11 XML Programmierung

- Programmatische Interfaces (API) zur Manipulation von XML (DOM und JDOM)
- Data Binding: Abbildung von DTD/Schema auf isomorphe Datenstrukturen
- Strombasierte Verarbeitung
- Integration von XML in Programmiersprachen

DSLs für XML

- DSL = domain specific language
- XPath: Spezialsprache zur Navigation in XML Dokumenten
- XSLT: Spezialsprache zur Transformation von XML Dokumenten
- (XQuery: datenbankmäßige Verarbeitung von XML)
- Helfen nur bedingt bei
  - Anwendungen, die XML-Daten lesen und schreiben müssen
  - Anwendungen, die XML-Daten visuell darstellen und editieren
  - Implementierung von XSLT, XQuery, etc

Ansätze zur XML Verarbeitung

- Strings lesen und schreiben
  nicht zufriedenstellend
  komplizierte Aufgabe:
  - Unicode, verschiedene Zeichenkodierungen
  - Wohlgeformtheit
  - Gültigkeit
- Spezialisierte APIs fürs
  - Parsen von XML $\rightarrow$ Repräsentation
  - Navigieren durch XML Repräsentation
  - Manipulieren von XML Repräsentation
  - Serialisierung in XML Text

11.1 Die DOM API

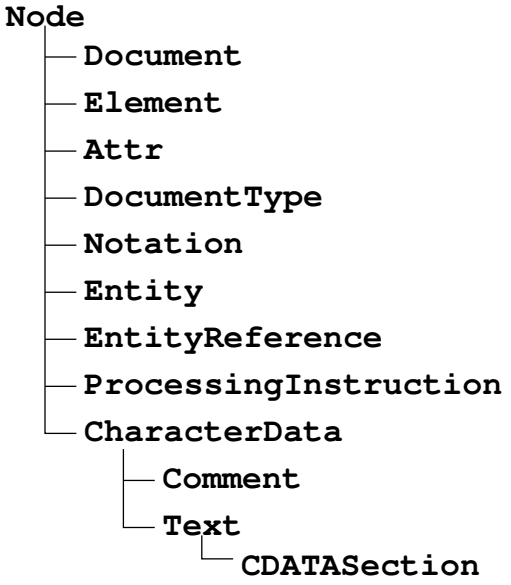
- DOM = Document Object Model
- W3C Empfehlung "DOM Level 3" `http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/`
- Selbstdarstellung:

  > ... a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of [XML] documents.

- Sprachunabhängigkeit durch Verwendung von IDL (OMG)
- Implementierungen für viele Sprachen: Java, JavaScript, Python, Perl, C#, Fortran, Ada, ...
- Implementiert in vielen Anwendungen (Webbrowser, OpenOffice, XMetaL, ...)

11.1.1 Das DOM Datenmodell

- Low-level Graphdarstellung für XML Dokumente
- Knotentypen (Interfaces, Ausschnitt)



Invarianten des Datenmodells

- Wichtigste Invariante: Graph muss immer eine Menge von Bäumen sein (d.h., höchstens ein Vaterknoten und keine Zyklen)
- Jeder Knoten gehört zu einem `Document` Knoten
- Nicht jeder Knoten darf als Kind eines anderen Knoten auftreten
  - `Document` darf nie als Kind auftreten (muss Wurzel sein)
  - `DocumentType` (höchstens einmal) nur Kind von `Document`
  - `Element`: `Document` (höchstens einmal), `Element`, `Entity`, `EntityReference`
  - `Attr` gilt nicht als Kind des zugehörigen `Element`
  - `Text`: `Element`, `Attr`, `Entity`, `EntityReference`
  - ...
- Invarianten werden nur informell in der API angegeben

11.1.2 XML Verarbeitung mit DOM

- Java Binding

```
Document dom = ...;
Element result = dom.createElement("span");
Attr at = dom.createAttribute("id");
at.nodeValue = "draw" + nr;
result.setAttributeNode (at);
```

Weitere Aspekte

- Navigation: über `Node`-Attribute `parentNode`, `childNodes`, `previousSibling`, `nextSibling`, `attributes`
- Manipulation: über `Node`-Methoden `insertBefore()`, `replaceChild()`, `appendChild()`, `removeChild()`
- Parsen und Serialisieren (Load and Save)
- Validieren gegen DTD und XML Schema
- Auswertung von XPath (1.0)
- Insgesamt etwa 200 Methoden

