XML in R

Ein kurzes Tutorial

Sascha Wolfer

10. August 2013

v0.1

Das Hauptpaket für die XML-Unterstützung in R heißt `XML`. `RCurl` hat außerdem nette Zusatz-Funktionen, die wir hier nicht wirklich benötigen. Wir werden lediglich die Funktion `getURL()` aus dem `RCurl`-Paket verwenden.

```
# install.packages(c('XML', 'RCurl')
library(XML)
library(RCurl)  # für getURL()

## Loading required package: bitops
```

Der Ablauf beim Verarbeiten von XML-Files ist immer der gleiche:

1. Einlesen
2. Parsen
3. Knoten selektieren
4. Rückgabe wählen und verarbeiten

Die Schritte (1) und (2) können dabei unter Umständen zusammenfallen. Beginnen wir damit, ein sehr einfaches XML-File direkt aus dem Internet zu parsen[1].

```
# Vorbereiten der URL
url <- getURL("http://www.w3schools.com/xml/simple.xml")
# Parsen
parsed.url <- xmlTreeParse(file = url, error = function(...) {
}, useInternalNodes = T)
# Nur für ein lokal abgespeichertes File
parsed.url <- xmlTreeParse(file = "/Users/sascha/Documents/R/XMLinR/simple.xml",
    error = function(...) {
    }, useInternalNodes = T)
```

`parsed.url` enthält jetzt eine Repräsentation des XML-Dokuments `simple.xml`, das eine kurze Speisekarte mit fünf Gerichten darstellt. Die ganze Karte umgibt der XML-Tag `breakfast_menu`, jedes Gericht ist umgeben vom `food`-Tag. Dieser enthält dann jeweils vier Tags: `name`, `price`, `description` und `calories`. Die Namen dieser Tags sind wohl selbsterklärend. Sie schließen die jeweilige Information zu jedem der fünf Gerichte ein. Das Dokument sollte man sich einmal auf der Konsole anschauen, indem man einfach `parsed.url` eingibt. Das wird hier wegen der Menge des Outputs nicht gezeigt. Schritte 1 und 2 (Einlesen und Parsen) sind schon erledigt. Nun werden wir relevante Knoten selektieren. Um für uns relevante Knoten zu finden, benötigen wir die sogenannte XPath-Syntax[2]. Als erstes wollen wir nun alle Namen der Gerichte extrahieren. Wir wollen also alle `name`-Tags aus dem Baum extrahieren. Die Funktion `xpathSApply()` wendet eine Rückgabe-Funktion auf alle Knoten an, die durch einen XPath-Ausdruck selektiert werden.

```
xpathSApply(doc = parsed.url, path = "//name", xmlValue)

## [1] "Belgian Waffles"             "Strawberry Belgian Waffles"
## [3] "Berry-Berry Belgian Waffles" "French Toast"
## [5] "Homestyle Breakfast"
```

Wir bekommen genau die Rückgabe, die wir wollten. Der XPath-Ausdruck bedeutet: „Selektiere alle `name`-Knoten, egal auf welcher Einbettungstiefe sie sich befinden.“. Die Rückgabefunktion

[1] Sollte dieses File nicht mehr existieren, funktioniert das natürlich nicht. Bitte in diesem Fall eine E-Mail an mich (sascha@wolferonline.de) schicken.

[2] Unter `http://www.w3schools.com/xpath/xpath_syntax.asp` gibt es eine gute Übersicht über die XPath-Syntax.

`xmlValue` sorgt dafür, dass alle Werte zurückgegeben werden, die jeweils zwischen dem öffnenden und dem schließenden Tag stehen. Wir können natürlich auch explizit formulieren, dass wir alle `name`-Knoten unter `food`-Knoten selektieren wollen. Die Rückgabe ist dann gleich.

```
xpathSApply(doc = parsed.url, path = "//food/name", xmlValue)

## [1] "Belgian Waffles"             "Strawberry Belgian Waffles"
## [3] "Berry-Berry Belgian Waffles" "French Toast"
## [5] "Homestyle Breakfast"
```

