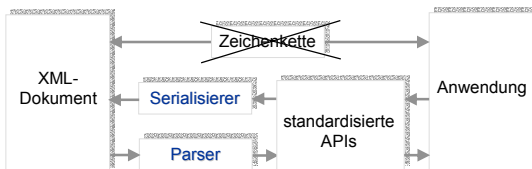


XML-Parser

Heutige Vorlesung

- Welche XML-Parser gibt es?
- Was sind ihre Vor- und Nachteile?
- Was sind Schema-Übersetzer?

Grundlegende Architektur



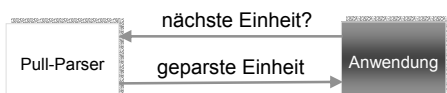
- Möglichst standardisierte APIs verwenden!
- **Parser**: Analysiert XML-Dokument und erstellt Parse-Baum mit Tags, Text-Inhalten und Attribut-Wert-Paaren als Knoten.
- **Serialisierer**: Generiert aus bestimmter Datenstruktur ein XML-Dokument.

Kategorien von Parser



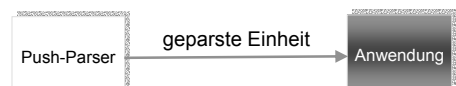
- **Pull- vs. Push-Parser**
Wer hat Kontrolle über das Parsen: die Anwendung oder der Parser?
- **Einschritt- vs. Mehrschritt-Parser** (*one-step vs. multi-step parsing*)
Wird das XML-Dokument in einem Schritt vollständig geparkt oder Schritt für Schritt?
- **Beachte**: Kategorien unabhängig voneinander, können kombiniert werden

Pull-Parser



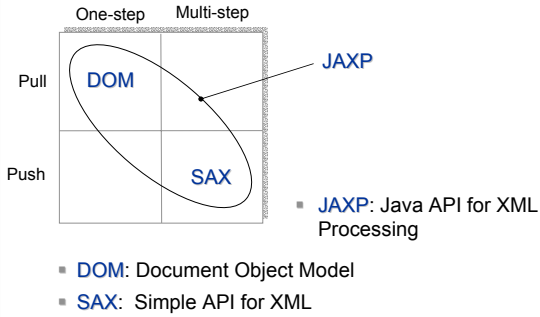
- Anwendung hat Kontrolle über das Parsen.
- Analyse der nächsten syntaktischen Einheit muss aktiv angefordert werden.
- **Beachte**: „Pull“ bezieht sich auf die Perspektive der Anwendung.

Push-Parser



- Parser hat Kontrolle über das Parsen.
- Sobald der Parser eine syntaktische Einheit analysiert hat, übergibt er die entsprechende Analyse.
- **Beachte**: „Push“ bezieht sich wiederum auf die Perspektive der Anwendung.

XML-Parser



© Klaus Schild, 2004

7

SAX-Parser



© Klaus Schild, 2004

8

Simple API for XML (SAX)

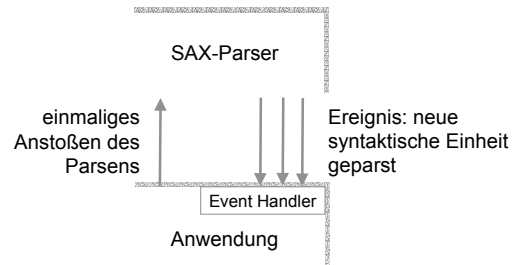


- Mehrschritt-Push-Parser für XML
 - standardisiertes API
 - ursprünglich nur Java-API, inzwischen werden aber auch viele andere Sprachen unterstützt
 - kein W3C-Standard, sondern *de facto* Standard
- <http://www.saxproject.org/>

© Klaus Schild, 2004

9

Ereignisbasiertes Parsen



© Klaus Schild, 2004

10

Beispiel



```
<priceList>
  <coffee>
    <name>
      Mocha Java
    </name>
    <price>
      11.95
    </price>
  </coffee>
</priceList>
```

Parser ruft startElement(...,priceList,...) auf.
Parser ruft startElement(...,coffee,...) auf.
Parser ruft startElement(...,name,...) auf.
Parser ruft characters("Mocha Java",...) auf.
Parser ruft endElement(...,name,...) auf.
Parser ruft startElement(...,price,...) auf.
Parser ruft characters("11.95",...) auf.
Parser ruft endElement(...,price,...) auf.
Parser ruft endElement(...,coffee,...) auf.
Parser ruft endElement(...,priceList,...) auf.

