CSE 154

LECTURE 23: XML

Storing structured data in arbitrary text formats (bad)

```
My note:
BEGIN
  FROM: Alice Smith (alice@example.com)
  TO: Robert Jones (roberto@example.com)
  SUBJECT: Tomorrow's "Birthday Bash" event!
  MESSAGE (english):
    Hey Bob,
    Don't forget to call me this weekend!
  PRIVATE: true
END                                                                  XML
```

- Many apps make up their own custom text format for storing structured data.
- We could also send a file like this from the server to browser with Ajax.
- What's wrong with this approach?

XML: A better way of storing data

```xml
<?xml version="1.0" encoding="UTF-8"?>
<note private="true">
  <from>Alice Smith (alice@example.com)</from>
  <to>Robert Jones (roberto@example.com)</to>
  <subject>Tomorrow's "Birthday Bash" event!</subject>
  <message language="english">
    Hey Bob, Don't forget to call me this weekend!
  </message>
</note>
```
XML

- **eXtensible Markup Language (XML)** is a format for storing nested data with tags and attributes
- essentially, it's HTML, but you can make up any tags and attributes you want
- lots of existing data on the web is stored in XML format

What is XML?

- XML is a "skeleton" for creating markup languages
  - you decide on an XML "language" of tags and attributes that you want to allow in your app
  - XML syntax is mostly identical to HTML's: `<element attribute="value">content</element>`
  - the HTML/XML tag syntax is a nice general syntax for describing hierarchical (nested) data
- when you choose to store data in XML format (or access external XML data), you must decide:
  - names of tags in HTML: `h1`, `div`, `img`, etc.
  - names of attributes in HTML: `id`/`class`, `src`, `href`, etc.
  - rules about how they go together in HTML: inline vs. block-level elements
- XML presents complex data in a human-readable, "self-describing" form

Anatomy of an XML file

```xml
<?xml version="1.0" encoding="UTF-8"?>      <!-- XML prolog -->
<note private="true">                       <!-- root element -->
  <from>Alice Smith (alice@example.com)</from>
  <to>Robert Jones (roberto@example.com)</to>
  <subject>Tomorrow's "Birthday Bash" event!</subject>
  <message language="english">
    Hey Bob, Don't forget to call me this weekend!
  </message>
</note>                                                                          XML
```

- begins with an `<?xml ... ?>` header tag (**prolog**)
- has a single **root element** (in this case, `note`)
- tag, attribute, and comment syntax is just like HTML

Uses of XML

- XML data comes from many sources on the web:
  - **web servers** store data as XML files
  - **databases** sometimes return query results as XML
  - **web services** use XML to communicate
- XML is the *de facto* universal format for exchange of data
- XML languages are used for music, math, vector graphics
- popular use: RSS for news feeds & podcasts

What tags are legal in XML?

- *any tags you want!* examples:
  - a library might use tags `book`, `title`, `author`
  - a song might use tags `key`, `pitch`, `note`
- when designing XML data, *you* choose how to best represent the data
  - large or complex pieces of data become tags
  - smaller details and metadata with simple types (integer, string, boolean) become attributes

```xml
<measure number="1">
  <attributes>
    <divisions>1</divisions>
    <key><fifths>0</fifths></key>
    <time><beats>4</beats></time>
    <clef>
      <sign>G</sign><line>2</line>
    </clef>
  </attributes>
  <note>
    <pitch>
      <step>C</step>
      <octave>4</octave>
    </pitch>
    <duration>4</duration>
    <type>whole</type>
  </note>
</measure>                                  XML
```

