Data Navigation

Faculty of Computer Science
Workflow Systems and Technologies

Interoperability SS 2023

Amolkirat Singh Mangat
Matthias Ehrendorfer
Florian Stertz

Introduction

What can we do if received data does not conform to our internal format/schema?

How can we do it?

Data Querying

- Enables the selection of elements in the data
- Relies on data following a certain schema
- Important for data transformation
- Languages for data querying include:
    - XPath (for XML documents)
    - JSONPath (for JSON documents)

XPath [1]

- Different versions:
  - **1.0: address parts of an XML document** [2]
  - 2.0: superset of XPath 1.0 which adds: richer set of data types, more functions [3]
  - 3.0: superset of XPath 2.0 which adds: string concatenation operator (“||”), mapping operator (“!”), dynamic function calls, inline function expressions, literal URLs in names [4]
  - 3.1: superset of XPath 3.0 which adds: maps (i.e., associative arrays) and arrays to the data model => can address nodes of JSON trees [5]

XPath 1.0 [2]

- Address parts of an XML document
- “Location Path” consists of “Location Steps” (separated by “/”)
- “Location Step” consists of:
  - axis (determine in which direction to go from context node)
  - node test (determine which node type is selected for the step)
  - predicate(s) (filters node set for the step)
- XPath can be used to query:
  - element nodes
  - attribute nodes
  - text nodes
  - others (root, namespace, processing, and comment nodes)

XPath 1.0 [2]

- Next step is relative to the context node

XML document

```
<beings>                                                 -----------> context node
  <being name="Batman" level="1000">                       -----------> child node
    <trait name="stealth" score="4"/>
    <trait name="charisma" score="1"/>
    <trait name="ruthlessness" score="5"/>
  </being>
  <being name="Storm" level="9900">                        -----------> child node
    <trait name="stealth" score="3"/>
    <trait name="charisma" score="4"/>
    <trait name="ruthlessness" score="4"/>
  </being>
</beings>
```

XPath 1.0 [2]

- Next step is relative to the context node

XML document

```
<beings>                                              ----------> parent node
  <being name="Batman" level="1000">                    ----------> context node
    <trait name="stealth" score="4"/>                     ----------> child node
    <trait name="charisma" score="1"/>                    ----------> child node
    <trait name="ruthlessness" score="5"/>                ----------> child node
  </being>
  <being name="Storm" level="9900">                     ----------> sibling node
    <trait name="stealth" score="3"/>
    <trait name="charisma" score="4"/>
    <trait name="ruthlessness" score="4"/>
  </being>
</beings>
```

XPath 1.0 [2]

- Next step is relative to the context node

XML document

```
<beings>
  <being name="Batman" level="1000">            ----------> parent node
    <trait name="stealth" score="4"/>             ----------> sibling node
    <trait name="charisma" score="1"/>            ----------> context node
    <trait name="ruthlessness" score="5"/>        ----------> sibling node
  </being>
  <being name="Storm" level="9900">
    <trait name="stealth" score="3"/>
    <trait name="charisma" score="4"/>
    <trait name="ruthlessness" score="4"/>
  </being>
</beings>
```

XPath 1.0 [2]

What should be selected?	**Syntax**	**Abbreviated Syntax**
select document root	/	/
select child elements <x>	child::<x> (child::* for all)	<x> (* for all)
select context node <x>	self::<x> (self::* for all)	. for all
select parent elements <x>	parent::<x> (parent::* for all)	.. for all
select attribute node <x>	attribute::<x> (attribute::* for all)	@<x> (@* for all)
select all descendants <x> (excluding/including self)	descendant::<x> / descendant-or-self::x	
select all ancestors <x> (excluding/including self)	ancestor::<x> / ancestor-or-self::<x>	
select following siblings <x>	following-sibling::<x>	
select preceding siblings <x>	preceding-sibling::<x>	

XPath 1.0 [2]

What should be selected?	**Syntax**	**Abbreviated Syntax**
select all text node childs	child::text()	text()
select all node childs	child::node()	node()
select all element descendants of root (including self)	/descendant-or-self::child::*	//*
add predicate to select nodes e.g., attribute <x> has value <y> attribute <x> exists there is a child with text “<y>” …	[<predicate>] [attribute::<x> = ”<y>”] [attribute::<x>] [self::*/child::*/text() = “<y>”]	[<predicate>] [@<x> = ”<y>”] [@<x>] [./*/text() = “<y>”]
select <number> element from the result set	[position()=<number>]	[<number>]
union of nodesets	<node-set> | <node-set>	

