Lecture 13
Pragmatic XML

Play

Outline

- XML Basic
- XML in PHP
- XPath
- XML and Ajax
- Programming with XML

What is XML?

- General specification for creating markup languages
- Data exchange between computer systems
  - System independent
  - Human readable
  - Most used on the web
  - Successor of SGML
- W3C recommendation
  - 1.0 1998 (last update 2008-11-26)
  - 1.1 2004 (last update 2006-08-16)

Use 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 by example

- XHTML
  - XML variant of the popular HTML
- RSS
  - Really Simply Syndication
  - Provide news / updates / … of websites
  - Read by special clients
  - Aggregation on portals / planets
- SVG
  - Scalable Vector Graphics
  - Describe vector graphics in XML
  - Potentially interactive / animated (via ECMAScript)

Related technologies - Schemas

- Schema
  - A schema defines the structure XML instance documents.
- XMLSchema
  - Written in XML
  - W3C recommendation
  - Popular
- RelaxNG
  - 2 syntax variants
    - XML based
    - Short plain text based
  - OASIS / ISO standard
  - Popular
- DTD
  - Plain text
  - W3C recommendation
  - Deprecated

Related technologies - Querying

- Query
  - A query extracts a sub-set of information from a data source.
- XPath
  - W3C recommendation
  - Navigation in XML documents
  - more on that later...
- XQuery
  - Functional programming language
  - Allows complex queries

An XML Document Example

Preamble

```
<?xml version="1.0" encoding="UTF-8"?>
<bookshelf>
    <book id="1">
        <title lang="en">Beautiful code</title>
        <author>A. Oram</author>
        <author>G. Wilson</author>
        <year>2007</year>
        <price currency="Euro">35.95</price>
    </book>

    <book id="2">
        <title lang="de">eZ Components - Das Entwicklerhandbuch</title>
        <author>T. Schlitt</author>
        <author>K. Nordmann</author>
        <year>2007</year>
        <price currency="Euro">39.95</price>
    </book>
</bookshelf>
```

Document element

Element

--can be nested

Attribute

Content

End Tag

Start Tag

XML Terminology

- tags: book, title, author, ...
- start tag: <book>, end tag: </book>
- elements: <book>...<book>,<author>...</author>
- elements are nested
- empty element: <red></red> abbrv. <red/>
- an XML document: single root element
- Attributes
- Name spaces

