SAX

(*S*imple *A*pplication Programming Interface for *X*ML)

Georg Kuschk 14.06.2002

Gliederung

Rückblick - DOM / Nachteile
Abhilfe : SAX

SAX - Funktionsweise

Der SAX Parser
- -Interface XMLReader
- -Interface ContentHandler
- -Interface ErrorHandler

Beispielprogramm

SAX 1.0 => SAX 2.0
Geschichte / Entstehung

Schlußfolgerungen / Zusammenfassung
Quellen

Rückblick : DOM / Nachteile

Baum-basiertes API :

- Baumstruktur des Dokuments beansprucht große Speichermenge (bis zu 5-10 mal die Größe des Dokuments)

- gesamte Datei muß in den Speicher geladen werden

- viele Anwendungen verwenden eigene Datenstrukturen
  ineffizient : - Konstruktion eines Baumes
  - Abbildung auf die eigene Datenstruktur
  - Verwerfung des Baumes

- Ungünstiges Beipiel : suchen eines Elementes innerhalb eines großen XML-Dokuments

Abhilfe : SAX

Ereignis(Event)-basiertes API :

- Event-basierte API‘s stelle einen Zugriff auf XML-Dokumente auf einer tieferen Ebene dar
- Durch sequentielle Abarbeitung der Events wird weniger Speicher als bei DOM basierten API‘s benötigt
- Anwendung kann mittels Callback-Event-Handler die Datenstruktur direkt konstruieren

SAX - Funktionsweise

- Mittels dem Interface XMLReader wird ein Parser initialisiert

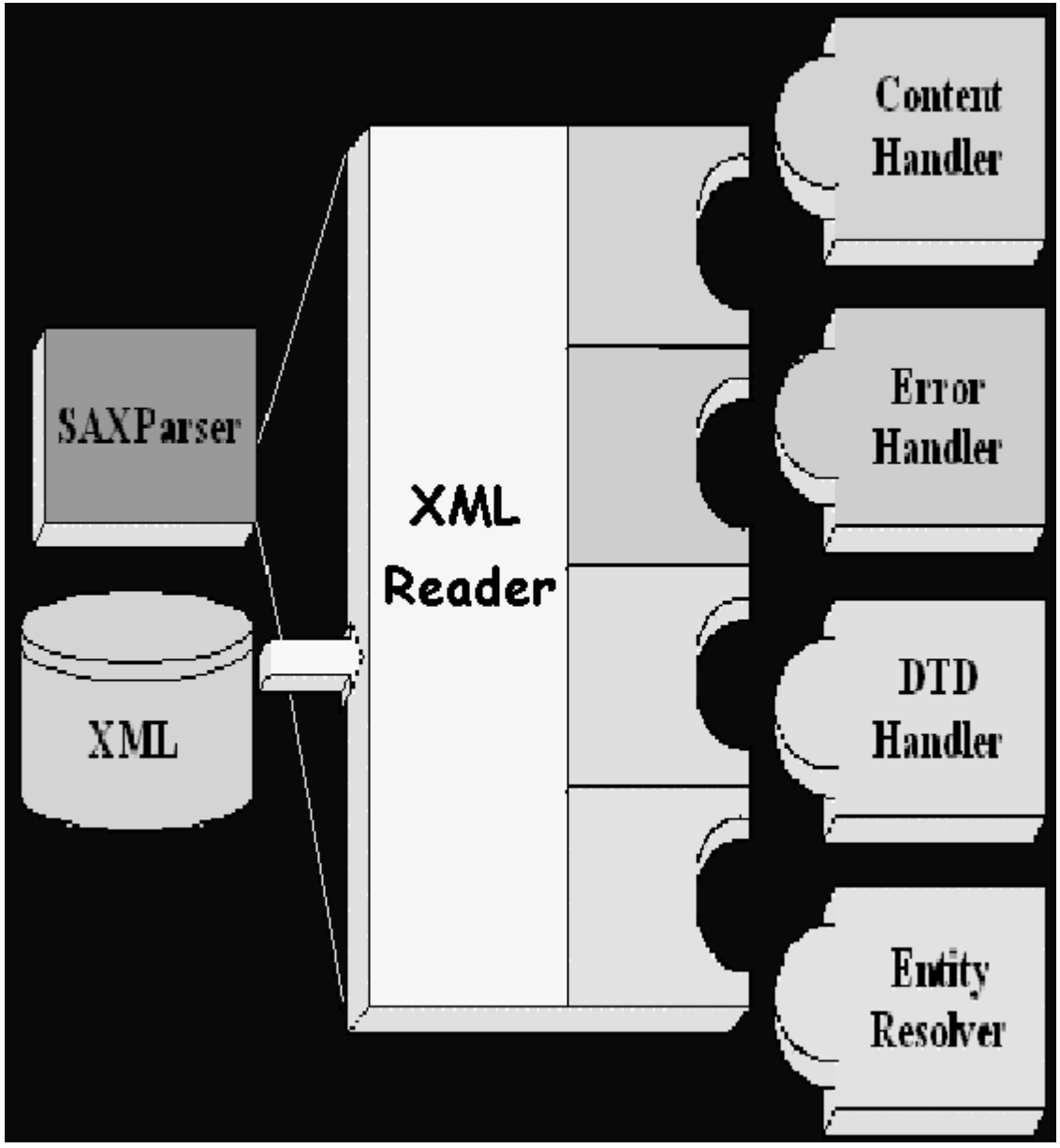
- Über diesen implementiert die Anwendung Handler, um die Events abzuarbeiten
  (z.B. durch setDocumentHandler, setErrorHandler)