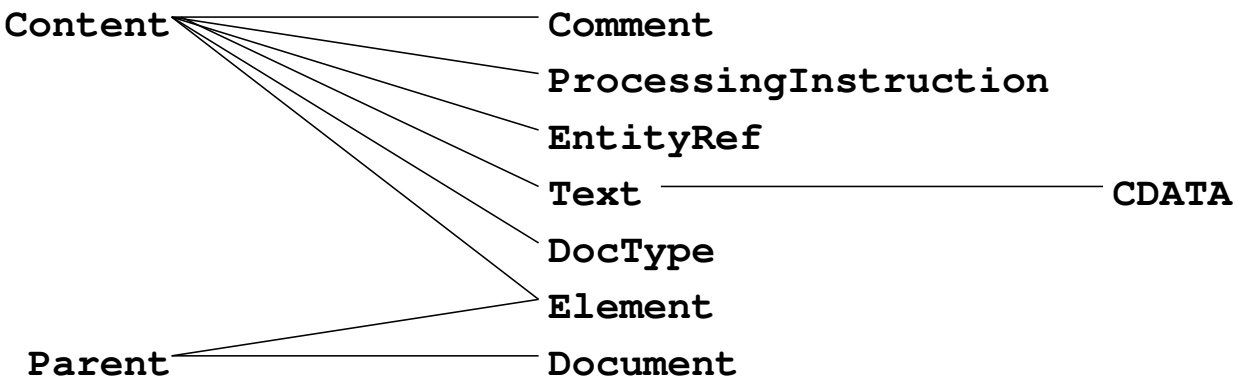
11.2 JDOM

- DOM oft unelegant, da
  - sprachunabhängig
  - universell einsetzbar
  - Erfahrung erforderlich
- JDOM
  - spezialisiert für Java
  - 80/20 Prinzip:
    - einfache und häufige Aufgaben unterstützt

11.2.1 JDOM Eigenschaften

- Kollektionen von Elementen und Attributen dargestellt durch `java.util.List` mit `Iterator`
- Datenmodell besteht aus (abstrakten) Klassen und Interfaces (Java)
- Implementierung kann für Java optimiert werden
- API einfacher, da die Indirektion über IDL wegfällt

11.2.2 JDOM Datenmodell



- `abstract class Content`, `interface Parent`
- ähnliche Operationen wie DOM

```
class Element {
  Element addContent(Content child);
}
```

- Bauminvariante wird zur Laufzeit geprüft

Beispiel: Höhe eines XML Baums

```
import java.util.*;
import org.jdom.*;
public class XmlHeight {
    int xmlHeight(Element e) {
        Iterator i = e.getContent.iterator();
        int max = 0;
        while (i.hasNext()) {
            Object c = i.next();
            int h = (c instanceof Element
                    ? xmlHeight((Element)c) : 1);
            if (h>max) max = h;
        }
        return max+1;
    }
}
```

Beispiel: Modifikation eines Dokuments

```
import org.jdom.filter.*;

  static void doubleSugar(Document d)
    throws DataConversionException {
    Namespace rcp = Namespace.getNamespace("http://www.brics.dk/ixwt/re
    Filter f = new ElementFilter("ingredient", rcp);
    Iterator i = d.getDescendants(f);
    while (i.hasNext()) {
      Element e = (Element)i.next();
      if (e.getAttributeValue("name").equals("sugar")) {
        double amount = e.getAttribute("amount").getDoubleValue();
        e.setAttribute("amount", new Double(2*amount).toString());
      }
    }
  }
```

11.2.3 Parsen, Validieren, Serialisieren

- JDOM enthält keinen Parser
- `import org.jdom.input.*;`
  Kann Dokumente aufbauen aus
  - externem SAX-Parser (Voreinstellung: JAXP) oder
  - existierendem DOM Dokument
- Validierung nur beim Parsen, nicht im Speicher
- `import org.jdom.output.*;`
  Ausgabe als
  - DOM Dokument
  - SAX-Events
  - XML-Strom

Beispiel: XML Lesen, Ändern, Schreiben

```
static void doit(String[] args)
    throws Exception {
  SAXBuilder b = new SAXBuilder();
  Document d = b.build(new File("recipes.xml"));
  Namespace rcp =
    Namespace.getNamespace("http://www.brics.dk/ixwt/recipes");
  d.getRootElement()
   .getChild("description", rcp)
   .setText("Cool␣recipes!");
  XMLOutputter outputter = new XMLOutputter();
  outputter.output(d, System.out);
}
```