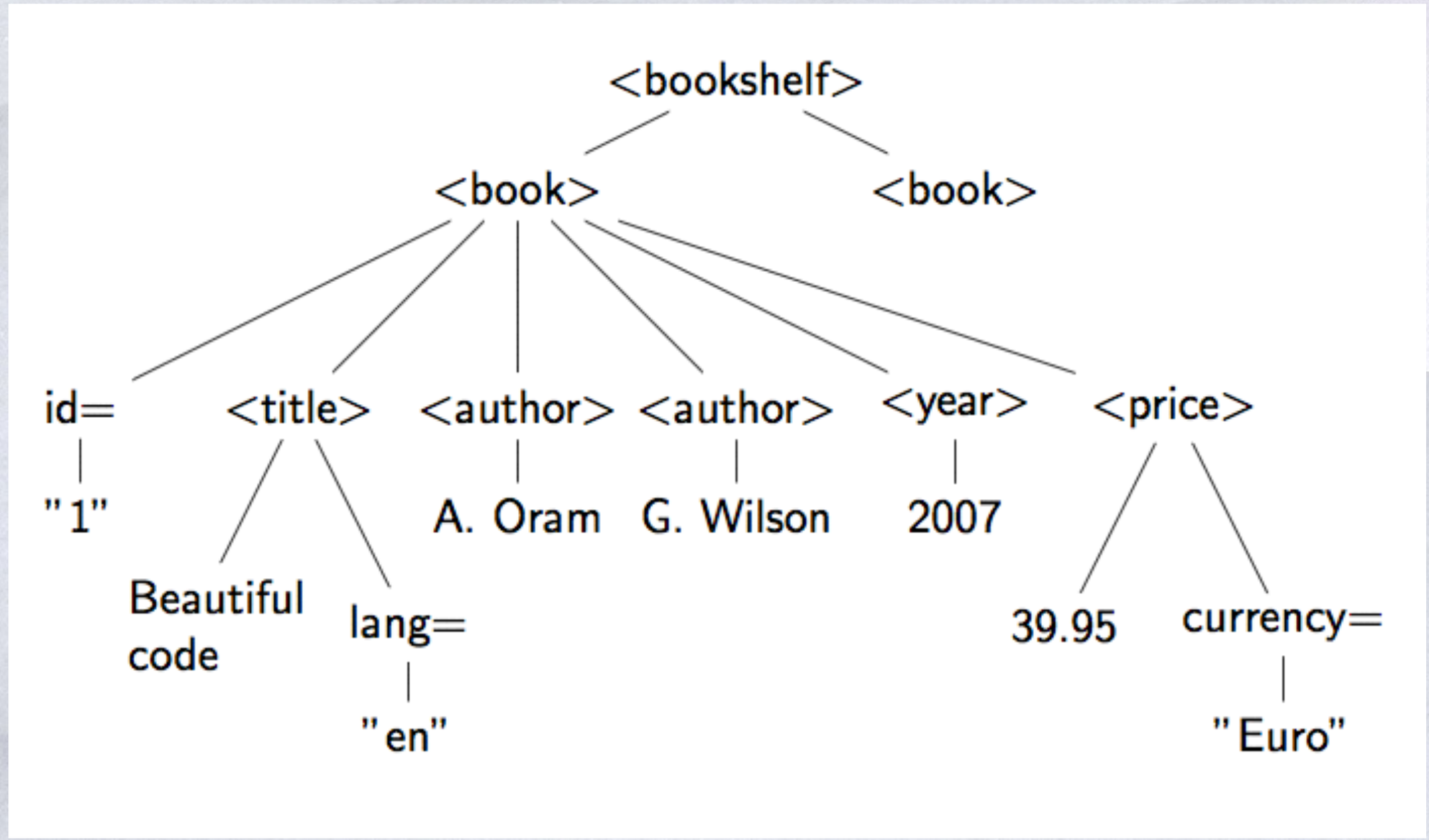
Pros and cons of XML

- 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 a good XML format

What tags are legal in XML?

- any tags you want!
- examples:
  - an email message might use tags called to, from, subject
  - a library might use tags called book, title, author
- when designing an XML file, you choose the tags and attributes that best represent the data
- rule of thumb: data = tag, metadata = attribute

The XML tree


<bookshelf>
<book>
<book>
id=
"1"
<title>
<author>
<author>
<year>
<price>
A. Oram
G. Wilson
2007
Beautiful
code
lang=
"en"
39.95
currency=
"Euro"

CDATA

CDATA:Avoid the escaping hell in text content.

Without CDATA

```
<bookshelf>
    <book id=”1”>
        <title lang=”en”>Beautiful code</title>
        <hint>
            Some examples make use of &lt;xml&gt;.
        </hint>
    </book>
</bookshelf>
```

With CDATA

```
<bookshelf>
    <book id=”1”>
        <title lang=”en”>Beautiful code</title>
        <hint>
            <![CDATA[
            Some examples make use of <xml>.
            ]]>
        </hint>
    </book>
</bookshelf>
```

The CDATA dilemma

```
<bookshelf>
    <book id=”1”>
        <title lang=”en”>Beautiful code</title>
        <hint>
            <![CDATA[
                Some examples show the usage of <![CDATA[]]>
            ]]> </hint>
    </book>
</bookshelf>
```

The CDATA dilemma workaround

```
<bookshelf>
  <book id=”1”>
      <title lang=”en”>Beautiful code</title>
      <hint>
        <![CDATA[
        Some examples show the usage of <!
        [CDATA[]]]><![CDATA[>
        ]]>
      </hint>
  </book>
</bookshelf>
```

