

Lecture Note 10

XML, AJAX, JSON

By Gabriel Fung, PhD
School of Information Technology and Electrical Engineering
The University Of Queensland

What is XML?

- eXtensible Markup Language
 - A markup language much like HTML
 - One of the most important technologies to support WWW
- Main difference between XML and HTML
 - HTML is about displaying information, while XML is about describing information.
 - XML was designed to describe data and to focus on what data is.
 - HTML was designed to display data and to focus on how data looks.
 - XML is not a replacement for HTML (by XHTML).

XML is a Meta-Language

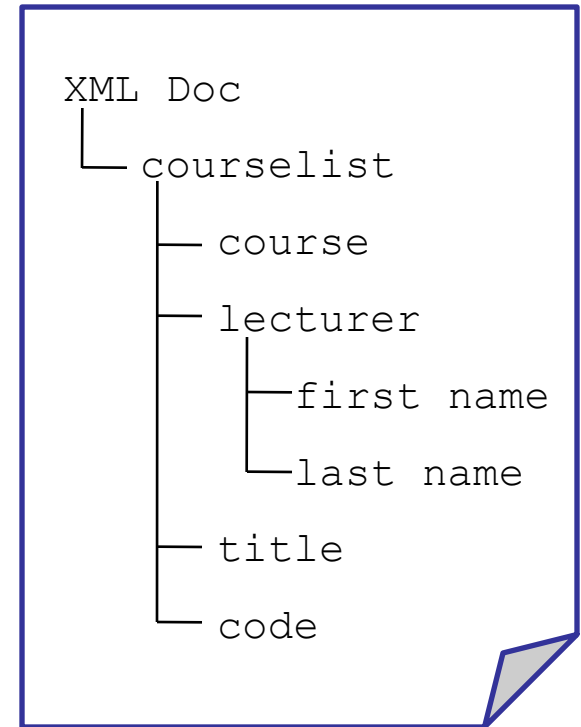
- A “language” used to define other languages
 - XML itself has no tags
 - Cannot be used directly to markup documents
 - It is used to define a set of tags
 - E.g. XHTML
- What is a “language”?
 - A set of tags
 - Vocabulary: elements and attributes
 - Grammar: where a tag can appear and how they can appear

Where to use XML

- XML is going to be everywhere.
- XML is ideal for:
 - e-commerce (in particular B2B), content management, Web Services, distributed computing, peer-to-peer (P2P) networking and the Semantic Web...
- However, it's huge in space
 - 3-20 times larger comparing to binary format
- Some people (not me) claim:
 - XML will be as important to the future of the Web as HTML has been to the foundation of the Web and that XML will be the most common tool for all data manipulation and data transmission.
 - Well... This can be true or not true
 - How about JSON? (we will discuss it shortly)

