웹 기술 및 응용

XML Path Language (XPath)

2018년 2학기

Instructor: Young-guk Ha
Dept. of Computer Science & Engineering

Contents

- Introduction
- Data Model
- Location Paths
- Expressions
- Core Function Library
- XPath 2.0

What is XPath?

❑ W3C Definition

- "***XPath*** is a language for addressing parts of an XML document, designed to be used by XSLT and XPointer"

❑ Features

- Addressing parts of an XML document
- A common syntax and semantics to be used with other XML specifications
  - E.g., XSLT and XPointer
- Manipulation of node-sets, strings, numbers, and booleans
- Models an XML document as a tree of nodes
  - Cf., DOM tree and DOM nodes
- Fully supporting XML Namespace

History

- Initially, XSLT contains XPath's semantics
  - Until 1999-04
- XPath 1.0 (Original Version)
  - 1999-07 ~ 1999-10: working draft
  - 1999-11: W3C recommendation
  - Basic and widely used: covered in this class
- XPath 2.0 (Second Edition)
  - 2010-12: W3C recommendation
- 기타
  - XPath 3.0: W3C recommendation (2014-04)
  - XPath 3.1: W3C recommendation (2017-03)

Basic XPath Expressions

- XPath includes 4 types of basic expressions
  - Location path expressions
  - Boolean expressions
  - Numeric expressions
  - Function calls
    - Boolean functions
    - Numeric functions
    - String functions

Data Model

- Evaluation result of an XPath expression is one of the following objects
  - Node-set
    - Returned by location path expressions
  - Boolean
    - Returned by boolean expressions or boolean functions
  - Number
    - Returned by numeric expressions or numeric functions
    - Floating point number (double-precision 64-bit)
  - String
    - Returned by string functions
    - A sequence of zero or more UNICODE characters

Data Model (Cont'd)

- XPath models an XML document as a tree of nodes
  - Some nodes have ***Expanded-name***
    - Namespace URI
    - Local part
  - Every node has ***String-value***

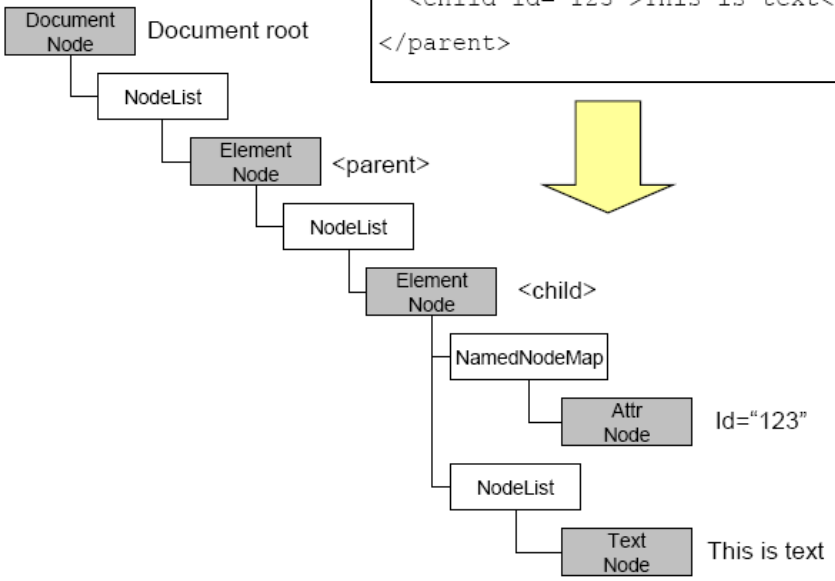
- Node types
  - Root node
  - Element node
  - Attribute node
  - Namespace node
  - Processing instruction node
  - Comment node
  - Text node

