

SAX – Simple API for XML

Vanessa de Paula Braganholo
UFRGS - Instituto de Informática
Agosto-Setembro/2002

Introdução

- Padrão desenvolvido na lista de e-mail XML-DEV
- <http://www.saxproject.org/>
- Apesar de não ter sido desenvolvida por um órgão oficial regulador de padrões, SAX tornou-se um padrão *de facto* e é um software livre para uso privado e comercial
- Primeira versão: maio de 1998

Introdução

- SAX é uma API padrão para processamento de dados XML baseado em eventos
- O *parser* entrega informação para o aplicativo disparando eventos

Call-backs e interfaces

- O aplicativo
 - ✓ instancia um objeto *parser*, fornecido por um desenvolvedor
 - ✓ manda executar o *parsing* de um documento ou *stream* de dados através do objeto *parser*
- Enquanto processa os dados XML, o *parser*
 - ✓ detecta partes significativas tais como *start-tag*, *end-tag*, erros, etc,
 - ✓ envia esta informação para o aplicativo utilizando um mecanismo de *call-backs*

Exemplo: *parsing SAX*

```
<pedido>
  <cliente>
    <razao_social>ABC</razao_social>
    <cgc>00.000.000/0001-00</cgc>
  </cliente>
  <itens_pedido>
    <item>
      <produto>caneta azul</produto>
      <quantidade>100</quantidade>
      <preco_unit>2</preco_unit>
    </item>
  </itens_pedido>
</pedido>
```

Exemplo: *parsing SAX*

```
start document
start element: pedido
start element: cliente
start element: razao_social
characters: ABC
end element: razao_social
start element: cgc
characters: 00.000.000/0001-00
...
end element: cliente
start element: itens_pedido
start element: item
start element: produto
characters: caneta azul
end element: produto
...
end document
```

Mecanismo de *call-backs*

- ❶ O aplicativo cria um ou vários objetos cujos métodos são utilizados pelo *parser* para informar a ocorrência de eventos
- ❷ O aplicativo passa referências a estes objetos para o *parser*
- ❸ O *parser* aceita os objetos e os utiliza no processamento dos dados

Interfaces

- Os objetos que o *parser* aceita devem pertencer a uma classe que define uma ou várias interfaces SAX (`org.xml.sax.*`)
- Deste modo, o *parser*
 - ✓ sabe que os métodos necessários estão presentes
 - ✓ executa chamadas aos métodos apropriados de acordo com a ocorrência dos eventos.

Interfaces SAX

- Parser
 - ✓ implementada pelo *parser* propriamente dito.
- DocumentHandler
 - ✓ manipula eventos básicos de marcação - início de documento, início de elemento, etc.
- AttributeList
 - ✓ manipula atributos
- ErrorHandler
 - ✓ manipula erros

Interfaces SAX

- EntityResolver
 - ✓ manipula entidades
- Locator
 - ✓ localização de cada erro - linha/coluna
- DTDHandler
 - ✓ informações sobre entidades externas que não estão em sintaxe XML - figuras, arquivos binários.
- HandlerBase
 - ✓ implementa todas as interfaces com um comportamento default.

Atividades não contempladas pelo padrão SAX

- Existem algumas atividades importantes envolvidas no *parsing*, navegação e processamento de documentos que o padrão SAX não cobre
 - ✓ escrever uma estrutura em formato XML está fora do escopo do padrão
- Estas características, quando implementadas, serão diferentes em cada *parser*
- Utilizaremos o parser "IBM for Java" em nossos exemplos.

Implementação Java

- Para utilizar o *parser*, a seguinte sentença é necessária:

```
import org.xml.sax.*;
```
- As classes abaixo implementam a interface Parser
 - org.apache.xerces.parsers.SAXParser
 - *parsing* sem validação.
 - com.ibm.xml.parser.ValidatingSAXParser
 - *parsing* com validação.

Parser

- O desenvolvedor do *parser* cria uma classe para executar o *parsing* de um documento XML ou um *stream* de dados
- Esta classe implementa a interface Parser

Interface Parser

```
void parse(InputSource src)
        throws SAXException, IOException;
void parse(String src)
        throws SAXException, IOException;
void setDocumentHandler(DocumentHandler doch);
void setErrorHandler(ErrorHandler errh);
void setDTDHandler(DTDHandler dtdh);
void setEntityResolver(EntityResolver entres);
void setLocale(Locale loc) throws SAXException;
```

Interface Parser

- Este métodos dividem-se em dois grupos
- O maior grupo contém os métodos do tipo *set* ✓ usados pela aplicação para registrar objetos no *parser* que correspondem a outras interfaces
- O segundo grupo é o grupo *parse* ✓ métodos deste grupo são utilizados pelo aplicativo para executar o *parsing*

Funcionamento

- Depois de registrar um ou mais objetos no *parser*, a aplicação chama um dos métodos *parse*
- O *parser* começa a ler os dados XML
- Quando um objeto significativo é encontrado, o *parser* pára e a informação é enviada para o aplicativo chamando o método apropriado através de um dos objetos registrados
- O *parser* espera o método retornar para continuar o processamento.

