



Unweit von hier befand sich das Basislager von dem aus
die wagemutigen BRÜDER FLORIAN UND MICHAEL BLOCH
mit ihren unerschrockenen Sherpas am
Sonntag den 28.6.2009 um 11:44:17 MEZ
zur ERSTBESTEIGUNG DES PATSCHERKOFEL MIT SAUERSTOFFMASKEN aufbrachen.
Um 13:35:02 MEZ konnte am Fuß des Gipfelkreuzes die Pfeifnudel-Flagge gehisst werden.

www.pfeifnudel.de

07-Input/Output-4-XMLJava

Objektorientierte Programmierung | Matthias Tichy

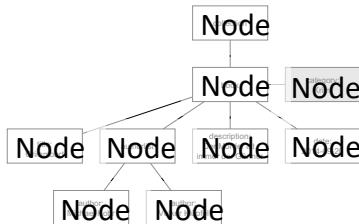
Lernziele

- XML in Java verarbeiten
 - DOM
 - Sax
- XPath
- JSON
- YAML

XML Verarbeitung

■ Prinzipiell zwei Möglichkeiten

1. Kompletten Baum im Speicher aufbauen → DOM
2. Tag für Tag verarbeiten → SAX



```
<?xml version="1.0" encoding="UTF-8"?>
<collection>
  <book category="Krimi">
    <!-- Kluftinger -->
    <title>Grimmbart</title>
    <authorlist>
      <author>Volker Klüpfel</author>
      <author>Michael Kobr</author>
    </authorlist>
    <description />
    <date>20.09.2014</date>
  </book>
</collection>
```

```
startDocument()
processingInstruction(...)
startElement(collection, ...)
startElement(book, ...)
startElement(title, ...)
characters(...)
endElement(title, ...)
startElement(authorlist, ...)
startElement(author, ...)
characters(...)
endElement(author, ...)
startElement(author, ...)
characters(...)
endElement(author, ...)
startElement(description, ...)
endElement(description, ...)
startElement(date, ...)
characters(...)
endElement(date, ...)
endElement(book, ...)
```

SAX

```
startDocument()  
processingInstruction(...)  
startElement(collection, ...)  
startElement(book, ...)  
startElement(title, ...)  
characters(...)  
endElement(title, ...)  
startElement(authorlist, ...)
```

- Simple API for XML
- De-facto-Standard
- viele Sprachen

- liest XML-Datei als sequentiellen Datenstrom (Stream)
- bei bestimmten Ereignissen werden definierte Methoden gerufen

- Aktueller "Zustand" bzw. Position im Dokument muss selbst verwaltet werden

- Praktisch keine Möglichkeit, Baum zu verändern
- Nur Werte können ausgelesen werden

SAX

```
startDocument()  
processingInstruction(...)  
startElement(collection, ...)  
startElement(book, ...)  
startElement(title, ...)  
characters(...)  
endElement(title, ...)  
startElement(authorlist, ...)
```

- SAX-Ereignisse:
 - startDocument
 - startElement
 - endElement
 - endDocument
 - characters
 - processingInstruction
 - ...
- startElement liefert auch die Attribute des Elements mit
- characters gibt auch alle whitespaces zwischen den Tags aus (z.B. Zeilenumbruch)

SAX - Example

```
<planet name="Arrakis" climate="Desert">
  <notableInhabitants>
    <inhabitant>
      <name>Paul Atreides</name>
      <role>Protagonist</role>
    </inhabitant>
    <inhabitant>
      <name>Baron Harkonnen</name>
      <role>Antagonist</role>
    </inhabitant>
  </notableInhabitants>
</planet>
```