Selektieren wir einen Knoten, der selbst nur andere Knoten als Wert enthält (also keinen Text, wie das beim eben selektierten `name`-Knoten noch der Fall war), wird der Text, der zwischen den untergeordneten Knoten steht, hintereinander geschrieben und zurückgegeben. Beachtenswert am nächsten Aufruf ist außerdem die absolute Pfadangabe, die bei der Wurzel („root“) des XML-Baums ansetzt, indem nur ein Slash zu Beginn des Pfads steht. Da jeder `food`-Knoten unter dem „Hauptknoten“ `breakfast_menu` hängt, muss `breakfast_menu` noch angegeben werden.

```
xpathSApply(doc = parsed.url, path = "/breakfast_menu/food", xmlValue)
# <Die Ergebnisse sind hier nicht ausgegeben.>
```

Bleiben wir doch einmal bei den `food`-Knoten. Diese haben selbst keinen Text als Wert, aber es wird – wie wir eben gesehen haben – der Text der untergeordneten Knoten ausgegeben. Nun ändern wir die Funktion, die hinter dem XPath übergeben wird. Diese steuert, welche Eigenschaften aus jedem Knoten zurückgegeben werden. Wir haben bisher immer die Funktion `xmlValue` verwendet. Wenn wir wissen möchten, welche Knoten unter dem selektierten Knoten hängen, benutzen wir die Funktion `xmlChildren`. Knoten, die unter einem bestimmten Knoten hängen, nennt man auch „Kinderknoten“, daher der Name der Funktion.

```
xpathSApply(doc = parsed.url, path = "/breakfast_menu/food", xmlChildren)

##             [,1] [,2] [,3] [,4] [,5]
## name        ?    ?    ?    ?    ?
## price       ?    ?    ?    ?    ?
## description ?    ?    ?    ?    ?
## calories    ?    ?    ?    ?    ?
```

Wir sehen nun, dass die `food`-Knoten die Kinderknoten `name`, `price`, `description` und `calories` haben. Wollen wir nun den Preis des vierten Gerichts wissen, können wir ganz einfach die Rückgabe des obigen Aufrufs indizieren. Ich indiziere im folgenden Beispiel mit dem Zeilennamen `price`, natürlich kann dieser auch einfach durch die Zeilennummer 2 ersetzt werden. Das Ergebnis ist dann dasselbe.

```
xpathSApply(doc=parsed.url, path="/breakfast_menu/food",
            xmlChildren)["price", 4]

## $price
## <price>$4.50</price>
```

Das nächste Ziel wird es sein, den Betrag als numerischen Wert zu extrahieren, damit wir damit besser in R umgehen können. Die Rückgabe des obigen Aufrufs war eine Liste mit einem Element, das `price` heißt. Inhalt dieses Elements ist ein XML-Knoten. Wert dieses Knotens ist der Preis mit einem vorangestellten Dollarzeichen. Mit den folgenden Aufrufen kann der Preis als numerischer Wert behandelt werden. Ich stelle die Lösung Schritt für Schritt vor, dann in einer Zeile.

```
price.list <- xpathSApply(doc=parsed.url, path="/breakfast_menu/food",
                          xmlChildren)["price", 4]
# Erstes (und einziges) Element aus Ergebnisliste extrahieren
price.node <- price.list[[1]]
# Wert des Knotens extrahieren
price.val <- xmlValue(price.node)
# Dollar-Zeichen entfernen
price.text <- gsub("$", "", price.val, fixed = T)
# In numerische Variable umwandeln
price <- as.numeric(price.text)
price
```

```
## [1] 4.5
```

```
# In einer Zeile (aber unübersichtlich)
as.numeric(gsub("$", "",
                xmlValue(xpathSApply(doc=parsed.url,
                                     path="/breakfast_menu/food",
                                     xmlChildren)["price", 4][[1]]), fixed = T))
```

```
## [1] 4.5
```

Wir haben nun also alle oben genannten Schritte durchgeführt: Bereits vorher haben wir die Datei `simple.xml` eingelesen und geparst. Wir haben dann alle `food`-Knoten selektiert und uns von diesen die Kinderknoten ausgeben lassen. Aus diesen Kinderknoten haben wir den Wert für den `price`-Knoten des vierten Gerichts extrahiert. Aus diesem Wert haben wir das Dollar-Zeichen entfernt und ihn dann in eine numerische Variable umgewandelt.