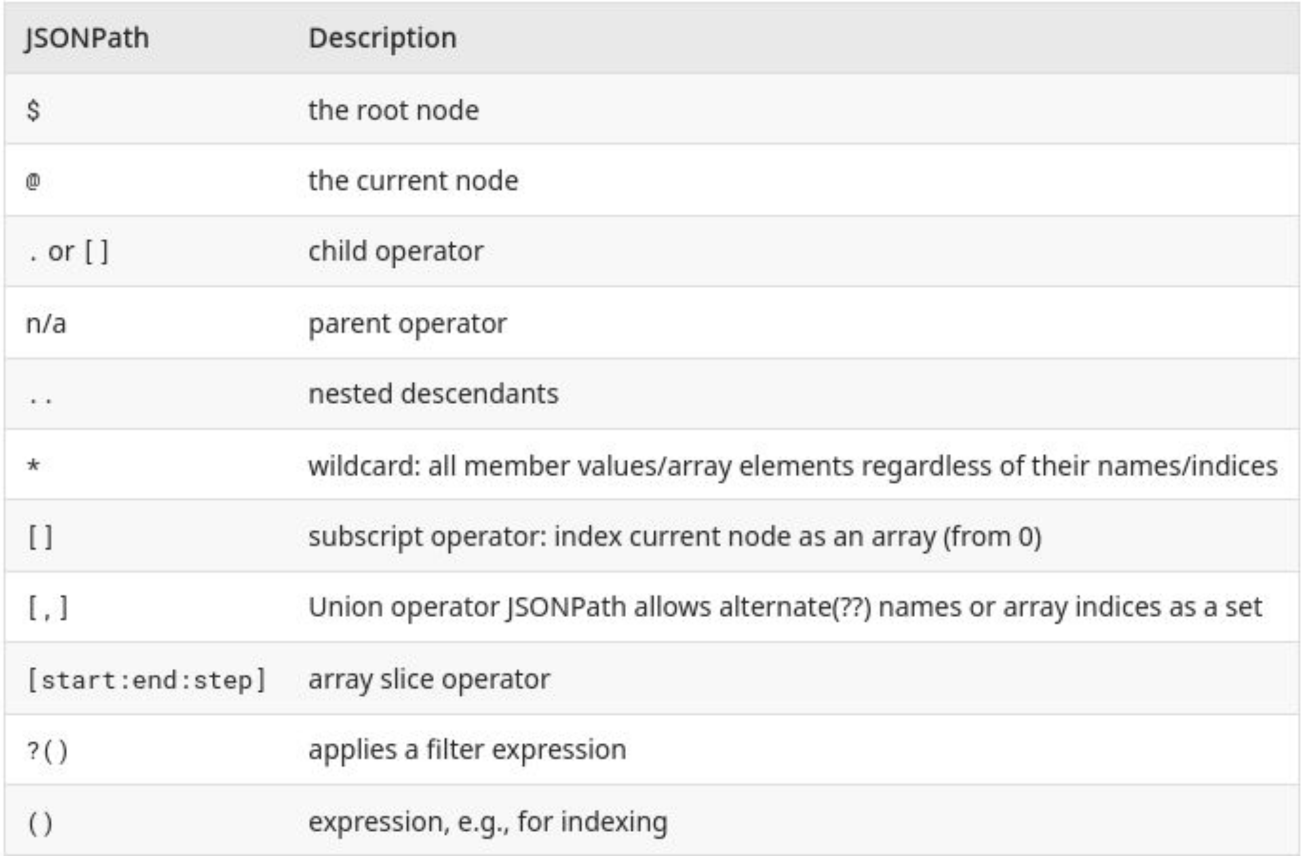
XPath 1.0 [2]

- Node set functions e.g.,
  - position() -> returns number equal to context position
  - count(node-set) -> returns number of nodes in node-set
  - name(node-set) -> returns name of first node in node-set
- String functions e.g.,
  - concat(string, string, string*) -> concatenates the arguments
  - starts-with(string, string) -> returns true if first string argument starts with second string argument
  - substring(string, number, number?) -> returns first argument from position of second argument to end or with length of third argument
- Also boolean and number functions available

JSONPath [3]

- Inspired by XPath
- XPath /beings/being[1]/trait looks like:
  - dot notation: $.beings.being[0].trait
  - bracket notation: $[‘beings’][‘being’][0][‘trait’]
- Similar syntax elements as in XPath are available e.g.:
  - root node: $
  - current node: @
  - wildcard: *
  - child operator: . or []
  - filter expression: ?()
  - ...

JSONPath [3]



JSONPath	Description
`$`	the root node
`@`	the current node
`.` or `[]`	child operator
n/a	parent operator
`..`	nested descendants
`*`	wildcard: all member values/array elements regardless of their names/indices
`[]`	subscript operator: index current node as an array (from 0)
`[,]`	Union operator JSONPath allows alternate(??) names or array indices as a set
`[start:end:step]`	array slice operator
`?()`	applies a filter expression
`()`	expression, e.g., for indexing

Source: https://www.ietf.org/archive/id/draft-ietf-jsonpath-base-03.html, IETF, last accessed: 30.03.2023

References

[1] https://www.w3.org/TR/xpath/, W3C, last accessed: 30.03.2023
[2] https://www.w3.org/TR/1999/REC-xpath-19991116/, W3C, last accessed: 30.03.2023
[3] https://www.w3.org/TR/2010/REC-xpath20-20101214/, W3C, last accessed: 30.03.2023
[4] https://www.w3.org/TR/2014/REC-xpath-30-20140408/, W3C, last accessed: 30.03.2023
[5] https://www.w3.org/TR/2017/REC-xpath-31-20170321/, W3C, last accessed: 30.03.2023
[6] https://www.ietf.org/archive/id/draft-ietf-jsonpath-base-03.html, IETF, last accessed: 30.03.2023