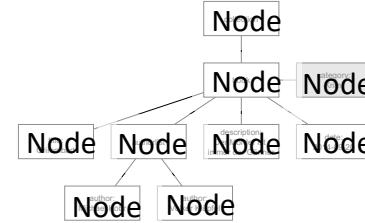
```
startDocument()
processingInstruction(...)
startElement(collection, ...)
startElement(book, ...)
startElement(title, ...)
characters(...)
endElement(title, ...)
startElement(authorlist, ...)
```

```
DefaultHandler handler = new DefaultHandler() {
  new *
  @Override
  public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {
    if(qName.equals("planet")) {
      System.out.println("Planet: " + attributes.getValue( qName: "name"));
      System.out.println("Planet: " + attributes.getValue( qName: "climate"));
    }
  }

  new *
  @Override
  public void endElement(String uri, String localName, String qName) throws SAXException {
  }

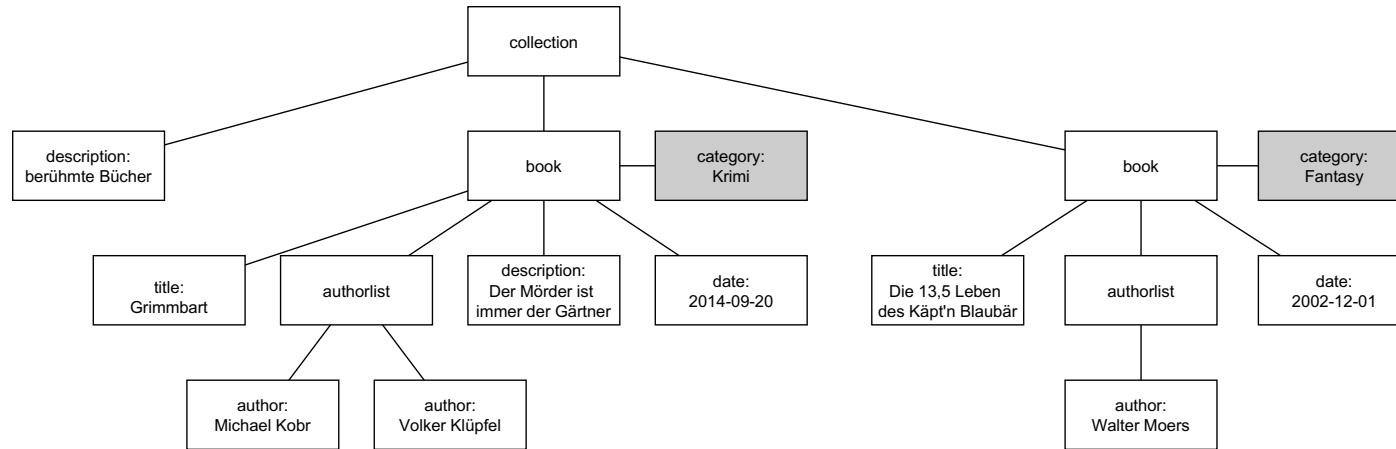
  new *
  @Override
  public void characters(char ch[], int start, int length) throws SAXException {
  }
};
```

DOM

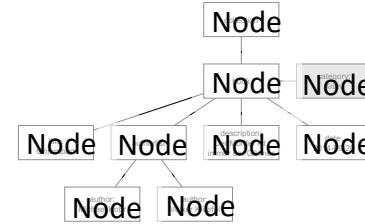


- Document Object Model
- Spezifikation von W3C (World Wide Web Consortium)
- Eigentlich kein "Model" sondern plattform- und sprachunabhängiges Interface
- Für Schnittstelle vorausgesetzte Baumrepräsentation ist "Model"
- Entstanden in den 90er Jahren → Browserkrieg
- JavaScript im Browser zur Manipulation von HTML

Baumnavigation

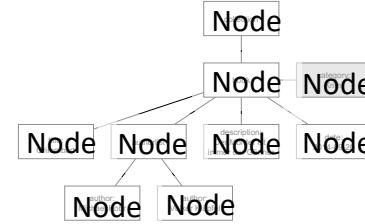


DOM



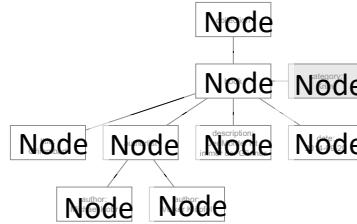
- Alles sind Nodes
- Methoden des Interface zur Baumnavigation
 - `NodeList getChildNodes()`
 - `Node getFirstChild()`
 - `Node getLastChild()`
 - `Node getPreviousSibling()`
 - `Node getNextSibling()`
 - `Node getParentNode()`
- Methoden zur Baummanipulation
 - `Node appendChild(Node newChild)`
 - `Node removeChild(Node oldChild)`
 - `Node replaceChild(Node newChild, Node oldChild)`

