

# XML-Frameworks in verschiedenen Programmiersprachen

Proseminar Textkodierung und Auszeichnung

Matthias Bethke [bethke@linguistik.uni-erlangen.de](mailto:bethke@linguistik.uni-erlangen.de)

Linguistische Informatik  
Universität Erlangen-Nürnberg

Sommersemester 2006

- 1 XML-Frameworks: Allgemeines
  - DOM
  - SAX
  - Beispieldokument
- 2 DOM
- 3 SAX

# Allgemeines: Document Object Model (DOM)

- Vom W3C standardisiertes, sprachneutrales Interface zum Zugriff auf Struktur und Inhalt eines Dokuments
  - Ursprünglich für HTML entwickelt (Marketing-Blah für \*script + DOM: *dynamic HTML*); vom W3C auch für →MathML und →SVG
  - Darstellung des Dokuments als *Baumstruktur von Objekten*
  - Folgerung: ganzes Dokument muss geparkt im Speicher vorgehalten werden
- Speicherintensiv, aber problemlos bei wahlfreiem Zugriff
  - Vorteilhaft für Fälle, wo hauptsächlich nichtlinear auf Dokumenteninhalte zugegriffen wird.

# Allgemeines: Simple API for XML (SAX)

- Datenstrom-orientierter Parser, speziell für XML; funktioniert prinzipiell aber auch für HTML
  - Dokument wird sequenziell gelesen; kein „Zurückspulen“ ohne Neuparsen
  - Reaktion auf bestimmte Strukturen mittels *Callbacks* (benutzerdefinierte Funktionen bzw. Methoden, die der Parser aufruft)
- Sehr geringer Speicherbedarf
  - Ungeeignet für wahlfreien Zugriff auf Elemente
  - Komplexere Transformationen mittels *State-Machines*, wird aber schnell unübersichtlich (CL-Analogie: Grammatikparser!)

# Beispieldokument für die Programme (1)

- XML-Datenbank aus dem „Madman“ Music Manager
- Listet Musikstücke mit Titel, Autor, Länge etc. auf
- Gekürzte Datei und Beispielprogramme unter `/projects/Textkodierung_SS06/DOM-SAX/`
- Aufgabe: Parsen des Dokuments und Ausgeben der Titel sämtlicher Musikstücke

# Beispieldokument für die Programme (2)

## Ausschnitt

```
<madmandb>
  <songs>
    <song tracknumber="04/09" title="Robot_Teaser" partialplaycount="1
      " lastmodified="1125345282" fullplaycount="0" artist="Suria"
      lastplayed="1142975853" existssince="1116629394" performer=""
      album="Logical_Evolution" duration="337" genre="
      Psychedelic" year="2004" uniqueid="6676" filesize="10800061"
      rating="2" filename="base64:L211ZG1h...=" />
  </songs>
  <song_set_node name="Root_Playlist" >
    <song_set criterion="~rating(>=4)" >
      <rendering>
        <song unique_id="8448" />
      </rendering>
    </song_set>
  </song_set_node>
</madmandb>
```

# Document Object Model: Perl

## DOM/Perl (1)

```
use XML::DOM;  # DOM-Parser-Modul einbinden

my $parser = new XML::DOM::Parser;
my $doc = $parser->parsefile("file.xml");

# Sammlung aller song-Tags als Objekt (case-sensitive!)
my $nodes = $doc->getElementsByTagName("song");

# Wie viele Elemente sind das?
my $n = $nodes->getLength;
```

# Document Object Model: Perl

## DOM/Perl (2)

```
# Alle TITLE-Attribute der SONG-Tags ausgeben
for(my $i = 0; $i < $n; $i++) {
    my $node = $nodes->item($i);
    my $title = $node->getAttributeNode("title") or next;
    print $title ->getValue . "\n";
}

# Aufräumen (wichtig wg. zirkulärer Referenzen!)
$doc->dispose;
```

# Document Object Model: Java

## DOM/Java

```
private static void parseDoc(String file )
{
    DocumentBuilderFactory fab = DocumentBuilderFactory.newInstance();
    try {
        Document doc = fab.newDocumentBuilder().parse(file);
        NodeList songs = doc.getElementsByTagName("song");
        int n = songs.getLength();
        System.out.println ("Found " + n + " songs: \n");
        for(int i=0; i<n; i++) {
            Node item = songs.item(i);
            Node title = item.getAttributes ().getNamedItem("title");
            if (null == title) continue;
            System.out.println ( title .getNodeValue());
        }
    } catch(Exception e) { e.printStackTrace (); System.exit (1); }
}
```

# Simple API for XML: Perl

## SAX/Perl (1)

```
use XML::SAX; # Parser-Modul einbinden
use IO::File; # Parser braucht objektorientierte E/A

my $parser = XML::SAX::ParserFactory->parser(
    Handler => MadmanXMLHandler->new
);
$parser->parse_file(IO::File->new("madman.xml"));

package MadmanXMLHandler; # Meine eigene Handler-Klasse
use base qw(XML::SAX::Base); # Oberklasse definieren

sub new {
    # Standard-Konstruktor
    my $class = shift;
    my $self = { __PACKAGE__."Nsongs" => 0 };
    return bless $self, $class;
}
```

# Simple API for XML: Perl

## SAX/Perl (2)

*# Handler-Methoden für jedes zu verarbeitende Ereignis*

```
sub start_element {  
    my ($self, $el) = @_;  
  
    return unless($el->{LocalName} eq "song");  
    ++$self->{__PACKAGE__."Nsongs"};  
    my $title = $el->{Attributes}->{"{}title"};  
    print $title ->{Value},"\n" if($title);  
}  
  
sub end_document {  
    my ($self) = @_;  
    print "\nTotal: ",$self->{__PACKAGE__."Nsongs"}," songs\n";  
}
```

# Simple API for XML: Java

## SAX/Java (1)

```
import javax.xml.parsers .*;
import org.xml.sax .*;
import org.xml.sax.helpers .*;

private static void parseDoc(String file ) {
    SAXParserFactory fab = SAXParserFactory.newInstance();
    try {
        fab.newSAXParser().parse( file , new MyHandler());
    }
    catch(Exception e) { e.printStackTrace (); System.exit (1); }
}
```

# Simple API for XML: Java

## SAX/Java (2)

```
static private class MyHandler extends DefaultHandler {
    int nSongs = 0;
    public void endDocument() {
        System.out. println ( "\nTotal: " + nSongs + " songs");
    }

    public void startElement( String uri , String localName,
        String qName, Attributes attributes ) {
        if (!qName.equals("song")) return;
        ++nSongs;
        String title = attributes .getValue("title");
        if ( null != title )
            System.out. println ( title );
    }
}
```