  - Note XPath tree model is different from the DOM tree model

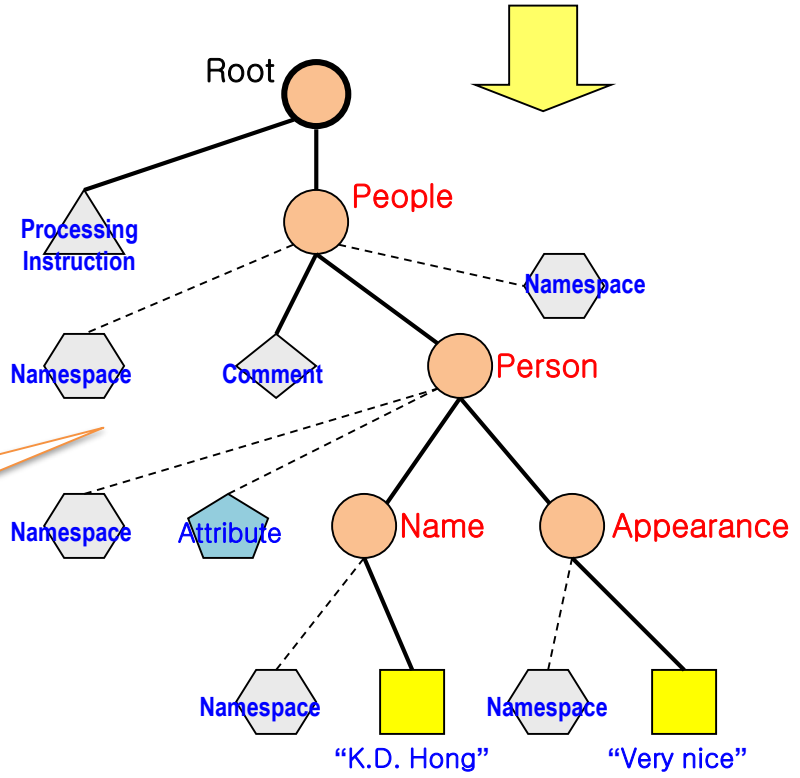
Data Model (Cont'd)

❑ DOM Tree vs. XPath Tree

```
<parent>
  <child id="123">This is text</child>
</parent>
```




```
<?xml version="1.0"?>
<?This is processing node?>
<People xmlns="http://www.abc.com/people" xmlns:w3c="http://w3c.org">
  <!-- List of people -->
  <Person pID="1234">
    <Name>K.D. Hong</Name>
    <Appearance>Very nice</Appearance>
  </Person>
</People>
```



- ✓ **Root = Document root**
- ✓ No **NodeList**
- ✓ No **NamedNodeMap**

Data Model (Cont'd)

❑ Root node

- ➢ Similar to the document root (**`Document`** node) of DOM
- ➢ Child node
  - o The root element node
  - o Processing instruction and comment nodes
- ➢ Expanded-name
  - o N/A
- ➢ String-value
  - o Concatenation of the string-values of all text node descendants

Data Model (Cont'd)

❑ Element node

- ➢ Child node
  - o Element nodes
  - o Comment, processing instruction and text nodes
- ➢ Expanded-name
  - o Namespace URI = URI for prefix of the element's QName
  - o Local part = Local part of the element's Qname
- ➢ String-value
  - o Concatenation of the string-values of all text node descendants

❖ Expanded name = Namespace URI + Local part
❖ Qualified name (QName) = Namespace prefix + Local part

Data Model (Cont'd)

❑ Attribute node

- Each element node has an associated set of attribute nodes
- Parent node
  - An element node
  - But NOT a child of its parent element node
- Expanded-name
  - Namespace URI = URI for prefix of the attribute's QName
  - Local part = Local part of the attribute's Qname
- String-value
  - Normalized attribute value

Data Model (Cont'd)

❑ Namespace node

- ➢ Each element has an associated set of namespace nodes
  - o For the default namespace and each distinct namespace prefix
- ➢ Parent node
  - o An element node
  - o But NOT a child of its parent element node
- ➢ Expanded-name
  - o Namespace URI = **null**
  - o Local part = Namespace prefix
- ➢ String-value
  - o Namespace URI

Data Model (Cont'd)