DOM



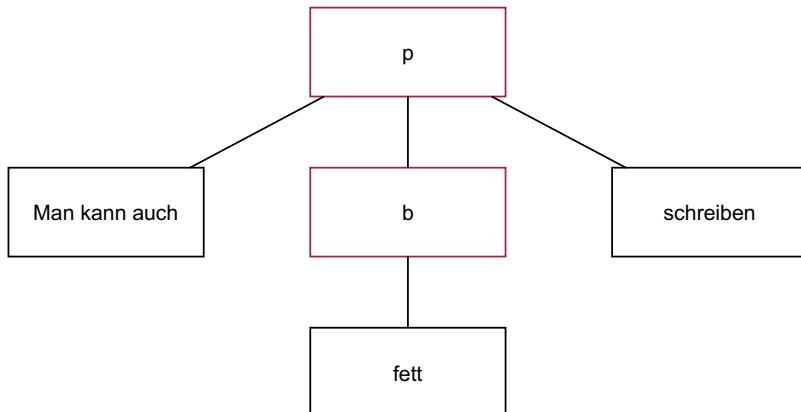
- Methoden zum Auslesen und Ändern von Inhalten:
 - `NamedNodeMap getAttributes()`
 - `String getNodeName()`
 - `String getNodeValue()`
 - `void setNodeValue(String value)`
 - `short getNodeType()`
- Node-Types (Auswahl):
 - `ATTRIBUTE_NODE`
 - `DOCUMENT_NODE`
 - `ELEMENT_NODE`
 - `ENTITY_NODE`
 - `TEXT_NODE`
- Pro Typ abgeleitete Interfaces mit speziellen Methoden

DOM



- Mixed-Content möglich:

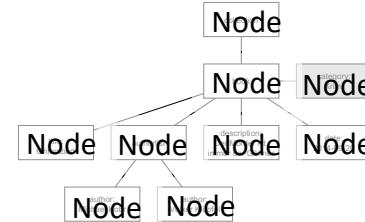
< p > Man kann auch < b > fett < /b > schreiben < /p >



```
<xs:element name="p">
  <xs:complexType mixed="true">
    <xs:choice minOccurs="0"
      maxOccurs="unbounded">
      <xs:element
        name="b" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

DOM - Example

```
<planet name="Arrakis" climate="Desert">
  <notableInhabitants>
    <inhabitant>
      <name>Paul Atreides</name>
      <role>Protagonist</role>
    </inhabitant>
    <inhabitant>
      <name>Baron Harkonnen</name>
      <role>Antagonist</role>
    </inhabitant>
  </notableInhabitants>
</planet>
```



```
NodeList nodeList = document.getElementsByTagName("planet");

for (int i = 0; i < nodeList.getLength(); i++) {
    Node node = nodeList.item(i);

    if (node.getNodeType() == Node.ELEMENT_NODE) {
        Element element = (Element) node;

