Web Data Collection with R

Session 5: Collecting Data from Static Websites: XPath

Sascha Göbel
sascha.goebel@uni-konstanz.de

December 2nd, 2020

Today: XPath

- identify information in HTML/XML
- download target HTML/XML
- parse the HTML/XML file
- extract the actual information using XPath queries $\Longleftarrow$

XPath

XML path language

- query language for extracting parts from XML/HTML files
- using characteristics and relations of nodes
- a “node” is the entire information encapsulated in tags
- a “node set” is a selection of multiple nodes

Why XPath?

- designed for XML-style files
- intuitive
- flexible
- efficient
- robust

XPath

XML path language

- query language for extracting parts from XML/HTML files
- using characteristics and relations of nodes
- a “node” is the entire information encapsulated in tags
- a “node set” is a selection of multiple nodes

Why XPath?

- designed for XML-style files
- intuitive
- flexible
- efficient
- robust

Queries

XPath query fundamentals

- based on hierarchical order of nodes
- expressed by sequence of nodes leading to the target node
- literally depicting a location path using /
- a character string that reads from left to right
- absolute path:
  - start from root node
  - proceed through all nodes on the way to the target
  - e.g., “/html/body/div/p/i”
- relative path:
  - jump between nodes using “//”
  - e.g., ”//p/i”
  - computationally more expensive

XPath query fundamentals

- wildcard "*" matches any node
- "." selects the current node
- ".." selects the node one level up
- "@" selects attributes
- separate multiple queries by "|" or pass as a vector
- queries have no length limit

How to write a query?

- define where the actual information is located in the DOM
- develop expression from there on, up the tree
- usually a trial-and-error process

XPath query fundamentals

- wildcard "*" matches any node
- "." selects the current node
- ".." selects the node one level up
- "@" selects attributes
- separate multiple queries by "|" or pass as a vector
- queries have no length limit

How to write a query?

- define where the actual information is located in the DOM
- develop expression from there on, up the tree
- usually a trial-and-error process

Using node relations

- we can identify nodes based on their relation to other nodes
- using notation based on family relationships
- describing the relation of node2 to node1
- “node1/relation::node2”

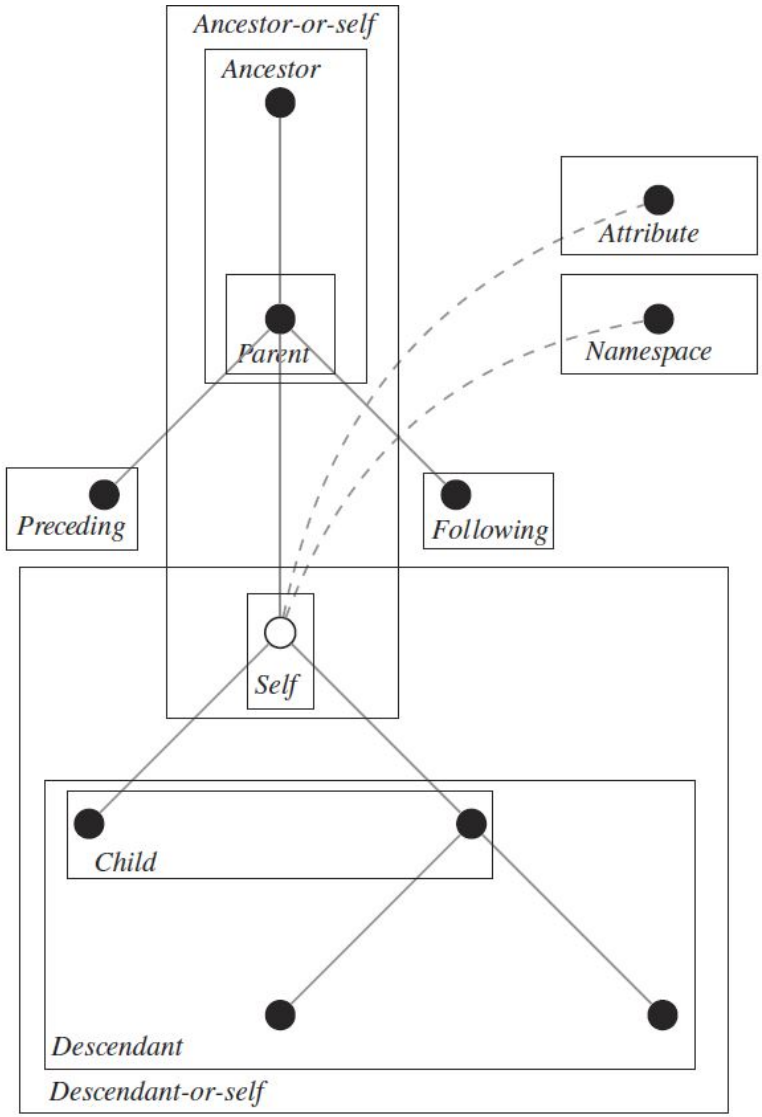

Figure 4.2 Visualizing node relations. Descriptions are presented in relation to the white node