- **Ereignisfluss:** Sobald Einheit geparkt wurde, wird Anwendung benachrichtigt.
- **Beachte:** Es wird *kein* Parse-Baum aufgebaut!

© Klaus Schild, 2004

11

Callback-Methoden



- **Callback-Methoden:** Methoden des Event-Handlers (also der Anwendung), die vom Parser aufgerufen werden.
- für jede syntaktische Einheit in XML eigene Callback-Methode:
 - startElement
 - endElement
 - characters
- **DefaultHandler:** Standard-Implementierung der Callback-Methoden, tun jeweils nichts.
- Standard-Implementierungen können aber überschrieben werden.

© Klaus Schild, 2004

12

Beispiel

```
<priceList>
  <coffee>
    <name>
      Mocha Java
    </name>
    <price>
      11.95
    </price>
  </coffee>
</priceList>
```

- **Aufgabe:** Gib den Preis von Mocha Java aus!
- Hierfür benötigen wir zwei Dinge:
 1. einen SAX-Parser
 2. passende Callback-Methoden

Wie bekomme ich einen SAX-Parser?

```
SAXParserFactory factory = SAXParserFactory.newInstance();
// liefert eine SAXParserFactory

SAXParser saxParser = factory.newSAXParser();
// liefert einen SAXParser

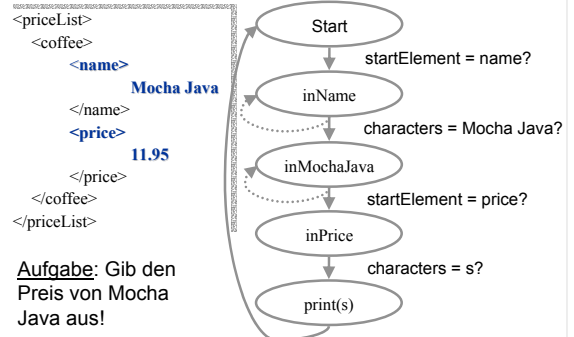
saxParser.parse("priceList.xml", handler);
```

- stößt SAX-Parser an
- priceList.xml: zu parsende Datei, kann auch URL oder Stream sein
- handler: Instanz von DefaultHandler, implementiert Callback-Funktionen

Exkurs: Factory Method

- Entwurfsmuster aus „Design Patterns“ von Gamma, Helm, Johnson, Vlissides (1995)
- liefert ein Objekt
- Objekt ist Instanz einer abstrakten Klasse oder einem Interface.
- wird von mehreren Klassen implementiert
- **Beispiel:** Iterator i = list.iterator();
- **Beispiel:** SAXParser saxParser = factory.newSAXParser();

Wie sehen die Callback-Methoden aus?



Die Callback-Methoden in Java

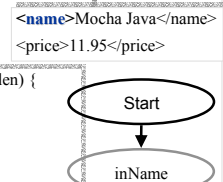
```
public void startElement(..., String elementName, ...) {
  if (elementName.equals("name")) { inName = true; }
  else if (elementName.equals("price") && inMochaJava) {
    inPrice = true;
    inName = false; } }
```

```
public void characters(char [] buf, int offset, int len) {
  String s = new String(buf, offset, len);
  if (inName && s.equals("Mocha Java")) {
    inMochaJava = true;
    inName = false; }
  else if (inPrice) {
    System.out.println("The price of Mocha Java is: " + s);
    inMochaJava = false;
    inPrice = false; } }
```

