostra como extrair informações ao máximo da exceção, transformando em uma string:

```
} catch (SAXParseException spe) {
    StringBuffer sb = new StringBuffer (spe.toString());
    sb.append("\n Line number: "+ spe.getLineNumber());
        sb.append("\n Column number: " + spe.getColumnNumber());
        sb.append("\n Public ID: " + spe.getPublicId());
        sb.append("\n System ID" + spe.getSystemId());
        return sb.toString();
}
```

4. Links Relacionados

W3C XML Page – http://www.w3c.org
Página da Sun – http://sun.java.com
Tecnologias XML e Java – http://www.sun.com.br/produtos-solucoes/software/apis.html
API Overview, JavaSoft XML APIs – http://java.sun.com/xml/jaxp-1.1/docs/tutorial/overview/3_apis.html

5. Bibliografia

Além dos links acima, foram consultados os seguintes livros:

- **XML Black Book – 2nd Edition – Ed Coriolis – Autor: Natanya Pits**