XPath 1.0 & 2.0

What is XPath ?

- XPath is a W3C standard
- XPath expressions are similar to URLs, hence the name
- XPath uses path expressions to navigate through the hierarchy of XML document tree; and address parts of an XML document
- XPath uses a compact, non-XML syntax to facilitate use of XPath within URIs and XML attribute values.
- XPath models XML document as a tree of nodes using Document Object Model (DOM)

XML data model

```
<library>
   <book>
     <author>ullman
     </author>
     <chapter> …
     </chapter>
     <chapter>
        <section>
           <para>… </para>
           <para>… </para>
        </section>
     </chapter>
   </book>

</library>
```

- library is the parent of book; book is the parent of two chapters
- The two chapters are the children of book; para is the child of the 1st section of the 2nd chapter
- The two chapters under the book are siblings (they have common parent)
- Library , book and chapter are the ancestors of para
- chapter section and the two paras are the descendants of book

Path Expressions

Absolute Path Expressions:

- Paths that begin with a / (slash)
- They match paths that start at the document root node
- Example: /library/book/chapter

Relative Path Expressions:

- Paths that do not begin with a / (slash)
- They match a path starting from the context node
- Example: chapter/section

- Paths that begin with // (double slash)
- They match a path that can start anywhere
- Example: //section/para

Predicates

- Predicates take either number or an Xpath expression
- Predicates are specified in square brackets

Examples:

/library/book/chapter[1] selects the first chapter of all book elements

//section/para[2] selects the second para of all section elements

//chapter[1]/section[2] selects the second section of 1st chapter

//book[author="Ullman"]//section selects all the sections in the book authored by Ullman

//book/chapter[last( )] selects the last chapter in all book elements

NOTE: The function last( ) selects the last matching element

Wild cards

Asterisk (*) selects all the children of the context node

Dot (.) refers to the context node

Dot-Dot (..) refers to the parent of the context node

Examples:

/library/book/chapter/* selects all the children of chapter

//*/*/para selects every para that has atleast two ancestors.

//* selects every element in the entire document.

/library/book/author[. = "Ullman"] selects all author elements having text as Ullman

//book/chapter[.. /author = "Ullman"] selects all chapters in the books written by Ullman

Attributes

- Attributes are represented as name-value pairs
  Eg., <book year="2004"> ..... </book>
  element book has an attribute year whose value is "2004"
- In Xpath, attribute is prefixed with @

Example:

@num selects *every* to attribute named num
//@* selects *every* attribute in the document
//book[@year] selects all books having an attribute year
//book[not(@year)] selects all books that do not have an year attribute

Attributes contd

//book[@*] selects all books having an attribute(s)

//book[not(@*)] selects all books that do not have any attribute(s)

//book[normalize-space(@year)="2004"] selects every book element with attribute year and with value 2004

NOTE: The function normalize-space( ) removes leading and trailing spaces from a value before comparison

XPath: Location paths

Location path is the main component of XPath. Each location path consists of one or more location steps. Each location step is of the form:

axis::nodetest [*predicate*(*s*)]

Example:

```
child::book[attribute::year="2004"]
```

XPath Axes

There are several XPath axes; each defines a node-set relative to the context node. They are

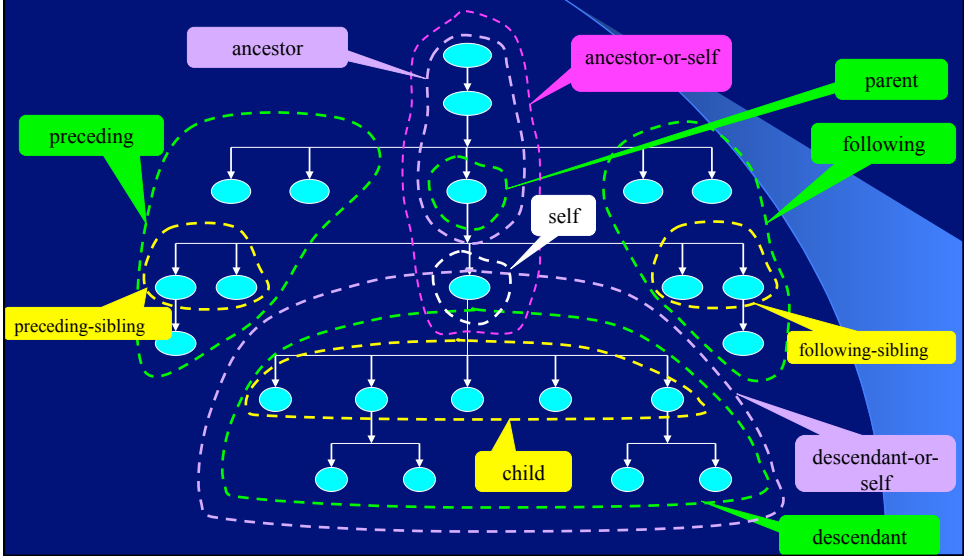
- self:: contains the set of context nodes
- child:: contains all the children of the context node(s)
- parent:: contains the parent of the context node(s)
- ancestor:: contains all ancestors (parent, grand parent, etc.) of the context node(s)
- descendant:: contains all descendants of the context node(s)
  (Note: never contains attribute or namespace nodes)
- ancestor-or-self:: contains the context node and all its ancestors.
- descendant-or-self:: contains the context node and all its descendants.