- In diesen wiederum können die Callback-Funktionen überschrieben werden um die Ereignisse (Events) entsprechend zu verarbeiten
  (z.B. startDocument, startElement, endElement)

Bsp. einer Callback-Funktion :

```
public void startElement(String name, AttributeList attrs)
  {
    out.print(name);
  }
```

SAX - Funktionsweise



- Parser initialisieren
- Handler implementieren und beim Parser registrieren
- Parser.parse() aufrufen

Event-Stream - Beispiel

XML-Dokument :

```
<?xml version =“1.0“?>
<Name>
  <Nachname>Künast</Nachname>
  <Vorname>Renate</Vorname>
</Name>
```

Events :

- start document
- ignorableWhitespace : /n
- start element : Name
- ignorableWhitespace : /n
- start element : Nachname
  characters : Künast
- end element : Nachname
- ignorableWhitespace : /n
- start element : Vorname
  characters : Renate
- end element : Vorname
- ignorableWhitespace : /n
- end element : Name
- end document

Der SAX Parser

Xerces-Parser von Apache :

Packet : org.apache.xerces.parsers

Klassen : org.apache.xerces.parsers.DOMParser

org.apache.xerces.parsers.SAXParser

org.xml.sax package : (SAX 2.0)

Interfaces :

XMLReader -Lesen des Dokuments mittels Callback-Funktionen

ContentHandler -Logischer Inhalt wird hier hinein geleitet

ErrorHandler -Für Fehlermeldungen zuständig

DTDHandler -Wird bei DTD-bezogenen Ereignissen benachrichtigt

EntityResolver -Löst Entities auf

+ XMLFilter / Locator / Attributes

Interface XMLReader

- **ContentHandler getContentHandler()**
  -liefert den aktuellen ContentHandler zurück
- **void setContentHandler(ContentHandler handler)**
  -registriert einen ContentHandler beim Parser
- **ErrorHandler getErrorHandler()**
  -liefert den aktuellen ErrorHandler zurück
- **void setErrorHandler(ErrorHandler handler)**
  -registriert einen ErrorHandler beim Parser
- **DTDHandler getDTDHandler()**
  -liefert den aktuellen DTDHandler zurück
- **void setDTDHandler(DTDHandler handler)**
  -registriert einen DTDHandler beim Parser
- **EntityResolver getEntityResolver()**
  -liefert den aktuellen EntityResolver zurück
- **void setEntityResolver(EntityResolver resolver)**
  -registriert einen EntityResolver beim Parser