Manipulador de Documentos

- O programador deve criar uma classe para implementar a interface DocumentHandler
- Através desta classe, o aplicativo trata os eventos disparados pelo parser

Interface Document Handler

```
void startDocument() throws SAXException;
void endDocument() throws SAXException;
void startElement(String name,
                  AttributeList atts) throws SAXException;
void endElement(String name)
                 throws SAXException;
void characters(char ch[], int start,
                int length) throws SAXException;
void ignorableWhitespace(char ch[], int start,
                         int length) throws SAXException;
void processingInstruction(String target,
                           String data) throws SAXException;
void setDocumentLocator(Locator myLoc);
```

Funcionamento

- O objeto *parser* recebe uma referência ao objeto DocumentHandler
- Este objeto é criado pela aplicação através de uma chamada ao método setDocumentHandler

Exemplo

- Implementação de uma classe que executa um *parsing*
- Implementação da classe Handler, que implementa a interface DocumentHandler

Exemplo

```
import org.xml.sax.*;
import com.ibm.xml.parsers.*;

public class Parse
{
    ValidatingSAXParser myParser;

    static public void main(String[] argv)
    {
        try
        {
            if (argv.length != 1)
            {
                // Usuário deve informar o nome do arquivo XML.
                System.err.println("Usage: Parse filename");
                System.exit(1);
            }
        }
    }
}
```

Exemplo

```
// Cria um novo handler para o parser
Parse p = new Parse();

// Cria uma instância do parser
Parser myParser = new ValidatingSAXParser();

// Cria uma instância da classe que implementa a
// interface DocumentHandler
Handler myHandler = new Handler();
myParser.setDocumentHandler(myHandler);
try
{
    myParser.parse(argv[0]);
}
catch (SAXException e)
{
    System.out.println(e.getMessage());
}
catch (Exception e)
{
    System.out.println(e.toString());
}
}
```

Exemplo – Implementação da Classe Handler

```
import org.xml.sax.*;
import com.ibm.xml.parsers.*;

public class Handler implements DocumentHandler
{
    Locator myLocator;

    public void startDocument() throws SAXException
    {
        System.out.println("Início do Documento");
    }

    public void endDocument() throws SAXException
    {
        System.out.println("Fim do documento");
    }

    public void startElement(String name, AttributeList atts)
        throws SAXException
    {
        System.out.println("Início de elemento");
    }

    public void endElement(String name) throws SAXException
    {
        System.out.println("Fim de elemento");
    }

    public void characters(char[] cbuf, int start, int len)
    {
        System.out.println("Texto");
    }

    public void setDocumentLocator(Locator loc)
    {
        myLocator = loc;
    }

    public void processingInstruction(String target, String data)
    {
        System.out.println("Instrução de processamento!");
    }

    public void ignorableWhitespace(char ch[], int start, int length)
    {
        System.out.println("Espaço ignorável!");
    }
}
```

Exemplo – Implementação da Classe Handler

```
public void endElement(String name) throws SAXException
{
    System.out.println("Fim de elemento");
}

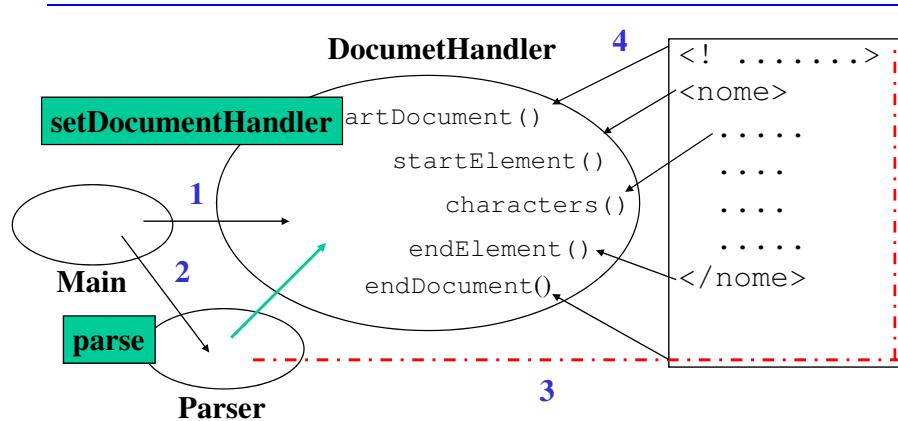
public void characters(char[] cbuf, int start, int len)
{
    System.out.println("Texto");
}

public void setDocumentLocator(Locator loc)
{
    myLocator = loc;
}

public void processingInstruction(String target, String data)
{
    System.out.println("Instrução de processamento!");
}

public void ignorableWhitespace(char ch[], int start, int length)
{
    System.out.println("Espaço ignorável!");
}
```

