Skriptsprachen
* Ruby *

10 - Ruby und XML

Naumann
Sommersemester 2016

10.1 GRUNDLAGEN

Da XML-Dokumente als einfache Textdateien (Standard: utf-8-Kodierung) gespeichert werden, können sie grundsätzlich von Ruby direkt manipuliert werden. Allerdings ist es dann je nach intendierte Aufgabe bzw. Anwendung erforderlich, Teile der Funktionalität nachzubilden, die ein XML-Parser sonst zur Verfügung stellt (z.B. Lesen/Ändern von Knoten des Dokumentbaums etc.).

Nahe liegender ist es daher, einen der für Ruby frei verfügbaren XML-Parser einzubinden, der die geforderte Funktionalität direkt zur Verfügung stellt. Nokogiri (http://nokogiri.org/) ist heute einer der bekanntesten und am weitesten verbreiteten XML-Parser für Ruby.

Nokogiri (鋸) is an HTML, XML, SAX, and Reader parser. Among Nokogiri's many features is the ability to search documents via XPath or CSS3 selectors.

FEATURES:

- XPath support for document searching
- CSS3 selector support for document searching
- XML/HTML builder

10.1 GRUNDLAGEN

Installation von Nokogiri

Nokogiri muss kompiliert und dynamisch gelinkt werden mit den Bibliotheken libxml2 und libxslt. Je nach verwendetem System (Linux / Mac OS X / Windows) müssen sie zunächst vorher installiert werden. Nokogiri kann dann einfach als gem installiert werden:

```
gem install nokogiri
```

bzw.

```
sudo gem install nokogiri
```

In Ruby genügt ein

```
require ‘nokogiri‘
```

um die von dem Paket zur Verfügung gestellten Methoden innerhalb eines Ruby-Programms nutzen zu können.

Wenn neben einfachen Dateien auch Webseiten verarbeitet werden sollen, ist es erforderlich, auch das open-uri gem zu installieren:

```
gem install open-uri
```

10.1 GRUNDLAGEN

Wir verwenden für die Beispiele in diesem Abschnitt unter anderem eine XML-annotierte Korpusdatei, die folgende Struktur aufweist (RELAX NG-Schema kompakt):

```
start =
  element corpus {
    element article {
      element header {
        element copyright { text },
        element identifier { text },
        element page { xsd:integer },
        element date { xsd:NMTOKEN },
        element length { text },
        element texttype { xsd:NCName },
        element author { text }
      },
      element body {
        element headings {
          element title { \token+ },
          element subtitle { clause+ }
        },
        element text {
          (clause
           | \token
           | element iwp { \token+ })+
        }
      }
    }
  }
```

```
\token =
  element token {
    attribute lemma { text },
    attribute type { xsd:NCName }?,
    attribute wclass { text },
    text
  }
clause =
  element clause {
    attribute complete { text },
    (clause | \token)*
  }
```

10.1 GRUNDLAGEN

Hier ein kurzer Auszug aus der Datei *taz.xml*:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<corpus>
 <article>
  <header>
   <copyright> … </copyright>
   …
   …
  </header>
  <body>
   <headings>
    <title> … </title>
    <subtitle> … </subtitle>
   </headings>
   <text>
    <clause complete="+">
        <token wclass="ART" lemma="d">Die</token>
        <token wclass="NN" lemma="Gruppe">Gruppe</token>
        <token wclass="NE" lemma="Underworld">Underworld</token>
        <token wclass="VAFIN" lemma="sein">ist</token>
        <token wclass="ART" lemma="ein">ein</token>
        <token wclass="NN" lemma="Betriebsunfall">Betriebsunfall</token>
        <token wclass="$," lemma=",">,</token>
        …
     </clause>
    …
   </text>
  </body>
 </article>
</corpus>
```

10.2 Parsen von Dokumenten

Nokogiri stellt Parser zur Verarbeitung von HTML- bzw. XML-Dokumenten bereit. Sie erzeugen ein Dokumentobjekt, dass dann weiterverarbeitet werden kann.

… aus einen String:

```
html_doc = Nokogiri::HTML("<html><body><h1>BMW Fan Club</h1></body></html>")
xml_doc  = Nokogiri::XML("<root><alien><name>Alf</name></alien></root>")
```

… aus einer Datei:

```
f = File.open("mordgedanken.xml")
  doc = Nokogiri::XML(f)
f.close
```

oder einfacher:

```
doc = Nokogiri::XML(File.open(„mordgedanken.xml“))
```

… aus einer Webseite:

```
require 'open-uri'
doc = Nokogiri::HTML(open("http://www.spiegel.de/"))
```

10.3 Suche in Dokumenten

Strukturierte Dokumente (HTML- bzw- XML-Dokumente) können mit Hilfe von XPATH- oder CSS-Ausdrücken durchsucht werden. Generell gilt:

- CSS-Ausdrücke (*dokument*.***css***(…)) sind tendentiell transparenter.
- XPATH-Ausdrücke (*dokument*.***xpath***(…)) sind leistungsfähiger; d.h. nicht alle Abfragen lassen sich mit CSS-Ausdrücken realisieren.

10.3 Suche in Dokumenten

```
[shows.xml]
<root>
  <sitcoms>
    <sitcom>
      <name>Married with Children</name>
      <characters>
        <character>Al Bundy</character>
        <character>Bud Bundy</character>
        <character>Marcy Darcy</character>
      </characters>
    </sitcom>
    <sitcom>
      <name>Perfect Strangers</name>
      <characters>
        <character>Larry Appleton</character>
        <character>Balki Bartokomous</character>
      </characters>
    </sitcom>
  </sitcoms>
  <dramas>
    <drama>
      <name>The A-Team</name>
      <characters>
        <character>John "Hannibal" Smith</character>
        <character>Templeton "Face" Peck</character>
        <character>"B.A." Baracus</character>
        <character>"Howling Mad" Murdock</character>
      </characters>
    </drama>
  </dramas>
</root>
```

kumente) können mit Hilfe von XPATH-

herell gilt:

tendentiell transparenter.

sind leistungsfähiger; d.h. nicht alle

n realisieren.

10.3 Suche in Dokumenten

[shows.xml]

```
<root>
  <sitcoms>
    <sitcom>
      <name>Married with Children</name>
      <characters>
        <character>Al Bundy</character>
        <character>Bud Bundy</character
        <character>Marcy Darcy</charact
      </characters>
    </sitcom>
    <sitcom>
      <name>Perfect Strangers</name>
      <characters>
        <character>Larry Appleton</char
        <character>Balki Bartokomous</c
      </characters>
    </sitcom>
  </sitcoms>
  <dramas>
    <drama>
      <name>The A-Team</name>
      <characters>
        <character>John "Hannibal" Smit
        <character>Templeton "Face" Peck</character>
        <character>"B.A." Baracus</character>
        <character>"Howling Mad" Murdock</character>
      </characters>
    </drama>
  </dramas>
</root>
```

okumente) können mit Hilfe von XPATH-

herell gilt:

tendentiell transparenter.

sind leistungsfähiger; d.h. nicht alle

realisieren

```
@doc = Nokogiri::XML(File.open("shows.xml"))

characters = @doc.xpath("//character")
# => ["<character>Al Bundy</character>",
#     "<character>Bud Bundy</character>",
#     "<character>Marcy Darcy</character>",
#     "<character>Larry Appleton</character>",
#     "<character>Balki Bartokomous</character>",
#     "<character>John "Hannibal" Smith</character>",
#     "<character>Templeton "Face" Peck</character>",
#     "<character>"B.A." Baracus</character>",
#     "<character>"Howling Mad" Murdock</character>"]
```

10.3 Suche in Dokumenten

Die Methode ***xpath*** liefert, wie das letzte Beispiel gezeigt hat, als Wert eine Knotenmenge, die sich aber in Ruby wie ein Array behandeln lässt:

```
characters[0].to_s # => "<character>Al Bundy</character>"
```

Oft sind CSS-Ausdrücke transparenter als XPATH-Ausdrücke, wie das folgende Beispiel verdeutlicht:

```
@doc.xpath("//dramas//character")
# => ["<character>John "Hannibal" Smith</character>",
#     "<character>Templeton "Face" Peck</character>",
#     "<character>"B.A." Baracus</character>",
#     "<character>"Howling Mad" Murdock</character>"]
```

Verwendet man die Methode ***css***, erhält man dasselbe Ergebnis durch den Term "drama character“, der alle character-Elemente findet, die Tochterelement eines drama-Elements sind.

```
plays = @doc.css("sitcoms name")
# => ["<name>Married with Children</name>", "<name>Perfect Strangers</name>"]
```

10.3 Suche in Dokumenten

Das erste bzw. letzte Element einer Knotenmenge kann man durch die Methoden first / last extrahieren.
Wenn es um das erste Element geht, kann man als Abkürzung auch einfach at_xpath bzw. at_css verwenden:

```
@doc.css("dramas name").first     # => "<name>The A-Team</name>"
@doc.at_css("dramas name")        # => "<name>The A-Team</name>“
```

Um auf ein Attribut eines Elementknotens zuzugreifen, gibt es verschiedene Möglichkeiten:

a) … Elementname[@Attributname] oder
b) … Elementname/@Attributname # Wenn für ein Element ein Attribut spezifiziert
   # ist, dann ist der Attributknoten ein
   in CSS-Notation: # Tochterelement des Elementknotens
c) … Elementname/Attributname

Natürlich kann für das Attribut auch ein/der Wert spezifiziert werden (= Wert).
Die Methode *text* liefert den Inhalt des Textknotens:

```
puts doc.xpath("//token[@wclass='VVFIN']").collect {|node| node.text}   # liefert alle
puts doc.css("token[wclass='VVFIN']").collect {|node| node.text}         # finiten Verben
```

10.3 Suche in Dokumenten

Weitere wichtige Methoden:

a) Methoden zur Navigation:
   parent, children, next_sibling, previous_sibling
b) Methoden für Attribute:
   keys, *values*

```
puts doc.xpath("//token")[1].keys        # => Attributnamen des 1. token-Elements
puts doc.xpath("//token")[1].values      # => Attributwerte des 1. token-Elements
```

c) Andere Methoden:

```
delete, remove, remove_attribute # Löschen eines Knotens bzw. Attributs
    puts doc.xpath("//token[not(contains(@wclass, 'VV'))]")[1].delete('wclass')
    # löscht das class-Attribut samt Wert
contains              # Substring-Suche
    puts doc.xpath("//token[contains(@wclass, 'VV')]")
    # findet alle token-Elemente, deren wclass-Attributwert lVVl enthält.
not                   # logische Negation
    puts doc.xpath("//token[not(contains(@wclass, 'VV'))]")
```

10.3 Suche in Dokumenten

Namensräume

Enthält ein Dokument Knoten, die verschiedenen Namensräumen angehören, dann kann die jeweilige Namensraum-URI bei der Suche angegeben werden, um nur Knoten des spezifizierten Namensraumes zu erhalten:

```
[parts.xml]
<parts>
  <!-- Alice's Auto Parts Store -->
  <inventory xmlns="http://alicesautoparts.com/">
    <tire>all weather</tire>
    <tire>studded</tire>
    <tire>extra wide</tire>
  </inventory>

