Extensible Markup Language (XML)

CSE 190 M (Web Programming), Spring 2008
University of Washington

What is XML?

- a specification for creating languages to store data; used to share data between systems
- a basic syntax of tags & attributes
- languages written in XML specify tag names, attribute names, and rules of use
- Example: XHTML is a "flavor" of XML
    - an adaptation of old HTML to fit XML's syntax requirements
    - XML specifies tag syntax: `<... ...="..."></...>`
    - HTML contributes tag names (e.g. `h1`, `img`) and attributes (`id`/`class` on all elements, `src`/`alt` on `img` tag)

An example XML file

```xml
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <subject>Reminder</subject>
  <message language="english">
    Don't forget me this weekend!
  </message>
</note>
```

XML syntax:

- begins with an xml header tag, then a single **document tag** (in this case, `note`)
- tag, attribute, and comment syntax is identical to XHTML's

What tags are legal in XML?

- *any tag you want*; the person storing the data can make up their own tag structure
- example: a person storing data about email messages may want tags named `to`, `from`, `subject`
- example: a person storing data about books may want tags named `book`, `title`, `author`
- "Garden State" XML: if you're feeling unoriginal, make up some XML nobody's ever done before
    - `<bloop bleep="flibbetygibbet">quirkleblat</bloop>`

Schemas

- **schema**: an optional set of rules specifying which tags and attributes are valid, and how they can be used together
- used to *validate* XML files to make sure they follow the rules of that "flavor"
    - XHTML has a schema; W3C validator uses it to validate
    - doctype at top of XHTML file specifies schema
- two ways to define a schema:
    - Document Type Definition (DTD)
    - W3C XML Schema
- (we won't use schemas in this course)

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 languages are used for music, math, vector graphics
- popular use: RSS for news feeds & podcasts

Pros and cons of XML

- pro:
    - easy to read (for humans and computers)
    - standard format makes automation easy
    - don't have to "reinvent the wheel" for storing new types of data
    - international, platform-independent, open/free standard
    - can represent almost any general kind of data (record, list, tree)
- con:
    - bulky syntax/structure makes files large; can decrease performance
        - example: quadratic formula in MathML
    - can be hard to "shoehorn" data into an intuitive XML format
        - won't need to know how for this class

Fetching XML using AJAX (template)

```
  new Ajax.Request(
    "url",
    {
      method: "get",
      onSuccess: functionName
    }
  );
  ...

function functionName(ajax) {
  do something with ajax.responseXML;
}
```

- `ajax.response`*`Text`* still contains XML code, but in plain text
- `ajax.response`*`XML`* is a pre-parsed DOM object representing the XML file (more useful)

Using XML data in a web page

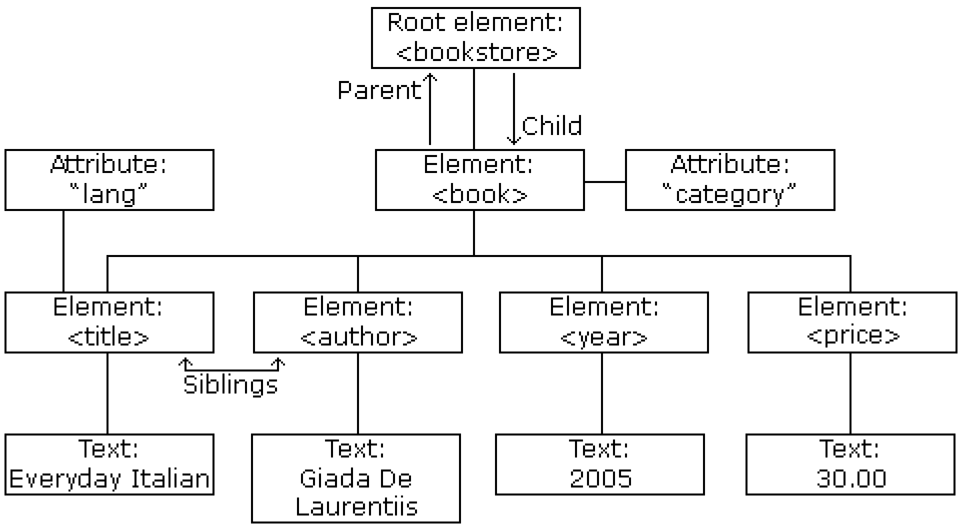
- custom flavor of XML needs to be converted to XHTML, then injected into page
- we will transform using Javascript XML DOM
- basic technique:
  1. fetch XML data using Ajax
  2. examine the `response`*`XML`* object, using DOM methods and properties
  3. extract data from XML elements and wrap them in HTML elements
  4. inject HTML elements into web page
- other ways to transform XML (not covered): CSS, XSLT