Auf <name> warten

```
public void startElement(..., String elementName, ...) {
  if (elementName.equals("name")) { inName = true; }
  else if (elementName.equals("price") && inMochaJava) {
    inPrice = true;
    inName = false; } }
```

```
public void characters(char [] buf, int offset, int len) {
  String s = new String(buf, offset, len);
  if (inName && s.equals("Mocha Java")) {
    inMochaJava = true;
    inName = false; }
  else if (inPrice) {
    System.out.println("The price of Mocha Java is: " + s);
    inMochaJava = false;
    inPrice = false; } }
```



Auf "Mocha Java" warten

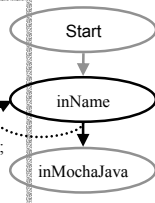
```
public void startElement(..., String elementName, ...){
    if (elementName.equals("name")){    inName = true; }
    else if (elementName.equals("price") && inMochaJava ){
        inPrice = true;
        inName = false; } }

```

```
<name>Mocha Java</name>
<price>11.95</price>
```

```
public void characters(char [] buf, int offset, int len) {
    String s = new String(buf, offset, len);
    if (inName && s.equals("Mocha Java")) {
        inMochaJava = true;
        inName = false; }
    else if (inPrice) {
        System.out.println("The price of Mocha Java is: " + s);
        inMochaJava = false;
        inPrice = false; } }

```



© Klaus Schild, 2004

19

Auf <price> warten

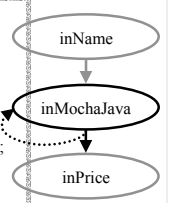
```
public void startElement(..., String elementName, ...){
    if (elementName.equals("name")){    inName = true; }
    else if (elementName.equals("price") && inMochaJava ){
        inPrice = true;
        inName = false; } }

```

```
<name>Mocha Java</name>
<price>11.95</price>
```

```
public void characters(char [] buf, int offset, int len) {
    String s = new String(buf, offset, len);
    if (inName && s.equals("Mocha Java")) {
        inMochaJava = true;
        inName = false; }
    else if (inPrice) {
        System.out.println("The price of Mocha Java is: " + s);
        inMochaJava = false;
        inPrice = false; } }

```



© Klaus Schild, 2004

20

Den Preis ausgeben

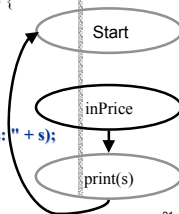
```
public void startElement(..., String elementName, ...){
    if (elementName.equals("name")){    inName = true; }
    else if (elementName.equals("price") && inMochaJava ){
        inPrice = true;
        inName = false; } }

```

```
<name>Mocha Java</name>
<price>11.95</price>
```

```
public void characters(char [] buf, int offset, int len) {
    String s = new String(buf, offset, len);
    if (inName && s.equals("Mocha Java")) {
        inMochaJava = true;
        inName = false; }
    else if (inPrice) {
        System.out.println("The price of Mocha Java is: " + s);
        inMochaJava = false;
        inPrice = false; } }

```

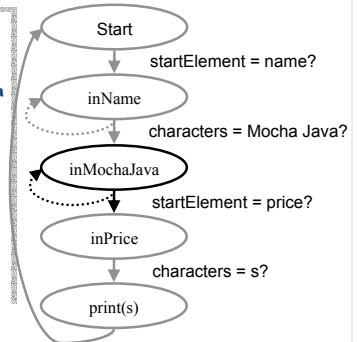


© Klaus Schild, 2004

21

Fehlerbehandlung

```
<priceList>
<coffee>
  <name>
    Mocha Java
  </name>
  <name>
    MS Java
  </name>
  <price>
    11.95
  </price>
</coffee>
</priceList>
```