Beispiel: . . . mit Validierung

```
SAXBuilder b = new SAXBuilder();
b.setValidation(true);
String msg = "No errors!";
try {
  Document d = b.build(new File(args[0]));
} catch (JDOMParseException e) {
  msg = e.getMessage();
}
```

11.2.4 JDOM mit XPath und XSLT

- Hooks für
  - XPath 1.0 Auswertung
    - Package `org.jdom.xpath`
    - Implementierung kann zur Laufzeit gesetzt werden
  - XSLT Transformation
    - Package `org.jdom.transform`
    - Implementierung kann zur Laufzeit gesetzt

Beispiel: JDOM mit XPath

```
import org.jdom.xpath.*;

  static void doubleSugar(Document d)
    throws JDOMException {
    XPath p = XPath.newInstance ("//rcp:ingredient[@name='sugar']");
    p.addNamespace("rcp", "http://www.brics.dk/ixwt/recipes");
    Iterator i = p.selectNodes(d).iterator();
    while (i.hasNext()) {
      Element e = (Element)i.next();
      if (e.getAttributeValue("name").equals("sugar")) {
        double amount = e.getAttribute("amount").getDoubleValue();
        e.setAttribute("amount", new Double(2*amount).toString());
      }
    }
  }
```

Beispiel: JDOM mit XSLT/Xalan

```
import org.jdom.transform.*;

static void applyXSLT(String file, String sheet)
    throws Exception {
  System.setProperty("javax.xml.transform.TransformerFactory",
          "org.apache.xalan.processor.TransformerFactoryImpl");
  SAXBuilder b = new SAXBuilder();
  Document d = b.build(new File(file));
  XSLTransformer t = new XSLTransformer(sheet);
  Document h = t.transform(d);
  XMLOutputter outputter = new XMLOutputter();
  outputter.output(h, System.out);
}
```

11.3 XML Data Binding

Typisches Anwendungsszenario

- Eingabedaten liegen in XML vor
- Ausgabedaten werden in XML erwartet
- Interne Verarbeitung mit herkömmlichen Objekten
  - spezialisiertes Interface einfacher/effizienter als allgemeines DOM/JDOM Interface
  - interne Verarbeitung und Klassenstruktur bereits vorhanden

Beispiel: Visitenkarten

```
class Card {
  public String name, title, email, phone, logo;
  public Card(String name, String title,
              String email, String phone,
              String logo) {
    this.name = name;
    this.title = title;
    this.email = email;
    this.phone = phone;
    this.logo = logo;
  }
}
```

Einlesen einer Liste von Visitenkarten (JDOM)

```
Vector doc2vector(Document d) {
  Vector v = new Vector();
  Iterator i = d.getRootElement().getChildren("card", b).iterator();
  while (i.hasNext()) {
    Element e = (Element)i.next();
    String phone = e.getChildText("phone", b);
    if (phone==null) phone = "";
    Element logo = e.getChild("logo", b);
    String uri = (logo==null) ? "" : logo.getAttributeValue("uri");
    Card c = new Card(e.getChildText("name", b),
                      e.getChildText("title", b),
                      e.getChildText("email", b),
                      phone, uri);
    v.add(c);
  }
  return v;
}
```

11.3.1 Binding Compiler

- Übersetzt Dokumentenspezifikation in (Java) Klassen
- Generiert getypte Methoden zum Parsen, Serialisieren, Zugriff, Modifikation der Dokumente
- Eliminiert dadurch Fehlerquellen
- Unterschiede zwischen Binding Compilern
  - Dokumentenspezifikation: DTD, XML Schema (oft nur teilweise), andere Schemasprachen
  - Klassenstruktur: festgelegt oder anpassbar
  - Verhalten bei Roundtripping: Informationsverlust bei Parsen gefolgt von Serialisieren?
  - Valierung: beim Parsen, im Speicher, nach Modifikation
  - Implementierung: ad-hoc oder DOM/JDOM basiert?

