Semistrukturierte Daten

Sommersemester 2008

Teil 6: SAX

6.1. Überblick
6.2. ContentHandler
6.3. weitere Event Handler
6.4. XMLReader
6.5. Benutzung von SAX in JAXP
6.6. SAX-Filter
6.7. Epilog

dbai

6.1. Überblick

- Entwicklung von SAX
- Funktionsweise von SAX
- XMLReader

Entwicklung von SAX

- SAX: Simple API for XML
- keine W3C Recommendation, aber ein de-facto Standard: http://www.saxproject.org/
- Entwickelt von XML-Entwicklern ("jeder" kann mitmachen über die Mailing List xml-dev@w3.org)
- Plattform- und Programmiersprachen-unabhängig (auch wenn SAX ursprünglich für Java entwickelt wurde)
- SAX Versionen:
  - SAX 1.0: entstanden auf Initiative von Peter Murray-Rust (mit dem Ziel, mehrere ähnliche aber inkompatible Java XML-APIs zusammenzuführen), im Mai 1998 freigegeben.
  - SAX 2.0: erweitert um Namespace-Unterstützung; führte zu inkompatibler Erweiterung von SAX 1.0

Funktionsweise von SAX

- Applikation registriert callback Funktionen beim SAX-Parser (d.h.: "XMLReader").
- Applikation startet SAX-Parser.
- SAX-Parser durchläuft das XML-Dokument einmal sequentiell.
- Parser erkennt Events (syntaktische Einheiten) beim Analysieren des XML-Dokuments.
- Parser ruft für jedes Event die entsprechende Callback Funktion auf, die die Applikation bereitstellt.
- Diese Callback Funktionen sind auf 4 Event Handler aufgeteilt: ContentHandler, ErrorHandler, DTDHandler, EntityResolver
- Der Speicherbedarf des SAX-Parsers ist konstant!
  (d.h.: unabhängig von der Größe des XML-Dokuments)

XML Reader

- Ist der eigentliche SAX-Parser, d.h.: liest das XML-Dokument und ruft die callback Funktionen auf.
- Erlaubt das Setzen/Auslesen bestimmter Properties/Features:

  `setFeature, setProperty, getFeature, getProperty`
- Benutzer registriert die Event Handler (mit den callback Funktionen):

  `setContentHandler, setDTDHandler,`
  `setEntityResolver, setErrorHandler`
- Analog dazu get-Methoden für die Event Handler:

  `getContentHandler, getDTDHandler,` etc.
- Methode zum Anstoßen des Parsers:

  `parse`

6.2. ContentHandler

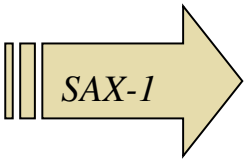
- Überblick
- einige ContentHandler-Methoden im Detail:
  - Dokument-Verarbeitung
  - Element-Verarbeitung
  - Attributes Interface
  - Text-Verarbeitung
  - Locator-Interface
- DefaultHandler

Überblick: Methoden des ContentHandler (1)

```
void startDocument()
void endDocument()
void     startElement(String namespaceURI,
      String localName, String qName, Attributes atts)
void  endElement(String namespaceURI, String localName,

      String qName)
void  characters(char[] ch, int start, int length)
void  setDocumentLocator(Locator locator)
void  processingInstruction(String target, String data)
// target: z.B. "xml-stylesheet".
// data: der unstrukturierte Rest
// d.h.: Pseudo-Attribute werden nicht erkannt wie z.B.:
// <?xml-stylesheet type="text/css" href="order.css"?>
```

Überblick: Methoden des ContentHandler (2)

```
void ignorableWhitespace(char[] ch, int start,
     int length)
// validierender Parser: meldet ignorable whitespace mit
// diesem Event und nicht als "characters"

void skippedEntity(String name)
// falls Parser die (externe) DTD nicht liest, meldet er
// eine Entity Referenz mit diesem Event (anstatt die
// Entity zu expandieren und als "characters" zu melden.

void startPrefixMapping(String prefix, String uri)
void endPrefixMapping(String prefix)
// nur interessant, wenn man auf ein Prefix in einem
// Attribut-Wert zugreift (z.B. bei XML-Schema)
```



Dokument-Verarbeitung

```
void startDocument()
void endDocument()
```

- Ein XMLReader + ContentHandler kann für die Verarbeitung mehrerer Dokumente (hintereinander!) verwendet werden.
  => Initialisierungen am besten in **`startDocument`**
- Wohlgeformtheitsfehler:
  - werden vom Parser erst erkannt, nachdem er schon etliche Events geliefert hat. => dessen muss man sich beim Verarbeiten von Events bewusst sein (d.h.: ev. Rückrollen erforderlich).
  - Allfälliger clean-up code in **`endDocument`** wird ev. nie ausgeführt.
- Reader + ContentHandler sind weder thread-safe noch reantrant.
  => Parallele Verarbeitung von mehreren Dokumenten erfordert mehrere Reader + ContentHandler Objekte.

