NAVIGATING DOCUMENTS AND TREES WITH XPATH

NEIL MOORE
KEEPING CURRENT SEMINAR
UNIVERSITY OF KENTUCKY
2022-01-19
OVERVIEW

• Background: XML and its document model
• XPath: History and basic syntax
• Examples (and demo)
• Where to go next?
OVERVIEW

• Background: XML and its document model
• XPath: History and basic syntax
• Examples (and demo)
• Where to go next?
XML: THE EXTENSIBLE MARKUP LANGUAGE

- XML is a markup language:
  - For representing text with markup (“tags” or annotations”)
  - …or for representing arbitrary hierarchical data
  - With a human-readable syntax
    - XML syntax isn’t that important in this talk: XPath focuses on the data model

- Introduced by the World Wide Web Consortium (W3C)
  - First draft version in 1996, final in 1998
  - Based on SGML (so a sibling of HTML)
    - In fact, the XML document object model (DOM) is shared with HTML
  - Supplanted for some uses by Javascript Object Notation (JSON) these days
STRUCTURE OF AN XML DOCUMENT

• An XML document is a tree consisting of nodes. Nodes may be:
  • **Text** nodes (sequences of characters with no tags)
    • Always leaves of the tree (cannot have children)
  • **Element** nodes (<tag>content</tag>)
    • Have a list of key-value **attributes** (which are nodes, but not “children” of the element!)
    • The “content” is a sequence of child nodes (elements or text)
  • A single unique **document** (or **root**) node, the root of the tree
    • With a single element child
  • A few other things (comments, processing instructions, CData, …)
EXAMPLE XML DOCUMENT (XML SYNTAX)

<?xml version="1.0" encoding="utf-8"?>
<books>
  <book year="1969">
    <title>The Left Hand of Darkness</title>
    <language>EN</language>
    <author>Ursula K. <surname>Le Guin</surname></author>
  </book>
</books>

Elements are in orange (running from <tag> to </tag>), attributes in yellow, and text in white. Purple can be thought of as representing the document node itself.
EXAMPLE XML DOCUMENT (AS A TREE)

```
document
  books
    book
      year: "1969"
        title
          The...
        language
          EN
        author
          Ursula K.
            Surname
              Le Guin
```

KEY

- text
- element attributes
- document
OVERVIEW

• Background: XML and its document model
• XPath: History and basic syntax
• Examples (and demo)
• Where to go next?
XPath Background

XPath is a language for querying XML (and HTML) documents according to their hierarchical structure.

- Introduced by W3C in 1999 (XPath 1.0)
  - Most recent version is 3.1, March 2017
  - We'll focus on 1.0: Most browsers don't even support 2.0 out of the box

- Declarative query language
  - Like basic SQL, it is not Turing-complete

- Most queries return a nodeset — an unordered list of nodes in the XML tree
  - XPath also has strings, numbers, and booleans
**XPath Queries**

- Queries are evaluated in a **context**: a node of the tree
  - And, for our purposes, they return a node set.

- The most basic kind of XPath expression is a **location path**
  - A sequence of steps indicating how to navigate from the context node to a set of other nodes.
  - Each step has an **axis** (which direction to go; default “child”),
  - …a **node test** (which nodes along that axis to select), and
  - …zero or more **predicates** (additional filters to restrict the results)
  - Each step is evaluated in the context(s) of the nodes selected by the previous step
LOCATION STEPS

• Location steps are separated by slashes
  • A location path beginning with a slash is “absolute” (context = document)
  • /books/descendant::surname/text()
    • First step: axis is child (the default: select children of the context node), node test is books (only select nodes with that name), and no predicates
    • Second step: axis is descendant, node test is author
    • Third step: axis is child, node test is text() (select nodes of that type)
  • “.” is shorthand for “self::node()”, and “..” for “parent::node()”
  • Double slash is shorthand for /descendant-or-self::node/"
  • So /books//surname/text() is similar to the first query, but not exactly the same
    • In particular, it has four steps, not three: This will matter later.
The leading slash means this is an absolute path, so the initial context node is the document.
Select children of the context node(s) named “books”
Select descendants of the context node(s) named “surname”
Select descendants of the context node(s) that are text nodes.
AXES AND NODE TESTS

• The axes in XPath include:
  • child, parent
  • self
  • descendant, ancestor
  • descendant-or-self, ancestor-or-self
  • following-sibling, preceding-sibling
  • following, preceding (in “document order”: preorder depth-first traversal)
  • attribute (shorthand: “@” means “attribute::”)

• A node test may be
  • The name of an element
  • “*” (any element; unless the axis is attribute)
  • “text()” (also “comment()” and “processing-instruction()”)
  • “node()” (matches anything)
A location step may end with any number of **predicates** to further filter results

- Written as `[expression]`
- The full syntax for expressions is too much to cover here, but it can include location paths, logical operators, relational operators, arithmetic, and calls to built-in functions
- If the result of the expression is a number, only select the node whose position in the results of this step (so far) is that number
  - `/descendant::book[2][1]` (the same (!): the first of the nodes selected by book[2])
- Otherwise, only select the node if the expression converted to boolean is **true**
  - `/descendant::book[position() > 2]` (all books after the second; already boolean)
- Most often a node set: Empty node sets are **false**, non-empty are **true**
  - `//book[author/pseudonym]` (books that have an author that has a pseudonym)
The leading slash means this is an absolute path, so the initial context node is the document.
Select all element descendants of the document.
For each selected node, find child text nodes.
document
books
book
  year: "1969"
  title
    The...
  language
    EN
  author
    Ursula K.
    Surname
      Le Guin

Only keep the selected nodes where the predicate was true (there was a text child)
Take the element children of the context nodes.

So this gives us all elements that have a text node sibling: Not the only way to express that!
OVERVIEW

• Background: XML and its document model
• XPath: History and basic syntax
• Examples (and demo)
• Where to go next?
EXAMPLES AND DEMO

• We’ll start with an expanded version of the list of books

• We’ll issue various queries and discuss why they return what they do

• Using the website: xpather.com
  • Could instead issue queries in your browser’s Javascript console with $x("/blah")
  • But it’s easier to see the results this way.
  • http://xpather.com/4gcYRTs6 (preloaded with our example document)
OVERVIEW

• Background: XML and its document model
• XPath: History and basic syntax
• Examples (and demo)
• Where to go next?
FOR ALL THE DETAILS

• We have covered just the basics of XPath 1.0
  • And occasionally simplified, perhaps over-simplified

• Online tutorials:
  • https://developer.mozilla.org/en-US/docs/Web/XPath
  • https://www.w3schools.com/xml/xpath_intro.asp

• Full specifications: https://www.w3.org/TR/xpath/
  • Not the easiest thing in the world to read!

• Formal semantics: https://www.w3.org/TR/xquery-semantics/ (XPath 2.0)
Naïve recursive evaluation of location steps can take exponential

- Alternating sequences of descendant and ancestor axes
- Nested predicates are even harder to deal with:
  /descendant:a[ancestor:b[descendant:a[ancestor:b[...]]]


- Shows XPath can be evaluated in polynomial time (on the sizes of the expression and of the document)
- Some useful subsets of XPath can be evaluated in linear time $O(|expr| \times |document|)$
- The trick: evaluate predicates “bottom up” (start with the most deeply nested)
THANK YOU!