The CDATA dilemma solution

- Base 64 in PHP
  - Use the built in functions base64 encode() and base64 decode().

```
<bookshelf>
   <book id=”1”>
      <title lang=”en”>Beautiful code</title>
      <hint>
      U29tZSBleGFtcGxlcyBzaG93IHRoZSB1c2FnZSBvZiA8IVtDREFUQVsgX
      V0+
      </hint>
   </book>
</bookshelf>
```

Comments

```
<bookshelf>
   <book id=”1”>
      <title lang=”en”>Beautiful code</title>
      <author>A. Oram</author>
      <author>G. Wilson</author>
      <year>2007</year>
      <price currency=”Euro”>35.95</price>
   </book>

   <!–– ... more books ... ––>

</bookshelf>
```

Namespaces

- Allow to avoid naming conflicts between different XML sources.

Single, default namespace

```
<bookshelf xmlns="http://example.com/book">
  <book id="1">
    <title lang="en">Beautiful code</title>
    <author>A. Oram</author>
    <author>G. Wilson</author>
    <year>2007</year>
    <price currency="Euro">35.95</price>
  </book>

</bookshelf>
```

Multiple namespaces

Multiple namespaces

```
<bookshelf
   xmlns =”http://example.com/book”
   xmlns:book=”http://example.com/book”
   xmlns:dc=”http://purl.org/dc/elements/1.1/ ”
>

   <book id=”1”>
      <dc:title book:lang=”en”>Beautiful code</dc:title>
      <author>A. Oram</author>
      <author>G. Wilson</author>
      <year>2007</year>
      <price currency=”Euro”>35.95</price>
   </book>

</bookshelf>
```

Outline

- XML Basic
- XML in PHP
- XPath
- XML and Ajax
- Programming with XML

XML in PHP

- XML APIs
  - PHP has quite some XML APIs.
- The most important are:
  - DOM
  - XMLReader/-Writer
  - SimpleXml (by example)
- Deprecated are:
  - DOM XML
  - XML Parser

Overview - DOM

- Document Object Model
- Standardized API to access XML tree
  - W3C recommendation
  - Level 1 in 1999
  - Currently: Level 3 (2004)
- Available in many languages
  - C
  - Java
  - Perl
  - Python ...
- Represents XML nodes as objects
- Loads full XML tree into memory

Overview - XMLReader/-Writer

- Popular approach to access XML data
- Similar implementations available in
  - Java
  - C#
- Pull / push based
- Does not load XML fully into memory

Overview - SimpleXml

- Very simple access to XML data
- Represents XML structures as objects
- Loads full XML tree into memory
- You don’t want to use SimpleXML, seriously!

APIs compared

	DOM	XMLReader/-Writer	SimpleXML
Read	⋆	⋆	●
Write	⋆	⋆	○
Manipulate	⋆	●	●
Full control	⋆	⋆	-
Namespaces	⋆	⋆	○
XPath	⋆	-	⋆
Validate	DTD Schema RelaxNG	DTD Schema RelaxNG	-
Comfort	●	○	⋆

- ⋆ Fully supported
- ● Supported but not nice
- ○ Poorly supported
- \- Not supported at all

Outline

- XML Basic
- XML in PHP
- XPath
- XML and Ajax
- Programming with XML

XPath

- Enables you to select information parts from XML documents
  - Traverse the XML tree
  - Select XML nodes
- W3C recommendation
  - Version 1: November 1999
  - Version 2: January 2007
- Fields of application
  - XSLT (XML Stylesheet Language Transformations)
  - Fetching XML nodes within programming languages

XML example reminder

```
<bookshelf>
  <book id="1">
    <title lang="en">Beautiful code</title>
    <author>A. Oram</author>
    <author>G. Wilson</author>
    <year>2007</year>
    <price currency="Euro">35.95</price>
  </book>
  <book id="2">
    <title lang="de">eZ Components - Das Entwicklerhandbuch</title>
    <author>T. Schlitt</author>
    <author>K. Nordmann</author>
    <year>2007</year>
    <price currency="Euro">39.95</price>
  </book>
</bookshelf>
```