        System.out.println("Planet: " + element.getAttribute("name"));
        System.out.println("Climate: " + element.getAttribute("climate"));
    }
}
```

JAXP

- "Java API for XML processing" fasst verschiedene APIs zusammen:
 - SAX
 - DOM
 - StAX (Streaming API for XML)
- Außerdem:
 - Möglichkeiten der Validierung (DTD, XSD)
 - Möglichkeit, XSLT auszuführen (siehe später)

StAX

SAX

- schnell
- wenig Speicherverbrauch
- unflexibel
- nur lesen

DOM

- langsam
- viel Speicherverbrauch
- beliebige Manipulationen

StAX

- Kombination der Vorteile von SAX und DOM
- Arbeitet wie SAX, aber kann im Dokument vor- und zurückspringen
- dadurch "wahlfreier" Zugriff wie bei DOM

XPath

- Pfadausdrücke zur Navigation in XML-Dokumenten
- beinhaltet Bibliothek mit Standardfunktionen
- Ist elementarer Teil von XSL (siehe später)
- Pfadausdrücke selektieren Mengen von Knoten ("|" ist Symbol für Vereinigung)
- Prädikate filtern diese Menge

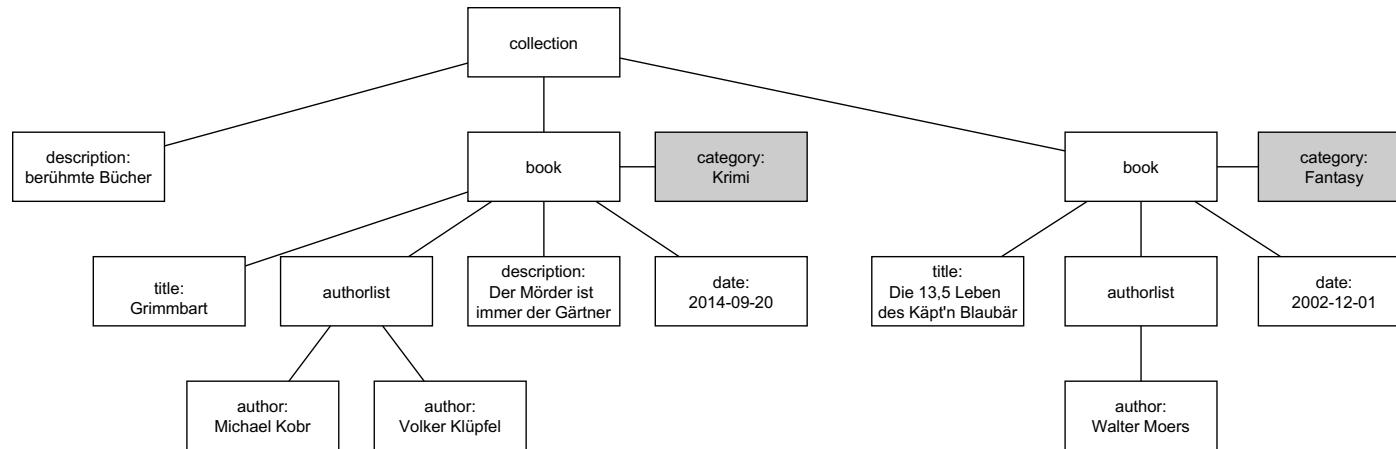
XPath – Pfadausdrücke

Symbol	Bedeutung
/	root-Node
<node-name>	alle Knoten mit diesem Namen
.	aktueller Knoten
..	parent-Node
//	alle (Sub-)Knoten ab dem aktuellen Knoten
@	Selektion von Attributen

XPath – Prädikate

Symbol	Bedeutung
[1]	erster Kindknoten
[last()]	letzter Kindknoten
[last()-1]	vorletzter Kindknoten
[position() < 3]	die ersten zwei Kindknoten
[@category]	alle Knoten mit Attribut "category"
