Programmierpraktikum SS 2012

Austauschbare Datenformate, Parsing

JSON, XML

DOM/Sax-Parser

4. Juni 2012

Max Schneiders

Inhalt:

I. JSON – JavaScript Object Notation

Allgemein:

- *Kompaktes Datenformat zum Datenaustausch zwischen Anwendungen*
- *Für Mensch und Maschine einfach lesbare Textform*
- *Jedes JSON-Dokument ist ein JavaScript und soll mit eval() interpretiert werden können*
- *Unabhängig von der Programmiersprache*
- *Ersatz für XML (vor allem dort, wo Ressourcen wie Speicherplatz, CPU-Leistung etc. gespart werden sollen, z.B. bei der Entwicklung desktopähnlicher Webanwendungen)*
- *Leicht zu erlernen*

Datenstruktur:

- *Es gibt folgende Datentypen:*
  - *Nullwert:* null
  - *boolescher Wert:* true, false
  - *Zahl:* 43, -27, 6.9, 1e-4, … (nur dezimal!)
  - *Zeichenkette:* „Hallo Welt!\n“
  - *Array:* [„String1“, „String2“, …], [1, 2, 3, 4, 5, 6, …], [], …
  - *Objekt:* Leeres Objekt: {}, hat Eigenschaften: (Schlüssel:Wert), wobei der Schlüssel eine Zeichenkette ist und der Wert ein beliebiger Datentyp (oder ein Array eines Datentyps) von oben
- *Leerzeichen, Tabulatorzeichen, Zeilenumbrüche beliebig verwendbar*
- *Also alles ähnlich wie in C oder Java*

Beispiel:

```
{
  "Herausgeber": "Xema",
  "Nummer": "1234-5678-9012-3456",
  "Deckung": 2e+6,
  "Währung": "EURO",
  "Inhaber": {
    "Name": "Mustermann",
    "Vorname": "Max",
    "männlich": true,
    "Hobbys": [ "Reiten", "Golfen", "Lesen" ],
    "Alter": 42,
    "Kinder": [],
    "Partner": null
  }
}
```

- *Objekt mit Eigenschaften Herausgeber, Nummer, Deckung, …*
- *Die Eigenschaft „Inhaber“ ist wieder ein Objekt mit Eigenschaften Name, Vorname, männlich, …*
- *=> Verschachtelung von Objekten problemlos möglich!*

II. XML – EXtensible Markup Language (erweiterbare Auszeichnungssprache)

Markup Language (Auszeichnungssprache):

- *Definieren die Bausteine eines Dokuments und legen die Beziehung fest, in denen die einzelnen Dokumente zueinander stehen*
- *Textbasiert und beschreiben den logischen Inhalt von Dokumenten, deren Struktur und Datenaustausch*
- *Jede Auszeichnungssprache hat eine eigene Syntax*
- *Wir betrachten XML*

XML allgemein:

- *Erweiterbare Auszeichnungssprache, d.h. andere Auszeichnungssprachen können um strukturierte Informationen erweitert werden*
- *Dient zur Darstellung hierarchisch strukturierter Daten in Textform*
- *Für den plattform- und implementationsunabhängigen Austausch von Daten zwischen Computersystemen, insbesondere über das Internet*
- *Bestehen aus Textzeichen, im einfachsten Fall ASCII-Kodierung*
- *Ebenfalls für den Menschen lesbar*

Datenstruktur:

- *Struktur für jeden Dokumenttyp anders*
- *Daher selbstdefinierbare Tags:* <tag>
- *Fassen Daten des gleichen Typs zusammen, z.B. <email>, <name>, …*
- ***Achtung:** XML unterscheidet zwischen Groß- und Kleinschreibung: <name> und <Name> sind unterschiedliche Tags*
- *Tags können Attribute haben zur näheren Beschreibung:* <person alter="25">, <tag attribut="wert">
- *Dabei tritt kein Attribut doppelt auf (pro Tag)*
- *Zu jedem öffnenden Tag <tag> gehört stets ein schließendes Tag </tag>*
- *Alles was zwischen öffnendem und schließendem Tag steht – es können reiner Text oder auch andere Tags sein – nennt man Unterelement dieses Tags*
- *Dies erlaubt eine hierarchische, baumartige Struktur*
- *Kommentare:* <!-- Kommentar --> (nicht in Tags)