2 variants to fetch all books

```
/bookshelf/book
// book
```

Addressing

- Every XPath expression matches a set of nodes (0..n)
- It encodes an “address” for the selected nodes
- Simple XPath expressions look similar to Unix file system addresses
- Two generally different ways of addressing are supported

Absolute addressing

/bookshelf/book/ title
/bookshelf/book/author

Relative addressing

book/author
../ title

Contexts

- Every expression step creates a new context
- The next is evaluated in the context created by the previous one

Contexts

/bookshelf/book/title

/	resets the context to global
bookshelf	selects all <bookshelf> elements in the global context
/	creates a new context, all children of <bookshelf>
book	selects all <book> elements in this context
/	creates a new context, all children of selected <book>s
title	selects all <title> elements in this context

→ A set of all title element nodes.

Attributes

- Select attributes
  - Prepend the attribute name with an @
- Select all currency attributes
  - /bookshelf/book/price/@currency

Steps

- Navigation
  - Navigation is not only possible in parent → child direction.

```
 Navigate to parent
../
/bookshelf/book/title /../author
```

```
 Navigate to descendants
// t i t l e
/bookshelf/book//@currency
```

Indexing

- Access nodes by position
  - It is possible to access a specific node in a set by its position.

Indexing
/bookshelf/book [2]
/bookshelf/book/author [1]

Start index
- Indexing generally 1 based
- Some Internet Explorer versions start with 0

Wildcards

- Wildcard search
  - A wildcard represents a node of a certain type with arbitrary name

Wildcards

/bookshelf/*/title
/bookshelf/book/@*

Union

- Union the node sets selected by multiple XPath expressions.

Union

```
/bookshelf/book/title |/bookshelf/book/author
```

A first PHP example

Querying first book title

```
$dom = new DOMDocument();
$dom–>load( ’sources/example.xml’ );

$xpath = new DOMXPath( $dom ) ;

$titles = $xpath–>query( ’/bookshelf/book[1]/ title ’ );

echo ’Title of first book is ’
      . $titles–>item( 0 )–>nodeValue . ”\n”;
```

Output

```
Title of first book is Beautiful code
```

Second PHP example

Querying all currencies

```
// ...

$currencies = $xpath–>query( ’//price/@currency ’ );

echo ”Following currencies occur :\n”;

foreach ( $currencies as $currency )
{
   echo $currency–>nodeValue . ”\n”;
}
```

Output

```
Following currencies occur:
Euro
Euro
```

XPath syntax

- An XPath query consists of steps
- Each step consists of:
  - axis:
    - child (default)
    - attribute (@)
    - descendant-or-self (//)
    - parent (..)
  - Node tests:
    - name of the node (default)
    - wildcard (*)
  - Predicate:
    - accessing a node by index

Step syntax

<axis >::<nodetest>[<predicate >]

Axis syntax

- <axisname >::<nodetest>

Child axis

```
/bookshelf/book
child :: bookshelf/child :: book
```

Descendant-or-self axis

```
// book
descendant–or–self::book
```

Attribute axis

```
//book/@id
descendant–or–self :: book/attribute :: id
```

Parent axis

```
//book/@id /..
descendant–or–self :: book/attribute :: id/parent :: node()
```

Ancestor axis

XPath with ancestor axis

```
//title/ancestor::*
```

Selected XML nodes

```
<?xml version="1.0" encoding="UTF-8"?>
<bookshelf>
   <book id="1">
      <title lang="en">Beautiful code</title>
      <author>A. Oram</author>
      <author>G. Wilson</author>
      <year>2007</year>
      <price currency="Euro">35.95</price>
   </book>
   <!-- ... -->
</bookshelf>
```