Interface XMLReader

- **boolean getFeature(java.lang.String name)**
  -liefert boolschen Wert zurück ob das entsprechende „Feature“ gesetzt ist

- **void setFeature(java.lang.String name, boolean value)**
  - setzt das entsprechende „Feature“ gemäß dem Parameter value

Beispiel :

```
try {
 (XMLReader)parser).setFeature( "http://xml.org/sax/features/validation", true);
 ((XMLReader)parser).setFeature( "http://xml.org/sax/features/namespaces", true );
      }
catch (SAXException e) ...
```

- **void parse(java.lang.String systemId)**
  -parsen des Dokuments

Interface ContentHandler

- void startDocument()
  -wird am Anfang eines Dokuments aufgerufen
- void endDocument()
  -wird am Ende eines Dokuments aufgerufen
- void characters(char[] ch, int start, int length)
  -wird bei auftretenden Zeichenketten benachrichtigt
- void ignorableWhitespace(char[] ch, int start, int length)
  -wird bei Zeilenumbrüchen und Leerzeichen benachrichtigt
- void skippedEntity(java.lang.String name)
  -wird bei zu überspringenden Entities aufgerufen
- void startElement(java.lang.String namespaceURI,
  java.lang.String localName, java.lang.String qName, Attributes atts)
  -wird bei einem öffnenden Element benachrichtigt
- void endElement(java.lang.String namespaceURI,
  java.lang.String localName, java.lang.String qName)
  -wird bei einem schließenden Element benachrichtigt

Interface ErrorHandler

- void warning(SAXParseException exception)
  -Warnung
- void error(SAXParseException exception)
  -behebbarer Fehler
- void fatalError(SAXParseException exception)
  - nicht behebbarer Fehler

Beispielprogramm

```
import util.Arguments;
import java.io.*;
import sax.helpers.AttributeListImpl;
import org.xml.sax.*;

public class SAXWriter
extends HandlerBase {
    protected PrintWriter out;

    public SAXWriter(boolean canonical) throws UnsupportedEncodingException
    { this(null, false); }

    protected SAXWriter(String encoding, boolean canonical) throws
     UnsupportedEncodingException
    {   if (encoding == null) { encoding = "UTF8"; }
        out = new PrintWriter(new OutputStreamWriter(System.out, encoding));
        this.canonical = false;
    }
```

Beispielprogramm

```
/** Output der SAX Callback-Funktionen. */
   public static void print(String parserName, String uri)
   {
       try {
           HandlerBase handler = new SAXWriter(false);
           Parser parser = ParserFactory.makeParser(parserName);

           parser.setDocumentHandler(handler);
           parser.setErrorHandler(handler);
           parser.parse(uri);
                 }
             catch (Exception e) { e.printStackTrace(System.err); }
   }


   // DocumentHandler Methoden
   public void startDocument()   { out.print("Start Document"); }
   public void endDocument()   { out.print("End document");  out.flush(); }
```

Beispielprogramm

```
public void startElement(String name, AttributeList attrs)
   {
      out.print('<');
      out.print(name);
      if (attrs != null) {
         attrs = sortAttributes(attrs);
         int len = attrs.getLength();
         for (int i = 0; i < len; i++) {
            out.print(' ');
            out.print(attrs.getName(i));
            out.print("=\"");
            out.print(attrs.getValue(i));
            out.print("");
         }        }
      out.print('>');
   }

   public void characters(char ch[], int start, int length)
   { out.print(ch); }
```