- *Entities (Ersetzungen):*
- <!ENTITY hhu „Heinrich-Heine-Universität Düsseldorf“>
- *Referenz:* &hhu; <student hochschule=“&hhu;“> … </student>
- *Vordefiniert:* ' (‘), " (“), > (>), < (<), & (&)
- *Entities können auch für andere Daten, z.B. Bilder stehen*
- *Wohlgeformtheit:*
    - *XML-Deklaration zu Beginn:* <?XML version=“1.0“?>
    - *Es muss immer ein Wurzelelement geben, das alle anderen Elemente beinhaltet:* <wurzel> Inhalt der XML </wurzel>
    - *Tags müssen immer geschlossen werden*
    - *Dabei ist auf die Verschachtelung zu achten, d.h. wird ein Tag innerhalb eines anderen Tags geöffnet, so wird erst das innere Tag geschlossen und dann das Äußere.*
    - *Der Wert eines Attributs steht in Anführungszeichen:* <tag attribut=“wert“>
    - *Sind diese Regeln erfüllt, so spricht man von einem wohlgeformten XML-Dokument*
    - *Statt <tag></tag> kann man auch <tag/> schreiben*

<u>Beispiel:</u>

```
<Kreditkarte
  Herausgeber="Xema"
  Nummer="1234-5678-9012-3456"
  Deckung="2e+6"
  Waehrung="EURO">
  <Inhaber
    Name="Mustermann"
    Vorname="Max"
    maennlich="true"
    Alter="42"
    Partner="null">
    <Hobbys>
      <Hobby>Reiten</Hobby>
      <Hobby>Golfen</Hobby>
      <Hobby>Lesen</Hobby>
    </Hobbys>
    <Kinder />
  </Inhaber>
</Kreditkarte>
```

Document Type Definition (DTD):

- *Die Beschreibung der XML-Tags wird in einer DTD verwaltet*
- *Beinhaltet Anzahl, Reihenfolge, Attribute von Tags eines XML-Dokuments*
- *Ein XML-Dokument, das zu einer bestimmten DTD konform ist, muss alles erfüllen, was in der DTD steht*
- *Wird im XML-Dokument deklariert:*
  <!DOCTYPE name SYSTEM "name.dtd">
- *Genügt ein XML-Dokument den Anforderungen einer DTD, so ist dies ein gültiges (valid) XML-Dokument*
- *Es muss keine DTD verwendet werden*
- *Dann deklariert man entsprechend:*
  <?XML version="1.0" standalone="yes"?>

Beispiel (adressdatei.dtd):

```
<!ENTITY kde "K Desktop Environment">
<!ELEMENT adressen (adresse*)>
<!ELEMENT adresse (vorname, nachname, mail*)>
<!ELEMENT vorname (#PCDATA)>
<!ELEMENT nachname (#PCDATA)>
<!ELEMENT mail (#PCDATA)>
<!ATTLIST mail type (privat | firma) #REQUIRED>
```

Erläuterung:

- *Definition einer Entity "kde"*
- *In jeder XML-Datei der Klasse adressdatei wird der Aufruf „&kde;" ersetzt durch „K Desktop Environment"*
- *Jede XML-Datei muss genau ein <adressen>-Tag mit beliebig vielen (auch keinen) <adresse>-Untertags enthalten*
- *Jedes <adresse>-Tag enthält genau ein <vorname>-, genau ein <nachname>- und beliebig viele <mail>-Untertags*
- *Diese drei Tags können reinen Text enthalten*
- *#PCDATA steht für* parsed character data*, d.h. der Text wird auf Entities überprüft, die ersetzt werden müssen (analog #CDATA)*