Element-Verarbeitung

```
void startElement(String namespaceURI,
     String localName, String qName, Attributes atts)
void endElement(String namespaceURI, String localName,
     String qName)
```

- Argumente:
  - namespaceURI: leerer String bei Element ohne Namespace
  - qName = Präfix + ":" + localName
  - atts: siehe nächste Folie
- SAX hat absolut kein "Gedächtnis".

  => Häufig verwendete Datenstruktur in der Applikation:

  Stack für die offenen Elemente:
  - bei **`startElement`**: push Element-Informationen
  - bei **`endElement`**: pop

Attributes Interface

- Die **`startElement`** Funktion liefert die Attribute dieses Elements als Attributes Objekt zurück.
- Zugriff mittels **`getLength()`** und Attribut-Index:

```
String getLocalName(int index)
String getURI(int index)
String getQName(int index)
String getType(int index)
```

- Zugriff mittels Namespace-qualified name:

```
int getIndex(String uri, String localName)
String getValue(String uri, String localName)
String getType(String uri, String localName)
```

- Analog: Zugriff mittels qualified (prefixed) name, z.B.:

```
int getIndex(String qualifiedName)
```

Text-Verarbeitung

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

- Darstellung von Text-Inhalt:
  - als Char-Array mit Start-Index und Längenangabe
  - (insbes. bei langem Text): SAX-Parser darf Text auf beliebig viele hintereinander folgende **`characters`** Events aufteilen.
- Häufige Verarbeitungsart:
  - Text-Inhalt in einem StringBuffer akkumulieren
  - bei startElement: Initialisierung des StringBuffer
  - bei endElement: Verarbeitung des gesamten Text-Inhalts.

Locator Interface

- Das Locator Interface erlaubt Zugriff auf die Stelle im Dokument, wo das letzte Event gefunden wurde.
- SAX-Parser sollte (muss aber nicht) Locator implementieren.
- Methoden, z.B.:

```
int getLineNumber()
int getColumnNumber()
```

- Typischer Code, um Locator-Information verfügbar zu haben:
  - Instanz-Variable für ContentHandler Objekt definieren:

```
private Locator locator;
```

  - Referenz auf Locator abspeichern:

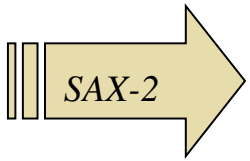
```
public void setDocumentLocator(Locator locator) {
   this.locator = locator;
 }
```

DefaultHandler

- Deklaration des DefaultHandlers (in org.xml.sax.helpers):

```
public class DefaultHandler extends Object
implements EntityResolver, DTDHandler,
    ContentHandler, ErrorHandler
```

- Enthält default-Deklarationen für alle callback Funktionen dieser 4 Event Handler
- Default-Verhalten: "do nothing"-Methoden
- Bequeme Definition eines eigenen SAX-Parsers:
  - von DefaultHandler erben
  - die tatsächlich benötigten callback Funktionen überschreiben



6.3. weitere Event Handler

- EntityResolver
- DTDHandler
- ErrorHandler

EntityResolver

- Einzige Methode:

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

- Idee:
  - Die **`resolveEntity`**-Methode wird vom Parser aufgerufen, wenn eine externe geparste Entity gefunden wurde.
  - externe geparste Entities können mittels SystemId oder PublicId angegeben werden.
  - Mittels **`resolveEntity`** Methode bekommt die Applikation die Möglichkeit (insbes. bei PublicId) dem Parser eine andere InputSource bereitzustellen.

DTDHandler

- Methoden:

```
void notationDecl(String name, String publicId,
 String systemId)
void unparsedEntityDecl(String name, String publicId,
 String systemId, String notationName)
```

- Idee:
  - Während der Bearbeitung der DTD meldet der Parser die Deklarationen von Notations und unparsed Entities.
  - Die Applikation speichert sich diese Informationen in eigenen Datenstrukturen (z.B. in Hash Tabelle)
  - Wenn der Parser Attribute vom Typ "NOTATION", "ENTITY" oder "ENTITIES" meldet, hat die Applikation die nötigen Informationen.