An XML Document

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- my first XML doc -->
<courselist>
  <course level="medium">
    <lecturer>
      <firstname>Gabriel</firstname>
      <lastname>Fung</lastname>
    </lecturer>
    <title>Web Information Systems</title>
    <code>INFS3202/INFS7202</code>
  </course>
</courselist>
```



`courselist` is the root element. `lecturer`, `title` and `code` are child `course`.
`lecturer`, `title` and `code` are siblings because they have the same parent.

XML Syntax

- An XML document are composed of:
 - An XML declaration: `<?xml version = "1.0" encoding="ISO-8859-1"?>`
 - This is optional in fact... but is a habit.
 - Used to identify the XML version and encoding schema to which the doc conforms
 - XML mark-up (6 kinds):
 - Elements (and attributes)
 - Entity references
 - Comments: `<!-- what ever you want to say -->`
 - Processing instructions: `<? instruction options ?>`
 - Marked sections
 - Document type declarations
 - Content
 - Well, this is optional in fact...

Elements and Attributes

- Elements (or Tags) (e.g. `<course> . . . </course>`)
 - All tags must be closed and nested properly (as in XHTML)
 - Empty tags are valid (E.g. `<course />`)
- Attributes (e.g. `<course level="undergraduate">`)
 - Additional information about an element
 - All values must be quoted (even for numbers)
 - Elements and attributes can be named in any way, except:
 - Must not start with a number or punctuation character or XML or Xml
 - Cannot contain spaces, the colon (:), greater-than (>), or less-than (<)
 - Avoid using the hyphen (-) and period (.)
 - No length limit and case sensitive
 - No names are reserved for XML (namespaces can solve naming conflicts)

Document Type Definitions (1/2)

- A DTD provides a means for defining what XML mark-ups can occur in an XML document
 - A mechanism to guarantee that an XML document complies with a well-defined set of rules for document structure and content
 - A parser reads a DTD and determines if an XML doc conforms to the DTD or not
 - DTD is optional for an XML doc, but is highly recommended
- Syntax (write in the first line of the file):
 - `<!DOCTYPE root [doctype-declaration...]>`
 - determines the name of the root element and contains the document type declarations
 - E.g.:
 - `<!DOCTYPE course SYSTEM "course.dtd">`
 - `<!DOCTYPE course PUBLIC "http://www.xxx....">`

Document Type Definitions (2/2)

- The process of checking to see if an XML document conforms to a schema is called validation
 - It is separate from XML's core concept of syntactic well-formedness.
 - All XML documents must be well-formed.
 - But it is not required that a document be valid unless the XML parser is "validating,"
 - in which case the document is also checked for conformance with its associated schema.

A DTD Example

- Consider:

- `<!-- course.dtd -->`
`<!ELEMENT courselist (course*)>`
`<!ELEMENT course (lecturer?, title, code)>`
`<!ELEMENT lecturer (firstname, lastname)>`
`<!ELEMENT firstname (#PCDATA)>`
`<!ELEMENT lastname (#PCDATA)>`
`<!ELEMENT title (#PCDATA)>`
`<!ELEMENT code (#PCDATA)>`
`<!ATTLIST course type (INFS|COMP|COMM) #REQUIRED>`
`<!ATTLIST course level (UG|PG|both) "both">`

- Occurrence constraints:

- *: 0 to many, +: 1 to many, ?: 0 or 1

- #PCDATA (parsable character data):

- element contains character data that the XML parser will interpret

- #CDATA (character data):

- element contains character data that the XML parser will not interpret

XML Schema

- XML Schema is a replacement for DTD pushed by W3C (final recommendation in mid 2001)
 - Also for specifying a valid construction of XML docs
 - It is an XML application
 - Thus can handle namespaces and has XML parser support
 - Elements can be inherited from other schemas
 - It describes relationships as well as structuring rules
 - It has more control over data types and restrictions
- An example:
 - <http://www.brics.dk/~amoeller/XML/schemas/xmlschema-example.html>
 - Reference: <http://www.brics.dk/~amoeller/XML/schemas/>

Highlights of XML Schemas

- Advantages over DTDs
 - More datatypes
 - 44+ versus 10
 - Written in the same syntax as instance documents
 - Object-oriented
 - Can express sets, i.e., can define the child elements to occur in any order
 - Can specify element content as being unique (keys on content) within a region
 - Can define multiple elements with the same name but different content
 - Can define elements with nil content
 - Can define substitutable elements

XQuery

- It is used to query against XML document:
 - Oracle and MySQL are relational databases
 - There are some databases called XML databases.
 - Store XML documents.
 - To query relational databases, we use SQL.
 - To query XML databases, we use XQuery.
 - E.g. Who teach the course INFS3202?
- A standard to query XML docs directly
 - <http://www.w3.org/TR/xquery-full-text/>

XQuery: Example

```
<courselist>
  {
    FOR $g IN document("courses.xml")/courselist/course
    WHERE $g/code/text() = "INFS3202"
    RETURN
      <myCourse>
        {$g/lecturer/lastname}
        {$g/lecturer/lastname}
      </myCourse>
  }
</courselist>
```

What is JSON?

- JSON stands for JavaScript Object Notation
 - It just like XML!
 - It is a lightweight data-interchange format.
 - It is easy for humans to read and write.
 - It is easy for machines to parse and generate.
 - It is NOT a mark-up language
- More information:
 - <http://www.json.org/>
 - <http://www.JSON.org/xml2006.ppt>

XML vs. JSON

- An XML:

- ```
<person>
 <name>Simon</name>
 <age currentYear="2009">25</age>
 <height>1.68</height>
 <urls>
 <url>http://simonwillison.net/</url>
 <url>http://www.flickr.com/photos/simon/</url>
 </urls>
</person>
```

- A JSON:

- ```
person={"name": "Simon",
  "age": {"currentYear": 2009, "value": 25},
  "height": 1.68,
  "urls": ["http://simonwillison.net/",
    "http://www.flickr.com/photos/simon/"]}
}
```

Why JSON?

- Personally:
 - JSON far more simple. Easier to read. Less bulky.
 - Use JSON for:
 - Simple Records
 - Use XML for:
 - Large complex document
- More information:
 - Many examples:
 - <http://www.json.org/example.html>
 - Some discussions:
 - <http://www.infoq.com/news/2006/12/json-vs-xml-debate>
 - <http://simonwillison.net/2006/Dec/20/json/>
 - <http://blogs.msdn.com/mikechampion/archive/2006/12/21/the-json-vs-xml-debate-begins-in-earnest.aspx>



JSON is Good!

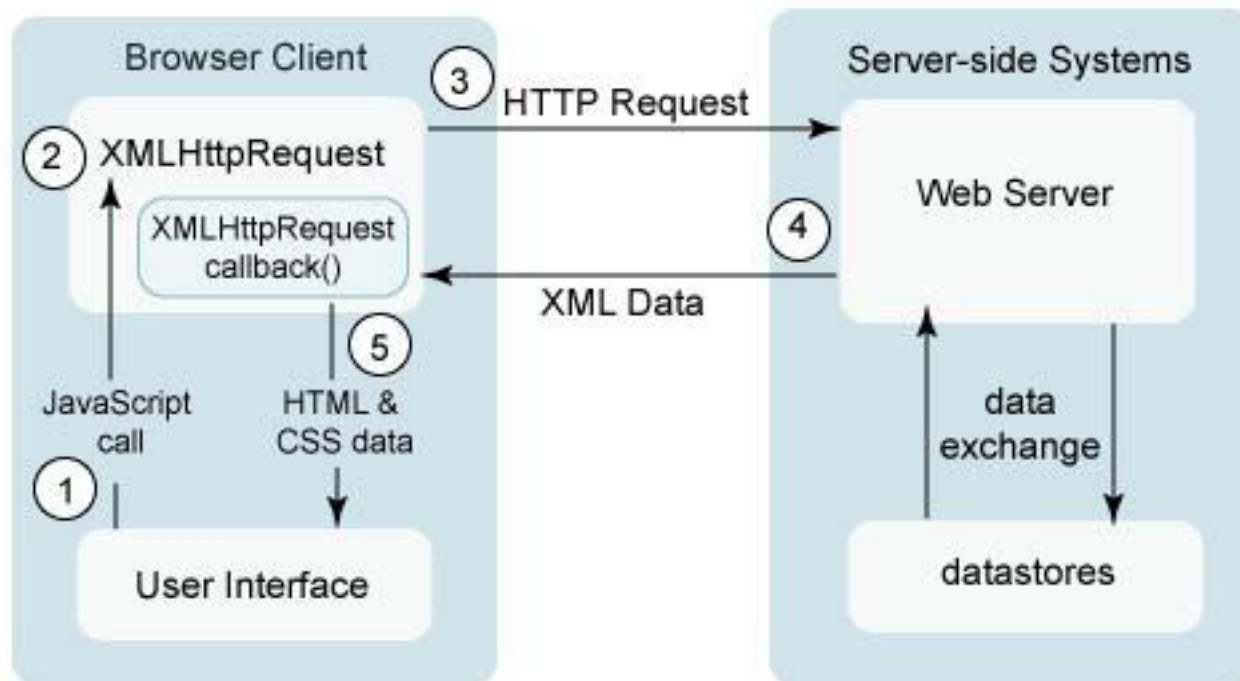
- It's simultaneously human- and machine-readable format
- It has support for Unicode, allowing almost any information in any human language to be communicated
- The self-documenting format that describes structure and field names as well as specific values
- The strict syntax and parsing requirements that allow the necessary parsing algorithms to remain simple, efficient, and consistent
 - Theoretically, parsing (i.e. read) a JSON document is far far far far more faster than parsing an XML document.
- The ability to represent the most general computer science data structures: records, lists and trees

Some Critics Against JSON

- JSON Doesn't Have Namespaces
 - JSON uses context to avoid ambiguity, just as programming languages do.
- JSON Has No Validator
 - Ultimately, every application is responsible for validating its inputs. This cannot be delegated.
 - More and more validator are developed.
- JSON Is Not Extensible
 - It doesn't need to be.
 - JSON is flexible. It can represent any non-recurrent data structure as is.
 - New fields can be added to existing structures without obsoleting existing programs.

What is AJAX?

- AJAX stands for:
 - Asynchronous JavaScript And XML
- AJAX is NOT a technology, but is a combination of existing technologies and a way of thinking.



Why AJAX?

- Looks good (no whole page refresh)
- Bandwidth usage – only download the part of the page you need updated
- Feels more interactive and responsive – like a desktop application
- Some AJAX applications:
 - Google Map
 - Google Suggest
 - Facebook
 - My web