- *Will man z.B. „&“ schreiben, so muss es als „&“ kodiert werden:* <name> Ernie & Bert </name> entspricht: Ernie & Bert (vgl. *interne Entities oben)*
- *Jedes <mail>-Tag muss (#REQUIRED) das Attribut* type *haben, welches entweder den Wert “privat” oder “firma” haben muss*

<u>Beispiel (MathML):</u>

```
<mrow>
       <mrow>
              <msup>
                     <mi>x</mi>
                     <mn>2</mn>
              </msup>
              <mo>-</mo>
              <mn>4</mn>
       </mrow>
       <mo>=</mo>
       <mn>0</mn>
</mrow>
```

<u>Erläuterung:</u>

- *Diese XML-Datei erzeugt die Gleichung:* $\mathrm{x}^2 - 4 = 0$
- *<mrow>: Gruppiert Elemente, die nebeneinander (also in einer Zeile) dargestellt werden*
- *<msup>: Höherstellen von Zeichen (superscript), wichtig für Potenzen; hat immer zwei Untertags (Basis und Exponent)*
- *<mi>: Hier stehen Bezeichner (identifier), z.B. die Variable x oder Funktionsnamen wie sin, cos, …*
- *<mn>: Hier steht eine Zahl (number)*
- *<mo>: Hier stehen Operatoren: „+“, „=“, …*

Alternative: XML Schema Definition (XSD):

- *Dient zum Definieren von Strukturen für XML-Dokumente*
- *Selbst ein XML-Dokument*
- *Unterstützung vieler Datentypen*
- *Unterscheidung zwischen einfachen Typen und komplexen Typen*
- *Einfache Typen (xs:simpleType):*
  - xs:string, xs:decimal, xs:integer, xs:float, …
  - *Dürfen keine Kindelemente oder Attribute besitzen*
  - *Auch Listen und Vereinigungen (unions) sind einfache Typen*

Beispiel (XML-Datentyp monatInt):

```
<xs:simpleType name="monatInt">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="1"/>
    <xs:maxInclusive value="12"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="monate">
  <xs:list itemType="monatInt"/>
</xs:simpleType>
```

Beispiel (XML-Typ monat als Vereinigung):

```
<xs:simpleType name="monatsname">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Jan"/>
    <xs:enumeration value="Feb"/>
    <xs:enumeration value="Mär"/>
    <!-- und so weiter … -->
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="monat">
  <xs:union memberTypes="monatsname monatInt"/>
</xs:simpleType>
```

- *Zu 1: <monate> 1 2 3</monate>*
- *Zu 2: <monat>Jan</monat> oder <monat>4</monat>*

- *Komplexe Typen (xs:complexType):*
  - *Hier sind Kindelemente und Attribute erlaubt*
  - *Drei unterscheidliche Arten:*
    - *xs:sequence:* Eine Liste von Kindelementen wird spezifiziert und müssen in der angegeben Reihe vorkommen, dabei bezeichnen die Attribute *minOccurs* und *maxOccurs* die Häufigkeit (Standard = 1)
    - *xs:choice:* Aus einer Liste von Kindelementen kann eines ausgewählt werden
    - *xs:all:* Aus einer Liste von Kindelementen darf jedes maximal einmal vorkommen, die Reihenfolge ist beliebig

Beispiel (xs:sequence):

```
<xs:complexType name="pc-Typ">
  <xs:sequence>
    <xs:element name="name"       type="xs:string"/>
    <xs:element name="hersteller" type="xs:string"/>
    <xs:element name="prozessor"  type="xs:string"/>
    <xs:element name="mhz"        type="xs:integer"
minOccurs="0"/>
    <xs:element name="kommentar"  type="xs:string"
minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:integer"/>
</xs:complexType>
```