ErrorHandler

- Methoden:

```
void fatalError(SAXParseException exception)
// non-recoverable error
void error(SAXParseException exception)
void warning(SAXParseException exception)
```

- Idee:
  - Bei Wohlgeformtheitsfehler wirft der Parser eine Exception und beendet den Parse-Vorgang.
  - Bei anderen Fehlern (insbes. Gültigkeitsfehler bei validierendem Parser) kann der Parser fortsetzen und wirft keine Exception.
  - Benachrichtigung der Applikation: Parser reicht SAXParseException an die entsprechende Methode des ErrorHandlers.

6.4. XMLReader

- Reader-Implementierung
- Features und Properties
- SAX-Ausgabe

Reader-Implementierung

- XMLReader Instanz mittels XMLReader Factory erzeugen
- Auswahl einer bestimmten Implementierung:
  - Default-Implementierung auswählen:

```
public static XMLReader createXMLReader()
    throws SAXException;
// Auswahl des SAX-Parsers laut system property
// org.xml.sax.driver (kann mit Kommandozeilen-
// parameter "-D" gesetzt werden)
```

  - Auswahl einer bestimmten Implementierung:

```
public static XMLReader createXMLReader(
    String className) throws SAXException;
// className = gewünschte Implementierung,
// z.B.: "org.apache.xerces.parsers.SAXParser"
```

Features und Properties

XMLReader-Methoden:

- `boolean getFeature(String name) throws SAXNotRecognizedException, SAXNotSupportedException`
- `Object getProperty(String name) throws SAXNotRecognizedException, SAXNotSupportedException`
- `void setFeature(String name, boolean value) throws SAXNotRecognizedException, SAXNotSupportedException`
- `void setProperty(String name, Object value) throws SAXNotRecognizedException, SAXNotSupportedException`

Features

- Haben einen boolean Wert: true / false
- Feature-Namen sind absolute URLs
- Standard-Features (z.B. validierend, Namespace-aware, ...)

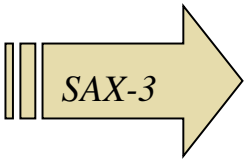
```
parser.setFeature(
 "http://xml.org/sax/features/validation", true);
parser.setFeature(
 "http://xml.org/sax/features/namespaces", true);
```

- Vendor-spezifische Features, z.B.:

```
boolean schemaValidierend = parser.getFeature(
 "http://apache.org/xml/features/validation/schema");
boolean schemaFullChecking = parser.setFeature(
 "http://apache.org/xml/features/validation/schema-full-
 checking");
```

Properties

- Haben einen "beliebigen" Typ, z.B.:
  - Property **"http://xml.org/sax/properties/lexical-handler"**
    (ermöglicht Zugriff auf Kommentare, CDATA Sections, …)
    => das konkrete handler-Objekt wird als Property gesetzt/gelesen.
- Property-Namen sind absolute URLs
- Weitere Standard-Properties, z.B.:
  - Property **"http://xml.org/sax/properties/xml-string"**
    read-only Property: liefert den Text-String, der das aktuelle SAX-Event ausgelöst hat.
- Vendor-spezifische Properties, z.B.:
  - Property **"http://apache.org/xml/properties/schema/external-schemaLocation"**: gibt an, wo der Parser nach XML-Schema Dateien suchen soll.



SAX-Ausgabe (1)

- Output des XMLReaders als XML-Dokument

  mittels "transformer", analog zur DOM-Ausgabe, d.h.:
  - Erzeuge mittels XMLReader eine SAXSource
  - Aufruf des transformers mit dieser SAXSource als Input

- Importe
  - wie bei DOM:

```
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamResult;
```

  - SAX-spezifische Importe:

```
import javax.xml.transform.sax.SAXSource;
import org.xml.sax.InputSource;
```

SAX-Ausgabe (2)

- Transformer-Instanzierung (wie bei DOM):

```
TransformerFactory tFactory =
 TransformerFactory.newInstance();
Transformer transformer = tFactory.newTransformer();
```