❑ Processing instruction node

- ➢ For every processing instruction
  - o XML declaration is NOT regarded as a processing instruction
- ➢ Expanded-name
  - o Namespace URI = null
  - o Local part = target of the processing instruction
- ➢ String-value
  - o Part that follows the target

Data Model (Cont'd)

❑ Comment node

- ➢ Expanded-name
  - o N/A
- ➢ String-value
  - o Content of the comment

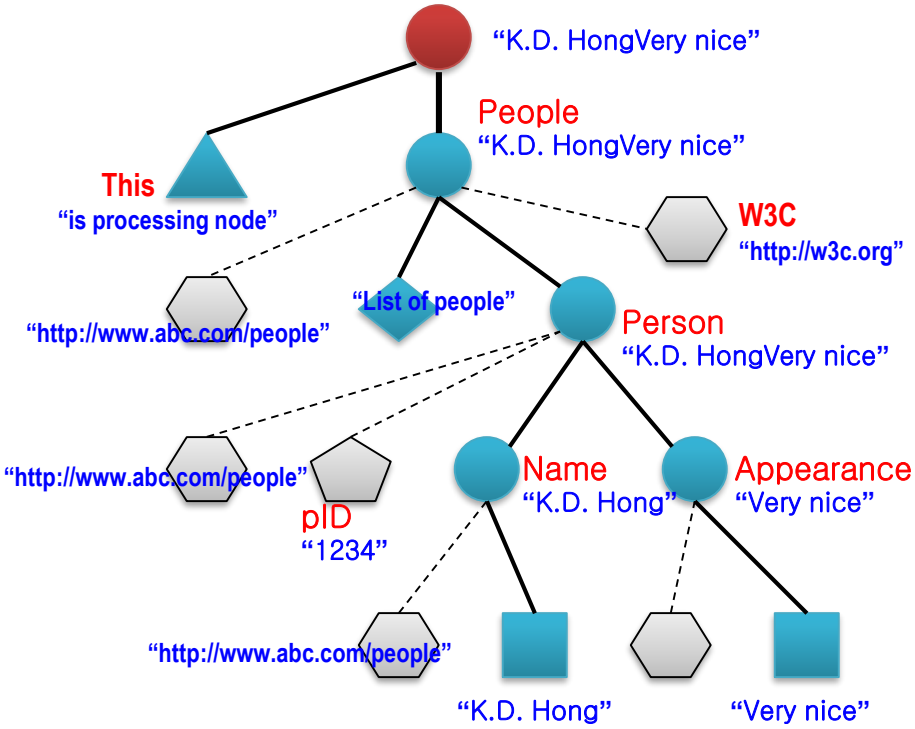
❑ Text node

- ➢ Character data
  - o Never have an immediately following/preceding sibling text node
- ➢ Expanded-name
  - o N/A
- ➢ String-value
  - o Text contents

Data Model (Cont'd)

[Tree representation example of an XML document]

```
<?xml version="1.0"?>
<?This is processing node?>
<People
   xmlns="http://www.abc.com/people"
   xmlns:w3c="http://w3c.org">
  <!-- List of people -->
  <Person pID="1234">
    <Name>K.D. Hong</Name>
    <Appearance>Very nice</Appearance>
  </Person>
</People>
```



Root Element Attribute

Namespace Text Comment

Processing instruction

—— Parent-child

----- Parent, but not child

- Local part of Expanded-name
- String-value

Grammar of XPath

```
LocationPath ::=              RelativeLocationPath |
                              AbsoluteLocationPath
AbsoluteLocationPath ::=      '/' RelativeLocationPath? |
                              AbbreviatedAbsoluteLocationPath
RelativeLocationPath ::=      Step |
                              RelativeLocationPath '/' Step |
                              AbbreviatedRelativeLocationPath
Step ::=                      AxisSpecifier NodeTest Predicate* |
                              AbbreviatedStep
AxisSpecifier ::=             AxisName '::' |
                              AbbreviatedAxisSpecifier
   ...                        ...
```