Grundlegende Datenstrukturen zur Repräsentation von Datentabellen in R sind Dataframes. Es wäre daher praktisch, wenn wir die XML-Datei in einem Dataframe darstellen könnten. In einer Spalte sollen dabei die Namen der Gerichte stehen, in der nächsten Spalte die Preise und in der letzten Spalte die Kalorien – den Beschreibungseintrag `description`, der ebenfalls in der XML-Datei zu finden ist (Tag `description`), lassen wir einmal unter den Tisch fallen. Das ist recht einfach zu bewerkstelligen mit den Mitteln, die wir schon kennengelernt haben.

```
names <- xpathSApply(parsed.url, "//name", xmlValue)
prices <- xpathSApply(parsed.url, "//price", xmlValue)
cals <- as.numeric(xpathSApply(parsed.url, "//calories", xmlValue))
# Umwandeln der Preise in numerische Werte
prices <- sapply(prices, FUN = function (price) {
  as.numeric(gsub("$", "", price, fixed = T))
})
food.df <- data.frame(name = names, price = prices, calories = cals,
                      row.names = 1:length(names))
food.df
```

```
##                          name price calories
## 1             Belgian Waffles  5.95      650
## 2  Strawberry Belgian Waffles  7.95      900
## 3 Berry-Berry Belgian Waffles  8.95      900
## 4                French Toast  4.50      600
## 5         Homestyle Breakfast  7.50      950
```

Mit diesem Dataframe können nun alle entsprechenden Operationen durchgeführt werden, die wir aus R kennen. Wir können beispielsweise den Mittelwert des Preises berechnen oder die Korrelation zwischen Preis und Kalorien.

```
mean(food.df$price)

## [1] 6.97

cor.test(food.df$price, food.df$calories)

##
##  Pearson's product-moment correlation
##
## data:  food.df$price and food.df$calories
## t = 3.568, df = 3, p-value = 0.03759
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.08416 0.99341
## sample estimates:
##    cor
## 0.8996
```

Die XPath-Syntax ist mächtig. So können wir beispielsweise Selektionsbeschränkungen für Knoten einführen, die sich auf Attribute dieser Knoten beziehen. Ein Attribut ist eine Eigenschaft eines Knotens. Jede Information, die zwischen den öffnenden (`<name>`) und den schließenden (`/<name>`) Tag eines Knotens geschrieben werden kann, kann grundsätzlich in Attribute dieses Knotens geschrieben werden. Keiner der Knoten in `simple.xml`, unserer bisher verwendeten Beispieldatei, hat ein Attribut. Wir benötigen also eine andere Beispieldatei, um dies zu demonstrieren. Wir werden eine Sample-Datei verwenden, die ein Auszug einer Datenbank eines Buchladens sein könnte. Wir werden zunächst wieder die Schritte 1 (Einlesen) und 2 (Parsen) für diese Datei durchführen[3].

```
url <- getURL("http://www.wolferonline.de/xmlR/books2.xml")
parsed.url <- xmlTreeParse(url, error = function (...) {}, useInternalNodes = T)
```

In der XML-Datei sind Informationen zu sechs fiktiven Büchern gespeichert. Für jedes Buch sind Titel- und Autor-Informationen in Knoten gespeichert. Außerdem haben die `book`-Knoten je zwei Attribute: `genre` und `pages`. Schauen wir uns zunächst die Titel aller Bücher an. Die zweite Ausgabe zeigt uns die Autoren-Knoten, unter denen jeweils zwei Knoten hängen. Die Werte dieser Knoten werden einfach aneinandergehängt und ausgegeben.

```
# Alle Titel
xpathSApply(parsed.url, "//book/title", xmlValue)

## [1] "Aliens from outer space"
## [2] "The man in the cupboard"
## [3] "The adventures of a hard-boiled detective"
## [4] "A long story about other planets"
## [5] "Space Quarrels XXII"
## [6] "Startled people run away"

# Alle 'values' unter Autoren-Knoten
xpathSApply(parsed.url, "//book/author", xmlValue)

## [1] "JohnDoe"          "JaneDoe"          "RobertChandelier"
## [4] "CeciliaWritemuch" "LucasGeorge"      "TeresaTugboat"
```