  <!-- Bob's Bike Shop -->
  <inventory xmlns="http://bobsbikes.com/">
    <tire>street</tire>
    <tire>mountain</tire>
  </inventory>
</parts>
```

10.3 Suche in Dokumenten

Namensräume

Enthält ein Dokument Knoten, die verschiedenen Namensräumen angehören, dann kann die jeweilige Namensraum-URI bei der Suche angegeben werden, um nur Knoten des spezifizierten Namensraumes zu erhalten:

```
[parts.xml]
<parts>
  <!-- Alice's Auto Parts Sto
  <inventory xmlns="http://a
    <tire>all weather</tire>
    <tire>studded</tire>
    <tire>extra wide</tire>
  </inventory>

  <!-- Bob's Bike Shop -->
  <inventory xmlns="http://b
    <tire>street</tire>
    <tire>mountain</tire>
  </inventory>
</parts>
```

```
@doc = Nokogiri::XML(File.read("parts.xml"))

car_tires = @doc.xpath('//car:tire', 'car' => 'http://alicesautoparts.com/')
# => ["<tire>all weather</tire>",
#       "<tire>studded</tire>",
#       "<tire>extra wide</tire>"]

bike_tires = @doc.xpath('//bike:tire', 'bike' => 'http://bobsbikes.com/')
# => ["<tire>street</tire>", "<tire>mountain</tire>"]
```

10.3 Suche in Dokumenten

Namensräume

Wenn ein oder mehrere *Namensräume* im Dokument verwendet wurden, ist ihre Angabe bei der Suche nach Knoten in der Regel obligatorisch:

Die Anfrage @doc.xpath('//tire') liefert z.B als Resultat eine leer Knotenmenge. Ein mit dem Wurzelknoten des Dokument assoziierter Namensraum wird von Nokogiri automatisch erkannt und mit dem in der Deklaration verwendetem Namen assoziiert:

```
<autos xmlns="http://www.autos.de/">
    <auto>
        ...
    </auto>
    <auto>
        ...
    </auto>
      ....
</autos>
```

@autos.xpath('//xmlns:autos') => Menge aller Auto-Knoten

Die Verwendung der Namensraumbezeichner bei Suchanfragen ist in diesen Fällen obligatorisch. Wer möchte, kann durch remove_namespaces alle Namensraumangaben aus einem Dokument entfernen.

10.4 Modifikation von Dokumenten

Modifikation von Dokumenten

a) Auslesen und Ändern des Inhalts eines Knotens

Nokogiri stellt mit content bzw. content= eine Methode zur Verfügung, um den Inhalt einen Knotens auszulesen bzw. zu verändern:

```
@doc = Nokogiri::HTML::DocumentFragment.parse <<-EOHTML
<body>
  <h1>Angriff der Killertomaten</h1>
  <div>Schlechte Filme</div>
</body>
EOHTML

h1 = @doc.at_css "h1"
h1.content = "Cosmopolis"

@doc.to_html
# => "<body>
#   <h1> Cosmopolis</h1>
#   <div>Schlechte Filme</div>
# </body>"
```

10.4 Modifikation von Dokumenten

b) Modifikation der Dokumentstruktur

Die einfachste Möglichkeit, die Struktur von XML- bzw- HTML-Dokumenten zu ändern besteht darin, Knoten andere Mutterknoten zuzuweisen:

```
h1  = @doc.at_css "h1"
div = @doc.at_css "div"
h1.parent = div

@doc.to_html
# => "<body>
#       <div>Schlechte Filme
#             <h1>Cosmopolis</h1>
#       </div>
#     </body>"
```

oder ihre Reihenfolge zu vertauschen:

```
div.add_next_sibling(h1)

@doc.to_html
# => "<body>
#       <div>Schlechte Filme</div>
#       <h1>Cosmopolis</h1>
#     </body>"
```

10.4 Modifikation von Dokumenten

c) Modifikation von Knoten und Attributen

Der Name eines Knotens kann durch die name-Methode geändert werden;
Attribute können angelegt bzw. ihre Werte modifiziert werden, indem sie als Selektor des Knotens verwendet werden:

```
h1.name = 'h2'                # Der Elementknoten ist jetzt ein Knoten vom Typ /h2/
h1['class'] = 'zeige-titel'   # Gab es bisher keinen entsprechenden Attributknoten, wird er
                              # mit dem spezifizierten Wert erzeugt; anderenfalls sein Wert
                              # aktualisiert.
@doc.to_html
# => "<body>
#   <h2 class="zeige-titel">Cosmopolis</h2>
#   <div>Schlechte Filme</div>
# </body>"
```

d) Erzeugung neuer Knoten

Wie aus Ruby vertraut, können Instanzen einer Klasse durch new erzeugt werden.
Dokumentknoten erzeugt man entsprechend durch Nokogiri::HTML::Node.new bzw. Nokogiri::XML::Node.new:

10.4 Modifikation von Dokumenten

```
h3 = Nokogiri::XML::Node.new "h3", @doc
h3.content = "2012"
h1.add_next_sibling(h3)

@doc.to_html
# => "<body>
#   <h1>Cosmopolis</h1>
# <h3>2012</h3>
#   <div>Schlechte Filme</div>
# </body>"
```

e) Einbettung aller Knoten einer Knotenmenge

```
nodes = @doc.css "h1,div"
wrapper = nodes.wrap("<div class='container'></div>")

@doc.to_html
# => "<body>
#   <div class="container"><h1>Cosmopolis</h1></div>
#   <div class="container"><div>Schlechte Filme</div></div>
# </body>"
```

10.5 Wohlgeformte Dokumente

Nokogiri ist (im begrenzten Umfang) in der Lage, nicht-wohlgeformte Dokumente zu korrigieren und in xml-konforme Dokumente zu verwandeln:

```
nicht_gut = <<-EOXML
<root>
  <open>foo
    <closed>bar</closed>
</root>
EOXML

dokument  = Nokogiri::XML nicht_gut

puts dokument        # => <?xml version="1.0"?>
                     #    <root>
                     #      <open>foo
                     #        <closed>bar</closed>
                     #    </open>
                     #    </root>
```

10.6 Erzeugung neuer Dokumente

Wenn man nicht bestehende Dokumente modifixzieren, sondern vollkommen neue Dokumente anlegen will, kann man den HTML- bzw. XML-Builder (Nokogiri::(XIHT)ML::Builder) verwenden. Der Builder nimmt entweder einen parameterlosen Block oder einen Block mit einem Parameter (erlaubt den Zugriff auf Umgebungsvariablen) als Argument :

```
builder = Nokogiri::XML::Builder.new do
root {
  products {
    widget {
      id_ "10"
      name "Tolles Widget"
    }
  }
}
end
```

```
builder = Nokogiri::XML::Builder.new do |xml|
xml.root {
    xml.products {
        xml.widget {
            xml.id_ "10"
            xml.name "Tolles Widget"
        }
    }
}
end
```

10.6 Erzeugung neuer Dokumente

Wenn man nicht bestehende Dokumente modifixzieren, sondern vollkommen neue Dokumente anlegen will, kann man den HTML- bzw. XML-Builder (Nokogiri::(XIHT)ML::Builder) verwenden. Der Builder nimmt entweder einen parameterlosen Block oder einen Block mit einem Parameter (erlaubt den Zugriff auf Umgebungsvariablen) als Argument :

```
builder = Nokogiri::XML::Builder.new do
root {
  products {
    widget {
      id_ "10"
      name "Tolles W
    }
  }
}
end
```

```
builder = Nokogiri::XML::Builder.new do |xml|
xml.root {
  xml.products {
    xml.widget {
      xml.id_ "10"
      xml.name "Tolles Widget"
    }
  }
}
end
```

```
builder.to_xml
<?xml version="1.0"?>
<root>
<products>
   <widget>
      <id>10</id>
      <name>Tolles Widget</name>
   </widget>
</products>
</root>
```