XPath Axes contd

- following:: contains everything after the context node in the entire XML document
- preceding:: contains everything before the context node in the entire XML document.
- following-sibling:: contains all the siblings after the context node.
- preceding-sibling:: contains all the siblings before the context node.
- attribute:: contains all attributes of the context node.
- namespace:: contains all the namespace nodes the context node.

XML Document Partitions

For any given context node *v*, the four major axes – ancestor, descendant , following , preceding – specify partitioning of the document containing *v*



Abbreviations for XPATH axes

(none)	is the same as	child::
@	is the same as	attribute::
.	is the same as	self::node()
.//X	is the same as	self::node()/descendant-or-self::node()/child::X
..	is the same as	parent::node()
../X	is the same as	parent::node()/child::X
//	is the same as	/descendant-or-self::node()/
//X	is the same as	/descendant-or-self::node()/child::X

XPath Axes Examples

- self::para : selects the context node if it is a para element
- //chapter/parent::* : selects the parent node(s) of all the chapter elements
- //chapter[2]/section[3]/following::* : selects every element after the third section of second chapter
- //book/descendant::para : selects every para element descendant of book
- child::chapter[position() = 5][attribute::title= 'Introduction'] : selects the 5th chapter of the context node if it has title 'Introduction'
- following-sibling::chapter[postion() != last()] selects the next sibling of the chapter if it is not the last chapter

XPath Axes Examples contd

- child::*[self::chapter or self::preface] : selects the chapter or preface children of context node (chapter|preface)
- .. /attribute::para : selects all the para attribute of the parent of the context node(s) (../@para)
- self::node()/descendant-or-self()/child::para : selects all the para descendants of the context node. (.//para)
- //*[count(@*) >= 3] : selects all nodes with more than three attributes
- //book[attribute::type= 'science'][3] : selects all book elements of type 'science' , and returns the third
- //book[3][attribute::type= 'science'] : selects the third book element, but only if it is of type 'science'

XPath Data types and Operators

An XPath expression yields either a node-set, a string, a boolean value (true/false), or a number

XPath operators:

|	Alternative, for example, section|para selects all section and para elements
or, and	Returns the or/and of two boolean values
=, !=	Equal or not equal, for booleans, strings, and numbers
<, >, <=, <=	Less than, greater than, less than or equal to, greater than or equal to --- for numbers only
+, -, *, div, mod	Add, subtract, multiply, floating-point divide, modulus operations

String-Value of an element

The string value of a complex element (non leaf elements) is determined by the concatenation of all the descendant text nodes. For a data element like this:

```
<para>
This paragraph contains some    <bold>XML</bold> stuff
</para>
```

The string-value of a <para> element is

"This paragraph contains some XML stuff"

Overview of XPath functions

XPath defines a set of functions on a collection of nodes. These functions return a string, a number, or a boolean value

Node-set functions:

- *number* count(*node-set*) : Returns the number of nodes in the argument *node-set*.
  - //section[count(section) = 0)]selects all sections that have no subsections
- *node-set* id(...) : Returns the node with specific id
  - id("algoChapt") selects the element with unique ID algoChapt
  - id("algoChapt")/child::para[position() = 5] selects the fifth para child of the element with unique ID algoChapt

String functions

- concat(*string*, *string*, ...) -- concatenates the string values
- starts-with(*string1*, *string2*) -- returns true if *string1* starts with *string2*
- contains(*string1*, *string2*) -- returns true if *string1* contains *string2*
- substring-before(*string1*, *string2*) -- returns the start of *string1* before *string2* occurs in it
- substring-after(*string1*, *string2*) -- returns the remainder of *string1* after *string2* occurs in it
- substring(*string*, *index*) -- returns the substring from the index position to the end, where the index of the first char = 1
- substring(*string*, *index*, *len*) -- returns the substring from the index position, of the specified length

String functions cont...

- string-length() -- returns the size of the context-node's string-value
- normalize-space() -- returns the normalized string-value of the current node (no leading or trailing whitespace, and sequences of whitespace characters converted to a single space)
- normalize-space(*string*) -- returns the normalized string-value of the specified string
- translate(*string1*, *string2*, *string3*) -- converts *string1*, replacing occurrences of characters in *string2* with the corresponding character from *string3*

Boolean functions

These functions operate on or return boolean values:

- not(...) -- negates the specified boolean value
- true() -- returns true
- false() -- returns false
- lang(*string*) -- returns true if the language of the context node (specified by xml:Lang attributes) is the same as the specified language.