Beispielprogramm

```
public void ignorableWhitespace(char ch[], int start, int length)
   { characters(ch, start, length); }

   public void endElement(String name) {
      out.print("</");     out.print(name);     out.print('>'); }


   // ErrorHandler Methoden
   public void warning(SAXParseException ex) {
      System.err.println("[Warning] " + getLocationString(ex)+": " + ex.getMessage()); }

   public void error(SAXParseException ex) {
      System.err.println("[Error] " + getLocationString(ex)+": " + ex.getMessage()); }

   public void fatalError(SAXParseException ex) throws SAXException {
      System.err.println("[Fatal Error] " + getLocationString(ex)+": " + ex.getMessage());
      throw ex; }
```

Beispielprogramm

```
private String getLocationString(SAXParseException ex) {
        StringBuffer str = new StringBuffer();

        String systemId = ex.getSystemId();
        if (systemId != null) {
            int index = systemId.lastIndexOf('/');
            if (index != -1)
                systemId = systemId.substring(index + 1);
            str.append(systemId);
        }
        str.append(':');   str.append(ex.getLineNumber());
        str.append(':');   str.append(ex.getColumnNumber());

        return str.toString();
    }
```

Beispielprogramm

```
/** Liefert sortierte Liste von Attributen zurück. */
   protected AttributeList sortAttributes(AttributeList attrs)
   {
      AttributeListImpl attributes = new AttributeListImpl();
      int len = (attrs != null) ? attrs.getLength() : 0;
      for (int i = 0; i < len; i++) {
         String name = attrs.getName(i);
         int count = attributes.getLength();
         int j = 0;
         while (j < count) {
            if (name.compareTo(attributes.getName(j)) < 0)
                     { break; }
            j++;
         }
         attributes.insertAttributeAt(j, name, attrs.getType(i),
                                 attrs.getValue(i));
      }
      return attributes;
   }
```

Beispielprogramm

```
public static void main(String argv[])
   {
      Arguments argopt = new Arguments();
      if (argv.length == 0)  { System.exit(1); }

      String  parserName = "org.apache.xerces.parsers.SAXParser";

      String arg = null;
      while ( ( arg =  argopt.getlistFiles() ) != null )
            {
         System.err.println(arg+':');
         print(parserName, arg, false);
      }
   } // main(String[])
} // class SAXWriter
```

SAX 1.0 => SAX 2.0

- Neue Namen für Interfaces : XMLReader für Parser , ContentHandler für DocumentHandler ...
- Unterstützung von XML-Namespaces
- Interface XMLFilter um Filter zwischen Anwendung und SAX-Treiber einzusetzen
- Einheitliche Schnittstelle zum Lesen und Schreiben von Properties und Features

Geschichte / Entstehung

- 13.12.1997 : Dav Megginson / Peter Murray Rust / Tim Bray beginnen mit der Entwicklung von SAX, welches auf John Tigues‘s XAPI-J aufbaut
- 11.05.1998 : SAX 1.0 wird veröffentlicht
- 05.05.2000 : SAX 2.0 wird veröffentlicht

Schlußfolgerungen / Zusammenfassung

DOM : ++ einfaches Navigieren und Modifizieren
von Dokumenten
-- Zwischenspeicherung des gesamten Dokuments
oo automatische Erstellung des Strukturbaumes

SAX : ++ keine Zwischenspeicherung nötig
++ schnelle Suchoperationen
-- während des Navigierens und Modifizierens
von Dokumenten müssen diese unter Umständen
mehrmals durchlaufen werden (kann nicht auf die
rückwertigen Daten zugreifen)
oo explizites Programmieren von Event-Methoden

Schlußfolgerungen / Zusammenfassung

- Weder SAX noch DOM ist das „bessere“ Programmier-Interface
- SAX und DOM ergänzen sich
- Wahl je nach Aufgabenspektrum
- Kleine Dokumente / Modifizieren von Dokumenten : DOM
- Große Dokumente, die nur gelesen werden : SAX

Quellen

http://xml.apache.org/
http://www.saxproject.org/
http://sourceforge.net/projects/sax/
http://www.wrox.com

XML Databases (Wrox - Kevin Williams)