Location Path

- The most important expression in XPath
- Relative location path
  - Returns node-set
    - Selects a set of nodes relative to the ***context node***
  - Consists of ***location steps***
    - Location step = axis specifier + node test + predicates
    - Relative location path = step / step / …
- Absolute location path
  - Starts with the document root (“/”)
    - Absolute location path = / relative location path
  - Note “/” is NOT the root element node
    - Actually, “/” refers to the parent node of the root element node
- Provides syntactic abbreviations
  - Provides abbreviated location path and steps

Location Step

❑ Syntax

```
AxisName '::' NodeTest ('[' PredicateExpr ']')*
```

❑ Examples

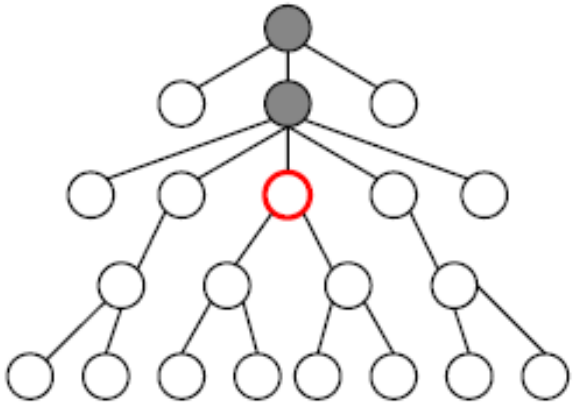
```
child::text()

child::title[position()=5]

descendant::student[position()=last()][@course="MS"]
```

Axes

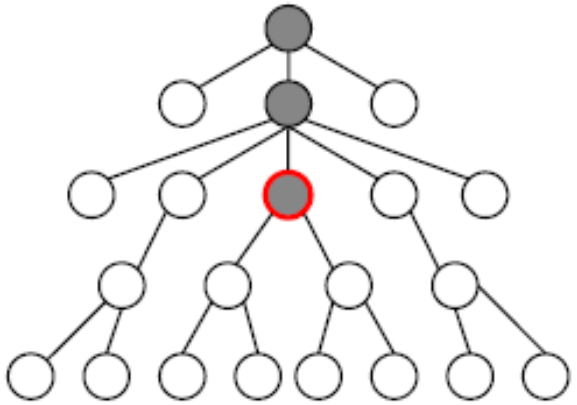
- An ***axis*** specifies relationship between the context node and the nodes selected by the location step

- Total 13 Axes
  - 11 axes for tree relationship
    - `ancestor, ancestor-or-self, child, descendant, descendant-or-self, following, following-sibling, parent, preceding, preceding-sibling, self`
  - 2 more axes for non-tree relationship
    - `attribute, namespace`
      - *Always returns **empty** unless the context node is an element*