  For example:

  lang("en") is true for <para xml:lang="en">...</para>

Numeric functions

These functions operate on or return numeric values.

- sum(...) -- returns the sum of the numeric value of each node in the specified node-set
- floor(*N*) -- returns the largest integer that is not greater than *N*
- ceiling(*N*) -- returns the smallest integer that is greater than *N*
- round(*N*) -- returns the integer that is closest to *N*

Conversion functions

These functions convert one data type to another.

- string(...) -- returns the string value of a number, boolean, or node-set
- boolean(...) -- returns a boolean value for a number, string, or node-set

  NOTE: A non-zero number, a non-empty node-set, and a non-empty string are all true
- number(...) -- returns the numeric value of a boolean, string, or node-set (true is 1, false is 0, a string containing a number becomes that number, the string-value of a node-set is converted to a number)

Namespace functions

These functions determine the namespace characteristics of a node.

- local-name() -- returns the name of the current node, minus the namespace prefix
- local-name(...) -- returns the name of the first node in the specified node set, minus the namespace prefix
- namespace-uri() -- returns the namespace URI from the current node
- namespace-uri(...) -- returns the namespace URI from the first node in the specified node set
- name() -- returns the expanded name (URI plus local name) of the current node
- name(...) -- returns the expanded name (URI plus local name) of the first node in the specified node set

XPath 2.0 overview

XPath 2.0 data model supports new datatypes for date, time, URIs etc., These new datatypes are based on XML schema:

XML Schema

xs:boolean	xs:gYear
xs:decimal	xs:gMonthDay
xs:float	xs:gDay
xs:double	xs:gMonth
xs:duration	xs:hexBinary
xs:dateTime	xs:base64Binary
xs:time	xs:anyURI
xs:date	xs:QName
xs:gYearMonth	xs:Notation

XPath 2.0: Sequences

Every XPath 2.0 expression evaluates to a sequence.

Definition of a sequence:

Sequence ← {Item*}
Item ← {Atomic|Node}
Atomic ← {XML Schema datatype}
Node ← {root|element|attribute|PI|comment|text|namespace}

- Sequence is an ordered list of items, where as *node-set* (in XPath 1.0) is unordered.
- Sequence may contain duplicate elements
  - not allowed in *node-set*

Examples of Sequences

(1, 2, 3) is a sequence, where comma is a sequence construction operator

(1 to 100) : creates a sequence using range operator 'to'

(1, (2, 3), ()) is not a valid XPath 2.0 sequence (nesting is not allowed) and is treated as (1, 2, 3)

(3, 6, 9)[2] : returns second item in the sequence, here it is 6.

(//chapter/title , //chapter/section/para) : results a sequence of titles and paras

(//title , //para , //title) is a valid sequence;
sequences allow duplicate elements, but *node-set*s don't

Working with Sequences

for expression is designed for handling sequences by looping, or iterating, over all items in a sequence.

Syntax: for *variable* **in** *sequence* **return** *expression*

for $c **in** //book/chapter **return** $c/title
iterates over all book elements, and for each chapter stored in $c, the expression returns title

Similarly, the average number of chapters a book contains can be found as:
avg (**for** $b **in** //book **return** count($b/chapter))

Quantifier Expressions

some expression requires that at least one item in a sequence satisfies a given expression to become *true*

Syntax: **some** *variable* **in** *sequence* **satisfies** *expression*

some $a **in** //author **satisfies** $a = "knuth"

returns true when at least one <author> element has text "knuth"

every expression requires that all items in a sequence satisfy a given expression to become *true*

Syntax: **every** *variable* **in** *sequence* **satisfies** *expression*

every $y **in** //year **satisfies** $y = "2004"

returns true when all <year> elements have text "2004"

Set Operations on Sequences

union operator:

//chapter[1]/* union //chapter[2]/*

intersect operator:

$authors intersect //book/author returns all the common elements in $authors and in the sequence //book/author

except operator:

$authors except //book/author returns all the elements in $authors and are not in the sequence //book/author

Additional Features

- Additional string functions such as upper-case, lower-case, string-pad, matches, replace, and tokenize are provided
- Can specify multiple node tests in location steps
  Eg., `//a/(b | c)/d` is equivalent to `//a/b/d | /a/c/d`
- Comments are specified between (: and :), and namespace axis is deprecated
- Provides additional expressions such as if, cast, castable, treat, and instance of
- Aggregate operations such as sum, avg, min, max are provided

References

XPath 1.0 : http://www.w3.org/TR/xpath
XPath 2.0 specification : http://www.w3.org/TR/xpath20/
XPath data model : http://www.w3.org/TR/xpath-datamodel/
The library of functions and operators supported by XPath 2.0
http://www.w3.org/TR/xquery-operators/
XPath 2.0 tutorial:
http://www.developer.com/xml/article.php/3344421