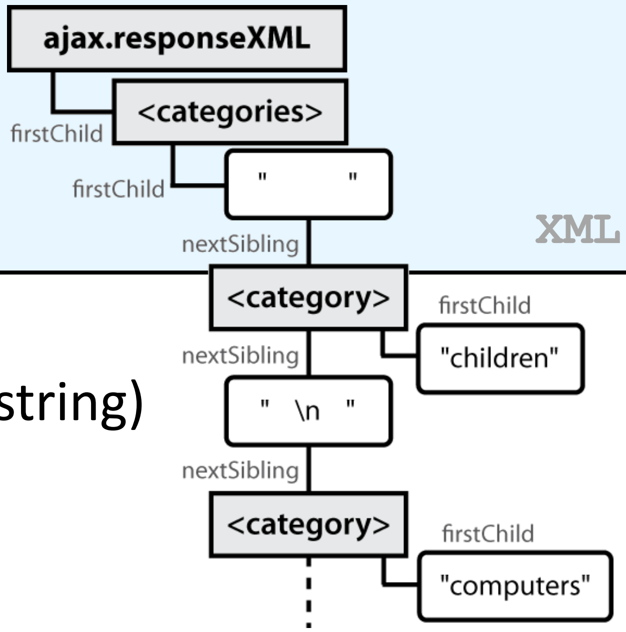
XML and Ajax

- web browsers can display XML files, but often you instead want to fetch one and analyze its data
- the XML data is fetched, processed, and displayed using Ajax
  - (XML is the "X" in "Ajax")
- It would be very clunky to examine a complex XML structure as just a giant string!
- luckily, the browser can break apart (**parse**) XML data into a set of objects
  - there is an XML DOM, similar to the HTML DOM

Fetching XML using Ajax (template)

```
var ajax = new XMLHttpRequest();
ajax.onload = functionName;
ajax.open("GET", url, true);
ajax.send();
...
function functionName() {
  do something with this.responseXML;
}
```

XML

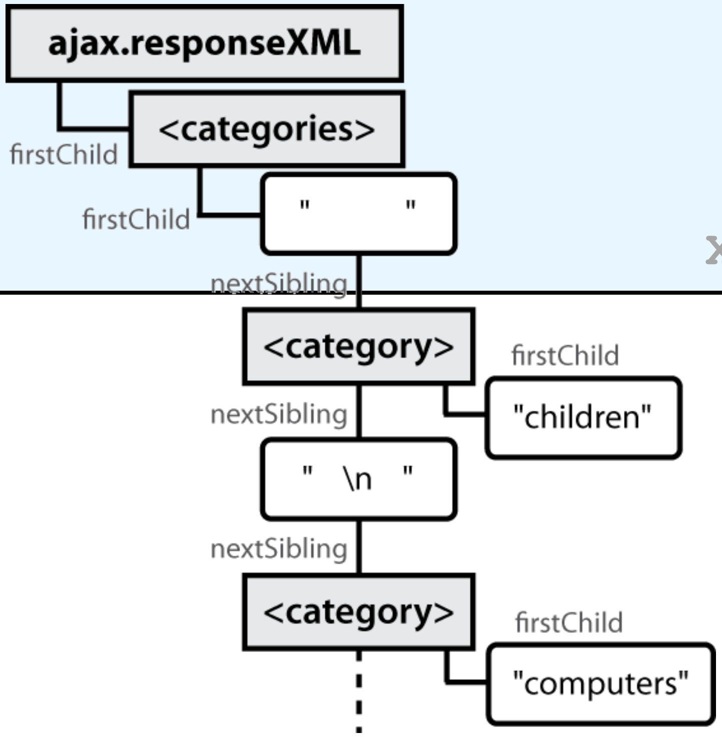


- `this.responseText` contains the data in plain text (a string)
- `this.responseXML` is a parsed XML DOM tree object
  - it has methods very similar to HTML DOM objects

XML DOM tree structure

```
<?xml version="1.0" encoding="UTF-8"?>
<categories>
  <category>children</category>
  <category>computers</category>
  ...
</categories>
```

XML



- the XML tags have a tree structure
- DOM nodes have parents, children, and siblings
- each DOM node object has properties/methods for accessing nearby nodes

Interacting with XML DOM nodes

To get an array of nodes:

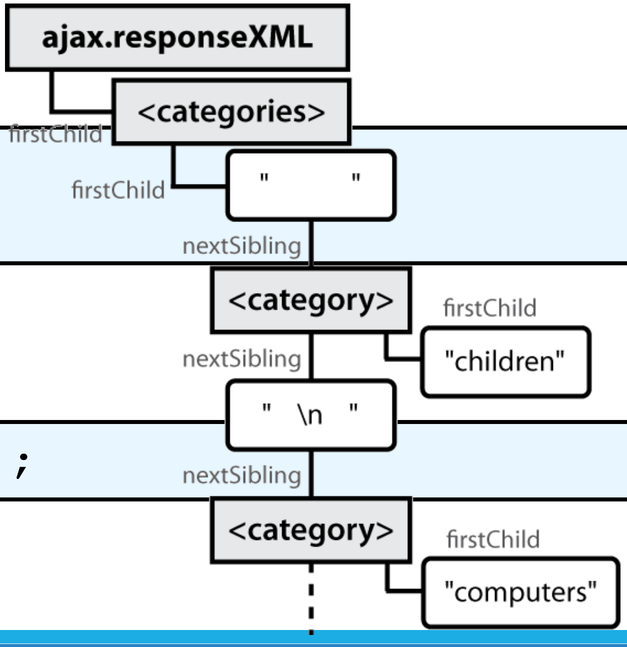
```
var elms = node.getElementsByTagName("tag");
var elms = node.querySelectorAll("selector");  // all elements
var elm  = node.querySelector("selector");     // first element      XML
```

To get the text inside of a node:

```
var text = node.textContent;  // or,
var text = node.firstChild.nodeValue;                                 XML
```

To get the value of a given attribute on a node:

```
var attrValue = node.getAttribute("name");                           XML
```



Differences from HTML DOM

Don't usually use `getElementById` because XML nodes don't have IDs or classes.

```
var div = ~~document.getElementById("main");~~                JS
```

Can't get/set the text inside of a node using `innerHTML`:

```
var text = ~~div.innerHTML;~~                JS
```

Can't get an attribute's value using `.`*attributeName*:

```
var imageUrl = ~~document.getElementById("myimage").src;~~                JS
```

Ajax XML DOM example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<employees>
  <lawyer money="99999.00" />
  <janitor name="Ed"> <vacuum model="Hoover" /> </janitor>
  <janitor name="Bill">no vacuum, too poor</janitor>
</employees>
```

XML

```javascript
// how much money does the lawyer make?
var lawyer = this.responseXML.querySelector("lawyer");
var salary = parseFloat(lawyer.getAttribute("money"));      // 99999.0
// array of 2 janitors
var janitors = this.responseXML.querySelectorAll("janitor");
var vacModel = janitors[0].querySelector("vacuum").getAttribute("model");
var excuse = janitors[1].textContent;  // "no vacuum, too poor"
```

- How would we find out the first janitor's name? *(use the Console)*
- How would we find out how many janitors there are?
- How would we find out how many janitors have vs. don't have vacuums?

Larger XML file example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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="computers">
    <title lang="en">XQuery Kick Start</title>
    <author>James McGovern</author>
    <year>2003</year><price>49.99</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year><price>29.99</price>
  </book>
  <book category="computers">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year><price>39.95</price>
  </book>
  <!-- you can play with this XML in the console
as 'this.responseXML2' -->
</bookstore>
```