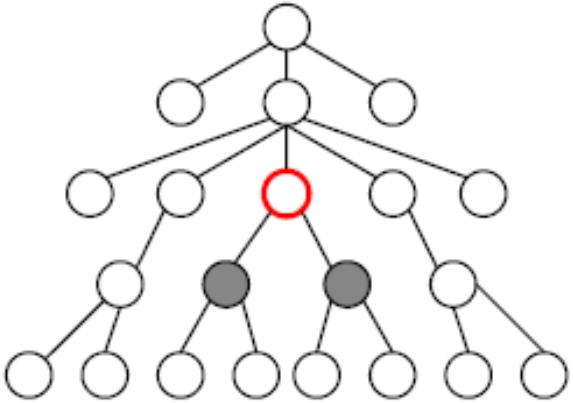
Axes for Tree



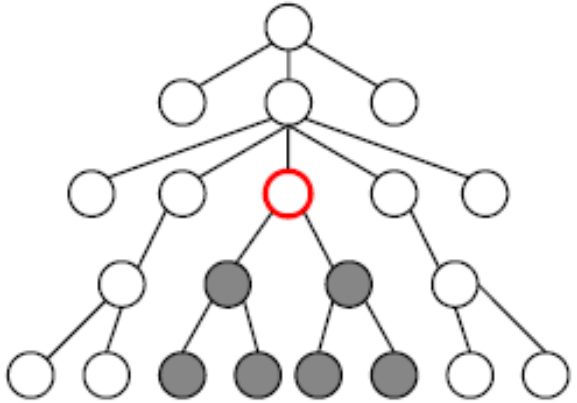
ancestor



ancestor-or-self



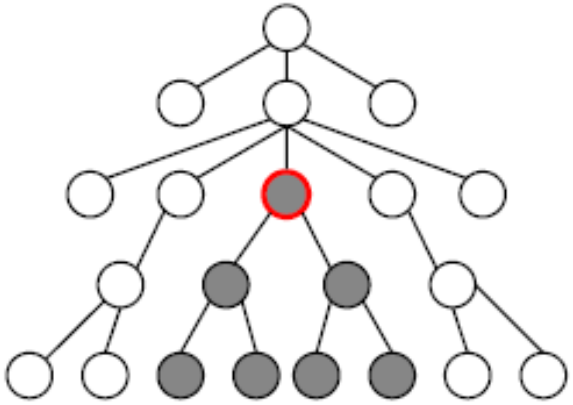
child



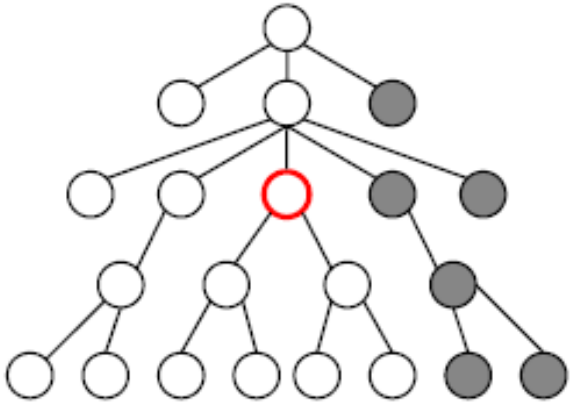
descendant

◯ : context node

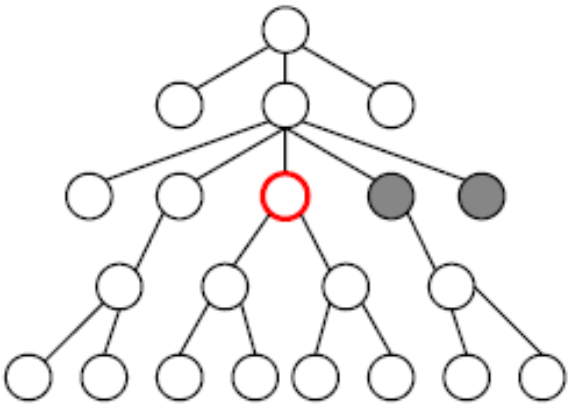
Axes for Tree (Cont'd)