Source: Munzert et al. 2015

Table 4.1 XPath axes

Axis name	Result
ancestor	Selects all ancestors (parent, grandparent, etc.) of the current node
ancestor-or-self	Selects all ancestors (parent, grandparent, etc.) of the current node and the current node itself
attribute	Selects all attributes of the current node
child	Selects all children of the current node
descendant	Selects all descendants (children, grandchildren, etc.) of the current node
descendant-or-self	Selects all descendants (children, grandchildren, etc.) of the current node and the current node itself
following	Selects everything in the document after the closing tag of the current node
following-sibling	Selects all siblings after the current node
namespace	Selects all namespace nodes of the current node
parent	Selects the parent of the current node
preceding	Selects all nodes that appear before the current node in the document except ancestors, attribute nodes, and namespace nodes
preceding-sibling	Selects all siblings before the current node
self	Selects the current node

Source: http://www.w3schools.com/xpath/xpath_axes.asp

Source: Munzert et al. 2015

Using node predicates

- functions applied to node name, value, or attribute
- evaluate whether a condition is true (selected) or false (not selected)
- "node1[predicate]"
- work in conjunction with operators
- when nesting predicates swap "=" with ","

Table 4.2 Overview of some important XPath functions

Function	Description	Example
`name(<node>)`	Returns the name of `<node>` or the first node in a node set	`//*[name()='title']`; Returns: `<title>`
`text(<node>)`	Returns the value of `<node>` or the first node in a node set	`//*[text()='The book homepage']`; Returns: `<i>` with value *The book homepage*
`@attribute`	Returns the value of a node's *attribute* or of the first node in a node set	`//div[@id='R Inventor']`; Returns: `<div>` with attribute *id* value *R Inventor*
`string-length(str1)`	Returns the length of `str1`. If there is no string argument, it returns the length of the string value of the current node	`//h1[string-length()>11]`; Returns: `<h1>` with value *Robert Gentleman*
`translate(str1, str2, str3)`	Converts `str1` by replacing the characters in `str2` with the characters in `str3`	`//div[translate(./@date, '2003', '2005')='June/2005']`; Returns: first `<div>` node with date attribute value *June/2003*
`contains(str1,str2)`	Returns `TRUE` if `str1` contains `str2`, otherwise `FALSE`	`//div[contains(@id, 'Inventor')]`; Returns: first `<div>` node with id attribute value *R Inventor*
`starts-with(str1,str2)`	Returns `TRUE` if `str1` starts with `str2`, otherwise `FALSE`	`//i[starts-with(text(), 'The')]`; Returns: `<i>` with value *The book homepage*
`substring-before (str1,str2)`	Returns the start of `str1` before `str2` occurs in it	`//div[substring-before(@date, '/')='June']`; Returns: `<div>` with date attribute value *June/2003*
`substring-after (str1,str2)`	Returns the remainder of `str1` after `str2` occurs in it	`//div[substring-after(@date, '/')=2003]`; Returns: `<div>` with date attribute value *June/2003*
`not(arg)`	Returns `TRUE` if the boolean value is `FALSE`, and `FALSE` if the boolean value is `TRUE`	`//div[not(contains(@id, 'Inventor'))]`; Returns: the `<div>` node that does not contain the string *Inventor* in its id attribute value
`local-name(<node>)`	Returns the name of the current `<node>` or the first node in a node set—without the namespace prefix	`//*[local-name()='address']`; Returns: `<address>`
`count(<node>)`	Returns the count of a nodeset `<node>`	`//div[count(.//a)=0]`; Result: The second `<div>` with one `<a>` child
`position(<node>)`	Returns the index position of `<node>` that is currently being processed	`//div/p[position()=1]`; Result: The first `<p>` node in each `<div>` node
`last()`	Returns the number of items in the processed node list `<node>`	`//div/p[last()]`; Result: The last `<p>` node in each `<div>` node

Source: Munzert et al. 2015

Table 4.3 XPath operators

Operators	Description	Example
|	Computes two node sets	`//i | //b`
+	Addition	`5 + 3`
-	Subtraction	`8 - 2`
*	Multiplication	`8 * 5`
div	Division	`8 div 5`
=	Equal	`count = 27`
!=	Not equal	`count != 27`
<	Less than	`count < 27`
≤	Less than or equal to	`count <= 27`
>	Greater than	`count > 27`
≥	Greater than or equal to	`count >= 27`
or	Or	`count = 27 or count = 28`
and	And	`count > 26 and count < 30`
mod	Modulo (division remainder)	`7 mod 2`

Source: Adapted from http://www.w3schools.com/xpath/xpath_operators.asp

Source: Munzert et al. 2015

XPath in R

R package “rvest”

- implements XPath 1.0
- no regex support
- workhorse functions:
  - `html_node()` - return first node matching the query
  - `html_nodes()` - return all nodes matching the query
  - `html_text()` - extract the value of a node
  - `html_attr()` - extract the specified attribute of a node
  - `html_table()` - parse a table into a data frame

Helpers

Tools for query construction

- Copy XPath from browser inspection tool
- Selector gadget extension
- Best to learn and code XPath yourself!

Next Session

XPath practice session

- please bring an idea and URL for web data collection
- the website should be static and require XPath, not (yet) API access

  $\longrightarrow$ no Twitter!
- apply and practice what you learned thus far
- chance to solve problems and ask questions in class