Beispiel (xs:choice):

```
<xs:element name="tagname">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded"
name="child" type="xs:integer"/>
      <!-- Weitere Elemente … -->
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- *Erweiterung und Einschränkung von Typen:*
  - *xs:extension:* neue Elemente werden einfach hinzugefügt
  - *xs:restriction:* Alte Elemente werden wiederholt und Veränderungen werden weggelassen
  - *Außerdem stehen für die Einschränkung Befehle zur Verfügung:* xs:length, xs:totalDigits, xs:enumeration, …

Beispiele:

1. *Extension:*

```
<xs:complexType name="myPC-Typ">
  <xs:complexContent>
    <xs:extension base="pc-Typ">
      <xs:sequence>
        <xs:element name="ram" type="xs:integer"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

2. *Restriction:*

```
<xs:complexType name="myPC2-Typ">
  <xs:complexContent>
    <xs:restriction base="pc-Typ">
      <xs:sequence>
       <xs:element name="name"       type="xs:string"/>
       <xs:element name="hersteller" type="xs:string"/>
       <xs:element name="prozessor"  type="xs:string"/>
       <xs:element name="mhz"        type="xs:integer"
minOccurs="0"/>
       <xs:element name="kommentar"  type="xs:string"
minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

3. *Restriction (Körpertemperatur): Dezimalzahl zwischen 35.0 und 42.5, drei Ziffern und eine Nachkommastelle*

```
<xs:simpleType name="celsiusKörperTemp">
  <xs:restriction base="xs:decimal">
    <xs:totalDigits value="3"/>
    <xs:fractionDigits value="1"/>
    <xs:minInclusive value="35.0"/>
    <xs:maxInclusive value="42.5"/>
  </xs:restriction>
</xs:simpleType>
```

4. *Restriction (Postleitzahl): optionales D und fünf Ziffern zwischen 0 und 9*

```
<xs:simpleType name="plz">
   <xs:restriction base="xs:string">
     <xs:pattern value="(D )?[0-9]{5}"/>
   </xs:restriction>
</xs:simpleType>
```

5. *Restriction (Enumeration):*

```
<xs:simpleType name="size">
  <xs:restriction base="xs:string">
    <xs:enumeration value="XS"/>
    <xs:enumeration value="S"/>
    <xs:enumeration value="M"/>
    <xs:enumeration value="L"/>
    <xs:enumeration value="XL"/>
  </xs:restriction>
</xs:simpleType>
```

III. XML-Parser (DOM und SAX)

XML-Parser allgemein:

- *Stellt der Anwendung die Informationen des XML-Dokuments zur Verfügung*
- *Stehen frei zur Verfügung*
- *Zwei Kriterien:* 1. validierend? 2. Schnittstelle?
- *Validierender Parser:* überprüft, ob das XML-Dokument gültig ist, d.h. ob es konform zu einer DTD ist (eine DTD ist in diesem Fall also erforderlich); auch die Wohlgeformtheit des Dokuments wird überprüft
- *Nicht validierender Parser:* überprüft lediglich die Wohlgeformtheit des XML-Dokuments; DTD ist nicht nötig, kann aber vorhanden sein
- *Allerdings:* Das Validieren durch den Parser ist sehr zeitaufwändig und sollte wenn möglich vermieden werden, vor allem wenn man auf andere Weise die Validität des XML-Dokuments sicherstellen kann!
- *In den meisten Browsern (Internet Explorer, Mozilla Firefox, …) gibt es bereits einen integrierten XML-Parser*
- *Wir betrachten zwei Schnittstellen:* DOM und SAX

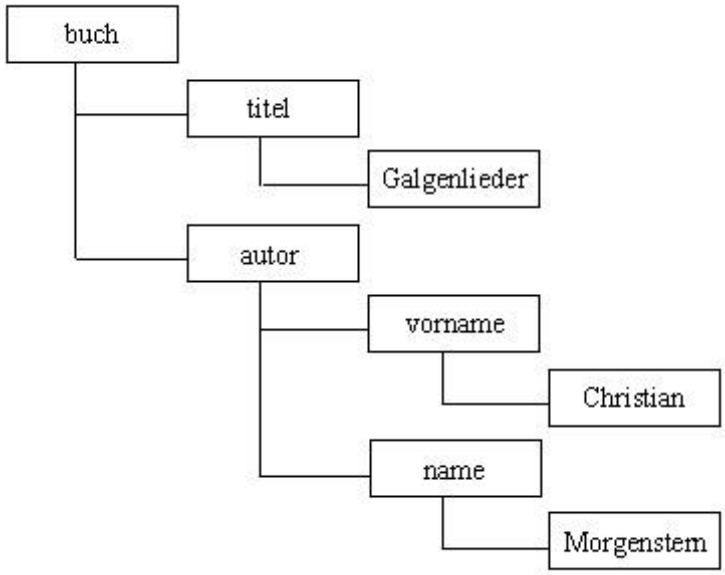
Document Object Model (DOM):

- *In Java implementiert*
- *Baut beim Parsen einen Baum auf:* DOM-Tree
- *Die Wurzel des XML-Dokuments ist dann die Wurzel des DOM-Trees*
- *Die Elemente des XML-Dokuments entsprechen dann den Knoten des Baumes*
- *Es gibt also Elementknoten (auch mit Attributen) und Textknoten*

- *Aus einem DOM-Tree lässt sich sehr leicht ein XML-Dokument aufbauen*
- *DOM-API bietet Werkzeuge zum Zugriff auf den Baum und zum Verändern der Baumstruktur; Ideal für Interaktive Anwendungen, da DOM die ganze Zeit im Speicher ist*
- *Nachteil: sehr langsam und speicherintensiv*

Beispiel:

```
<buch>
    <titel>Galgenlieder</titel>
    <autor>
        <vorname>Christian</vorname>
        <name>Morgenstern</name>
    </autor>
</buch>
```



- *Java-Beispiele:*
  http://www.uzi-web.de/parser/parser_dom_nonval.htm

Simple API for XML (SAX):

- *Ebenfalls in Java implementiert*
- *"event-based":* Der Parser löst beim Lesen eines XML-Konstrukts ein Ereignis aus
- *Können vom Entwickler ganz individuell behandelt werden, durch Schreiben eines eigenen Eventhandlers, der beim Parser registriert wird*
- *Ereignisse werden nicht gespeichert, d.h. für die Speicherung ist man selbst zuständig*
- *DefaultHandler:*
    - *startDocument() und endDocument()*
    - *startElement() und endElement()*
    - *characters()*

Beispiel:

```
<zitat>
    Drum prüfe wer sich ewig bindet, ob sich nicht was
Besseres findet.
</zitat>
```

- *Der Parser würde folgende Ereignisse auslösen:*
    - *Öffnendes Tag*
    - *Text*
    - *Schließendes Tag*
- *Jedes Ereignis ruft einen Eventhandler (vom Entwickler geschriebene Methode) auf, der das Ereignis verarbeitet, z.B. könnte man das Zitat auf dem Bildschirm ausgeben*
- *SAX ist im Vergleich zu DOM schnell und einfach hat aber kein Object Model, worauf man zugreifen kann*
- *Ideal für Anwendungen, die das XML-Dokument nur einmal durchlesen müssen, z.B. um es anzuzeigen*
- *Java-Beispiele:*
  http://www.uzi-web.de/parser/parser_sax_grundlagen.htm

IV. Quellen:

JSON:

- http://de.wikipedia.org/wiki/JavaScript_Object_Notation

XML:

- http://de.wikipedia.org/wiki/Extensible_Markup_Language
- http://de.wikipedia.org/wiki/Auszeichnungssprache
- http://de.wikipedia.org/wiki/XML_Schema
- http://wwwbayer.in.tum.de/lehre/WS2000/PS00-Ausarbeitungen/dueck-xmlproseminar.html
- http://www.uzi-web.de/

DOM/SAX-Parser:

- http://www.uzi-web.de/parser/parser_sax_grundlagen.htm
- http://de.wikipedia.org/wiki/Java_API_for_XML_Processing
- http://www.uzi-web.de/