XML and JSON Theory

IT Learning &
Outsourcing Center

Lector: Dimitar Topuzov

www.pragmatic.bg

E-mail: dtopuzov@gmail.com
LinkedIn: http://bg.linkedin.com/pub/dimitar-topuzov/18/470/833/en

Content

- XML Theory
  - XML Tree
  - XML Syntax
  - XPath
  - XML Validation
- JSON Theory
  - Comparison with XML
  - JSON Syntax
  - JSON And JavaScript
- Demos
  - Tools for formatting XML and JSON
  - Syntax check tools
  - Tools for testing XPath expressions
- Exercises

We already know…

- What is a Web Service
- What is the difference between SOAP and REST
- How XML looks like
- How JSON looks like

We already know…

XML

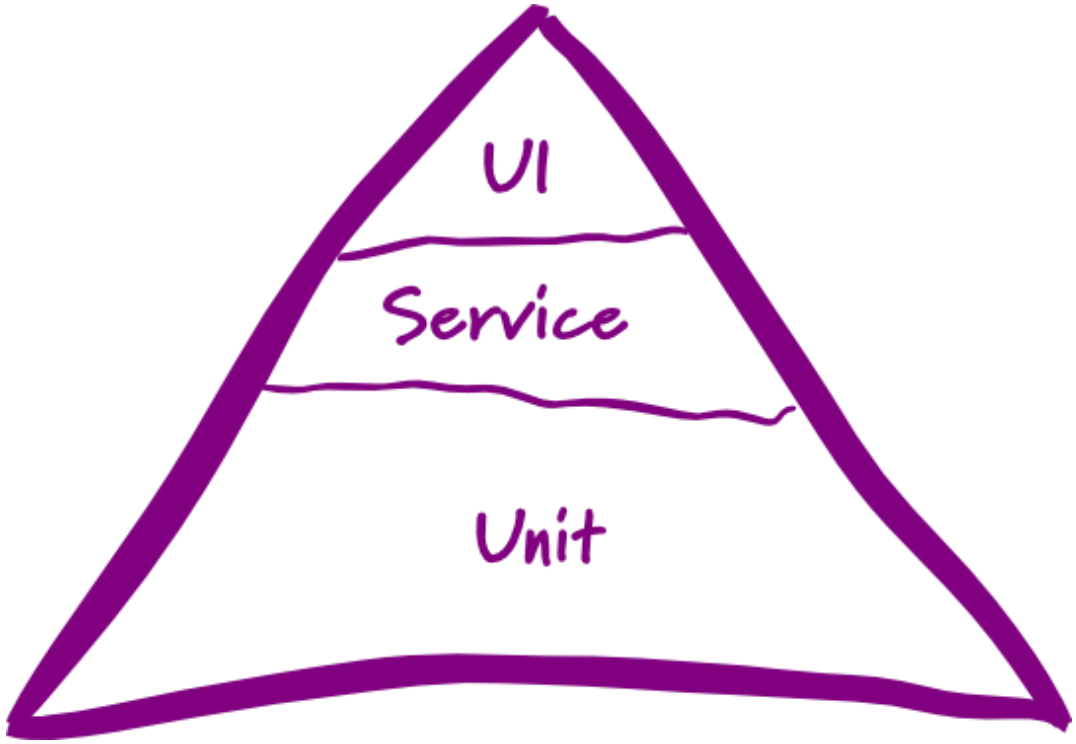
```
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

JSON

```
{"employees":[
  {"firstName":"John", "lastName":"Doe"},
  {"firstName":"Anna", "lastName":"Smith"},
  {"firstName":"Peter", "lastName":"Jones"}
]}
```

We already know…

- Why we test at Web Service level



Offtopic

- Great resources I found this week
  - Emanuil Slavov's blog
    - **The Three Pillars of Automated Testing**
    - **Fix Your Unstable Automated UI Tests**

What is XML

- XML stands for Extensible Markup Language
- XML was designed to describe data, not to display data
- XML tags are not predefined. You must define your own tags
- XML is designed to be self-descriptive
- XML is a W3C Recommendation
- XML Does Not DO Anything
  - It is just information wrapped in tags. Someone must write a piece of software to send, receive or display it.

XML Tree

- Sample XML

```
<root>
   <child>
            <subchild1>.....</subchild1>
            <subchild2>.....</subchild2>
   </child>
</root>
```

- The terms parent, child, and sibling are used to describe the relationships between elements.
  - Parent elements have children.
  - Children on the same level are called siblings.

XML Elements

- Definition
  - An XML element is everything from (including) the element's start tag to (including) the element's end tag.
- An element can contain
  - Other elements
  - Text
  - Attributes
  - Or a mix of all of the above…

XML Elements

- XML elements must follow these naming rules
  - Element names are case-sensitive
  - Element names must start with a letter or underscore
  - Element names cannot start with the letters xml (or XML, or Xml, etc)
  - Element names can contain letters, digits, hyphens, underscores, and periods
  - Element names cannot contain spaces

XML Attributes

- XML elements can have attributes, just like HTML.
- Attributes provide additional information about an element
- Examples:

```
<file type="gif">computer.gif</file>

<person gender="female">
```

Elements vs. Attributes

- Gender as attribute

```
<person gender="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

- Gender as element

```
<person>
  <gender>female</gender>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

- There are no rules about when to use attributes or when to use elements

Elements vs. Attributes

- Avoid using Attributes
  - attributes cannot contain multiple values (elements can)
  - attributes cannot contain tree structures (elements can)
  - attributes are not easily expandable (for future changes)
- Use elements for data
- Use attributes for information that is not relevant to the data
- Bad example:

```
<note day="10" month="01" year="2008"
to="Tove" from="Jani" heading="Reminder"
body="Don't forget me this weekend!">
</note>
```

XML Syntax (1)

- All XML Elements Must Have a Closing Tag
  - This is a paragraph.
- XML Tags are Case Sensitive
  - The tag <Letter> is different from the tag <letter>
  - Opening and closing tags must be written with the same case
- XML Elements Must be Properly Nested
  - Wrong: This text is bold and italic
  - Correct: This text is bold and italic
- XML Documents Must Have a Root Element
  - XML documents must contain one element that is the **parent** of all other elements

XML Syntax (2)

- XML Attribute Values Must be Quoted
  - Wrong: <note date=12/11/2007>
  - Correct: <note date="12/11/2007">
- Entity References
  - Wrong: <message>if salary < 1000 then</message>
  - Correct: <message>if salary < 1000 then</message>

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

XML Syntax (3)

- Comments in XML
  - <!-- This is a comment -->
- Well Formed XML
  - XML documents that conform to the syntax rules above are said to be "Well Formed" XML documents.

XPath

- XPath is a syntax for defining parts of an XML document
- XPath uses path expressions to navigate in XML documents
- XPath expressions can also be used in JavaScript, Java, XML Schema, PHP, Python, C and C++, and lots of other languages

XPath Syntax and Semantics (1)

/nodename

Selects all nodes with the name "nodename" from the root node

//nodename

Selects all nodes with the name "nodename" no matter where they are

.

Selects the current node

..

Selects the parent of the current node

@

Selects attributes

XPath Syntax and Semantics (2)

/nodename[1]

Selects first of all nodes with the name "nodename" from the root node

//nodename[last()]

Selects last of all nodes with the name "nodename" no matter where they are

//nodename[last()-1]

Selects last but one of all nodes with the name "nodename" no matter where they are

//nodename[position()<3]

Selects first two of all nodes with the name "nodename" no matter where they are

/nodename[1] | /nodename[last()]

Selects first AND last of all nodes with the name "nodename" from the root node

XPath Syntax and Semantics (3)

/nodename/text()

Selects inner text of all nodes with the name "nodename" from the root node

count(//nodename)

Selects count of all nodes with the name "nodename" no matter where they are

sum(//nodename/text())

Selects sum of all texts (assuming that text is number) of all nodes with name “nodename” no matter where they are

//nodename[contains(text(), 'XML')]

Selects nodes with name “nodename” which have text elements containing ‘XML’

//nodename[contains(@lang(), en')]

Selects nodes with name “nodename” which have “lang” attribute containing “en”

XPath Example (1)


```
<bookstore>

<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

</bookstore>
```

/bookstore/book[1]

Selects the first book element that is the child of the bookstore element

/bookstore/book[last()]

Selects the last book element that is the child of the bookstore element

/bookstore/book[last()-1]

Selects the last but one book element that is the child of the bookstore element

XPath Example (2)


```
<bookstore>

<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

</bookstore>
```

/bookstore/book[position()<2]

Selects book elements with position less than 2.

XPath Example (3)


```
<bookstore>

<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

</bookstore>
```

/bookstore/book[position()<3]

Selects book elements with position less than 3.

XPath Example (4)


```
<bookstore>

<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="CHILDREN">
  <title lang="bg">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

</bookstore>
```

```
//title[@lang]
```

Selects all the title elements that have an attribute named lang

XPath Example (5)


```
<bookstore>

<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="CHILDREN">
  <title lang="bg">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

</bookstore>
```

```
//title[@lang='en']
```

Selects all the title elements that have a "lang" attribute with a value of "en"

XML Validation

- XML validation is the process of checking if a document
  - Is well-formed
  - Follows a defined structure

XML Schema

- XML Schema
  - An XML Schema describes the structure of an XML document
  - An XML document with correct syntax is called "Well Formed".
  - An XML document validated against an XML Schema is both "Well Formed" and "Valid"
- Why Use an XML Schema?
  - With XML Schema, independent groups of people can agree on a standard for interchanging data.

XML Schema Example

- XML Schema Example