Funcionamento



Funcionamento

- O primeiro método chamado é `startDocument` e o último é `endDocument`
- Métodos úteis para inicializar variáveis, abrir e fechar arquivos, etc

```
public void startDocument() {  
    total_pedido=0;  
}
```

Funcionamento

- O método `startElement` é chamado quando uma *start-tag* é encontrada no fluxo de dados
- O método `endElement` é chamado quando uma *end-tag* é encontrada no fluxo de dados.

Funcionamento

- Exemplo: identificar a razão social do cliente

```
public void startElement(String name, AttributeList  
atts)  
{  
    if (name.equals("razao_social"))  
        InRazaoSocial = true;  
}  
  
public void endElement(String name)  
{  
    if (name.equals("razao_social"))  
        InRazaoSocial = false;  
}
```

Funcionamento

- O método `characters` é chamado quando um *string* de texto é encontrado. O exemplo abaixo converte o *array* de entrada para um *string* simples e armazena o nome do cliente.

```
public void characters(char ch[], int start,
                      int length)
{
    String chars = new String(ch, start, length);
    if ( InRazaoSocial )
        NomeCliente = chars;
}
```

Funcionamento

- `ignorableWhitespace`
 - ✓ este método é chamado quando um *string* de caracteres que podem ser ignorados é encontrado
 - ✓ espaços em branco que aparecem em elementos que somente podem conter elementos são considerados caracteres ignoráveis
 - ✓ quando uma DTD não é utilizada este método não é chamado, pois o *parser* não é capaz de distinguir entre elementos que podem conter texto e elementos que só podem conter outros elementos

Funcionamento

- `processingInstruction`
 - ✓ este método é chamado quando uma instrução de processamento é encontrada
 - ✓ o método recebe como parâmetros
 - nome da aplicação de destino
 - instrução de processamento

Funcionamento

```
public void processingInstruction(String
                                  target, String data)
{
    if ( target.equals("ACME") )
    {
        // o aplicativo é o processador ACME
        if ( data.equals("new_page") )
        {
            // quebra de página aqui...
        } ...
    }
}
```

Atributos

- Quando o *parser* informa para a aplicação que uma *star-tag* foi encontrada, ele chama o método `startElement`
- Uma *start-tag* pode conter um ou mais atributos

Atributos

- Problema:
 - ✓ não existe limite no número de atributos que um elemento pode conter
 - ✓ portanto, esta informação não pode ser passada como parâmetro
- Solução:
 - ✓ criar um objeto para encapsular todos os detalhes dos atributos
 - ✓ este objeto deve implementar a interface `AttributeList`

Interface AttributeList

```
int getLength();  
String getName(int i);  
String getType(int i);  
String getType(String name);  
String getValue(int i);  
String getValue(String name);
```

Interface AttributeList

- `getLength`
 - ✓ retorna um valor inteiro que representa o número de atributos que o elemento possui
 - ✓ o valor zero indica que não existem atributos
 - ✓ cada atributo é identificado por um valor de índice, começando em zero.
- `getName`
 - ✓ recebe um valor de índice e retorna o nome do atributo correspondente.