[@category='Krimi']	alle Knoten deren Attribut "category" den Wert "Krimi" hat
[price > 35.00]	alle Knoten, deren Element "price" größer als 35.00 ist

Baumnavigation

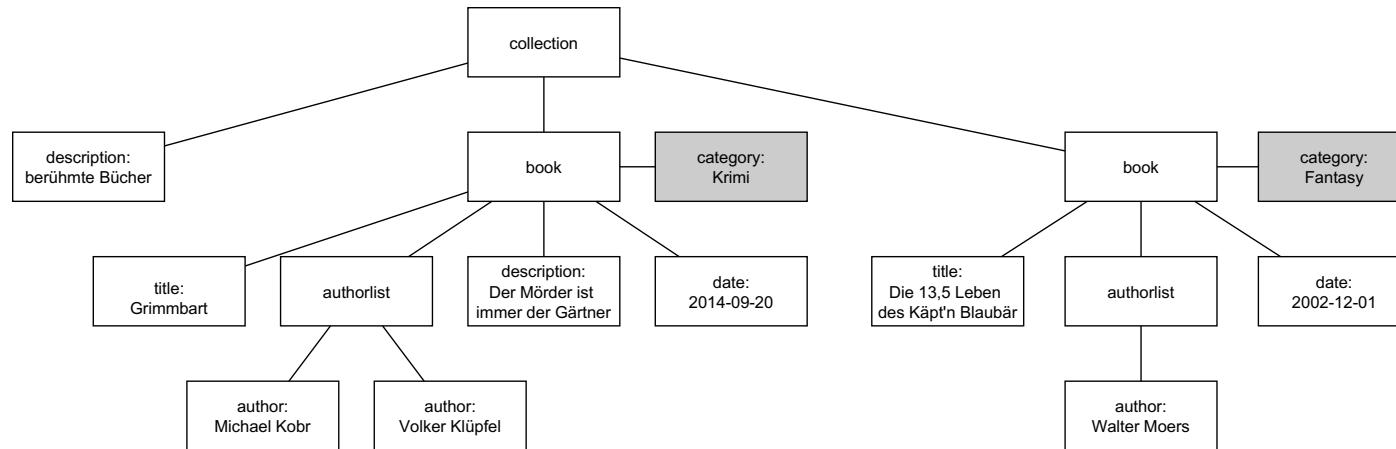


absolute Pfade:

/collection/description → eindeutig

/collection/book/authorlist/author → mehrdeutig: Menge von 3 Knoten

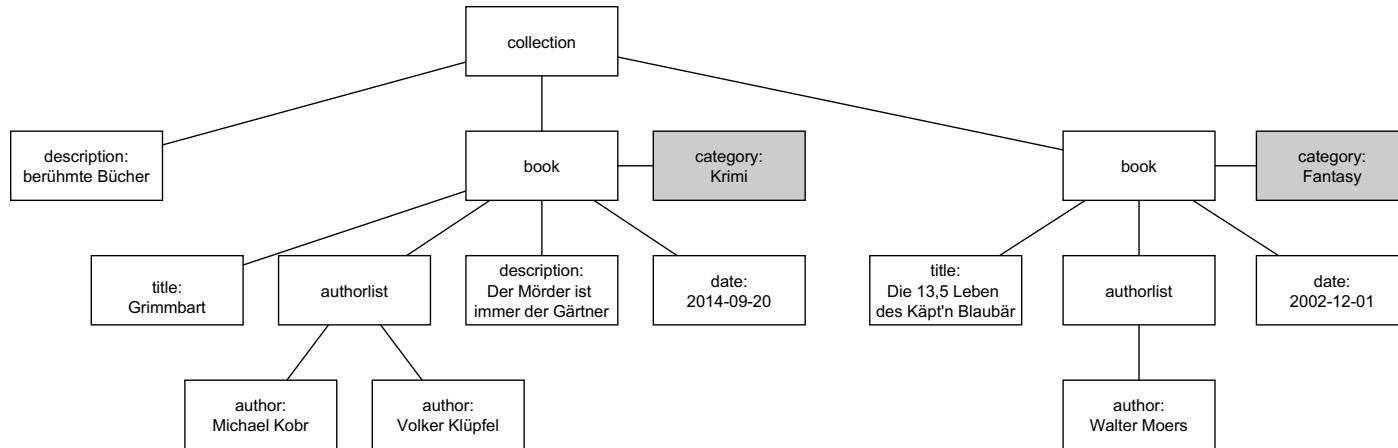
Baumnavigation



Ausgehend von "title:Grimmbart":
Ausgehend von "date:2002-12-01":

../authorlist/author
../authorlist/author

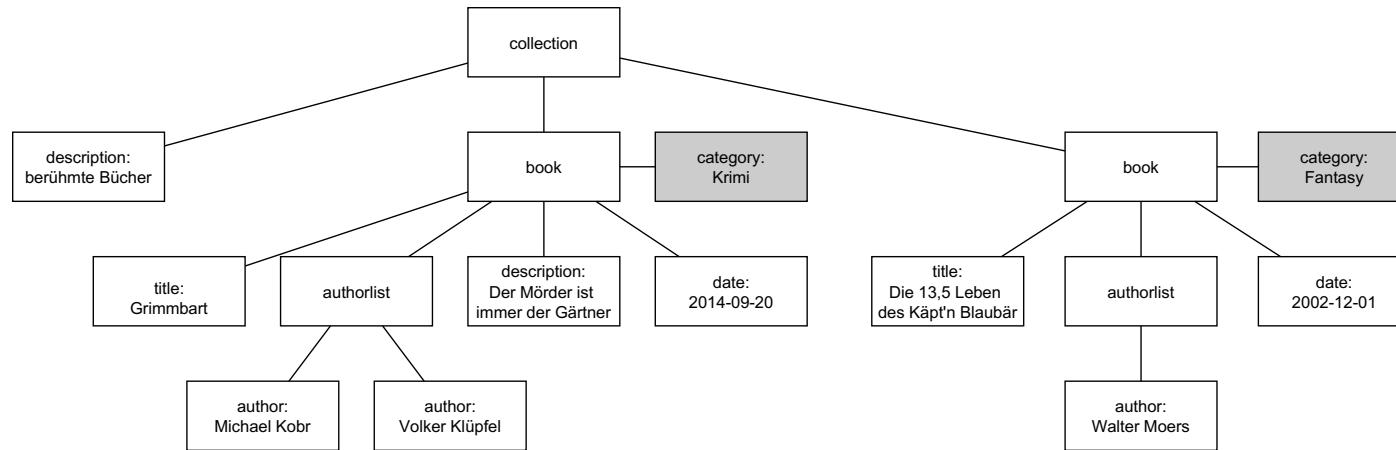
Baumnavigation



Alle Knoten deren "category-Attribut = Krimi" ist

Alle Beschreibungen von Büchern

Baumnavigation



Die Titel aller Bücher, die mindestens zwei Autoren haben

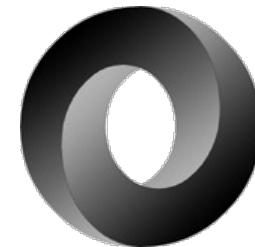
Demo in Java

- XPathExample.java
- UniformCircles.java

Warum überhaupt XML?

- Häufig zu viel Text für zu wenig Inhalt
- Parsen/Formatiertes Schreiben oft nicht so einfach

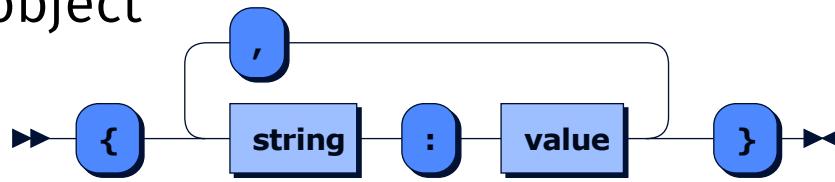
→ JSON (Javascript Object Notation)



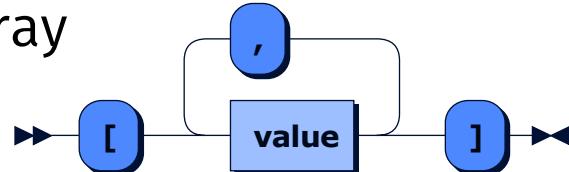
- direkt Javascript Syntax, daher ohne Parser verarbeitbar
- sehr weit verbreitet, gerade für Datenaustausch im Web

JSON

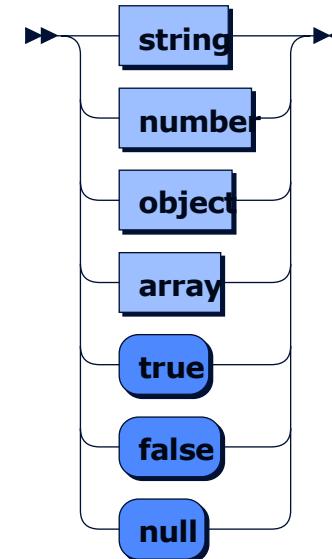
object



array



value



- Kein Text als Kind-Knoten (#PCDATA), nur mit Attribut realisierbar

JSON vs. XML

Google Maps: Höhenanfragen