Recall: Javascript XML (XHTML) DOM

All of the DOM properties and methods we already know can be used on XML nodes:

- properties:
  - `firstChild`, `lastChild`, `childNodes`, `nextSibling`, `previousSibling`, `parentNode`
  - **`nodeName`**, **`nodeType`**, **`nodeValue`**, **`attributes`**
- methods:
  - `appendChild`, `insertBefore`, `removeChild`, `replaceChild`
  - **`getElementsByTagName`**, **`getAttribute`**, **`hasAttributes`**, **`hasChildNodes`**
- Prototype methods:
  - `ancestors`, `childElements`, `descendants`, `firstDescendant`, `descendantOf`, `next`, `previous`, `siblings`, `previousSiblings`, `nextSiblings`, `adjacent`

XML DOM tree structure



- the XML tags have a tree structure
- DOM nodes have parents, children, and siblings

Analyzing a fetched XML file using DOM

Assume the following XML file is returned via an Ajax request:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<foo bloop="bleep">
  <bar/>
  <baz><quux/></baz>
  <baz><xyzzy/></baz>
</foo>
```

XML

We can use DOM properties and methods on `ajax.responseXML`:

```javascript
// zeroth element of array of length 1
var foo = ajax.responseXML.getElementsByTagName("foo")[0];

// same
var bar = foo.getElementsByTagName("bar")[0];

// array of length 2
var all_bazzes = foo.getElementsByTagName("baz");

// string "bleep"
var bloop = foo.getAttribute("bloop");
```

Recall: Pitfalls of the DOM

Using the same file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<foo bloop="bleep">
  <bar/>
  <baz><quux/></baz>
  <baz><xyzzy/></baz>
</foo>
```

XML

We are reminded of some pitfalls of the DOM:

```javascript
// works - XML prolog is removed from document tree
var foo = ajax.responseXML.firstChild;

// WRONG - just a text node with whitespace!
var bar = foo.firstChild;

// works
var first_baz = foo.getElementsByTagName("baz")[0];

// WRONG - just a text node with whitespace!
var second_baz = first_baz.nextSibling;

// works - why?
var xyzzy = second_baz.firstChild;
```

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>
</bookstore>
```
XML

Navigating the node tree

- don't have `id`s or `class`es to use to get specific nodes
- `firstChild`/`nextSibling` properties are unreliable
- best way to walk the tree is using `getElementsByTagName`:

```js
node.getElementsByTagName("tagName")
```
JS

  - get an array of all *node*'s children that are of the given tag (`"book"`, `"subject"`, etc.)
  - can be called on the overall XML document or on a specific node

-
  ```js
  node.getAttribute("attributeName")
  ```
  JS

  - gets an attribute from a node (e.g., `category`, `lang`)
- Prototype methods also useful: `childElements`, `siblings`, `next`/`previous`, etc.

Navigating node tree example

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

    // make an XHTML <p> tag based on the book's XML data
    var p = document.createElement("p");
    p.textContent = title + ", by " + author;
    document.body.appendChild(p);
  }
}
```
JS

A historical interlude: why XHTML?