Interface AttributeList

- Exemplo: recupera o nome do último atributo

```
String lastAttribute = null;  
int totalAtts = atts.getLength();  
if ( totalAtts > 0 )  
    lastAttribute =  
        atts.getName(totalAtts - 1);
```

Interface AttributeList

- getValue

- ✓ retorna o valor de um atributo
- ✓ é possível utilizar tanto um valor de índice quanto um nome de atributo.

```
lastAttValue = atts.getValue(totalAtts - 1);
```

Interface AttributeList

- getType

- ✓ retorna informações sobre o tipo do atributo
- ✓ quando uma DTD é utilizada, cada atributo possui um tipo de dados (CDATA, ID ou NMOKEN)
- ✓ quando o *parser* não possui acesso à DTD, o tipo *default* (CDATA) é utilizado

Interface AttributeList

- Exemplo: reconstrói a lista de atributos original

```
public void startElement(String name, AttributeList  
                         atts)  
{  
    System.out.print( "<!ATTLIST " + name + " " );  
    for( int i =0; i < atts.getLength(); i++ )  
    {  
        System.out.print( atts.getName(i) + " " +  
                        atts.getType(i) + " " +  
                        "#IMPLIED \"\" "+  
                        atts.getValue(i) + "\" \"\n" );  
    }  
    System.out.print( "> \n" );  
}
```

Interface AttributeList

- `getValue`
 - ✓ recupera o valor de um atributo nomeado
 - ✓ o nome do atributo deve ser passado como parâmetro
 - ✓ se o atributo não existe, um valor nulo é retornado

Interface AttributeList

- Exemplo: recupera o valor do atributo `numero` do elemento `pedido`

```
public void startElement(String name,
    AttributeList atts)
{
    if (name.equals("pedido"))
    {
        num_pedido =
            atts.getValue("numero");
    }
}
```

Interface ErrorHandler

- Um aplicativo implementa a interface `ErrorHandler` para ser informado sobre erros e *warnings*
- Métodos:

```
void warning(SAXParseException err)
    throws SAXException;
void error(SAXParseException err)
    throws SAXException;
void fatalError(SAXParseException err)
```

Interface ErrorHandler

- Tipicamente o mesmo objeto que manipula eventos normais também manipula eventos de erro
- O método `setErrorHandler` registra no parser o objeto ao qual os eventos de erro devem ser enviados

```
class MyClass implements DocumentHandler,
    ErrorHandler
{
    ...
    myParser.setDocumentHandler( this );
    myParser.setErrorHandler( this );
```

Interface ErrorHandler

- Exemplo: Mostra mensagem cada vez que um *warning* é disparado

```
public void warning (SAXParserException err)
{
    System.out.print( "WARNING: " + err);
}
```

Interface Locator

- O *parser* pode informar ao aplicativo a entidade, o número de linha e o número do caractere que originou o erro, através de um objeto específico
- Este objeto deve ser instância de uma classe que implementa a interface Locator

Interface Locator

- Métodos definidos na interface Locator

```
int getLineNumber();
/* retorna -1 se a informação não
está disponível */

int getColumnNumber();
/* idem getLineNumber */

String getSystemId();
String getPublicId();
```

Interface Locator

- O método `setDocumentLocator` da interface `DocumentHandler` registra o objeto que manipula a informação sobre erros passada pelo *parser*

```
Locator myLocator;
...
public void setDocumentLocator(Locator
aLocator)
{
    myLocator = aLocator;
}
```

Interface Locator

- Exemplo: como acessar informação sobre erros através da interface Locator

```
Locator myLocator;  
...  
public void error(SAXParseException err)  
{  
    int ln = myLocator.getLineNumber();  
    int ch = myLocator.getColumnNumber();  
    String ent = myLocator.getSystemID();  
    System.out.println("ERROR (" + err + ")" +  
        " at " + ln + ":" + ch +  
        " in file " + ent );  
}
```

Interface DTDHandler

- Entidades externas que não seguem a sintaxe XML não podem ser processadas pelo *parser*.
 - ✓ Por isso não são passadas para a aplicação
 - ✓ A aplicação pode nem ficar sabendo da existência delas
- A interface DTDHandler foi definida para contornar esta limitação
 - ✓ Informa a aplicação sobre
 - Entidades binárias
 - Declarações de notação

Interface DTDHandler

- A interface DTDHandler deve ser implementada por uma classe no aplicativo
- O *parser* recebe uma referência para um objeto instância dessa classe

```
class MyClass implements DTDHandler {  
    ...  
    myParser.setDTDHandler(this);
```

Interface DTDHandler

- Métodos definidos na interface DTDHandler

```
void notationDecl(String name,  
                  String publicId, String systemId)  
    throws SAXException;  
  
void unparsedEntityDecl(String name,  
                        String publicID, String systemID,  
                        String notationName)  
    throws SAXException;
```

InputSource

- É possível executar o *parsing* de um fluxo de caracteres ou fluxo de *bytes* (além de uma URL)
- Isto pode ser muito útil quando um aplicativo lê dados diretamente de outro aplicativo
- A classe InputSource possui métodos que especificam a natureza da fonte de dados a ser manipulada