```
{  
  "results": [  
    {  
      "elevation": 1608.637939453125,  
      "location": {  
        "lat": 39.7391536,  
        "lng": -104.9847034  
      },  
      "resolution": 4.771975994110107  
    },  
    {  
      "elevation": -50.78903579711914,  
      "location": {  
        "lat": 36.455556,  
        "lng": -116.866667  
      },  
      "resolution": 19.08790397644043  
    }  
  "status": "OK"  
}
```

```
<ElevationResponse>  
  <status>OK</status>  
  <result>  
    <location>  
      <lat>39.7391536</lat>  

```

https://maps.googleapis.com/maps/api/elevation/json?locations=39.7391536%2C-104.9847034%7C36.455556%2C-116.866667&key=YOUR_API_KEY
<https://developers.google.com/maps/documentation/elevation/requests-elevation?hl=de>

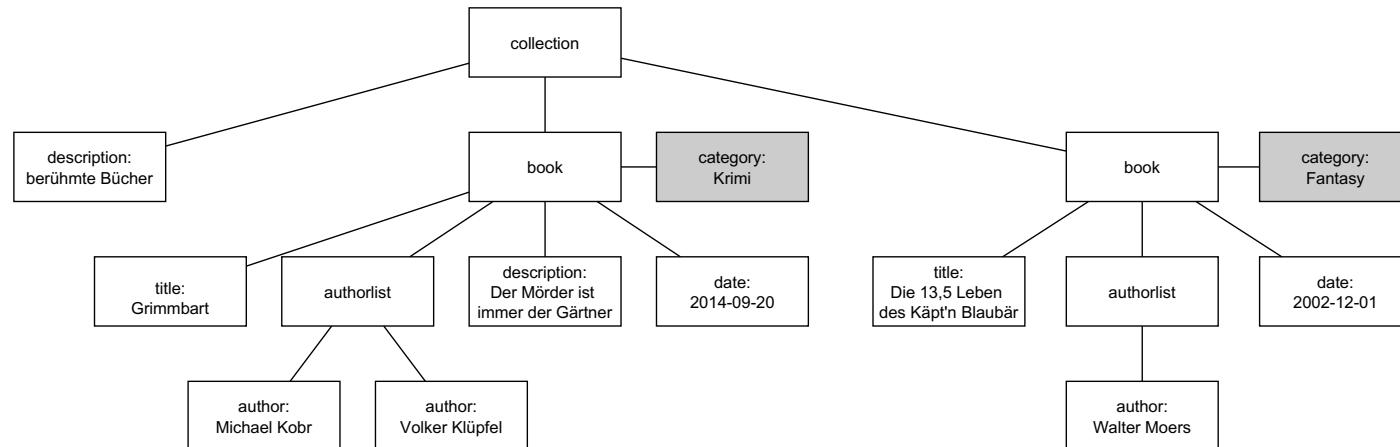
JSONPath

XPath		JSONPath	
Symbol	Bedeutung	Symbol	Bedeutung
/	root-Node	\$	root-Node
<node-name>	alle Knoten mit diesem Namen	.node-name	alle Knoten mit diesem Namen
.	aktueller Knoten	@	aktueller Knoten
..	parent-Node	n/a	parent-Node
//	alle (Sub-)Knoten ab dem aktuellen Knoten	..	alle (Sub-)Knoten ab dem aktuellen Knoten
@	Selektion von Attributen	n/a	Selektion von Attributen

JSONPath

Die Titel aller Bücher, die mindestens zwei Autoren haben:

XPath: `//authorlist[count(author) > 1]/../title` **JSONPath:** `$.collection.book[?(@.authorlist.length > 1)].title`



XSD for JSON?