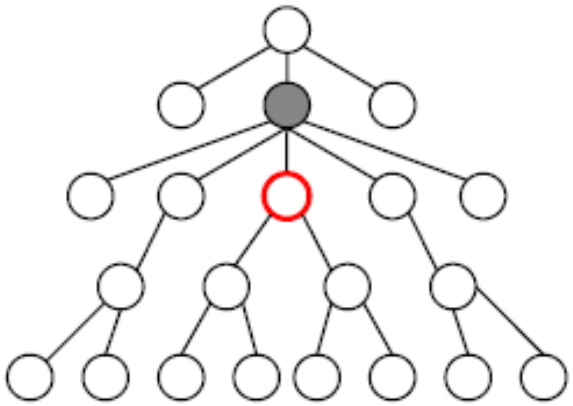
descendant-or-self



following



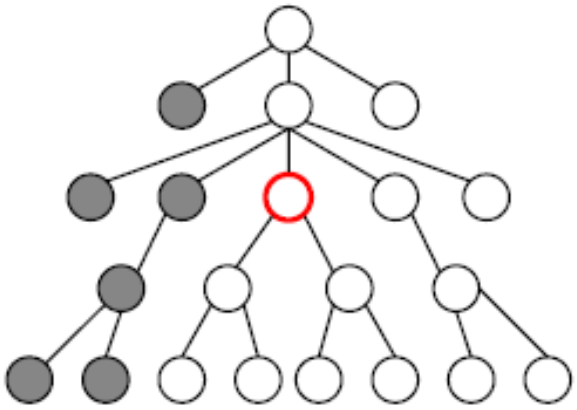
following-sibling



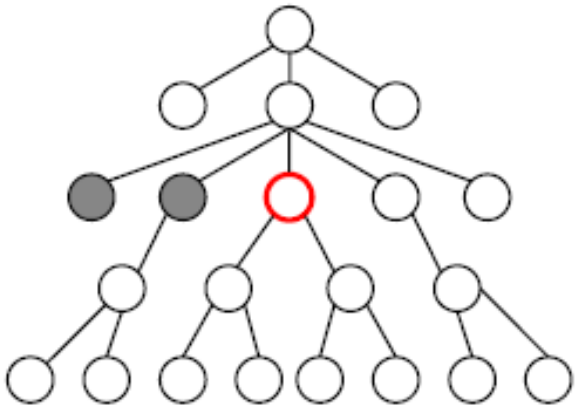
parent

○ : context node

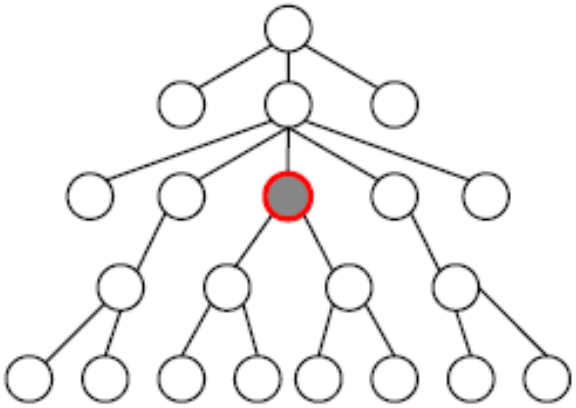
Axes for Tree (Cont'd)



preceding



preceding-sibling



self

◯ : context node

* All the nodes in the document (without overlapping)
= self + ancestor + descendant + preceding + following

Node Test

❑ Note test specifies

- name or type of the nodes

❑ Name test

- *: **true** for any node name of the principal node type
- NCName or QName: **true** for the given name of XML Document

E.g.)

```
child::*          child::para
attribute::*      attribute::href
```

Node Test (Cont'd)

❑ Type test

- **`text()`**: true for any text node
- **`comment()`**: true for any comment node
- **`processing-instruction()`**: true for any processing instruction
- **`node()`**: true for any node of any type

Ex) **`child::text()`** **`child::comment()`**

`decendent::node()`

`self::processing-instruction()`

Predicates

- Filters a node-set produced by axis and node test processing

```
[PredicateExpr_1][PredicateExpr_2]...[PredicateExpr_n]
```

  - Evaluated for each node to be filtered
  - Each expression can contain
    - Boolean expression
    - Numeric expression
    - Core functions (numeric, boolean, string)
  - All the predicate expressions are combined with logical AND operations

Location Path Examples

- `/child::doc`
  - Selects the root element “doc”
- `/child::doc/child::chapter`
  - Selects all the child elements named “chapter”
- `/descendant-or-self::section`
  - Selects all the “section” elements in the same document
- `child::section[position()=2]/child::para`
  - Selects all the “para” elements in the second “section” child of the context node
- `preceding-sibling::chapter[position()=1]`
  - Selects the first previous “chapter” sibling of the context node
- `child::para[position()=last()][attribute::type="warning"]`
  - Selects the last “para” child of the context node that has a “type” attribute with value “warning”

Abbreviated Syntax

Abbreviation	Full XPath Syntax
(no axis specifier)	child::
@	attribute::
.	self::node()
..	parent::node()
//	/descendant-or-self::node()/
[x]	[position()=x]