© Klaus Schild, 2004

22

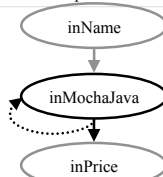
Fehlerbehandlung

```
public void startElement(..., String elementName, ...){
    if (elementName.equals("name")){    inName = true; }
    else if (elementName.equals("price") && inMochaJava ){
        inPrice = true;
        inName = false; } }

```

```
<name>Mocha Java</name>
<name>MS Java</name>
<price>11.95</price>
```

- inMochaJava erwartet price-Element
- Kommt stattdessen name-Element, wird aktueller Zustand nicht verändert.
- Kommt danach price-Element, wird der aktueller Zustand inPrice.
- Dadurch wird Preis von MS Java ausgegeben!



© Klaus Schild, 2004

23

Fehlerbehandlung

- SAX-Parser überprüft immer Wohlgeformtheit eines XML-Dokumentes.
- kann aber auch die *Zulässigkeit* bzgl. einer DTD oder eines Schema überprüfen
- Syntax- und Strukturfehler fängt bereits der SAX-Parser ab.
- Callback-Methoden können von einem wohlgeformten und zulässigen Dokument ausgehen.

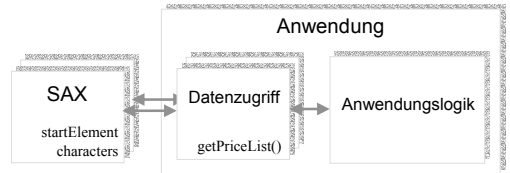
© Klaus Schild, 2004

24

Vor- und Nachteile von SAX

- + sehr effizient, auch bei großen XML-Dokumenten
- Kontext (Parse-Baum) muss von Anwendung selbst verwaltet werden.
- abstrahiert nicht von XML-Syntax
- nur Parsen möglich, keine Modifikation oder Erstellung von XML-Dokumenten

Zusätzliche Schicht zum Datenzugriff



- Immer Anwendungslogik durch zusätzliche Schicht von dem Datenzugriff trennen (nicht nur bei SAX).
- Z.B. könnte getPriceList() eine Liste von Ware-Preis-Paaren liefern.
- sollte nicht nur von SAX-APIs, sondern auch von der XML-Syntax abstrahieren

DOM-Parser

Document Object Model (DOM)

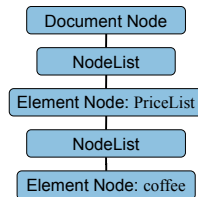


- W3C-Standard
- abstrakte Schnittstelle zum Zugreifen, Modifizieren und Erstellen von Parse-Bäumen
- unabhängig von Programmiersprachen
- nicht nur für XML-, sondern auch für HTML-Dokumente
- im Ergebnis ein Einschnitt-Pull-Parser

DOM-Parse-Bäume

```

<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
  
```

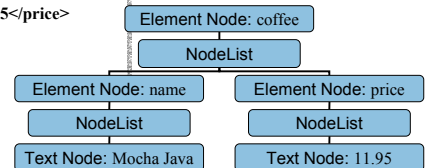


- **Beachte:** priceList ≠ Dokument-Wurzel
- DOM führt virtuelle Dokument-Wurzel ein, um syntaktische Einheiten außerhalb von priceList (wie version="1.0") repräsentieren zu können.

DOM-Parse-Bäume

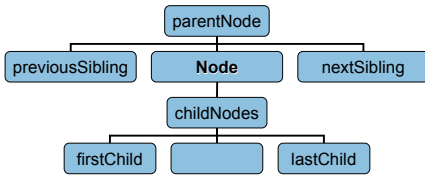
```

<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
  
```



- **Beachte:** Text-Inhalte werden als eigene Knoten dargestellt.