Navigating node tree example

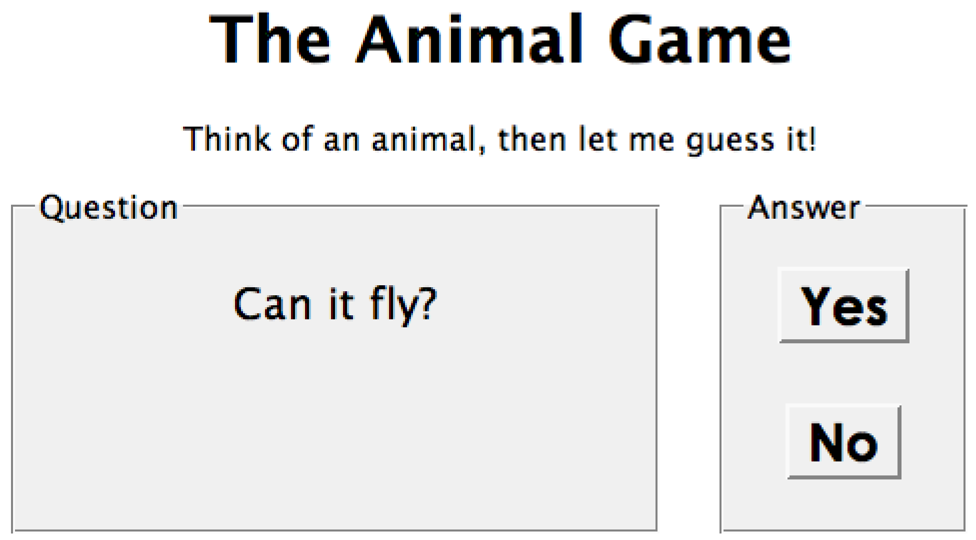
```
// make a paragraph for each book about computers
var books = this.responseXML.getElementsByTagName("book");
for (var i = 0; i < books.length; i++) {
  var category = books[i].getAttribute("category");
  if (category == "computers") {
    // extract data from XML
    var title = books[i].querySelector("title").textContent;
    var author = books[i].querySelector("author").textContent;

    // make an HTML <p> tag containing data from XML
    var p = document.createElement("p");
    p.innerHTML = title + ", by " + author;
    document.body.appendChild(p);
  }
}
```
JS

Exercise: Late day distribution

- Write a program that shows how many students turn homework in late for each assignment.
- Data service here: http://webster.cs.washington.edu/cse154/hw/hw.php
  - parameter: `assignment=hw`*N*

Exercise: Animal game

- Write a program that guesses which animal the user is thinking of. The program will arrive at a guess based on the user's responses to yes or no questions. The questions come from a web app named `animalgame.php`.



The Animal Game

Think of an animal, then let me guess it!

Question

Can it fly?

Practice problem: Animal game (cont'd)

The data comes in the following format:

```
<node nodeid="id">
  <question>question text</question>
  <yes nodeid="id" />
  <no nodeid="id" />
</node>                                                         XML
```

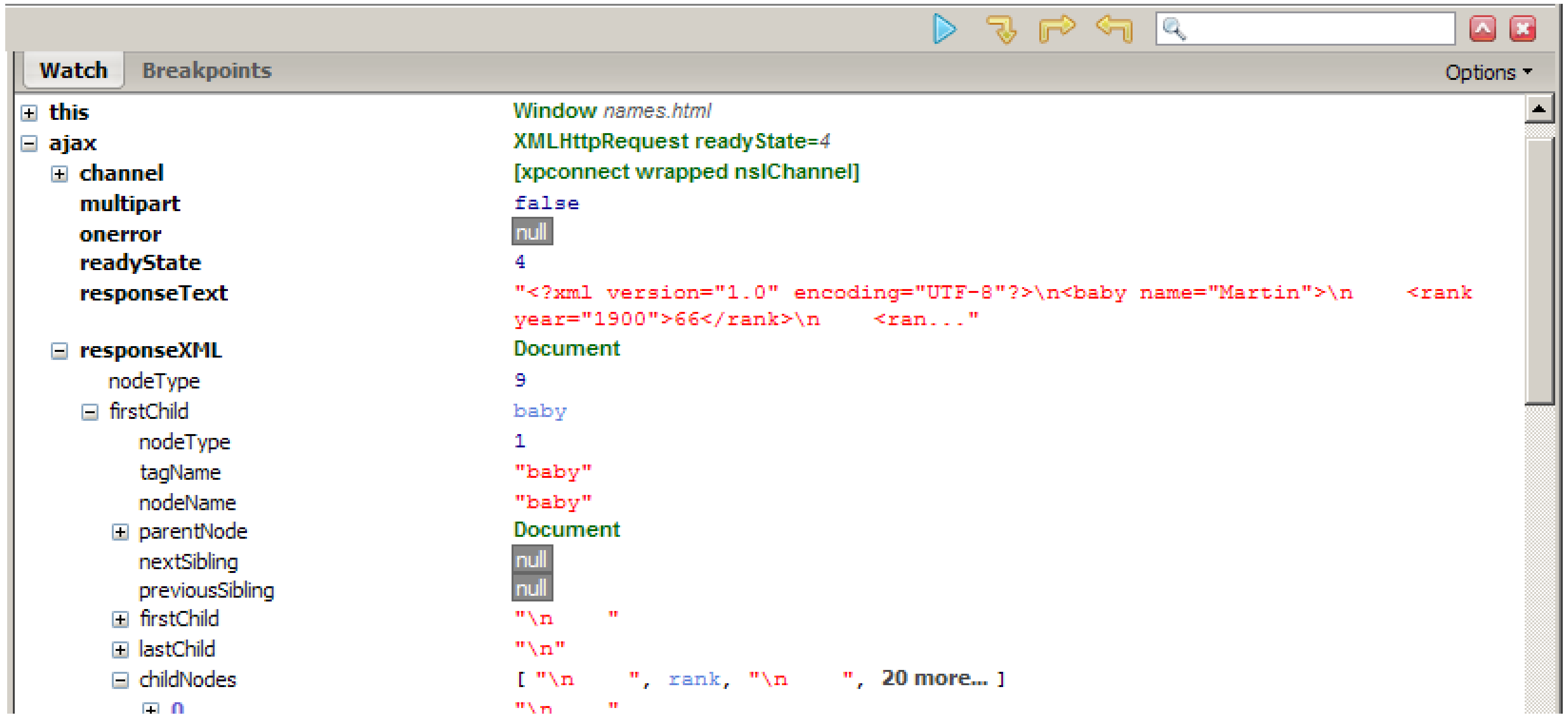
```
<node nodeid="id">
  <answer>answer text</answer>
</node>                                                         XML
```