 - There are many many AJAX applications...



Why Not AJAX?

- Browser Integration
 - Breaks the Back button
 - No history
 - Hard to bookmark
 - Browser loading icon won't activate on requests
- Response-time concerns:
 - Network latency to many requests
 - Reliance on JavaScript
 - This is often implemented differently by different browsers or versions of a particular browser
- No good practical way to upload files using AJAX
 - Well... remember that this is not a problem of AJAX, just a limitation.

When to Use AJAX?

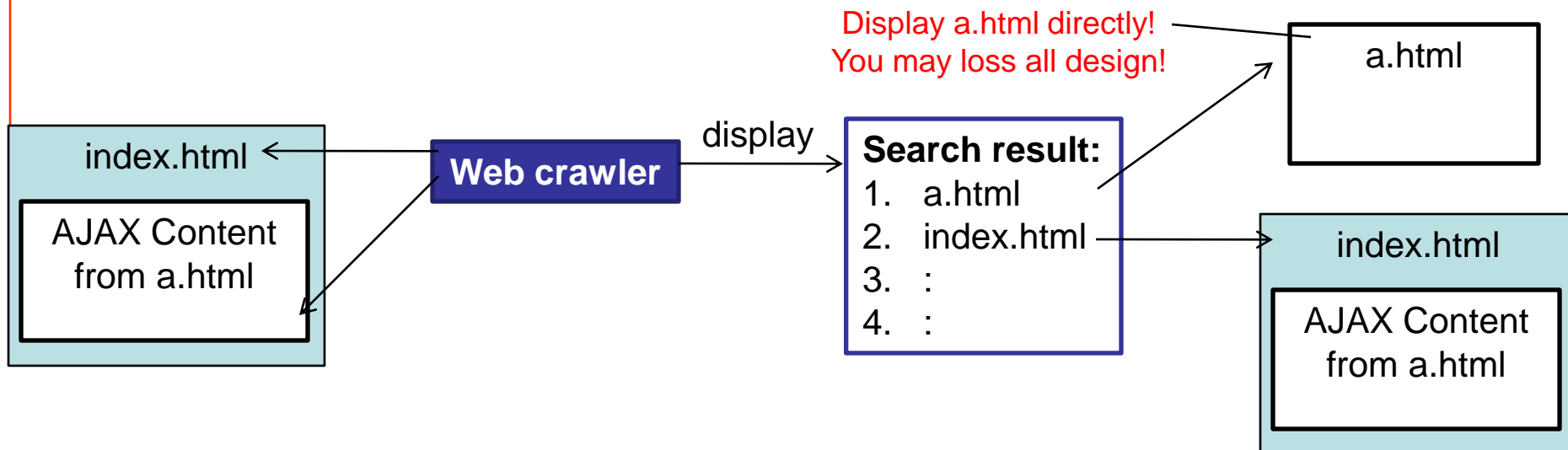
- When only a small portion of a site needs to change at a time. Good examples are:
 - Adding comments inline
 - E.g. Commenting on a blog post
 - Adding/Deleting elements
 - E.g. Deleting a row from a table – no need to reload the whole page. Just send a delete command using Ajax and remove the `<tr>`
 - Complex validation
 - E.g. Checking user input against database records
 - Checking for changes at the server without bothering the user
 - E.g. Chat rooms.

When Not to use AJAX?

- If a large portion of the information in the site you want to change
 - Then you better not use AJAX
 - P.s. “Better not use” but not “Never use”!!!
- Just for the sake of it
 - The biggest newcomer mistake is using Ajax for absolutely everything. It is not supposed to replace conventional browsing
- To “reinvent” browsing conventions
 - The average user understands the idea of Back, Forwards, clicking links, adding bookmarks, etc.
 - Do not make it hard for users to navigate your site

Search Engine Problem (1/2)

- One of the major problem of AJAX is indexing:
 - It is difficult for a search engine to index your site properly!
 - Web crawlers will only search for HREF, SRC or some other link tags.
 - Normal web crawler will not search for AJAX JavaScript proxies.
 - Some “abnormal” web crawler exists, but most are still under research status.
 - If you AJAX refer to another HTML document...
 - The web crawlers will index “the other HTML document” as well:



Search Engine Problem (2/2)

- Simple Solution:
 - If you want to build web sites that will be reachable through search engines you have to write two sites
 - One with the usage of AJAX for the human visitor
 - One for web crawlers that will need the complete HTML/text of the page.