11.3.2 Beispiel: JAXB

- JAXB = Java Architecture for XML Binding
- Data binding framework
- Dokumentenspezifikation: XML Schema
- Klassenstruktur: anpassbar durch Annotationen im Schema
- Roundtripping: bis auf CDATA
- Validierung: beim Parsen und im Speicher (ganzes Dokument)
- Implementierung: nur Interfaces vorgegeben, konkrete Repräsentation kann geändert werden

11.3.3 Beispiel: Visitenkarten mit JAXB

- XML Schema Spezifikation für Namespace
  `http://businesscard.org`
- Generierte Interfaces und Klassen (Implementierungen)

XML Schema für Visitenkarten

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:b="http://businesscard.org"
        targetNamespace="http://businesscard.org"
        elementFormDefault="qualified">
  <element name="cardlist" type="b:cardlist_type"/>
  <element name="card" type="b:card_type"/>
  <element name="name" type="string"/>
  <element name="email" type="string"/>
  <element name="phone" type="string"/>
  <element name="logo" type="b:logo_type"/>

  <attribute name="uri" type="anyURI"/>
  <!-- Typdefinitionen -->
</schema>
```

XML Schema für Visitenkarten, II

```
<complexType name="cardlist_type">
  <sequence>
    <element name="title" type="b:cardlist_title_type"
             minOccurs="0"/>
    <element ref="b:card"
             minOccurs="0" maxOccurs="unbounded"/>
  </sequence></complexType>

<complexType name="cardlist_title_type" mixed="true">
  <sequence>
    <any namespace="http://www.w3.org/1999/xhtml"
         minOccurs="0" maxOccurs="unbounded"
         processContents="lax"/>
  </sequence></complexType>
```

XML Schema für Visitenkarten, III

```
<complexType name="card_type">
  <sequence>
    <element ref="b:name"/>
    <element name="title" type="string"/>
    <element ref="b:email"/>
    <element ref="b:phone" minOccurs="0"/>
    <element ref="b:logo" minOccurs="0"/>
  </sequence>
</complexType>
<complexType name="logo_type">
  <attribute ref="b:uri" use="required"/>
</complexType>
```

Ausgabe von JAXB

Package `org.businesscard`

- *`Cardlist`*, *`CardlistType`*, `CardlistImpl`, `CardlistTypeImpl`;
- *`CardlistTitle`*, *`CardlistTitleType`*, `CardlistTitleImpl`, `CardlistTitleTypeImpl`;
- *`Card`*, *`CardType`*, `CardImpl`, `CardTypeImpl`;
- *`Name`*, `NameImpl`;
- *`Email`*, `EmailImpl`;
- *`Phone`*, `PhoneImpl`;
- *`Logo`*, *`LogoType`*, `LogoImpl`, `LogoTypeImpl`; and
- `ObjectFactory`.

Das *CardType* Interface

```
public interface CardType {
  java.lang.String getEmail();
  void setEmail(java.lang.String value);

  org.businesscard.LogoType getLogo();
  void setLogo(org.businesscard.LogoType value);

  java.lang.String getTitle();
  void setTitle(java.lang.String value);

  java.lang.String getName();
  void setName(java.lang.String value);

  java.lang.String getPhone();
  void setPhone(java.lang.String value);
}
```

11.4 XML Streaming mit SAX

- DOM, JDOM, JAXB laden vor der Verarbeitung das gesamte Dokument in den Speicher
- Verarbeitung von großen Dokumenten (> 500MB) nicht möglich
- Lösung: *Streaming*, d.h.
  - Verarbeitung des Dokuments während des Lesens
  - ohne dass vollständige Dokumentenrepräsenation erstellt wird
- SAX
  - Framework fürs Streaming von XML Dokumenten
  - (nicht vom W3C geschaffen)

11.4.1 Grundidee von SAX

- Abstraktion von der Zeichenebene
- XML Dokument = Strom von XML-Ereignissen (*events*)
  - Start des Dokuments
  - Start eines Elements
  - Ende eines Elements
  - Namespace-Deklaration gelesen
  - Leerzeichen gelesen
  - Textdaten gelesen
  - Ende des Dokuments

11.4.2 Ereignisbasierte API

- SAX Implementierung erzeugt XML-Ereignisse
- API spezifiziert Callback-Methode für jedes Ereignis
- Programmierer liefert Menge von Callback-Methoden zur Implementierung der Anwendung
- Klasse `DefaultHandler` liefert triviale Callback-Methoden, die durch Subklassen überschrieben werden können

11.4.3 Beispiel: Verwendung der SAX API

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class Trace extends DefaultHandler {
  void printIndent() { ... }
  // Weitere Methoden ...
  public static void main(String[] args) {
      Trace tracer = new Trace();
      XMLReader reader = XMLReaderFactory.createXMLReader();
      reader.setContentHandler(tracer);
      reader.parse(args[0]);
  }
}
```