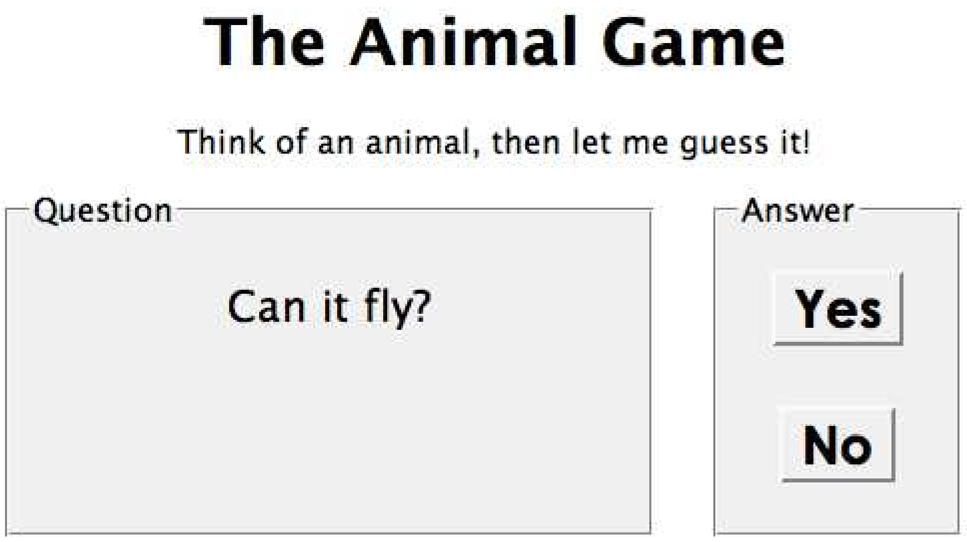
- in XML, different "flavors" can be combined in single document
- theoretical benefit of including other XML data in XHTML
  - nobody does this
- most embedded data are in non-XML formats (e.g., Flash)
  - non-XML data must be embedded another way (we'll talk about this later on)
- requires browser/plugin support for other "flavor" of XML
  - development slow to nonexistent
  - most XML flavors are specialized uses

Why XML in AJAX?

- most data you want are provided in XML
  - the *de facto* universal format
- the browser can already parse XML (i.e., XHTML) into DOM objects
  - DOM only defined for XML-based formats, may not map directly to another format
- would have to manually parse a different format
  - simple formats can be parsed manually from `ajax.responseText`
  - most data are easier to manipulate as DOM objects than to parse manually

Practice problem: 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`.



Practice problem: Animal game (cont'd)

- The data comes in the following format:

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

```xml
<node nodeid="id">
  <answer>answer</answer>
</node>
```

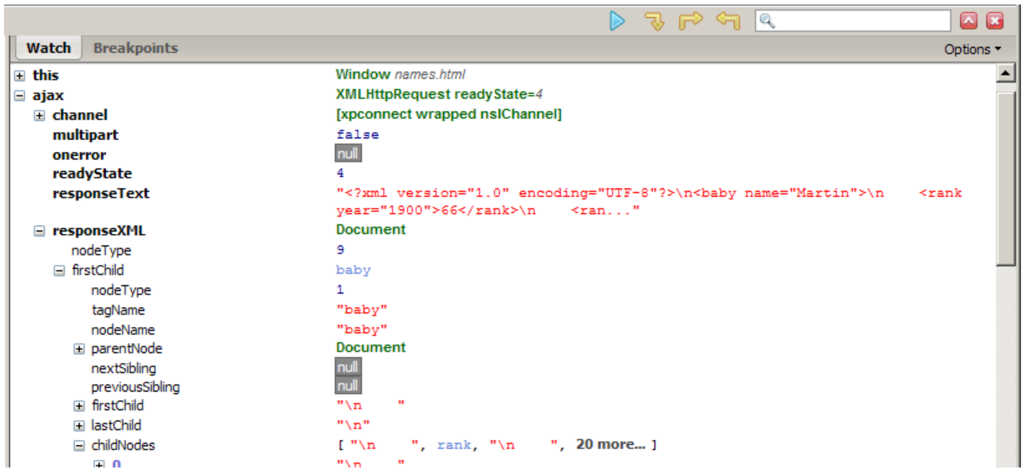
- to get a node with a given id: `animalgame.php?nodeid=id`
- start by requesting the node with `nodeid` of `1` to get the first question

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