```
{  
    "$schema": "http://json-schema.org/draft-07/schema#",  
    "$id": "http://example.com/bookcollection.json",  
    "definitions": {  
        "book": {  
            "type": "object",  
            "properties": {  
                "title": {  
                    "type": "string",  
                    "description": "The title of the book"  
                },  
                "author": {  
                    "type": "string",  
                    "description": "The author of the book"  
                },  
                "isbn": {  
                    "type": "string",  
                    "description": "The ISBN of the book"  
                }  
            },  
            "required": ["title", "author", "isbn"]  
        }  
    },  
    "targetNamespace": "bookcollection",  
    "title": "Book Collection",  
    "type": "object",  
    "properties": {  
        "collection": {  
            "type": "array",  
            "items": {  
                "type": "object",  
                "properties": {  
                    "book": {  
                        "type": "array",  
                        "items": {  
                            "type": "object",  
                            "properties": {  
                                "title": {  
                                    "type": "string",  
                                    "description": "The title of the book"  
                                },  
                                "author": {  
                                    "type": "string",  
                                    "description": "The author of the book"  
                                },  
                                "isbn": {  
                                    "type": "string",  
                                    "description": "The ISBN of the book"  
                                }  
                            },  
                            "required": ["title", "author", "isbn"]  
                        }  
                    }  
                },  
                "required": ["book"]  
            }  
        }  
    }  
}
```

Lesbarkeit?