Beispiel: Verwendung der SAX API, II

```
public void startDocument() {
  System.out.println("start␣document"); indent++;
}
public void endDocument() {
  System.out.println("end␣document"); indent--;
}
public void startElement(String uri, String localName,
                         String qName, Attributes atts) {
  printIndent();
  System.out.println("start␣element:␣"+qName); indent++;
}
public void endElement(String uri, String localName, String qName) {
  indent--; printIdent();
  System.out.println("end␣element:␣"+qName);
}
```

Beispiel: Verwendung der SAX API, III

```
public void ignorableWhitespace(
    char[] ch, int start, int length) {
  printIdent();
  System.out.println("whitespace, length "+length);
}
public void processingInstruction(
    String target, String data) {
  printIdent();
  System.out.println("processing instruction: "+target);
}
public void characters(
    char[] ch, int start, int length) {
  printIdent();
  System.out.println("character data, length "+length);
}
```

11.4.4 Beispiel: Höhe eines XML Dokuments

```
public class Height extends DefaultHandler {
  int h = -1;
  int max = 0;
  public void startElement(String uri, String localName,
                           String qName, Attributes atts) {
    h++; if (h>max) max = h;
  }
  public void endElement(String uri, String localName,
                         String qName) {
    h--;
  }
  public void characters(char[] ch, int start, int length){
    if (h+1>max) max = h+1;
  }
```

Beispiel: Höhe eines XML Dokuments, II

```
  public static void main(String[] args) {
    try {
      Height handler = new Height();
      XMLReader reader = XMLReaderFactory.createXMLReader();
      reader.setContentHandler(handler);
      reader.parse(args[0]);
      System.out.println(handler.max);
    } catch (Exception e) {
      System.err.println(e.getMessage());
    }
  }
}
```

Beispiel: Höhe eines XML Dokuments, III

Vergleich

- DOM Version verarbeitet maximal 18MB Eingabe
- SAX Version verarbeitet 1GB in einer Minute

11.4.5 SAX Filter

- Ereignisströme können *gefiltert* werden
- Ein Filter kann Ereignisse selbst verarbeiten oder sie weiterreichen
- Eine Anwendung kann mehrere Filter zusammensetzen
  $\Rightarrow$ modularer Aufbau
- Umsetzung
  - `XMLFilterImpl` ist Filterklasse, die alle Ereignisse weiterreicht
  - Änderung des Verhaltens durch Subklassen
- Beispiel: Drei Filter (Entfernung von PIs, Erzeugung von `id`-Attributen für alle Elemente, Berechnung der Länge aller Textdaten)

Ausfiltern von PIs

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
class PIFilter extends XMLFilterImpl {
  public void processingInstruction(String target,
                                    String data)
    throws SAXException {
  }
}
```

Generierung von `id`-Attributen

```
class IDFilter extends XMLFilterImpl {
  int id = 0;
  public void startElement(String uri, String localName,
                           String qName, Attributes atts)
    throws SAXException {
    AttributesImpl idatts = new AttributesImpl(atts);
    idatts.addAttribute("",
                        "id",
                        "id",
                        "ID",
                        new Integer(id++).toString());
    super.startElement(uri, localName, qName, idatts);
  }
}
```

Länge der Textdaten

```
class CountFilter extends XMLFilterImpl {
  public int count = 0;
  public void characters(char[] ch, int start, int length)
    throws SAXException {
    count = count+length;
    super.characters(ch, start, length);
  }
}
```

Komposition der Filter

```
public class FilterTest {
  public static void main(String[] args) {
    try {
      XMLReader reader = XMLReaderFactory.createXMLReader();
      PIFilter pi = new PIFilter();
      pi.setParent(reader);
      IDFilter id = new IDFilter();
      id.setParent(pi);
      CountFilter count = new CountFilter();
      count.setParent(id);
      count.parse(args[0]);
      System.out.println(count.count);
    } catch (Exception e) {
      System.err.println(e.getMessage());
    }
  }
}
```