[3]Es bietet sich an, die Datei außerdem im Internetbrowser anzuschauen. Dazu muss einfach die URL der Datei in die Adresszeile eines Browsers kopiert werden. Die meisten Browser unterstützen die Anzeige von XML-Dateien (getestet auf Chrome und Safari).

Nun lassen wir uns die Attribute der `book`-Knoten ausgeben. Wir sehen für jedes der sechs Bücher das zugeordnete Genre sowie die Anzahl der Seiten, die das Buch umfasst.

```
xpathSApply(parsed.url, "//book", xmlAttrs)

##       [,1]    [,2]       [,3]    [,4]    [,5]    [,6]
## genre "scifi" "thriller" "crime" "scifi" "scifi" "thriller"
## pages "302"   "578"      "680"   "1899"  "240"   "452"
```

Über die Anpassung des XPaths können wir nun nur jene Buchknoten selektieren, die dem Genre `scifi` angehören. Diese Selektionsbeschränkung wird hinter den `book`-Knoten gehängt, denn zu diesem Knoten gehört das Attribut `genre`. Zwei eckige Klammern führen eine Selektionsbeschränkung ein. Hinter einem at-Zeichen folgt dann der Name des Attributs. Nach einem *einfaches* Gleichheitszeichen[4] wird dann der Wert definiert, der selektiert werden soll. Von den selektierten Büchern wollen wir die Titel ausgeben. Der XPath wird also einfach nach der Selektionsbeschränkung fortgesetzt.

```
xpathSApply(parsed.url, "//book[@genre = 'scifi']/title", xmlValue)

## [1] "Aliens from outer space"          "A long story about other planets"
## [3] "Space Quarrels XXII"
```

Wir sehen: Statt sechs Buchtiteln werden nur noch drei ausgegeben. Es sind diejenigen, an deren übergeordneten `book`-Knoten das Attribut `genre` gleich `scifi` ist. Lässt man das at-Zeichen weg, werden übrigens Knoten unter dem aktuellen Knoten überprüft. Der nächste XPath ist relativ komplex. Er bedeutet natürlichsprachlich: „Selektiere alle `title`-Knoten, die unter `book`-Knoten hängen, für die gilt: Der Knoten `last-name` unter dem `autor`-Knoten muss `Doe` lauten.“ Wir bekommen also alle Titel von Büchern, deren Autoren mit Nachnamen „Doe“ heißen.

```
xpathSApply(parsed.url, "//book[author/last-name = 'Doe']/title", xmlValue)

## [1] "Aliens from outer space" "The man in the cupboard"
```

Als letztes Beispiel soll nun noch das Geschlecht der Autoren ausgegeben werden, für die wir Bücher mit mehr als 500 Seiten in unserer Datenbank haben. Das Geschlecht des Autors ist über das Attribut `sex` direkt am `author`-Knoten kodiert. Um an diese Information zu kommen, müssen wir die Selektion von Buchknoten einschränken und auf die Attribute des Autorenknotens zugreifen.

```
longbookauthors.sex <- xpathSApply(parsed.url, "//book[@pages > 500]/author",
    xmlAttrs)
longbookauthors.sex

##      sex      sex      sex
## "female"   "male" "female"

# und direkt tabellieren
table(longbookauthors.sex)

## longbookauthors.sex
## female   male
##      2      1
```

[4] Achtung! In R wird in solchen Fällen typischerweise das doppelte Gleichheitszeichen verwendet. Nicht so in der XPath-Syntax!

Das schließt dieses Kurztutorial zu XML in R ab. Es wurde bisher lediglich ein Bruchteil der Fähigkeiten des XML-Pakets dargestellt. Diese werden gegebenenfalls in weiteren Versionen dieses Tutorials aufgegriffen.