```
<xs:element name="note">

<xs:complexType>
  <xs:sequence>
    <xs:element name="to" type="xs:string"/>
    <xs:element name="from" type="xs:string"/>
    <xs:element name="heading" type="xs:string"/>
    <xs:element name="body" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

</xs:element>
```

JSON in Details

- JSON stands for JavaScript Object Notation
- JSON is a lightweight data-interchange format
- JSON is language independent *
  - JSON uses JavaScript syntax, but the JSON format is text only, just like XML
  - Text can be read and used as a data format by any programming language
- JSON is "self-describing" and easy to understand

Comparison with XML

- This is same
  - Both JSON and XML is "self describing" (human readable)
  - Both JSON and XML is hierarchical (values within values)
  - Both JSON and XML can be parsed and used by lots of programming languages
- This is different
  - JSON doesn't use end tag
  - JSON is shorter
  - JSON is quicker to read and write
  - JSON can use arrays
  - XML has to be parsed with an XML parser, JSON can be parsed by a standard JavaScript function

JSON Syntax

- The JSON syntax is a subset of the JavaScript syntax.
  - Data is in name/value pairs
  - Data is separated by commas
  - Curly braces hold objects
  - Square brackets hold arrays

JSON Data

- A name/value pair example:
  - "firstName":"John"
- JSON Values:
  - A number (integer or floating point)
  - A string (in double quotes)
  - A Boolean (true or false)
  - An array (in square brackets)
  - An object (in curly braces)
  - null

JSON Objects

- JSON objects are written inside curly braces
- JSON objects can contain multiple name/values pairs
- JSON objects example:
  - {"firstName":"John", "lastName":"Doe"}

JSON Arrays

- JSON arrays are written inside square brackets.
- JSON array can contain multiple objects
- JSON array example

```
"employees":[
  {"firstName":"John", "lastName":"Doe"},
  {"firstName":"Anna", "lastName":"Smith"},
  {"firstName":"Peter","lastName":"Jones"}
]
```

  - In the example above, the object "employees" is an array containing three objects. Each object is a record of a person (with a first name and a last name).

JSON And JavaScript

- JSON syntax is derived from JavaScript object notation
  - Very little extra software is needed to work with JSON within JavaScript.
- With JavaScript you can create an array of objects and assign data to it, like this:

```
var employees = [
   {"firstName":"John", "lastName":"Doe"},
   {"firstName":"Anna", "lastName":"Smith"},
   {"firstName":"Peter","lastName": "Jones"}
];
```

- JavaScript object array can be accessed like this:

```
// Returns John Doe
employees[0].firstName + " " + employees[0].lastName;
```

- Data can be modified like this

```
// Modify Johnto Gilbert
employees[0].firstName = "Gilbert";
```

Additional Resources

- XML
  - http://www.w3schools.com/xml
- JSON
  - http://www.w3schools.com/json

Questions

Demos

- Tools for formation JSON/XML and syntax check
  - http://jsonlint.com/
  - http://www.freeformatter.com/
  - http://chris.photobooks.com/xml/default.htm

Demos

- Online XPath tester
  - http://www.freeformatter.com/
  - http://codebeautify.org/Xpath-Tester
- Online XPath generator
  - http://xmltoolbox.appspot.com/
- Online JSON selector
  - http://jsonselector.com/
- Online JSON to XML and XML to JSON convertors
  - http://www.utilities-online.info/xmltojson
  - http://www.freeformatter.com/xml-to-json-converter.html

Exercises

- Define XML and JSON objects with data for several cars
  - Car object should contain following info
    - Make
    - Model
    - Year
    - Engine
  - Engine object should contain following info
    - FuelType
    - Cylinders
    - Displacement

Exercises

- Locate following items in Lecture-03-ParentChild.xml file
  - All 'child' elements
  - All child elements with name 'Child_1‘
  - All 'child' elements with id <= 3
  - Texts of all 'child' elements
  - Text of first 'child' of last 'Parent'
  - Text of last 'child' of last 'Parent '
  - Texts of first and last 'child ' under first 'Parent'

Exercises

- Locate following items in Lecture-03-Bookstore.xml file
  - Price of books
  - Count of book elements
  - Sum of prices of all books
  - Title of books which contains 'XML'
  - Books with Bulgarian edition (lang attribute contains bg)

Homework

- Locate following items in Lecture-03-Weather-5days file:
  - <country> element
  - Text of <country> element
  - Periods <time> elements when windSpeed will be "Light breeze"
  - mps attributes of all windSpeed elements
  - sum of mps attribute values of all windSpeed elements
  - count of mps attribute values of all windSpeed elements