InputSource

```
InputSource();  
InputSource(String systemId);  
InputSource(InputStream byteStream);  
InputSource(Reader characterStream);  
  
void setPublicId(String publicId);  
void setSystemId(String systemId);  
void setByteStream(InputStream byteStream);  
void setCharacterStream(Reader  
    characterStream);  
void setEncoding(String encoding);
```

InputSource

```
String getPublicId();  
String getSystemId();  
InputStream getByteStream();  
Reader getCharacterStream();  
String getEncoding();
```

InputSource

- Nesta classe,
- métodos *set...* são alternativas de uso dos construtores
- métodos *get...* são utilizados pelo parser para extrair a informação fornecida pelo aplicativo

InputSource

- Exemplo: parsing de um fluxo de caracteres

```
String text = "<note>Esta é uma nota</note>";
StringReader myReader = new StringReader(text);
InputSource myInputSource = new InputSource();
myInputSource.setCharacterStream(myReader);
...
myParser.parse(myInputSource);
```

Interface EntityResolver

- Ao utilizar os métodos e classes discutidas até agora, as aplicações não enxergam a estrutura física dos dados XML
- O *parser* contém um gerenciador de entidades que esconde estas complexidades da aplicação, a qual vê os dados como um simples stream de dados

Interface EntityResolver

- Entretanto, pode ser útil saber mais sobre cada entidade, interceptar uma referência a uma entidade e redirecionar o parser para outro recurso ou para uma cópia local do recurso considerado, entre outras coisas.
- É possível interceptar referências a entidades utilizando a interface EntityResolver

Interface EntityResolver

- Uma classe pertencente ao aplicativo implementa esta interface
- A interface define um único método:

```
InputSource resolveEntity(String publicId,
                         String systemId);
```

Interface EntityResolver

- O *parser* detém o processamento quando encontra uma referência para uma entidade externa (não binária) e passa o identificador para o aplicativo utilizando o método `resolveEntity`, e espera o método retornar
- O retorno pode ser um valor, um arquivo de dados ou fluxo de dados. No caso de valor nulo, o *parser* continua normalmente
- Caso contrário, o parser deve processar os dados recebidos antes de continuar. Os dados são retornados através de um objeto `InputSource`

Interface EntityResolver

- Exemplo: ilustra o uso do método `resolveEntity`
- Retorna um identificador de sistema válido (arquivo "disc.xml" no diretório "xml") quando a entidade encontrada possui um identificador de sistema com o valor "Disclaimer" ou um identificador público com o valor
"–//MyCorp//TEXT Disclaimer//EN"

Interface EntityResolver

```
public InputSource resolveEntity(String publicId, String
    systemId)
{
    if (systemId.equals("Disclaimer") ||
        publicId.equals("–//MyCorp//TEXT Disclaimer//EN"))
        return (new InputSource("file://xml/disc.xml"));
}
```

Interface HandlerBase

- Interface `HandlerBase` implementa um comportamento padrão para cada evento
- Utilidade:
 - ✓ quando a aplicação precisa fazer algo muito simples com a fonte de dados XML
- Não é necessário implementar todas as interfaces descritas anteriormente

Interface HandlerBase

```
import org.xml.sax.HandlerBase;  
...  
public class myHandler extends HandlerBase {  
    public void myHandler() {}  
    public void processinInstruction  
        (String target, String content) {  
        System.out.println(target + "\t" + content);  
    }  
}
```

Exemplo: espaços em branco ignoráveis

- SAXSample.java

- ✓ Conta o número de elementos e o número de espaços em branco ignoráveis de um documento XML

Exemplo para processar pedidos

- ProcessaPedido.java

Exercício 1

- Modificar a classe ProcessaPedido.java para exibir os itens do pedido e o valor de cada item.

Exercício 2

- Modificar a classe ProcessaPedido.java para exibir a nota fiscal referente ao pedido.
 - ✓ Processe com o documento pedido3.xml, que possui dados do endereço do cliente.

ABC	00.000.000/0001-00
Rua das Flores, 75	Porto Alegre RS

Produto	Quant P.Unit. P.Total

caneta azul	100 2 200
papel	100 8 800

	1000

Exercício 3

- Crie uma classe que encontra as instruções de processamento do arquivo view1.xml e as exibe na tela
- Atenção!! Exibir somente as instruções de processamento que contêm instruções SQL!!

Exercício 4

- Imprimir na tela o documento XML que está sendo lido