Following axis

XPath with ancestor axis

```
//book/following ::*
```

Selected XML nodes

```
<?xml version="1.0" encoding="UTF-8"?>
<bookshelf>
  <book id="1">
    <title lang="en">Beautiful code</title>
    <!-- ... -->
  </book>

  <book id="2">
    <title lang="de">eZ Components - Das Entwicklerhandbuch</title>
    <!- ... ->
  </book>
</bookshelf>
```

Following-sibling axis

XPath with ancestor axis

```
//book/following-sibling ::*
```

Selected XML nodes

```
<?xml version="1.0" encoding="UTF-8"?>
<bookshelf>
  <book id="1">
    <title lang="en">Beautiful code</title>
    <!-- ... -->
  </book>

  <book id="2">
    <title lang="de">eZ Components - Das Entwicklerhandbuch</title>
    <!- ... ->
  </book>
</bookshelf>
```

Namespace axis

XPath with namespace axis

namespace : : *

```
<bookshelf
   xmlns=”http://example.com/book”
   xmlns:book=”http://example.com/book”
   xmlns:dc=”http://purl.org/dc/elements/1.1/”
>
   <book id=”1”>
      <dc:title book:lang=”en”>Beautiful code</dc:title>
      <author>A. Oram</author>
      <author>G. Wilson</author>
      <year>2007</year>
      <price currency=”Euro”>35.95</price>
      </book>
</bookshelf>
```

Predicate syntax

- You already saw indexing with numeric predicates
- Predicates can also be booleans
- Functions and operators allow fine grained tests

Predicate syntax

```
<nodetest >[<predicate >]
```

Select all books with ID 1

```
//book[@id = ’1’]
```

Select all books that have any attribute at all

```
//book [@*]
//book/@*/..
```

Select all books with price round 40

```
//book[round( price ) = 40]
```

Select all authors with first name initial T

```
//book/author [ substring (. , 1, 1) = ’T’]
```

Operator overview

- Mathematical operators
  - +, −, *: Addition, subtraction, multiplication
  - div: Division
  - mod: Modulo operation
- Comparison operators
  - =: Check for equality
  - !=: Check for inequality
  - <, <=: Less than and less than or equal
  - >, >=: Greater than and greater than or equal
- Logical operators
  - or: Logical or
  - and: Logical and
- Logical negation
  - not() is a function in XPath!

Functions by example

- String functions
  - string-join() Concatenates 2 strings
  - substring() Extracts a part from a string
- Node set functions
  - count() Returns number of nodes in a set
  - position() Returns the position index of each node
- Boolean functions
  - not() Negates the received boolean expression
  - true() Boolean true
- Mathematical functions
  - round() Rounds the given number to the next integer
  - floor() Returns the next integer smaller than the given number
- Function overview
  - An overview on all functions can be found on http://www.w3.org/TR/xpath-functions/