Navigationsmodell



Beispiel

```
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

- **Aufgabe:** Gib des Preis von Mocha Java aus!
- Hierfür benötigen wir zwei Dinge:

1. einen DOM-Parser
2. eine passende Zugriffsmethode

Wie bekomme ich einen DOM-Parser?

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

- liefert DocumentBuilderFactory

```
DocumentBuilder builder = factory.newDocumentBuilder();
```

- liefert DOM-Parser

```
Document document = builder.parse("priceList.xml");
```

- DOM-Parser hat Methode parse().
- liefert DOM-Parse-Baum

Wie sehen die Zugriffsmethoden aus?

```
NodeList coffeeNodes = document.getElementsByTagName("coffee");
for (int i=0; i < coffeeNodes.getLength(); i++) {
  thisCoffeeNode = coffeeNodes.item(i);
  Node thisNameNode = thisCoffeeNode.getFirstChild();
  String data = thisNameNode.getNodeValue();
  if (data.equals("Mocha Java")) {
    Node thisPriceNode = thisNameNode.getNextSibling();
    String price = thisPriceNode.getFirstChild().getNodeValue();
    break; } }
```

Java-Programm, das DOM-Methoden benutzt

Gib mir die Liste aller coffee-Elemente!

```
NodeList coffeeNodes = document.getElementsByTagName("coffee");
```

- getElementsByTagName: direkter Zugriff auf Elemente über ihren Namen
- egal, wo Elemente stehen
- Resultat ist immer eine NodeList.

```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

Betrachte alle Elemente der coffee-Liste!

```
NodeList coffeeNodes = document.getElementsByTagName("coffee");
for (int i=0; i < coffeeNodes.getLength(); i++) {
  thisCoffeeNode = coffeeNodes.item(i);
  ...
}
```

coffeeNodes.item(0)

```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

Gib mir erstes Kind-Element von coffee!

```
NodeList coffeeNodes = document.getElementsByTagName("coffee");
for (int i=0; i < coffeeNodes.getLength(); i++) {
    thisCoffeeNode = coffeeNodes.item(i);
    Node thisNameNode = thisCoffeeNode.getFirstChild();
    String data = thisNameNode.getFirstChild().getNodeValue();
    if (data.equals("Mocha Java")) {
        Node thisPriceNode = thisCoffeeNode.getNextSibling();
        String price = thisPriceNode.getFirstChild();
        break; } }
```

firstChild

```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

Gib mir den Inhalt von name!

```
NodeList coffeeNodes = document.getElementsByTagName("coffee");
for (int i=0; i < coffeeNodes.getLength(); i++) {
    thisCoffeeNode = coffeeNodes.item(i);
    Node thisNameNode = thisCoffeeNode.getFirstChild();
    String data = thisNameNode.getFirstChild().getNodeValue();
    if (data.equals("Mocha Java")) {
        Node thisPriceNode = thisCoffeeNode.getNextSibling();
        String price = thisPriceNode.getFirstChild();
        break; } }
```

- **Beachte:** getFirstChild() liefert Text-Knoten, nicht dessen Inhalt „Mocha Java“!

firstChild

```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

Gib mir das nächste Kind-Element!

```
NodeList coffeeNodes = document.getElementsByTagName("coffee");
for (int i=0; i < coffeeNodes.getLength(); i++) {
    thisCoffeeNode = coffeeNodes.item(i);
    Node thisNameNode = thisCoffeeNode.getFirstChild();
    String data = thisNameNode.getFirstChild().getNodeValue();
    if (data.equals("Mocha Java")) {
        Node thisPriceNode = thisNameNode.getNextSibling();
        String price = thisPriceNode.getFirstChild();
        break; } }
```

nextSibling