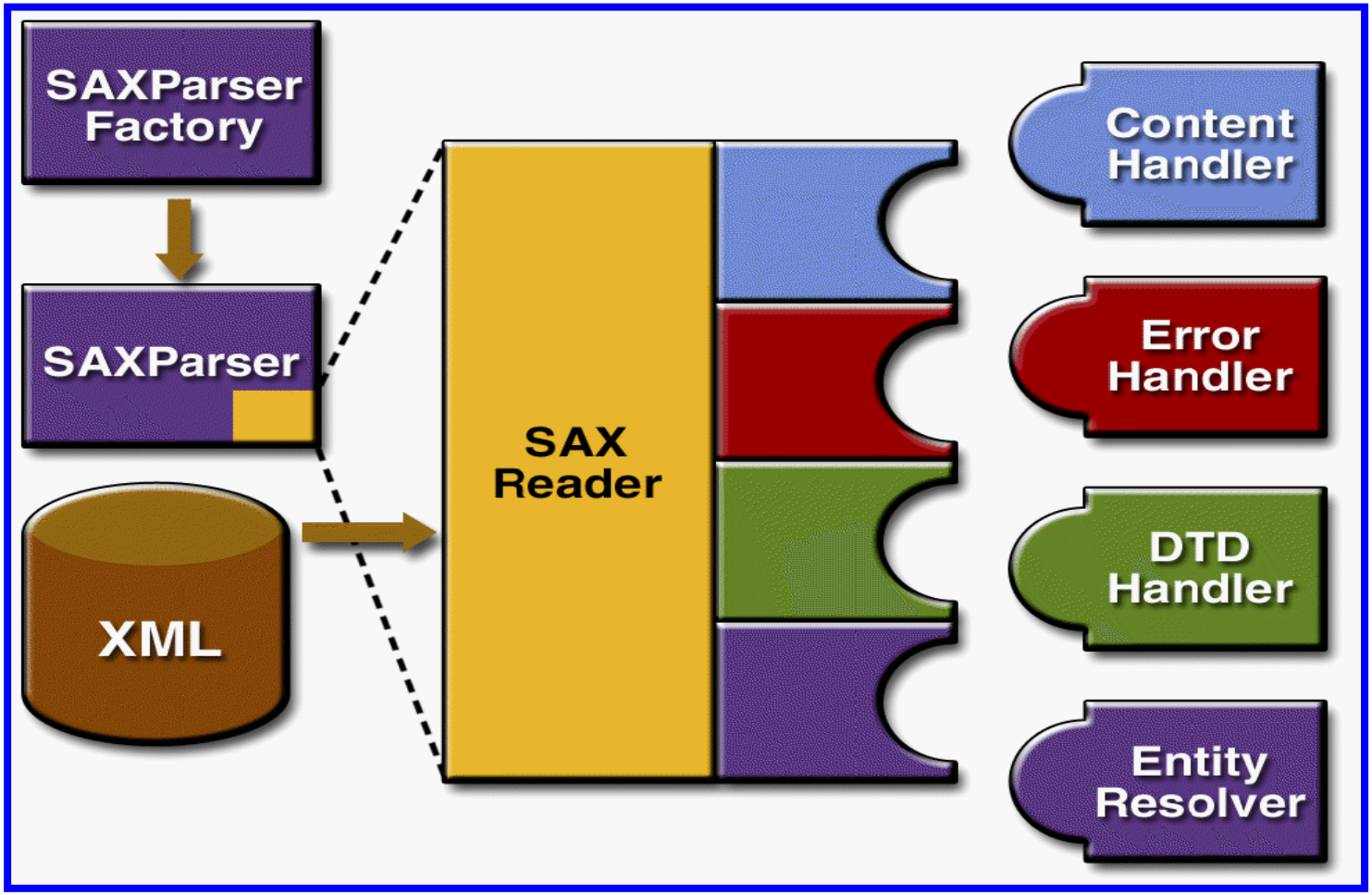
- Ausgabe (d.h.: erzeuge SAXSource mittels XMLReader)

```
SAXSource source =
 new SAXSource(reader, new InputSource(quelle));
StreamResult result = new StreamResult(new File(ziel));
transformer.transform(source, result);
```

6.5. Benutzung von SAX in JAXP

- Überblick
- SAXParser Interface
- Features und Properties

Überblick



SAXParser-Interface (1)

- Das SAXParser-Interface in JAXP ist einfach ein Wrapper um den XMLReader.
- Erzeugung mittels statischer Methode `newSAXParser()` der Klasse `SAXParserFactory`.
- Mit `getXMLReader()` hat man Zugriff auf den XMLReader.
- Registrierung der Event Handler:
  - Keine eigenen Set-Methoden erforderlich
  - Statt dessen: DefaultHandler als Parameter beim Aufruf der parse()-Methode des SAXParsers