```
<planet name="Arrakis" climate="Desert">
  <notableInhabitants>
    <inhabitant>
      <name>Paul Atreides</name>
      <role>Protagonist</role>
    </inhabitant>
    <inhabitant>
      <name>Baron Harkonnen</name>
      <role>Antagonist</role>
    </inhabitant>
  </notableInhabitants>
</planet>
```

```
{
  "planet": {
    "name": "Arrakis",
    "climate": "Desert",
    "notableInhabitants": [
      {
        "inhabitant": {
          "name": "Paul Atreides",
          "role": "Protagonist"
        }
      },
      {
        "inhabitant": {
          "name": "Baron Harkonnen",
          "role": "Antagonist"
        }
      }
    ]
  }
}
```

YAML

- YAML (Yet Another Markup Language)
- YAML ist eine human-readable data serialization language

```
planet:  
  name: Arrakis  
  climate: Desert  
  notableInhabitants:  
    - inhabitant:  
        name: Paul Atreides  
        role: Protagonist  
    - inhabitant:  
        name: Baron Harkonnen  
        role: Antagonist
```

```
<planet name="Arrakis" climate="Desert">  
  <notableInhabitants>  
    <inhabitant>  
      <name>Paul Atreides</name>  
      <role>Protagonist</role>  
    </inhabitant>  
    <inhabitant>  
      <name>Baron Harkonnen</name>  
      <role>Antagonist</role>  
    </inhabitant>  
  </notableInhabitants>  
</planet>
```

OpenAPI

- Definition von Schnittstellen zwischen Webdiensten
 - Fokus hier auf der Typ-Schemadefinition der JSON-Objekte
-
- Mehr in Vorlesung: Web Engineering

OpenAPI

YML-Format

```
type: object
title: ElevationResponse
required:
  - status
  - results
properties:
  error_message:
    description: | When [...]
    type: string
    example: "Invalid request. Invalid 'locations' parameter."
  status:
    $ref: "./ElevationStatus.yml"
  results:
    type: array
    items: $ref: "./ElevationResult.yml"
```

<https://github.com/googlemaps/openapi-specification/blob/main/specification/schemas/ElevationResponse.yml>

OpenAPI

YML-Format

```
type: object
title: ElevationResult
required:
  - elevation
  - location
properties:
  elevation:
    description: The elevation of the location in meters.
    type: number
  resolution:
    description: The [...]
    type: number
  location:
    description: A [...]
    $ref: "./LatLngLiteral.yml"
```

<https://github.com/googlemaps/openapi-specification/blob/main/specification/schemas/ElevationResult.yml>

OpenAPI

YML-Format

```
type: object
title: LatLngLiteral
description: An object describing a specific location with Latitude and Longitude in decimal degrees.
required:
  - lat
  - lng
properties:
  lat:
    type: number
    description: Latitude in decimal degrees
  lng:
    type: number
    description: Longitude in decimal degrees
```

<https://github.com/googlemaps/openapi-specification/blob/main/specification/schemas/LatLngLiteral.yml>

OpenAPI

Schema Object Specification

- Spezifikation von Input- und Outputdaten
- Obermenge von JSON Schema
- Typen:
 - boolean, number, string
 - object, array
 - null
- Children Array Sub-Schemata: u.a. Items (item)
- Children Object Sub-Schemata: u.a. Attribute (properties)
- Kombination von Schemata:
 - anyOf, allOf, oneOf, not
 - In Abhängig von Existenz eines Attributes: dependentSchemas
- Noch viel mehr...

<https://spec.openapis.org/oas/latest.html#schema-object>

Lernziele

- Parsing XML
 - DOM
 - Sax
- XPath
- JSON
- YAML