```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

Gib mir den Inhalt von price!

```
NodeList coffeeNodes = document.getElementsByTagName("coffee");
for (int i=0; i < coffeeNodes.getLength(); i++) {
    thisCoffeeNode = coffeeNodes.item(i);
    Node thisNameNode = thisCoffeeNode.getFirstChild();
    String data = thisNameNode.getFirstChild().getNodeValue();
    if (data.equals("Mocha Java")) {
        Node thisPriceNode = thisNameNode.getNextSibling();
        String price = thisPriceNode.getFirstChild().getNodeValue();
        break; } }
```

firstChild

```
<?xml version="1.0" ?>
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

Vor- und Nachteile von DOM

- + Kontext (Parse-Baum) muss *nicht* von Anwendung verwaltet werden.
- + direkter Zugriff auf Elemente und Attribute über ihre Namen
- + nicht nur Parsen, sondern auch Modifikation und Erstellung von XML-Dokumenten
- speicherintensiv

SAX oder DOM?

SAX

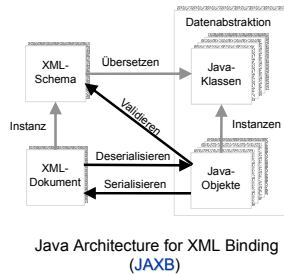
- geeignet, um gezielt bestimmte Teile von XML-Dokumenten herauszufiltern, ohne zu einem späteren Zeitpunkt andere Teile des Dokumentes zu benötigen

DOM

- geeignet, um auf unterschiedliche Teile eines XML-Dokumentes zu verschiedenen Zeitpunkten zuzugreifen
- Erstellen und Modifizieren von XML-Dokumenten

Schema-Übersetzer

Schema-Übersetzer



Beispiel

```
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
</priceList>
```

PriceList-Schema

JAXB

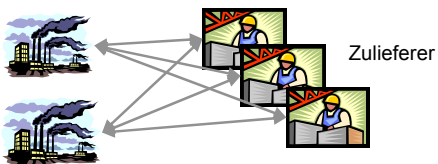
```
public interface PriceList {
    java.util.List getCoffee();
    public interface CoffeeType {
        String getName();
        void setName(String value)
        java.math.BigDecimal getPrice();
        void setPrice(java.math.BigDecimal value) }
}
```

Warum noch XML lernen?

- Schema-Übersetzer: XML-Schema kann in Java- oder C#-Klassen übersetzt werden.
- Hiermit können XML-Dokumente gelesen, modifiziert und erstellt werden, *ohne* dass XML sichtbar ist.

Warum sich also noch mit XML und XML-Schemata beschäftigen?

Typisches E-Business-Projekt



- Branchenvertreter wollen miteinander Geschäfte über das Internet machen.
- Sie müssen sich auf ein Austauschformat einigen.

Typisches E-Business-Projekt: Phase I

- Welche Geschäftsdaten sollen ausgetauscht werden?
- Gibt es bereits einen passenden Branchenstandard?
- Wie sollen diese Geschäftsdaten in XML repräsentiert werden?
- Gibt es bereits einen geeigneten XML-Standard?
- Ziel:** Branchenstandard in Form eines XML-Schemas

Software-Architekten entwickeln gemeinsam einen XML-basierten Branchenstandard.

Typisches E-Business-Projekt: Phase II



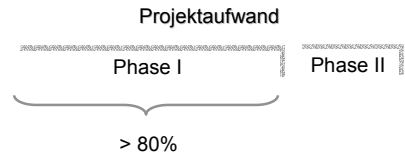
- **gegeben:**
 - Branchenstandard in Form eines XML-Schemas
 - gemeinsames Verständnis der XML-Syntax
- **Aufgabe:** Realisierung der Schnittstelle zwischen betriebsinterner Software und dem XML-Standard.

Programmierer können Schema-Übersetzer einsetzen und von XML abstrahieren.

Warum sich noch mit XML beschäftigen?



- **Phase I:** Software-Architekten beschäftigen sich intensiv mit entsprechender Branche, XML und XML-Schemata.
- **Phase II:** Programmierer können Schema-Übersetzer einsetzen und von XML abstrahieren.



Wie geht es weiter?



heutige Vorlesung

- Sax- und DOM-Parser
- Vor- und Nachteile von SAX und DOM
- Schema-Übersetzer

nächste Woche

- Transformation von XML-Dokumenten mit XSLT