SAXParser-Interface (2)

- Importe: SAXParser/SAXParserFactory

```
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParser;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
```

- Factory-Instanzierung:

```
SAXParserFactory factory = SAXParserFactory.newInstance();
```

- Parser-Instanzierung und Parsen:

```
DefaultHandler handler = new MyDefaultHandler();
try{
   SAXParser saxParser = factory.newSAXParser();
   saxParser.parse(new File(args[0]), handler);
} catch (Exception e)…
```

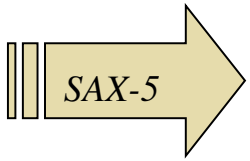
Features und Properties

- Einstellungen des Parsers (z.B. validierend: ja/nein)

```
SAXParserFactory factory =
    SAXParserFactory.newInstance();
factory.setValidating(true);
factory.setNamespaceAware(true);
```

- Methoden zum Zugriff auf Properties und Features:

```
Object getProperty(String name) throws … ;
void setProperty(String name, Object value) throws … ;
boolean isNamespaceAware();
boolean isValidating();
```



6.6. SAX-Filter

- Funktionsweise eines SAX-Filters
- Verwendung eines SAX-Filters

Funktionsweise eines SAX-Filters

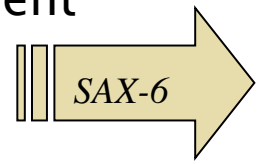
- Filter beschränken/manipulieren auftretende Events
- Filter können verschachtelt verwendet werden
- Falls sich Dokumentstruktur ändert, kann eine Adaption des Filters ausreichend sein (also: ohne die Applikation zu ändern).
- Bemerkung: Filter kann auch für nicht XML-Input verwendet werden (ohne dass die Client-Applikation das merkt...)

Funktionsweise eines SAX-Filters (2)

- Vorbereitung:
  - Applikation teilt dem Filter mit, auf welchen Reader er horchen muss
  - Applikation registriert ihre Event Handler beim Filter
- Start des Parse-Vorgangs:
  - Applikation ruft parse()-Methode des Filters auf
  - Filter ruft parse()-Methode des Readers auf
- Parse-Vorgang:
  - Reader erzeugt Events => ruft callback Funktionen *des Filters* auf
  - Filter ruft innerhalb seiner callback Funktionen die callback Funktionen der Applikation auf.
- Der Filter ist also gleichzeitig Event Handler und Reader.

Verwendung eines SAX-Filters

- XMLFilter Interface: erweitert XMLReader um 2 Methoden:
  - `void setParent(XMLReader parent)` und `XMLReader getParent()`
  - "parent" = Reader, auf den der Filter horchen muss
  - Mittels **`setContentHandler`**, etc. werden die Event Handler der Applikation beim Filter registriert.
- Implementierung des XMLFilter Interface:
  - "per Hand": ziemlich aufwändig (XMLReader hat 14 Methoden)
  - Eleganterer Weg: mittels XSLT Stylesheet kann ein XMLFilter automatisch erzeugt werden (späterer VL-Termin).
  - Die Klasse XMLFilterImpl in org.xml.sax.helpers stellt einen Default-Filter bereit, der die Requests in beiden Richtungen transparent durchreicht.



6.7. Epilog

- DOM vs. SAX
- Literatur

DOM vs. SAX

- DOM:
  - Baut gesamten XML-Baum im Speicher auf => wahlfreier Zugriff
  - Manipulation des Baums möglich
  - Hoher Speicherbedarf, langsamer
- SAX:
  - XML-Dokument wird einmal durchlaufen => sequentieller Zugriff
  - "streaming" möglich (d.h.: Bearbeiten und Weiterreichen, bevor das ganze Dokument übertragen ist).
  - Geringerer Speicherbedarf, höhere Geschwindigkeit
  - Falls mehrmaliger Zugriff auf Knoten erforderlich: Applikation ist selbst für das Puffern verantwortlich.
  - Low level (DOM-API benutzt SAX-API)

Literatur

- Spezifikationen:
  - http://www.w3.org/DOM/
  - http://www.saxproject.org/
- (Online) Bücher und Artikel:
  - Elliotte Rusty Harold: "Processing XML with Java"
    http://www.cafeconleche.org/books/xmljava/
  - J2EE Tutorial (Kap. 4-7):
    http://java.sun.com/j2ee/1.4/docs/tutorial/doc/