Outline

- XML Basic
- XML in PHP
- XPath
- XML and Ajax
- Programming with XML

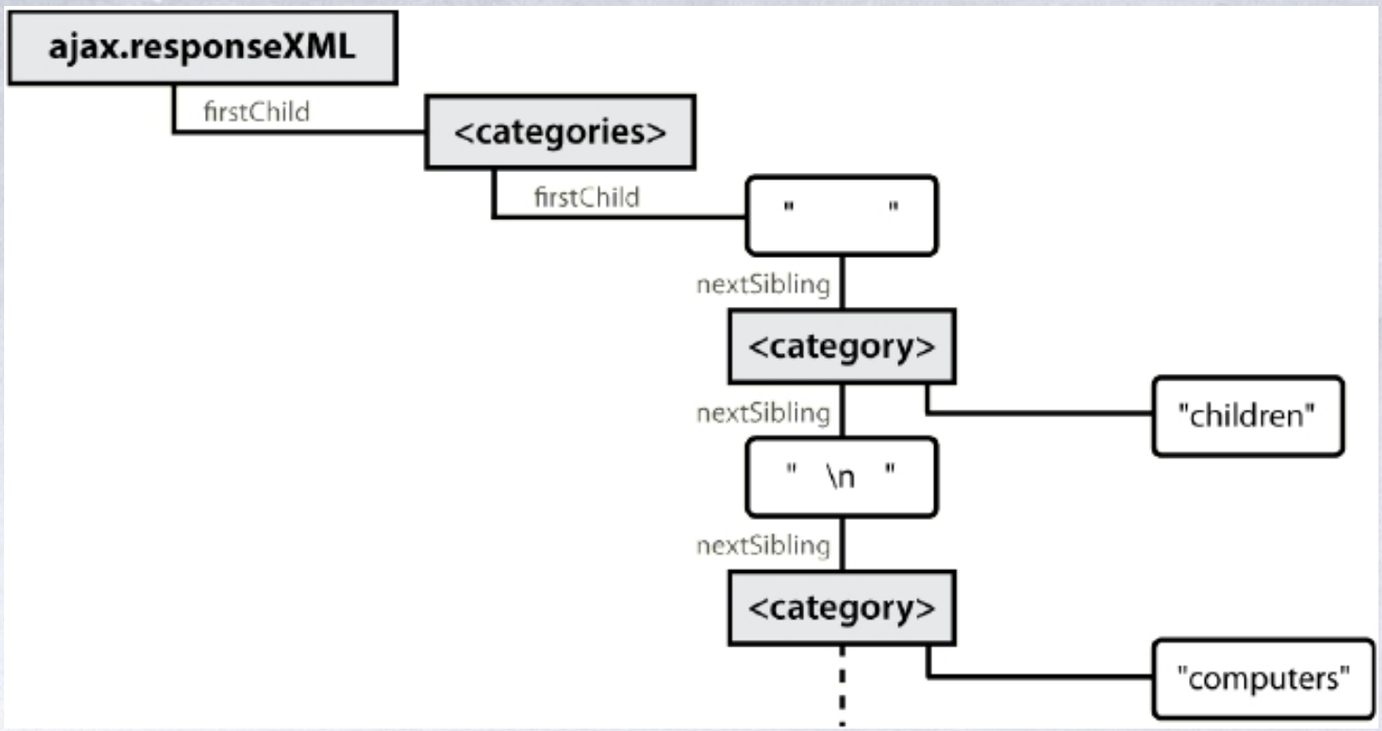
XML and Ajax

- 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 chunky 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, very similar to the (X)HTML DOM

XML DOM tree structure

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

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



Recall: Javascript XML (XHTML) DOM

- 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
- caution: cannot use HTML-specific properties like innerHTML in the XML DOM!

  * (though not Prototype's, such as up, down, ancestors, childElements, descendants, or siblings)

Navigating the node tree

- caution: can only use standard DOM methods and properties in XML DOM. HTML DOM has Prototype methods, but XML DOM does not!
- caution: can't use ids or classes to use to get specific nodes
  - id and class are not necessarily defined as attributes in the flavor of XML being read
- caution: firstChild/nextSibling properties are unreliable
  - annoying whitespace text nodes!
- the best way to walk the XML tree:

```
var elms = node.getElementsByTagName("tagName")
```
JS

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

Outline

- XML Basic
- XML in PHP
- XPath
- XML and Ajax
- **Programming with XML**

Using XML data in a web page

Procedure:

1. use Ajax to fetch data
2. use DOM methods to examine XML:
   - XMLnode.getElementsByTagName()
3. extract the data we need from the XML:
   - XMLelement.getAttribute(), XMLelement.firstChild.nodeValue, etc.
4. create new HTML nodes and populate with extracted data:
   - document.createElement(), HTMLelement.innerHTML
5. inject newly-created HTML nodes into page
   - HTMLelement.appendChild()

Fetching XML using Ajax (template)

- ajax.responseText contains the XML data in plain text
- ajax.responseXML is a pre-parsed XML DOM object