- to get a node with a given id: `animalgame.php?nodeid=`*id*
- start by requesting the node with `nodeid` of `1` to get the first question

Attacking the problem

- Questions we should ask ourselves:
- How do I retrieve data from the web app? (what URL, etc.)
- Once I retrieve a piece of data, what should I do with it?
- When the user clicks "Yes", what should I do?
- When the user clicks "No", what should I do?
- How do I know when the game is over? What should I do in this case?

Debugging responseXML in Firebug



```
⊞ this                      Window names.html
⊟ ajax                      XMLHttpRequest readyState=4
  ⊞ channel                 [xpconnect wrapped nsIChannel]
    multipart               false
    onerror                 null
    readyState              4
    responseText            "<?xml version="1.0" encoding="UTF-8"?>\n<baby name="Martin">\n    <rank
                            year="1900">66</rank>\n    <ran..."
  ⊟ responseXML             Document
      nodeType              9
    ⊟ firstChild            baby
        nodeType            1
        tagName             "baby"
        nodeName            "baby"
      ⊞ parentNode          Document
        nextSibling         null
        previousSibling     null
      ⊞ firstChild          "\n    "
      ⊞ lastChild           "\n"
      ⊟ childNodes          [ "\n    ", rank, "\n    ", 20 more... ]
        ⊞ 0                 "\n    "
```

- can examine the entire XML document, its node/tree structure

Full list of XML DOM properties

- properties:
  - **`nodeName`**, **`nodeType`**, **`nodeValue`**, **`attributes`**
  - `firstChild`, `lastChild`, `childNodes`, `nextSibling`, `previousSibling`, `parentNode`
- methods:
  - `getElementById`, **`getElementsByTagName`**, `querySelector`, **`querySelectorAll`**, **`getAttribute`**, **`hasAttribute`**,**`hasChildNodes`**
  - `appendChild`, `insertBefore`, `removeChild`, `replaceChild`
- full reference

Schemas and Doctypes

- "rule books" describing which tags/attributes you want to allow in your data
- used to *validate* XML files to make sure they follow the rules of that "flavor"
  - the W3C HTML validator uses an HTML schema to validate your HTML (related to `<!DOCTYPE html>` tag)
- these are optional; if you don't have one, there are no rules beyond having well-formed XML syntax
- for more info:
  - W3C XML Schema
  - Document Type Definition (DTD) ("doctype")