- Examples

```
parent::node()/child::title                    → ../title
/descendant-or-self::node()/child::para        → //para
/child::para[attribute::type="warning"][position()=5]
                                  → /para[@type="warning"][5]
```

Other Expressions

❑ Union expressions

- Computes the union of the result node-sets of location path expressions

```
LocationPathExpr "|" LocationPathExpr "|" …
```

❑ Boolean expressions

- Operators: "**or**", "**and**", "=", "!=", "<=", "<", ">=", ">"
- "<" and "<=" must be quoted like "<" and "<=" in XML documents
- Left associative
  - E.g., 3 > 2 > 1 evaluates false

    because 3 > 2 > 1 = (3 > 2) > 1 = (true) > 1 = (1) > 1 = false

Other Expressions (Cont'd)

❑ Numeric expressions

- Returns a floating-point number in 64-bit double-precision format including
  - Positive & negative infinity
  - NaN (Not-a-Number) value (e.g., square root of -1)
- Operators
  - "+", "-", "*", "div", "mod"
  - Note "-" operator typically needs to be preceded and followed by a whitespace
    - E.g.) `foo-bar` (x)
      `foo - bar` (o)

❑ Function calls

```
FunctionName(Argument_1, Argument_2, ...)
```

Core Function Library

- Must be included in every XPath implementation
- Node set functions
  - **number last()**
  - **number position()**
  - **number count(*node-set*)**
  - **node-set id(*object*)**
    - E.g., `id("foo")` selects the element with unique ID “foo”
  - **string local-name(*node-set*?)**
    - Returns local part of the first node
  - **string namespace-uri(*node-set*?)**
    - Returns namespace URI of the first node
  - **string name(*node-set*?)**
    - Returns QName of the first node

Core Function Library (Cont'd)

String Functions

- **string string(*object*?)**
  - E.g., `string(0)` = "0"
  - E.g., `string(true())` = "true"
  - E.g., `string(node-set)` = string-value of the first node
- **string concat(*string*, *string*, *string**)**
- **boolean starts-with(*string*, *string*)**
- **boolean contains(*string*, *string*)**
- **string substring-before(*string*, *string*)**
  - E.g., `substring-before("1999/04","/")` = "1999"
- **string substring-after(*string*, *string*)**
- **string substring(*string*, *number*, *number*?)**
  - E.g., `substring("12345",2,3)` = "234"
- **number string-length(*string*?)**
- **string normalize-space(*string*?)**
- **string translate(*string*, *string*, *string*)**
  - E.g., `translate("aabbcc", "abc", "ABC")` returns "AABBCC"

Core Function Library (Cont'd)

❑ Boolean functions

- **boolean boolean(*object*)**
  - A number is true, if and only if number != 0 nor NaN
  - A node-set is true, if and only if node-set != empty
  - A string is true, if and only if string-length(string) != 0
- **boolean not(*boolean*)**
- **boolean true()**
- **boolean false()**
- **boolean lang(*string*)**
  - E.g., `lang("en")` is true,
    if context node is `<para xml:lang="en"/>`

Core Function Library (Cont'd)

❑ Numeric functions

- **number number(*object*?)**
  - number(true()) = 1
  - number("123") = 123
  - number(node-set) = number(string(node-set))
- **number sum(*node-set*)**
- **number floor(*number*)**
- **number ceiling(*number*)**
- **number round(*number*)**

XPath 2.0

❑ Latest version (Second Edition)

- ➢ Released on December 2010

❑ Feature

- ➢ Improved ease of use
- ➢ Improved interoperability
- ➢ Improved internationalization support
- ➢ Improved processor efficiency
- ➢ Simplified manipulation of XML Schema-type content
- ➢ Simplified manipulation of string content
- ➢ Support for related XML standards
- ➢ Maintenance of backward compatibility

References

- XML Path Language (XPath) 1.0
  - http://www.w3.org/TR/xpath
- XML Path Language (XPath) 2.0 (Second Edition)
  - http://www.w3.org/TR/xpath20/

W3C WORLD WIDE WEB consortium Ubiquitous Web domain