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

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

Analyzing a fetched XML file using DOM

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

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

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

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

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

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

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

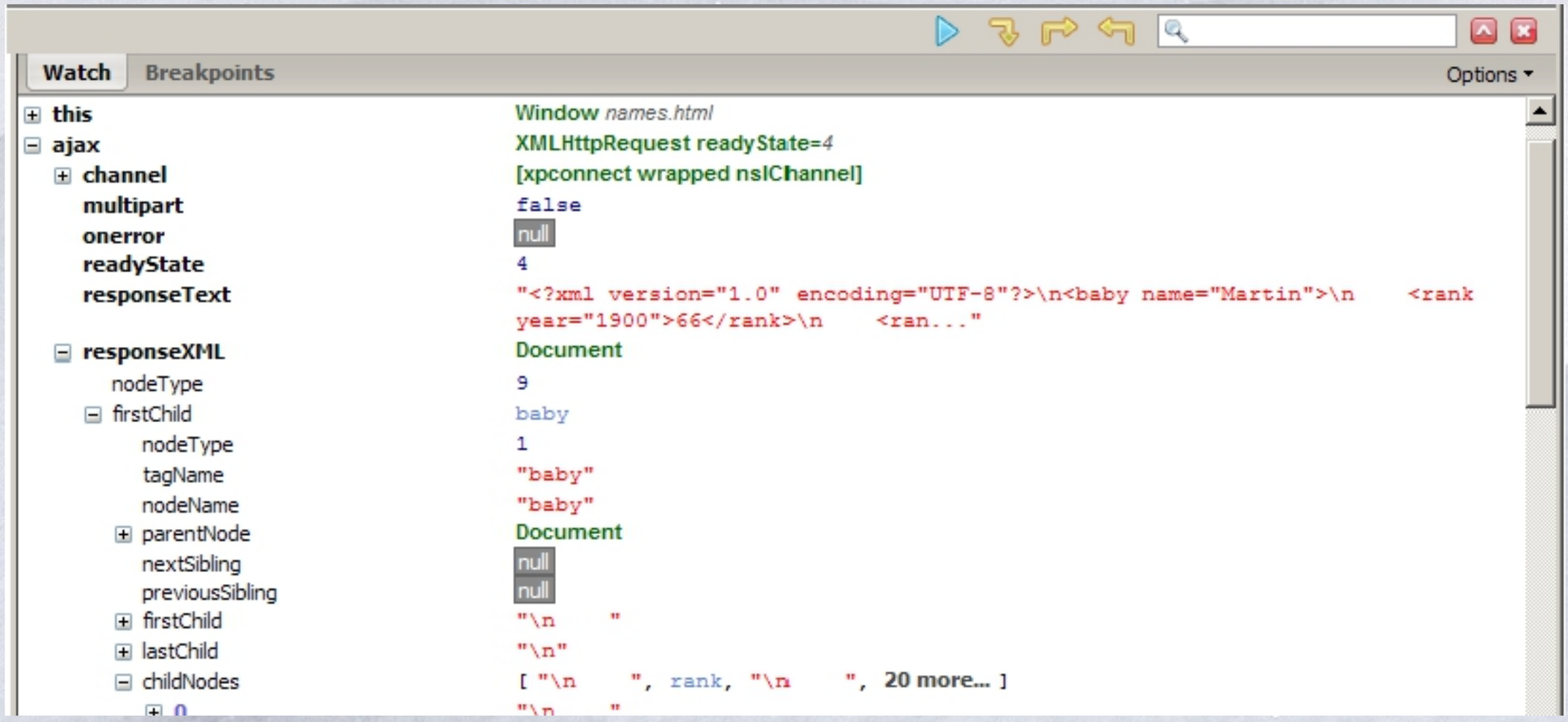
Navigating node tree example

```
// 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") {
    // extract data from XML
    var title = books[i].getElementsByTagName("title")[0].firstChild.nodeValue;
    var author = books[i].getElementsByTagName("author")[0].firstChild.nodeValue;

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

JS

Debugging responseXML in Firebug



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

Thanks!!!